

1992

An intelligent database for PSUBOT, an autonomous wheelchair

Dieudonne Mayi
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Mayi, Dieudonne, "An intelligent database for PSUBOT, an autonomous wheelchair" (1992). *Dissertations and Theses*. Paper 4332.

<https://doi.org/10.15760/etd.6216>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF Dieudonne Mayi for the Master of Science in Electrical and Computer Engineering presented February 7, 1992.

Title: An Intelligent Database for PSUBOT, an Autonomous Wheelchair.

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:



Marek A. Perkowski, Chair



Michael A. Driscoll



Maria Balogh



Jean Scholtz

In the design of autonomous mobile robots, databases have been used mainly to store information on the environment in which the device is to operate. For most of the models and ready systems, the database when used, is not a stand alone component in the system, rather it is only intended to keep static information on the disposition and properties of objects on the map.

In this thesis is implemented an intelligent database. This database is called intelligent because it is knowledge-based. It combines static facts to build more information. An intelligent database such as this one will be a plus for an intended autonomous machine such as the PSUBOT wheelchair being developed in the Department of Electrical Engineering. The database will make intelligent decisions as an intelligent function of the the central control module of the system (i.e, find a global optimum path, recognize details in the building, support sensory integration). The database will also serve as the core of pattern recognition and localization of the wheelchair inside the building.

The thesis starts with the definition of a model of the environment in which the wheelchair is confined to operate, then a relational database is designed to keep this information. The second part of the thesis concentrates on finding a global shortest path using knowledge-based method combined with the Djikstra's shortest path method. The third part of the development consists of the implementation of image matching. Image matching is used to simulate the localization of the wheelchair within the building assuming that the most recent location visited is known. The evaluation of the database is based on the accuracy of the path planning results and the percentage of success of the best match of two images. The percentage of success of localization is measured as the accuracy of the database to recognize a location among a set of candidate locations. The global evaluation of the database rests upon the speed of the processing of information (path planning, image matching and localization) compatible with the operation of the wheelchair at reasonable speed.

The database has been implemented on a PC-386SX with 640K base memory and a clock resolution of 20MHz. It is composed of a relational database, a knowledge base, a set of management routines and programs to perform tasks such as global path planning and image matching for localization. The relational database has been implemented using the BORLAND PARADOX3 database management system. The knowledge base part of

the database has been implemented in a combination of BORLAND Turbo Prolog (for intelligent tasks) and BORLAND Turbo C++ (for tasks requiring faster computation power). Global path planning has been implemented with a knowledge-based approach rather than a conventional graph method in order to match the hierarchical description of a building and to add more intelligence power. Localization is the key problem for which we use image matching. The wheelchair needs to possess the capability to recognize where it is in the building. Therefore, in order to perform this task, the matching of the current image of the scene with template images of candidate locations is performed. The image matching has been implemented to match two images described each as a set of straight lines. The matching method is correspondence matching between the two sets of features with the criteria being the best acceptable match. Image matching is used for localization of the wheelchair inside the building based on matching of the currently perceived image with template images of locations previously stored in memory.

**AN INTELLIGENT DATABASE FOR PSUBOT,
AN AUTONOMOUS WHEELCHAIR**

by

DIEUDONNE MAYI

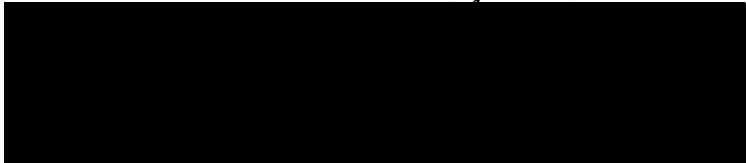
A thesis submitted in in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING**

**Portland State University
1992**

TO THE OFFICE OF GRADUATE STUDIES:

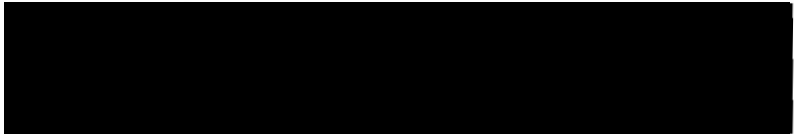
The members of the Committee approve the thesis of Dieudonne Mayi
presented February 7, 1992.



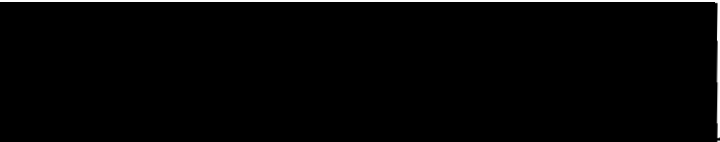
Marek A. Perkowski, Chair



Michael A. Driscoll

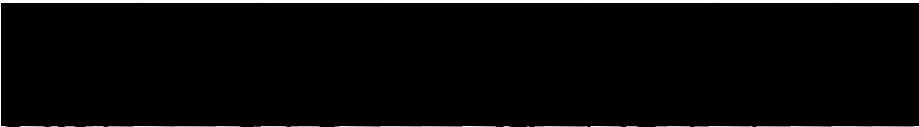


Maria Balogh

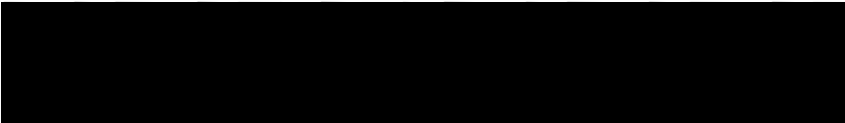


Jean Scholtz

APPROVED:



Rolf Schaumann, Chair, Department of Electrical Engineering



C. William Savery, Vice Provost for Graduate Studies and Research

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser Dr. Marek A. Perkowski who encouraged me, counseled me, and patiently guided me throughout this work.

I also would like to thank Dr. Michael Driscoll, Dr. Maria Balogh and Dr. Jean Scholtz for their suggestions, critiques and comments to make this thesis a better work.

I express my gratitude to the PSUBOT team, especially Kevin Stanton and Cecilia Espinoza for their ideas and critiques.

I thank all those who have helped me with the writing of this thesis.

I would like to thank the US Agency for International Development and the Government of Cameroon for sponsoring my studies during this training program.

I would like especially to thank my wife Bibiane Mayi, my friends Abdi Hassan, Monique Grone and Dan Shea who have counseled and supported me spiritually during difficult moments in the completion of this work.

Finally, I will never express my gratitude enough to God for helping me with His grace to complete this work.

Portland, Oregon

Dieudonne Mayi

TABLE OF CONTENTS

	PAGE
ACKNOWLEDGEMENTS	iii
LIST OF TABLES	vii
LIST OF FIGURES	viii
 CHAPTER	
I INTRODUCTION	1
I.1 The PSUBOT Wheelchair	3
I.2 Problem Statement	7
I.3 Basic Assumptions	8
I.4 Development Methodology	9
I.5 Goals	11
I.6 Evaluation	12
II AUTONOMOUS MOBILE ROBOTS: THE NAVIGATION AND LOCALIZATION PROBLEMS	14
II.1 Models of Autonomous Wheelchairs and Mobile Robots	14
II.2 Navigation of Autonomous Mobile Robots: Definition of the Problem	26
II.3 Localization of Autonomous Mobile Robots	28
III MODEL OF THE WORLD	30
III.1 Definition of the Problem	30
III.2 Environment Modelling for Autonomous Mobile Robots ..	32

	III.3	A priori Knowledge: Description of the World Around the Robot	34
	III.4	Models of Selected Patterns in the Indoor Scene	45
IV		DATABASES, OBJECT BASES, AND KNOWLEDGE BASES	47
	IV.1	Introduction	47
	IV.2	Database Management Systems	50
	IV.3	Data Models for Database Systems	57
	IV.4	Relational Databases	60
	IV.5	Knowledge Bases	62
	IV.7	Representation of Data in Mobile Robots	74
V		PATH PLANNING FOR MOBILE ROBOTS	76
	V.1	General Considerations	76
	V.2	Different Approaches to Path Planning	78
	V.3	Optimum Path Problem	81
	V.4	Global Path Planning for PSUBOT	84
	V.5	Implementation of Global Path Planning.....	91
VI		PSUBOT DATABASE ORGANIZATION	100
	VI.1	General Characteristics and Assets of the Database Approach	100
	VI.2	Overview of the Database System	102
	VI.3	The Static Relational Database	104
	VI.4	The Dynamic Knowledge Base	112
	VI.5	The Management Routines	114
	VI.6	The Interface Routines.....	123
	VI.7	Interconnection with the Other Modules of the PSUBOT System	123

VII	MATCHING TWO IMAGES	125
	VII.1 Definition of the Problem	125
	VII.2 Different Approaches to the Matching Problem	133
	VII.3 Image Matching for the PSUBOT Wheelchair:	
	Proposed Approach	138
	VII.4 A Structural Approach to Image Matching:	
	Proposed Methodology	155
VIII	LOCALIZATION OF THE WHEELCHAIR	160
	VIII.1 Problem Statement	160
	VIII.2 Localization Strategy for PSUBOT	161
IX	TESTING AND EVALUATION OF THE DATABASE SYSTEM	167
	IX.1 Justification of the Choice of Languages	167
	IX.2 Evaluation of the Database	168
X	CONCLUSIONS AND FUTURE WORK	178
	X.1 Conclusions	178
	X.2 Future Work	180
	REFERENCES	181
	APPENDICES	
	A EXAMPLES OF APPLICATION	185
	B PSEUDOCODES OF PROGRAMS	192
	C SOURCE CODES OF PROGRAMS	205

LIST OF TABLES

TABLE		PAGE
I	Experimental Results with Habib's Model	20
II	Evolution of Database Systems	49
III	Path on Corridor: Computation Time as Function of the Number of Observation Points	93
IV	Path Planning Results - Example 1	94
V	Path on Floor: Computation Time as Function of the Number of Observation Points of the Graph	97
VI	Path Planning Results - Example 3	99
VII	Map_table	105
VIII	Buildings_table	106
IX	Bridges_table	106
X	Floors_table	107
XI	Rooms_table.....	107
XII	Corridors_table	108
XIII	Corridor_ends_table	109
XIV	Objects_table	109
XV	Image Matching Results - Image P201	150
XVI	Image Matching Results - Image P203	151
XVII	Results - Simulation of Localization	154
XVIII	Estimation of the Speed of the Wheelchair	172

LIST OF FIGURES

FIGURE	PAGE
1. General Organization of the PSUBOT System	4
2. Hierarchical Description of the Static Map	35
3. Corridor End-points	37
4. Standard Shapes for a Corridor	41
5. Normalization of the Shapes of Corridors	41
6. An Example of a Hall in a Building	42
7. A Simple Example of a Floor Plan	43
8. Simplified Model of the Interior of a Room	44
9. Communication between the DBMS and the Application Program	56
10. General Structure of a Database System	56
11. Diagram of the Hierarchy of Normalization	62
12. Hierarchical Approach to Global Path Planning	85
13. Example of a Problem Showing the Hierarchical Approach to Global Path Planning.....	86
14. Query Simplification	88
15. A Problem from Madarasz's Model	93
16. Evaluation of the Speed of the Path Planning Algorithm	93
17. Computation Time vs Number of Observation Points on the Corridor	96
18. Computation Time vs the Problem Size	97
19. Structure of the PSUBOT Database System	103

20. Communication between the Database and the other Modules of the PSUBOT System.....	124
21. Adding Uncertainty to Image Matching and Feature Extraction	130
22. An Output Image from Low-to-medium Image Processing [3] (level 1 to 4).....	131
23. An Output Image from Low-to-medium Image Processing [3] (level 5 to 8).....	132
24. Correspondence Matching of Two Images	141
25. Criteria for Matching of Two Lines	143
26. Experiment 1: Changing the Angle of the Camera	144
27. Line Extraction Process for Image P201 (level 1 to 3)	145
28. Line Extraction Process for Image P201 (level 4 to 7).....	146
29. Template Images of Location P201 (+3 degrees to +10 degrees).....	147
30. Template Images of Location P201 (+15, +20, -3, -6 degrees)	148
31. Template Images of Location P201 (-8 degrees to -20 degrees)	149
32. Computation Time versus Total Number of Lines, Image P201.....	150
33. Image P203 and Disposition of Locations in the Room	152
34. Computation Time versus Total Number of Lines, Image P203	152
35. Locations P202 and P204	153
36. Image Matching Approach using Higher Level Features	157
37. Models of Higher Level Features: Corners, Rectangles, Parallelogram	158
38. Models of Higher Level Features, Corridor	158
39. Extraction of a Simple Corner	159
40. Floor Plan of the First Floor of the Building PCAT	186
41. Floor Plan of the Basement of the Building PCAT	187
42. Floor Plan of the First Floor of the Building B1	187

43. Problem Formulation as an Edged Graph	
(basement of PCAT and first floor of B1)	188
44. Problem Formulation as an Edged graph	
(first floor of PCAT).....	188

CHAPTER I

INTRODUCTION

This thesis involves the integration of an intelligent database to the PSUBOT autonomous wheelchair system [1, 2, 3]. The PSUBOT wheelchair is an autonomous electric wheelchair capable of understanding voice commands from the user and being able to navigate by itself in an indoor environment while carrying a paraplegic or blind person. An autonomous system such as the PSUBOT should be able not only to use information that is provided to it (for example sensor readings and world description), but as an autonomous device it should be able to build more knowledge from the data in order to accomplish its task. In this sense the major contribution of the intelligent database to the PSUBOT system is to be the core of intelligence for the wheelchair, scene recognition and intelligent localization of the robot.

In the literature, many models of autonomous mobile robots and wheelchairs have been proposed. Each model tries to overcome a particular problem, for example, drop-off detection, navigation and collision avoidance, uncertainty consideration, model of the world etc.. We will refer to some of these models which in our judgement are close to the PSUBOT, either because of some aspect of their formulation or because the intended user is a handicapped person.

The model proposed by Madarasz [4] is an autonomous electric wheelchair, equipped with an on-board computer, a TV camera and range sensors. This model can compute its global path and navigate inside the building at very low speed. The data representing the model of the world are stored in simple file structures. The model does not explicitly use knowledge-based method to build more information from the static map.

The model proposed by Rao et Kuc [5] is a prototype without any intelligence mechanism at all. It has the particularity of detecting drop-offs on sharp edges. The model of the world is a confined work space with obstacles. No specific database structure is used.

Habib and Yuta [6] use a model with a well defined world model stored in a relational database. However, the information is static and used as such. There is not a knowledge support mechanism per se.

As it will be expanded in Chapter II, none of those systems use a database as a unified module of the system and the task of intelligence handling is not done at the level of the database. This is a major difference of the PSUBOT approach to have a stand-alone database. Such a realization seems to have the following main advantages:

- 1) the database is the center of all the intelligence handling and keeps all the data about the PSUBOT. Therefore, the status of the wheelchair operations can be well known in all the modules of the system.
- 2) a realization around an intelligent database is more flexible in a sense that more information can be built from available information. For example, the neighborhood of a room on a corridor can be retrieved as the set of rooms located on that corridor. The available information is the corridor and the set of objects located along it (rooms, obstacles and other observation points). The new information built is the neighborhood of a given room of that corridor. Therefore, the intelligent database allows the integration of more intelligent algorithm because of its knowledge-based nature. Also, sensor integration can be done because the perception of the environment will be the result of the combined model retrieved from all the sensors (vision and sonar).
- 3) An intelligent database using knowledge capability (a language such as Prolog being

an example) can be used by the user to ask on-line queries and to obtain answers from the database. The voice-queries from the user will be interpreted into knowledge queries (declarative queries) and an answer can be given back to the user in the same way. The database will store a query base for the system. It can be pointed out that queries written in a declarative language such as Prolog have several advantages while interfacing the application program to a standard relational database [7].

4) Finally, an intelligent database of this kind allows for Artificial Intelligence approach of pattern matching on abstract level and intelligent localization of the wheelchair. The structured method of pattern recognition [8] allows storage of less information but ability to build more complex information from the basic knowledge. An example of application can be found in Recognition By Components technique [9, 10]. Instead of storing the whole model as one entity of information, its basic parts can be stored and the whole model can be addressed using rules which refer to relations between the basic facts. Pattern matching at the abstract level can be used in image matching in order to provide at the same time information on recognizable details in the scene. Intelligent localization necessitates the definition of models of items to recognize at certain locations inside the building. These items serve as landmarks for the localization of the wheelchair.

I.1 THE PSUBOT WHEELCHAIR

PSUBOT is a conventional electric wheelchair that is being automatized by adding sensory (visual and sonar) perception, and intelligent knowledge mechanisms to make decisions and route guidance capabilities. The goal is to make it completely autonomous and self contained. The PSUBOT is intended to navigate inside a building, carrying a handicapped person who may be a paraplegic in the worse case, or a blind person or any motor-impaired user. The system (see Figure 1) includes three main modules: the

Navigation module (Navigator and Pilot [2]), the sensory module (Vision [3], sonar) and the Intelligent database. Currently voice and motor control have been implemented on the wheelchair [1], vision and sonar systems have been implemented and tested but not yet integrated to the wheelchair.

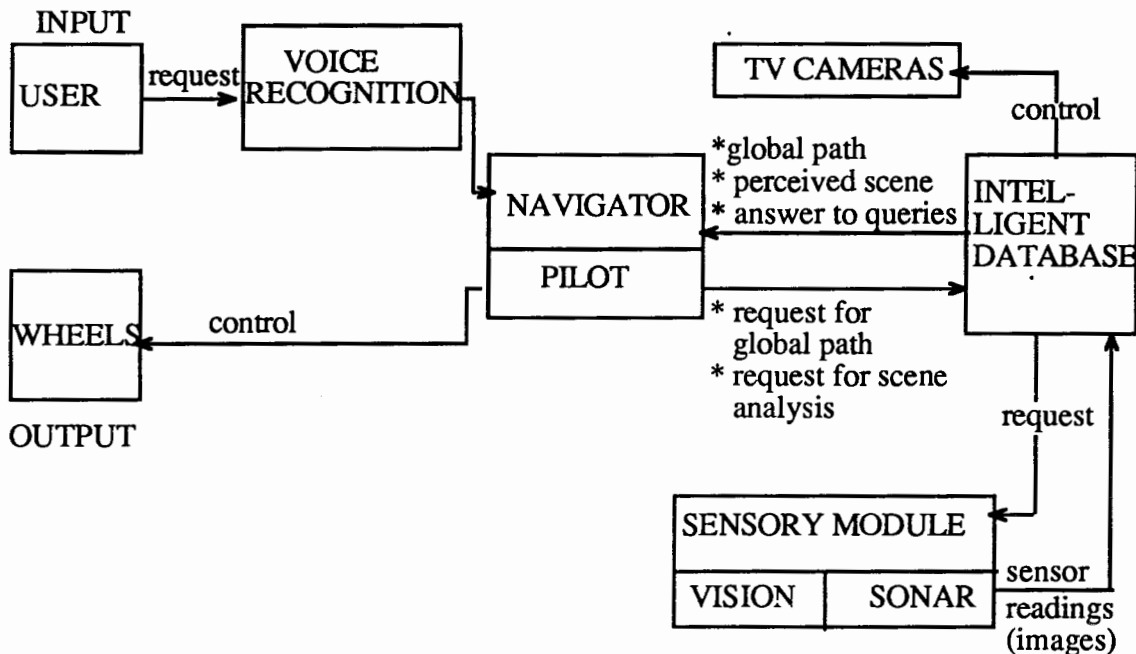


Figure 1. General organization of the PSUBOT system.

The PSUBOT model differs from other autonomous robots and wheelchairs in literature by many respects. First of all, rather than being a general purpose mobile robot, PSUBOT is intended to meet the needs of the paraplegic or the blind. The device is designed with the thought that its user is a severely handicapped. That is why (for example) it is voice activated. Secondly, the PSUBOT is intended, more than many robots of this kind, to be completely self-contained and implemented (vision etc) on PC. Systems that are self contained on PC [6] do not use vision because, regular vision processing is computational intensive and time consuming on PC. A challenge has been to implement

such a vision system on PC with some additional hardware. Thirdly, the structure of the PSUBOT system includes an intelligent database as a stand alone module of the system and dedicated to knowledge processing. In current implementations, databases are used just to store data and are not modules per se of the system. Another source of difference is that the PSUBOT system contrary to many other systems [11, 12], relies much more on vision for localization and navigation of the wheelchair. Other systems rely more on sonar maps and range perception. Vision processing for the PSUBOT implements a new method of line feature extraction using hierarchical Hough Transform. Many systems mix lower level image processing, and medium level image processing in the sense that they extract their lines directly from pixels. Some [13] explain the advantage of doing so as to include uncertainty from the pixel level to the subsequent higher levels. From the point of view of vision, the PSUBOT is particular because it implements low-to-medium image processing fully. This approach deals with higher level image features such as straight lines, which can be used to describe most of the details encountered inside a building. From the higher level line features, more complex higher level features such as door frames, can be extracted. These patterns will be used for scene recognition and structured image matching.

The approach of building systems around a database is not new. However, specifically in this topic, the use of a database as a stand alone module has some essential advantages. The database is responsible for localization of the wheelchair and sensory management. The approach to build the wheelchair system around an intelligent database has some advantages and some pitfalls. The advantages include:

- **Interfacing;** There will be no need to build dedicated interfaces between the modules of the system. All shared data can be stored in the database. Therefore interfacing is between the database and each module of the system as opposed to many interfaces between all modules. This asset becomes especially important when the modules of the system are implemented in different programming languages.

- Global control of the wheelchair; All the data on the wheelchair will be available at any time in the database. Therefore, part or all of the data produced by a module will be at any time available to other modules of the system with some delay due to processing and protection mechanisms.
- Intelligence handling; The active database will support the artificial intelligence mechanisms for the robot. The intelligent database will be the core of scene recognition and image matching, as well as localization of the wheelchair. The database is able not only to store facts, but also to derive new facts. Since the database communicates with all the other parts of the system and keeps all their information, the database will act as a shared memory for all the modules of the system. This structure positions the database as the centralized command of the system.
- Modifying the structure of a critical part of the system such as the Navigator or adding a new module will affect the other modules to a lesser extent; thereby the structure of the whole system will not be considerably affected.

The inconveniences of a centralized database include: the need for data protection because the data will be seen by all the parts of the system, the prevention of the bottleneck situation that may result if the database mechanisms fail or be not fast enough, and the synchronization to avoid false readings by other modules of the data created by one module. It seems, however, that most of these problems can be solved using the distributed Object Oriented approach [14, 15, 16]. The database will be the parent process for the sensory module. Actually the database concentrates on managing the map data, providing global path, acquiring sensor data (vision), and most of all, it will be responsible for the localization of the wheelchair. The general organization of the wheelchair is shown in Figure 1.

I.2 PROBLEM STATEMENT

It appears that for almost all approaches from the references, there are two real issues: the navigation in a real or partial modelled environment, and the robot localization. Sensor imperfection, speed of processing, best modelling of the real world, real time localization of the robot are some of the actual key issues in the navigation of autonomous mobile robots. As stated earlier, sensor imperfection affects navigation because it gives an inaccurate interpretation of the environment of the robot. Description of the real world is also an issue. Therefore, providing a model that gives a description of the world more complete and yet simple is the ideal. The speed of processing needs to be acceptable so that the robot will navigate at a reasonable speed and successfully avoid obstacles on its way. Real time localization is important to keep a reasonable speed for the wheelchair and to provide safer navigation. Some approaches try to solve the problem by using powerful mathematical and relational models for scene analysis [13], others [17] include fuzziness and uncertainty in the representation of the world model, yet another [18, 19] use special architectures such as MIMD computers to speed up the processing.

Finding an ideal model to describe a real environment, such as the inside of a building, is not easy. Also, dealing with an approximate model doesn't give much information to the robot. An approach which combines deductive knowledge capability with perception of various sensors seems ideal. Deductive knowledge capability will exploit the static information provided by the map data to build more information. Deductive knowledge can also be applied to sensor readings, especially vision data, to extract more information on the scene. The major problem to solve is to provide the robot with knowledge (static and dynamic information) of its surroundings, sufficient for its safe operation at reasonable speed. It is then important to process knowledge fast enough to give the needed information to the wheelchair Navigator module.

Our approach tries to use knowledge-based deduction and artificial intelligence to build more information and to speed up the processing of information. Our approach, similar to Fennema's [18], matches the current image with the template images of specific locations of the building in order to locate the robot. However, our approach differs from the one from [18] in that our method does not try to project the template into the image. We base the matching on a correspondence between the features of the two images. The two images match successfully if a best and satisfactory correspondence can be found between the two images. This is different from the approach of projecting template details into the current image. Similar method using correspondence matching is used by Crowley [20] and Hebert [11]. We think that projecting the template into the current image is more adequate for the double purpose of image matching and object recognition. The reason is that when a model has been successfully projected into the current image, then there is information that the specific detail represented by that model was present on the scene. Fennema et. al. have implemented their system on Suns3 mainframe architecture. Our implementation is intended to be done on a PC.

The contribution sought in this work is to integrate an intelligent database to the PSUBOT wheelchair system. The database will act as a small expert system. As such, the intelligent database will add knowledge capability to the system and flexibility to global path planning, based on static description of the world. The database will also serve as the core of pattern matching and object recognition for the whole system. The capability of intelligently localizing the wheelchair in the building will be a key improvement for future work on this topic.

I.3 BASIC ASSUMPTIONS

As the result of previous work done, planning and coordination with other member of the PSUBOT project, six basic assumptions have been made at the onset of this thesis.

- (1) The environment in which the robot navigates is the indoor of a public building, private home or an apartment. The building is assumed to be a multiple-story building and has facilities such as automatic doors, elevators, ramps etc..
- (2) The operation of the PSUBOT wheelchair should not bring major modification to the current interior of the building. Therefore, landmarks and other localization tools that may require modification of the environment are to be avoided.
- (3) The PSUBOT is designed to meet the needs of three categories of users: the blind, the blind who cannot walk and the blind who cannot walk or use his/her hands. Therefore, the PSUBOT is voice-activated to take voice commands and has a knowledge-based capability to find its own path and guide itself on its route.
- (4) The PSUBOT architecture is composed of three main parts: Navigator, Vision/sonar and the Intelligent database. Each module has its own processor and they talk to each other either by file sharing or by object messages.
- (5) The low to medium image processing has been implemented. The output image after the hierarchical line extraction [3] is a description of the scene, made of straight lines.
- (6) The speed of processing of the entire system should be reasonable enough to allow the wheelchair to navigate at a reasonable speed.
- (7) An initial position (location) of the robot is assumed known.

I.4 DEVELOPMENT METHODOLOGY

The general approach implemented in this thesis for localization based on correspondence matching of the current image with the template image. Template images of "landmark" locations are taken during the learning stage of the PSUBOT. At run time, the robot starts at a known location. The user issues a voice command to the Navigator to go to a given location on the floor or in the building. The Navigator asks for a global path to the database. After the global path has been computed, the Pilot starts the navigation on the

hallway. An image is taken periodically, then processed to extract line features [3]. The output image is composed of line features only. Knowledge is applied to determine the best guess on the most probable location of the robot. Then the image is matched with the template images in the neighborhood, to issue a best decision on the location of the robot. The methodology used in this implementation can be described in five steps.

Step 1: I will define a static model of the world and test the routines to enter, update, query and read the static map of the world. The static model of the world is a map, described as a network of buildings and as the interior of a building. Part of a known campus (PSU) is entered and tested. The user friendliness, the ease of use of the programs, and the consistency of data are evaluated.

Step 2: Implement global path planning. At this step, the work includes implementing a global path finder. The static map is processed (interfacing) to build a knowledge base for the wheelchair. This part of the knowledge base is uniquely concerned with the description of the world. The path finder uses the static map described as an edged simple graph to compute an optimum path between two points. The global path planning program will be tested extensively for correctness and speed.

Step 3: Match two images successfully. Two images of a given location, taken with slightly changed angle of the camera will be matched. The experience will be reproduced on different images to draw a heuristic and to have some statistics on the accuracy of the matching algorithms.

Step 4: Localization of the wheelchair. An algorithm will be proposed and simulation will be conducted. The image (template) of a given location in the building will be taken, processed and stored in memory. A series of images will be taken in the neighborhood of that location, including an image taken at that location exactly. The images will be processed and stored in memory. To simulate the localization of the robot, the images will be submitted one by one in a ordered sequence, following their geographical location on the

corridor. The template will be projected in each of the images, and a match will be done to determine if the robot is at any known location. This strategy can be completed only after the design and integration of other PSUBOT modules (which has not been yet done).

Step 5: Test of the database integrated with the other modules of the system. First, a simulation of the communication between the database and Vision/sonar will be conducted. Thereafter, a simulation of the communication between the database and the Navigator will be conducted. To end, a simulation of the integrated system including the database and all the modules will be done.

Because of the scope of this project, only steps (1) thru (3) will be completed, i.e., implemented and tested. The other steps need the integration of other modules of the system, in order to be tested effectively.

I.5 GOALS

The main goal of integrating an intelligent database to the PSUBOT can be broken into four sub-goals.

- (1) Define a model of the world for the wheelchair that will provide sufficient information on its surrounding. This sub-goal involves implementing an adequate database structure to keep the world model.
- (2) Successfully find a global optimum path for the wheelchair between two given points on the map.
- (3) Find an algorithm for the best match of two images and implement it.
- (4) Find an algorithm to localize the wheelchair using knowledge-based approach and techniques.

The main hypothesis is that template matching works better when applied on abstracted “feature” images and not pixel images. In fact, pixels are the smallest entities that describe a digital image. The other features (edges, lines etc.) are built from these basic features with

appropriate processing. The matching of images at the level of pixels would be a highly inaccurate one because pixels are atoms and don't include uncertainty and fuzziness. On the contrary, higher level features such as straight lines, are made up of many pixels which are not necessarily aligned strictly. There is a margin of error allowed. Template matching in cases such as image matching should include a margin of error because of sensor imperfection and because of the fact that the sensor image and the template images are not taken exactly with the same camera angle.

The thesis is divided in four parts. The first three chapters focus on different approaches to the design of autonomous wheelchairs and mobile robots, world modelling and navigation. The next two chapters focus on databases, knowledge bases and path algorithms for mobile robots. Chapter VI describes the PSUBOT database. Chapter VII concentrates on the image matching. Chapter VIII focuses on localization of the autonomous mobile robot. The last two chapters focus on the evaluation of the intelligent database system and future directions for the research on this area.

In this thesis we have accomplished the following goals: (1) definition of a model of the world for the wheelchair and implementation of a relational database to keep the information on the map of the world, (2) implementation of a knowledge-based algorithm to compute the shortest global path for the wheelchair (3) implementation of a matching algorithm based on correspondence matching of two images described as sets of straight lines and simulation of localization of the wheelchair based on recognition of a location among a set of candidate locations.

I.6 EVALUATION

The evaluation of the database will be conducted sub-module by sub-module. For the global path planner, the evaluation will be based on the speed of path retrieval. Numerous tests applications will be run for different data and problem sizes to guess an approximate

speed of the algorithm. The evaluation of the matching has two criteria: speed of processing and degree of accuracy. Regarding the degree of accuracy, the image matcher should be able to include fuzziness in order to take into account the difference in orientation of the camera when the pictures are taken. It means that matcher should be able to match pictures of the same location taken with a very slight difference in camera angle and come up with the result that they all represent the same location. Applications will be run to estimate the percentage of success of the method. The time will also be recorded.

The evaluation of the localization method will be conducted on the base of the percentage of success of the guess for a given target location.

The global evaluation of the database will be made on the base of the speed of processing of knowledge from the database, compared to the speed of vision processing. An estimation of the speed of the wheelchair will be made from the average speed of the database knowledge processing and vision image processing. The projected speed will be compared to speeds of other wheelchairs and mobile robots from the literature. The overall speed of processing as implemented on PC will be scaled, compared to the speed of processing in robots using mainframe computers, in order to know if any improvement on the speed can be justified. Also, a judgement of the speed of knowledge processing by the database will help to determine if an immediate obstacle can successfully be detected and avoided on time.

The requirements for the computer hardware for future PSUBOT based on this research experience will be formulated.

CHAPTER II

AUTONOMOUS MOBILE ROBOTS: THE NAVIGATION AND LOCALIZATION PROBLEMS

II.1 MODELS OF AUTONOMOUS WHEELCHAIRS AND MOBILE ROBOTS

As mentioned earlier, models of autonomous wheelchairs and robots differ mainly with respect to 1) the structure of the system, 2) the modelling of the world, and 3) the method of navigation. There are many prototypes or models that have been proposed in the literature. We will not cover all the existing models, but we have chosen the most often referenced ones. For each model, we will identify the specific problem solved, the method used, the hardware versus software, the implementation and the results, the major difficulties and limitations, and finally, the trade-offs. The models presented are either wheelchairs or other autonomous mobile robots. The problem to solve is that of providing the device with the capability of self-navigation and decision-making. This remains the common objective in the design of autonomous wheelchairs and other mobile robots. One source of differences in the design approaches is the focus placed on the user of the device. For the case of a wheelchair, the user of the device is a handicapped person. Therefore, in the design of PSUBOT, for example, special consideration is put on the fact that the user of the wheelchair may be a quadriplegic or a blind person.

II.1.1 R.C Madarasz et al. [4]

The specific problem is to design a self-navigating wheelchair that can carry a disabled person within a building environment from a given location, to a designed room, by giving only a room number. In other words, the problem is that of an autonomous navigation in a

crowded environment. To solve this problem, they first define a model of the world, then implement the global and local path planning strategies. The robot navigates by keeping track of landmarks and hallway intersections. The tracking of the landmarks is done using a TV camera.

The environment is modelled as a crowded interior of a building. The model of the world is the indoor of an office building, described by a floor plan showing the geographical disposition of rooms, corridors and their intersections, and elevators, without any specification of distances between these elements. Knowledge on the location of obstacles is provided at run-time. The system includes a sensory module with vision and range finder, and a Navigator module responsible for global and local path planning. The global path of the robot is determined as the shortest path, i.e, a path that contains less rooms on its way, or a path that contains obstacles with longer age. The age of an obstacle is the time period after which the obstacle is assumed to be removed. This path is not necessarily the shortest path, but they assume it is close. Their system is designed such that it can detect and identify both static landmarks and dynamic objects. The output of the planning system is a series of primitive operations, i.e, commands such as to rotate, to move, presence of an elevator ahead, etc.. These commands are executed sequentially. The sensing is done with a TV camera for vision and an ultrasonic range finder. They make three major assumptions. Their first assumption is that with monocular vision, depth cannot be perceived. The second assumption is that vision is computationally expensive. The last one is that images are ambiguous to interpret and are often affected by the illumination. As a result, they use vision only for specific purposes which are: to locate and recognize known objects, to determine the status of the elevator and to keep the wheelchair centered on the hallway by tracking the wall-floor intersection line. The Ultrasonic range finder is used to determine the distance between the wheelchair and its immediate surroundings. It can be used to generate depth maps by scanning. The main problem with

this sensor is false readings due to the reflection of sound waves on surfaces. It is used primarily to orient the chair with respect to walls of the hallways. The system doesn't include a data base as a module, but rather uses a simple data structure to store the map of the modelled world. It seems that the program does not refer to artificial intelligence methods to assist the wheelchair to find its path or to recognize locations and objects of the building.

The system was implemented using as hardware an on-board PC 320KB, two floppy, parallel I/O, digital to analog converters, a RAM disk to speed up the operations, a digital camera (128x128) resolution and a wide angle of 35°. The camera is connected to the processor through a DMA interface capable of taking images recorded at a rate of 11 frames/sec., Polaroid ultrasonic range finder scans 360° with 3° interval. They have built a special DOS-based operating system on the PC to ease the transfer of files between modules of the robot system. The functions are presumably written in C language.

The wheelchair was tested in a crowded building. For the current implementation the system is capable of planning a path through the building, orienting itself in a hallway and performing visual guidance down a hallway. However, the processing of these tasks is not fast enough for a practical case of the wheelchair running at a speed of few miles per second. They have evaluated their system compared to the objectives, i.e, the functions to achieve rather than the speed of the processing. The main concern was to check that the system works for one building, before extending it to a map with many buildings.

The particularity of the method is the ability of the robot to travel in a dynamic, crowded, as well as imprecise, environment. Instead of defining general features to describe the scene, they rather track fixed details such as landmarks in order to guide the robot on the hallway. Tracking only landmarks is a severe limitation for the visual perception of the robot. Therefore, for future extension of the research, they envision a

system that uses more general features, rather than fixed details such as landmarks. They also envision a system with each module having its own processor.

Major problems seem to be false readings from the range finder and collisions that may occur with obstacles that are out of sight of the wheelchair camera. An obstacle that has been supposed passed and no more in the view of the camera can be dangerous. This problem may be solved by adding another camera or range finder on the back of the wheelchair. Also, the age of obstacles is an assumption that has to be revised. The age of an obstacle is dependent on the users of the building, therefore may not be well defined. Using a disposition of obstacles assisted with sensor information may give better results. Information on the disposition of obstacles will be entered at the learning stage or by the programmer of the system. Every time the sensors detect a new obstacle or fail to detect the presence of an existing one, the information should be updated immediately.

II.1.2 S. Rao and R. Kuc [5]

The problem is the design of an autonomous mobile robot with the capability of detecting drop-offs, such as stairs down. The model INCH is a prototype equipped with a Motorola 68HC11 Microcontroller Evaluation Board to control the whole system, six Polaroid ultrasound transducers and sonar sensors. The sonar sensors, placed along the side of the robot and facing the ground, are used to detect drop-offs.

The robot was tested on a rather confined model of the world made of scattered obstacles on top of a table. The test made of INCH was to avoid the obstacles and to detect the drop-offs at the ends of the table. The robot navigates by sense, i.e, it uses range finders to locate obstacles in front. Not much is said about how it avoids the obstacles and how it finds its path. The system seems not to use any artificial intelligence, database or vision at all. This prototype is mainly sensor-based and does not seem to make many intelligent decisions. Two main problems are encountered. The first is that in order to

secure safe braking, the sensors need to be placed on extensions distant enough from the wheelchair. The second problem is that smooth drop-offs or those covered with carpet are not detected by the range finders.

One major particularity of this system is its ability to detect drop-offs, making it very suitable for a blind person. However, the lack of both a description of the world and a knowledge support, make this robot more a prototype than a machine capable of carrying autonomously a handicapped person in a building environment.

II.1.3 M. K Habib and S. Yuta [21]

The problem solved is that of the design and implementation of an intelligent navigation system for an autonomous robot using hierarchical world map representation. The model YAMABICO M-12, is an autonomous robot, the 12th of a series that started in 1975. The system uses an off-board PC composed of five Motorola 68000 processors. A special multi-processor operating system and a real time language ROBOL/0 (real time language) using C-programming language, are used respectively to ease the file sharing between the modules and to speed-up the execution of commands.

The system is composed of the following: a global path planner to find a global path, the Map manager responsible for storing the map in a small database (28K-64K), the Locomotion module responsible for motor control and a sensory module made of Ultrasonic Range finders. Each module has its own processor and there is the central processor "master" to monitor the whole system. The system is equipped with voice synthesizer for control of the wheelchair by the user. However no vision and database are implemented in this system.

The model of the world is an indoor environment corresponding to the inside of a building. The map of the world is described using three levels of hierarchy: the bridge,

which is a path between two buildings, the corridor between two rooms and the objects along a hallway or inside a room.

What is particular in their method is that they distinguish two categories of path planning: on-corridor path planning and in-room path planning. On-corridor path planning is concerned with the problem of finding an optimum path on a vertex-graph built from the world model. The vertex-graph is made of intersecting corridors and bridges between buildings. The in-room path planning uses PRA (Prime Rectangular Areas) to find a path inside a room. A PRA is a rectangle in the room so that an optimum path between two points in the PRA is included in the same PRA.

Dead reckoning is used as the navigation method in this model. The Dead Reckoning method applied here produces as output an “observed track” obtained by adding together the position vectors received from sensor data. The global path is then the union of all these piece-wise sections. It seems that the authors are not concerned with the problem of localization of the robot. But the position of the robot is constantly updated by comparing it to wall information stored in the database. Distance is recorded until the goal position is reached. However the recorded distance may not be representative of the real distance between the two points. This situation may occur if the wheelchair turns around its vertical axis without advancing towards the goal point. Also it is not clear if the robot perceives that it has reached the target destination.

The speed of the wheelchair was not estimated necessarily. However the total time for the computation of the route map and global path planning was estimated. The experimental results are shown in Table I. There are some problems at issue. First, path planning in a room is a free space problem. It is necessary to process knowledge instantaneously so that the robot can produce decisions on-line. Very little has been done to improve the speed of knowledge processing because it is difficult to represent the actions of the intelligent robot

in a single instruction arranged sequentially. Industrial robots, for instance, are controlled in real time by concurrent multi-processor architectures [22].

TABLE I
EXPERIMENTAL RESULTS WITH HABIB' S MODEL

Start and goal point in the same building.	Start and goal point in the same building.but different corridors.	Start and goal point in the same building and on the same corridor.
7 sec	5 sec	2 sec

Source: [6]

II.1.4 C. Fennema [18]

The problem solved is the autonomous navigation of a mobile robot. The robot experimented with is HARVEY, a platform manufactured by Denning Mobile Robotics intended to navigate through offices, hallways, and university grounds as it carries out commands.

The system is composed of modules. Among the modules is a model manager which stores and manages the hierarchical representation of shapes of the items in the scene. Key features are defined to represent items in the environment and they serve as models that will be used for recognition of locations. Another module is the Navigator responsible for plan sketching. Another module is responsible for intelligent decisions: recognize a milestone and execute a sub-goal. A milestone is made of a set of features that are expected to be visible in the scene at a given location of the map. A module is responsible of matching 2-D line features. The matching is made between the features of the current image and the set of features of a location on the map.

The model of the world is a partial model described as a graph or network representation of the world made of partially modeled and unchanging environment using landmark milestones. The method of navigation used is a goal-oriented navigation, i.e, the

details expected in the scene at certain locations are known ahead of time. The system starts by capturing an image of the scene. Then the milestones are projected into the current image until there is a correspondence with one of the milestones. The projection is made by superposing the features of the milestone into the details of the current image until they match perfectly. If the matching is not good, then the current location is not estimated. Their navigation method is goal-oriented and map-matched navigation. It is map-matched because it relies on matching the current image with the template images representing a location on the map. When the matching has been successful, the robot can then be located.

The system was implemented on VAX11-750 using Common Lisp and C. A TV camera is used for vision and range finders for the detection of obstacles. The speed of this robot has not been mentioned specifically. However, the speed of computations of the matching of milestones was measured. It took 5 minutes to successfully recognize three sketches. Sketches are just the features such as door frames, electric poles on the street and corridor way.

There are two problems at issue. First, landmarks alone don't give enough information on the environment. Such information includes the relative geographic position of objects. This is an unrealistic situation for an autonomous robot. Landmarks in general give little information. The description may be improved if assisted with perpetual servoing (feedback from sensors). Secondly, the real timeliness is still an issue. The navigation is a computationally demanding task. Trying to project milestone features into an image can be time consuming in terms of computations. A better and more simple method may be to extract features from the template (milestone) and from the current image, and then match the features of the two images.

II.1.5 J. L. Crowley [20]

The topic of interest in this research is the autonomous navigation of a mobile robot. The specific problem solved is the design of a navigation system based on a dynamically maintained model of the environment. The model was tested on the robot platform IMP (intelligent mobile platform). The system uses a rotating ultrasonic range finder and a touch sensor. The robot plans and executes its paths as a sequence of straight lines. This method is similar to the methods used by Yakahama [4].

This model integrates the information from the rotating range sensor, the robot's touch sensor, and the pre-learned global model, as the robot moves through the environment. The IMP is able to use its network of places to plan a path to a place; to use its sensing, modeling and navigation abilities to execute this plan, and to modify the plan dynamically in reaction to unexpected events. The IMP is to serve as a foundation for mobile household, business, and factory robots which require intelligent navigation. The system doesn't use vision explicitly nor a database.

Navigation is based on a dynamically maintained model of the local environment, called the composite model. The model of the world is either (1) a finite and pre-learned domain such as a house, an office or a factory, (2) a global re-learned model or (3) a network of pre-learned places. Navigation is divided into two kinds: local navigation and global navigation. Global navigation provides a path of landmark points. Local navigation estimates the robot position with respect to the global model and plans local steps to avoid unexpected obstacles. Global navigation may operate on pre-learned model. Local navigation requires a model that reflects the state of the immediate environment, including changes as the plan is being executed. The local position is estimated and corrected by matching the local model stored in memory with sensor model.

The speed of the wheelchair was not estimated, but it can be expected that the system is slow because it constructs a local model of the environment. The problems at issue seem to be the speed of the device and the difficulty to explicitly recognize a given room.

II.1.6 L.M Waxman [23]

The problem solved is the design of a visual navigation system for autonomous land vehicles. The experiment was done on DARPA an Autonomous Land Vehicle (ALV) of the Martin Marietta Corporation, Aerospace Division, Denver, Co. This vehicle is not a wheelchair prototype. The system includes vision and navigator as the main components. The robot is intended to navigate in an indoor or an outdoor environment.

The world is described in terms of models of features expected in the scenes. Such features are, for example, the road landmarks. The system refers to these features to interpret the scene perceived with vision sensor. The robot is guided on its way by tracking landmarks. The navigator is responsible for locating the robot position from known landmarks sighted. The approach uses image matching and image analysis. Key features are defined for the scene. The features are extracted and an image knowledge base is built.

The system was implemented using an image processing adapted to run on a VICOM image processor. The VICOM uses a pipelined architecture. The whole system of the autonomous robot was implemented as a set of concurrent communicating processes on a VAX 11/785 using Berkeley UNIX 4.3. The robot's test speed was estimated at 3km/h while navigating and computing road models from 20 consecutive frames.

What is particular in this formulation is the distinction of two modes of image processing: a bootstrap mode and feedforward mode. In the bootstrap mode, all the image is analyzed. For the feedforward mode, only certain details of the image are closely examined. Another particularity in this system is the navigation method. There are three modes of navigation: long-range, intermediate-range and short-range. In the long-range

navigation mode, the environment is decomposed into regions sharing common properties such as uniform visibility of landmarks and navigability of the terrain. Intermediate-range navigation is invoked to select a corridor of free space through which the vehicle is next to travel. Short-range navigation is responsible for selecting the actual path to be traversed through the established corridor of free space.

Two main problems remain at issue. First, the system has very limited road-following capabilities and could be easily confused with patchy shadows, water on the road, etc.. Additionally, the system cannot yet recognize intersections of corridors or execute turns at intersections (of course). Secondly, there may be problems of accuracy because the system relies more on vision which is known to be intrinsically corrupted by noise [24].

II.1.7 R.C Arkin [25, 26, 27]

The problem to solve is to design an autonomous robot architecture which takes into account uncertainty and fuzziness in the representation of the world. The robot model is AuRA (The Autonomous Robot Architecture). The system is made up of four modules : the navigator, the pilot and the sensory module (vision and range finders).

The model of the world is a partially modelled world. A *meadow* map of the world is built and stored in LTM (long-term memory). The model of the world built from sensor perception is stored in STM (short term memory). The meadow map is a hybrid vertex-graph of free-space model. In this formulation, the free space is modelled as a collection of convex polygons. The LTM memory can be understood as permanent information stored in memory, such as the meadow map. STM holds temporary and continuously updated information, such as the current sensory image of the environment.

The navigation method is a sensor-based navigation with the meadow map. The optimum path is computed using the A*(A*-1 and A*-3) algorithm, followed by a path

improvement process. The implementation of the software for the control and guidance of the robot was made on a mainframe computer.

What is particular in this formulation is the meadow map representation of the world. The meadow map as a layered representation allows showing of free spaces that the robot can use for its path. Another particularity is the use of a feature editor to extract the features seen on the scene.

The computational penalty with the A* algorithm may be significant if the search space is large--specifically for systems with low speed such as regular PCs.

II.1.8 Conclusion of the Review

For almost all the models referenced in the previous paragraphs, two main problems seem to be at issue: modelling of the world and the realtimeliness of the whole processing of information. Most of the methods used seem not to put much emphasis on assisting the system with some intelligence expertise. By intelligence expertise, we mean deductive methods to build more information from the static information stored in the system, such as a static map. These intelligence mechanisms, although they may not have the full characteristics of known artificial intelligence methods, can be taken as artificial sources of intelligence to assist path planning and scene analysis. For example, simple features can be defined as models to represent some details in the scene. Complex models can be described from the simple models using hierarchical or inheritance properties and deductive reasoning. Also, most of the approaches referenced use a partially modelled world described as a vertex graph. Intelligence-based methods could improve the knowledge of the world by finding rules and by deducting from the static data. For example, more information can be built from a rule base combined with a set of facts, than from a bare relational database where information is stored into tables. Queries could be used, but the scope of queries is limited. Making the whole processing of the system fast enough is still a major issue,

especially if there is a consideration that the autonomous robot should navigate at a reasonable speed. One way of solving this problem is to use parallel processing techniques or parallel architectures, for example, pipelining and the hypercube architecture.

II.2 NAVIGATION OF AUTONOMOUS MOBILE ROBOTS: DEFINITION OF THE PROBLEM

Autonomous navigation of a mobile robot in an environment requires [from the vehicle's control] the knowledge of the disposition of objects in space. This information is provided to the robot by the sensors and the “a priori” knowledge of the world. Therefore there are currently two main approaches to autonomous robot navigation: built-map-based navigation and a-priori-map-matched navigation¹. In the built-map-based approach [5] a map is built from the sensor perception and this map is then used for robot's navigation. This method is similar to human perception. A human perceives surroundings with senses and moves (navigates) based on what (s)he perceives. Of course, humans always refer to some kind of knowledge about the world, but we do not need absolute geometrical dispositions of objects. This method gives more accurate results and is more flexible, but requires smart sensors and very fast algorithms because the robot navigates while creating its own map of the world around. The other method, the a-priori-map-matched method [6,4] uses sensor perception, but matches the sensor perceived scene with some templates of milestones previously stored in the system during the learning process. Obviously this method is less flexible because the map of the system is static and the sensors don't necessarily have the same parameters (position and focus) during the learning stage as they have during the run-time. Navigation of the autonomous mobile robot serves only the specific goal defined by the user.

¹The terms built-map-based navigation and a-priori-map-based navigation are due to this author.

Navigation of the autonomous robot has some factors it depends on heavily. First, it is subject to the type of environment the robot is supposed to navigate in. For example, the problem of the navigation of a wheelchair in a crowded environment such as a sidewalk of a busy street at a rush-hour is far different and more demanding in precautions and strategy than the navigation of the same wheelchair in an open area with no obstacles at all. The first deals with moving and unpredictable obstacles in the environment, the second assumes no moving and no not-known items in the perceived environment. Secondly, the knowledge of the environment is a key factor. Knowledge of the environment is taken here in a wide sense. The robot needs to have some information about the environment it is navigating in, so that it can make decisions based on sensor perception fused from sensors, and the knowledge of the environment. For example, an autonomous wheelchair may need to know the characteristics of a given door in order to deal with it as an obstacle. Information about a door on corridor so as whether it opens in, opens out or slides, may be useful for the Pilot of the wheelchair in order to adopt a given strategy to guide the wheelchair through the door. The wheelchair may have to know the shape of an obstacle in order to choose a proper strategy to avoid it or manipulate it (door, elevator). Therefore the navigation problem will be more or less facilitated depending on whether the environment is known or unknown.

Finally, the type of the robot's activity influences the method of its navigation. An autonomous robot used for military purposes may be designed to act blindly on anything else except certain types of targets. Therefore, the vehicle may run randomly over any obstacle on its way. A wheelchair is supposed to carry the handicapped in an inhabited environment. Therefore, care must be taken not to hurt its user or persons walking by. All these considerations play an important role in the navigation problem. Another issue for autonomous mobile robots is the localization problem.

II.3 LOCALIZATION OF AUTONOMOUS MOBILE ROBOTS

Autonomous mobile robots are designed to assist the user in the accomplishment of a given task. However, it is important for the user of the device to control the robot and to provide it with some mechanism that will locate its position so that it doesn't get lost, thereby jeopardizing the operation. The wheelchair for the disabled and all mobile robots of this category need some support mechanism to help locate itself in the environment (building in the case of the PSUBOT). When the user orders the robot to go to a given room on the floor, the autonomous robot needs to have in its system a mechanism that will tell it whether or not the desired location has been reached successfully. Also, when the robot is lost, it should be able to re-localize itself by answering the question "where am I?". This is the problem of localization of the autonomous robot.

Localization methods vary with the researcher and the navigation method used. Some approaches use landmarks to locate the robot [6,4], some other use specific features in milestone image templates [18]. Landmarks are special signs or features on the scene on which the robots relies to follow its track on the hallway. A landmark is different from a milestone in the sense that a milestone is a set of defined models of features that are expected at a given location in the building. A landmark is a sign that is put at a place as an indicator. The robot will use these points for its localization. The robot is in the 3D world and needs a human-like perception. Locating the robot is difficult so approximations are used. Also, scene analysis and perception of the environment need to be robust so that a misinterpretation of the sensor's readings or poor performance of the sensory module don't lead to the lost control of the wheelchair. The problem of the autonomous robot's localization is made more complex when the environment is a real one, as opposed to limited and well-defined model environments. The wheelchair may lose its path. This needs to be taken care of by the error recovery mechanisms.

C. Fennema[18], for example, uses milestone templates. An image is taken, then features of the most probable milestones are projected into the current image. If the matching is satisfactory, then it is concluded that the robot is at the corresponding milestone.

Hebert [11] describes each image as a set of features. To match the two images, an estimation of the displacement between them is carried out. This displacement can be understood for example as the average displacement between the features of the two images. Then prediction regions are built. A prediction region is defined as a set of features that are at a Cartesian distance lower than a given threshold value, and at an angular distance lower than a certain other threshold value. After the prediction regions have been built, the search for matches between the two images is carried out. The result of the search is a set of possible matching between the two sets of features. The matching that realizes the minimum value of the error function is taken as the best. This method seems to be computational intensive.

Section VIII.1 of the thesis will examine different approaches to the localization problem for wheelchairs and autonomous land vehicles.

CHAPTER III

MODEL OF THE WORLD

III.1 DEFINITION OF THE PROBLEM

As mentioned in Chapter II, there are two main methods in autonomous land vehicle navigation: built-map-based navigation and a-priori-map-matched navigation. Both of the two methods need a model of the world in which the autonomous machine will travel. While the first method may need knowledge of the world more in terms of "abstract model" only (example: a door way is rectangle), the second method needs explicit models of objects in the world, their relative disposition, shapes and locations. An image of the locations of interest is previously stored in memory and later on is matched to the current image of the scene. If the current image satisfactorily matches an image of a location, then the robot knows that it is at that location or at least in its immediate neighborhood.

The PSUBOT wheelchair implementation applied in this project uses the a-priori-map-based navigation approach. This method has been preferred to the built-map-based one for three reasons: 1) building a map from sensor data takes a lot of processing time and requires fast processors, which is not satisfied with the regular PC architecture used, 2) the latter method requires accurate sensors which were not available to us, 3) sensor integration has not been completed in the architecture of the PSUBOT system.

The problem to solve in this chapter is to provide the autonomous wheelchair with a description as complete as possible of the world around. The PSUBOT wheelchair needs

information about its surroundings. This information can be provided to the wheelchair's database in different forms:

- as a map which gives static information about the nature and geographical disposition of the elements of the world [4, 6],
- relationships between the elements of the world, so that more knowledge can be constructed from these relations using for instance the major object-oriented principles of data abstraction and inheritance [9].

The inside of a building usually has an abundance of information: different objects, shapes, disposition of objects, colors etc.. The design and organization of the inside of a building is left to the choice and appreciation of the users of the facility. Therefore, because of the diversified nature of the information related to the inside of the building, computer acquisition of this knowledge may be complex. The best description may seem to be a picture of the scene stored in the computer as an image frame. The image can be colored or black and white (nowadays, colored image are stored and processed in computers as well). However the passive image needs to undergo some processing so that the robot or the computer can use the information behind it. This is a critical issue that is dealt with by new image processing techniques. One way to simplify the problem for an application such as the wheelchair seems to be building and storing models of certain objects that can be encountered in the building. These objects will be selected among others based on the fact that they are the most likely. Their models will be used for scene analysis and recognition. For instance, a door frame is a common object that is encountered in a building. Therefore a model for the door frame may be built and stored. The models can be stored either as analytical data or as semantic knowledge supported by some basic information. Similar approach is used by Fennema et al.[18] and Sanderson and Foster [28] in their models.

Generally, the inside of a building (except the objects inside rooms) can be described as lines (straight lines or curved lines) representing details such as wall-floor intersection,

door frames, etc.. Some approaches [20] use lines as features of interest to describe a scene in the matching process. The sensor image is processed, lines are extracted. The image is then matched to a template image of a location stored in memory.

III.2 ENVIRONMENT MODELLING FOR AUTONOMOUS MOBILE ROBOTS

The inclusion of the world model is essential for an autonomous robot which relies on an a priori knowledge of the disposition of elements of the world to plan its own navigation. Therefore, for autonomous land robots it is necessary to keep in memory a model of the world in terms of disposition of the objects and their characteristics.

R. L. Madarasz et al.[4] propose a model of the world as a symbolic structure of the inside of a building. The symbolic structure shows relative locations of offices, rooms, corridors and hall intersections. However, the model doesn't explicitly give the distances separating these elements. The features are static and the knowledge about the location of obstacles in this approach is very important for the navigation of the wheelchair. A "duration age" is associated to each obstacle. The duration age is the time interval during which an obstacle is assumed to be at a given location point.

M.K. Habib and S. Yuta [6] represent the world for their robot Yamabico-M12 as the inside of a building with three levels of hierarchy: the corridors, the bridges connecting two buildings and the objects. The elements of the inside of the room (such as walls) have specific data attached to them to describe certain information, for instance their attributes, the position of door, etc.. Definitely, their model comprises such entities as the building, the corridor, the room and the object. The map is described as a connected simple graph of vertices, each vertex corresponding to an intersection of corridors.

Other researchers in the area, for instance C. Fennema et al.[18] don't use an a priori map. Rather they explicitly adopt models of important features. These features represent

specific objects or specific details in the scene of the environment. These features will be used for scene recognition or interpretation.

It is however important to point out again that the representation of the world a priori is somehow not flexible and very limited. Some researchers [11] prefer not to have an a priori map but to build a map of the world from sensors. This method, although costly in terms of the reduced speed,¹ seems more sound and gives better results in terms of accuracy of navigation of the mobile robot. The next sections examine the model of the world proposed for the PSUBOT wheelchair.

One assumption made in this thesis about the PSUBOT wheelchair is that the processing of information should be fast enough to allow the device to run at reasonable speed. Building a map from sensor readings certainly will slow down the speed of the device. Another consideration is that the PSUBOT robot needs information on the world surrounding it in addition to information about its path. This information may be needed to service voice requests from the user or the Navigator Module, such as to provide him/her with information (name for instance) of a specific room of the floor. The approach of the PSUBOT is justified also by the nature of the hierarchical description of a building. A public building (even a private one) can be described in terms of its levels (floors), hallways (corridors) on each level and locations (rooms) or objects along each corridor. However, an interesting point in the PSUBOT is that more than other models (for instance the model presented by Hebert [11]), it will use artificial intelligence approach of knowledge deduction and inference reasoning to construct more information from the static map, which is rather complicated to do from a map built from sensor readings. Let us take

¹ The robot has to build the map first, then uses that map to navigate. This process slows down the robot. There will be some situations where the robot will be advancing without having any new information at all on its route map because it will have not yet been built from sensors. That is what I would call "blind navigation" cases. This problem will be much reduced if the map building process is fast enough compared to the speed of the wheels.

for example, a situation where the user wants to know which room is at the right end of a given corridor. The sensor map may not be of great help to answer this question. Our approach of an a priori map assisted with knowledge deduction and construction mechanisms seems appropriate for the wheelchair because information on the static disposition of elements is provided as well as the capacity to extract relationships between elements in the static description. This method will allow to answer simple queries from the user on the building. We would like to make it clear that we consider our system to take benefit of expert-system-like knowledge deduction and inference when we reference artificial intelligence. Our system doesn't implement nor use standard artificial intelligence methods which often require a lot of processing time and memory storage.

III.3 A PRIORI KNOWLEDGE : DESCRIPTION OF THE WORLD AROUND THE ROBOT

III.3.1 Overview of the Map Structure

It is an assumption of this research that the wheelchair will be navigating inside a campus environment made up of buildings. There are five levels of hierarchy in the description of this formulation of the map. This description was adopted because of the way a building is commonly perceived (floors, corridors, rooms, elevators, doors, etc.) and the way buildings are interconnected on a campus. The description of a map from the campus level to the building level can be seen as a hierarchy (see Figure 2).

At the highest level the map is described as a site (map site), i.e, a collection of neighboring buildings connected to each other by bridges. A bridge is just a path in the campus that connects two buildings and can be used by the PSUBOT wheelchair. At the next level, the map corresponds to the indoor and floor by floor description of the building. The following level is the floor of a building which is described by its major elements: the rooms and the corridors.

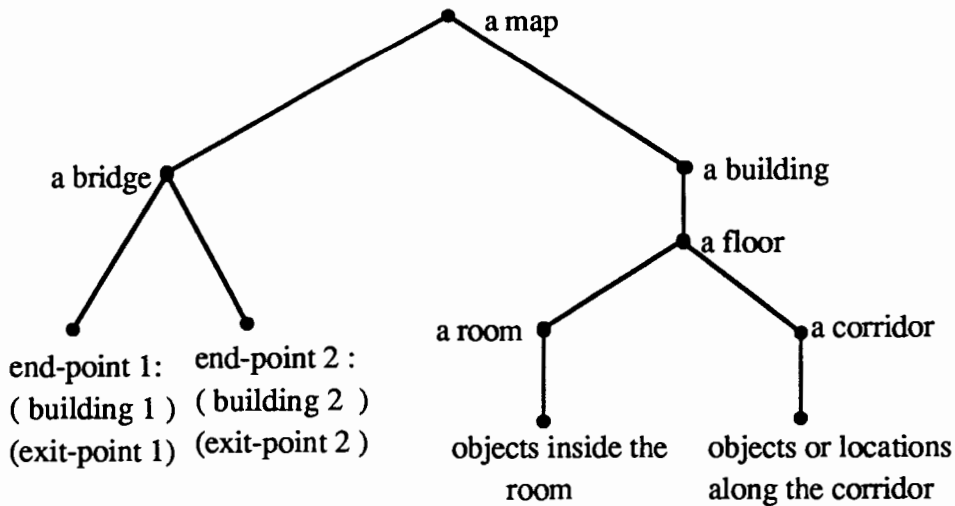


Figure 2. Hierarchical description of the static map.

The corridor comes after the floor as one of the most important elements of the hierarchy. In fact, the corridor is the path that the wheelchair will take to go from one place to another on the floor. As described in the next sub-section, the corridor will be characterized by its shape, dimensions and information on its end-points. The dimensions of a corridor are its length, its width or its radius. The end-points of a corridor correspond to its intersection points with other corridors.

At the lowest level of the map hierarchy, are placed the objects on the corridor way. Objects on the corridor include room entrances, obstacles, doors on the corridor way that are not room doors, signs, special signs or special points (locations) on the corridor such as the end-points of the corridor and fountains. Obstacles are defined here as special objects which, if placed on the corridor way, may endanger or prevent normal operation of the wheelchair or even block the way to the wheelchair. One example of such an element is stairs.

The formulation and choice of the present structure for the a priori map is influenced by way the things commonly appear in regular buildings and the traffic of moving obstacles as a function of the time period during the day.

III.3.2 Map Structure

The map model structure proposed in this thesis can be formulated as follows.

<map>	= <map name><list of buildings>
<building>	= <building name><map name><number of floors> <access to public ><access facilities for motor-handicapped people>
<bridge>	= <bridge name><map name><building #1> <end from building #1><building #2> <end from building #2> <length><traffic frequency> <obstacle density>
<floor>	= <floor name><building name><floor above><elevator >
<corridor>	= <corridor name><floor name>building name><type><length> <traffic frequency><obstacle density>
<corrend>	= <corridor name><floor name>building name> <end-point #1><angle><orientation> <end-point #2><angle><orientation>
<room>	= <room name><floor name><building name> <main entrance door><exit door><in room information>
<object>	= <object name><floor name><building name><corridor name> <side of corridor on><distance to end-point #1 of corridor> <attributes of the object><next object on same side>

An example of a map description is given in the Appendix A.

The information on the status of the building (whether it is public or has access facilities for the handicapped) is important. The battle for providing more access facilities to

the handicapped in public buildings and means of transportation is still an issue for governmental decision makers in the USA especially [29]. So it is important to know if a building has ramps, elevators and automatic doors to help the handicapped. Additionally, a building on the map site may be of restricted access to the general public.

A bridge is an accessible walk way linking two buildings in such a way that the wheelchair can use it. In fact, to be more precise, it links two entrances of two distinct buildings. It should be mentioned in the description of floors if a functioning elevator exists between any two floors. An elevator is necessary if the handicapped persons needs to go from one floor to another .

The term "corrend" stands for corridor end-points information. At each end-point the corridor shape makes a certain angle with the x-axis of the reference taken for the floor (see Figure 3). The orientation of the corridor end-point designates the geographical orientation of that point (N,S,E,W and combinations). The in-room information is the information on the disposition of obstacle in the room such as furniture. The model of the inside of a room will be defined in the next section.

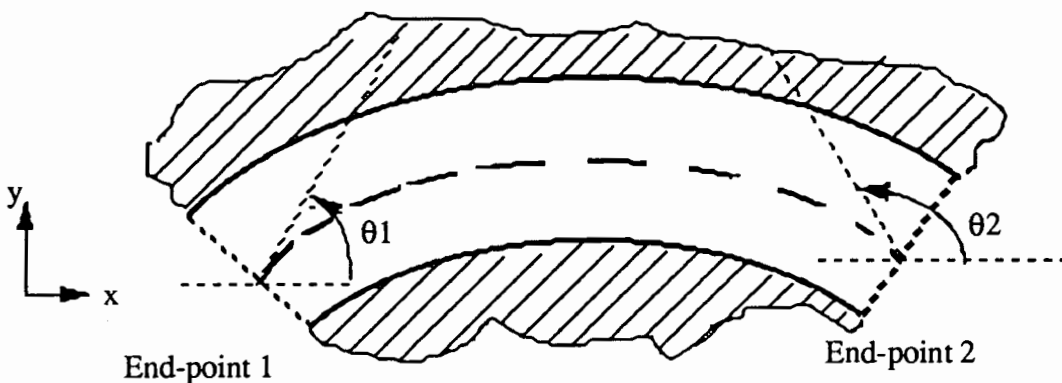


Figure 3. Corridor end-points.

Attached to a corridor or a bridge are three measures: length, traffic frequency, and

obstacle density. The traffic frequency is a real number that measures the periodicity of moving objects on the corridor during the hours of the day. In other words, the traffic frequency indicates a measure of the occupancy of the corridor by moving obstacles. Some corridors are more solicited than others. For example, a crowded corridor at rush hour may not be very suitable for the autonomous wheelchair. A path with slightly longer length but less traffic frequency may be more convenient to take. The traffic frequency could be specified temporally in the sense that it can be described as a function of the time of the day. It will therefore take into account the readings from the on-board clock of the PC attached to the wheelchair.

Another measure attached to a corridor is the obstacle density. The obstacle density is defined as the number of fixed obstacles that physically exist per unit length of the given corridor. Obstacles are doors on the corridor, as well as other objects that can block the wheelchair from traveling on the corridor. These two densities can be considered to add a correction measure to the length of the corridor at a given speed of the wheelchair. This consideration needs to be taken into account for the wheelchair's operation. The Navigator may use the strategy to record the distance from a starting point for the purpose of locating the end of a corridor. If, while the Navigator is recording the distance, the wheelchair is stopped in its travel to avoid a sudden obstacle which is in its way, the estimate of the distance covered will be inaccurate, especially if the wheelchair has used many trials and errors to avoid the obstacle. The problem may be simplified by stopping the recording of the distance covered whenever there is a need to avoid an obstacle on the way.

The objects (room doors, exit doors, obstacles and special locations) are located along the corridor on four sides (left, right, both ends) as indicated in the example of Figure 7. The data structure chosen gives information on the relative position of an object with its neighbors and its absolute location with reference to one end of the corridor.

In the graph representation of a floor map, we have chosen the ends of corridors as the vertices and the corridors themselves as arcs of the graph. Room doors and special observation points could have been chosen as vertices of the graph, but it is obvious that such a consideration would result in a larger graph (number of nodes and arcs). This would not help because it increases the memory demand and doesn't change the nature of the problem. The optimum path (shortest path) is computed in the simplified graph and all the locations along the path (with the exception of nodes) will be retrieved using deductive reasoning and the backtracking method.

III.3.3 Models of the Corridor

As mentioned in section III.1 of this chapter, the corridor is the key element in the description of the inside of a building. In this section the models of the corridor are described .

A corridor as a hall way is described by its length, its shape, its end-points and the list of elements that are located along it. The shape of a corridor (see Figure 4) will be either piecewise linear or piecewise circular. In the model only one direction of convexity is allowed for a corridor (see Figure 5). The convexity of a corridor is the direction of curvature of that corridor. It is true that a corridor can have two directions of curvature -- or more in the worse case. In our formulation, a corridor of non-standard form will be normalized by being broken into pieces of corridor of standard form. The orientation of a corridor is taken from its first end-point (reference end-point) to the other end-point. It means practically that a corridor $C(E1,E2)$ where $E1$ is the reference end-point and $E2$ the other end-point, has the orientation $E1 \rightarrow E2$.

Two major advantages of this formulation need to be pointed out. First, for a corridor with only one convexity, it is possible to track the orientation of the wheelchair much more easily by just comparing the actual angle of the front wheels with the angles of the two end-

points of the corridor. The angles of the two end-points of the piece of corridor define an interval in which the angles of the wheelchair wheels need to be while in normal following of the track. Also, the orientation attached to each end-point of the corridor along with the single convexity of the corridor are a source of knowledge on the orientation of the direction the wheelchair is heading. This can help in path planning. However, normalization of corridors may create a lot of pieces of corridors, thereby increasing the amount of atomic data to process.

Secondly, identifying corridors from intersection points may help to reduce ambiguity. A serious question may be in which category to put a hall. A hall (see Figure 6) may be taken as a room with many doors or as an “empty” intersection point of corridors.

Equation (1). In constructing an equation for the corrected length of a corridor, the assumptions that we make are the following:

- (a) the wheelchair runs at a constant speed V_0
- (b) avoidance of each obstacle slows down the wheelchair by a time t_{01}
- (c) because of moving obstacles (traffic frequency) the total equivalent delay is t_{02}

It will take

$$Obst_dens \times L \times t_{01},$$

time to avoid all the obstacles along the corridor of length L .

The corrected length will be

$$L' = L \left(1 + \frac{Obst_dens \times t_{01}}{V_0} \right)$$

Similarly it will take

$$L'' = \alpha \frac{L}{V_0}$$

extra length because of moving objects. Where α designates a real coefficient, proportional to the traffic frequency Thus the corrected length is

$$L = L(1 + \frac{Obst_dens \times t_{01}}{V_0} + \alpha \frac{1}{V_0})$$

Obst_dens designates the obstacle density. The coefficient α is function of the time of the day. Values of alpha can be stored in a look up table in the database. The time of the day will be accessed using the clock of the on-board computer.

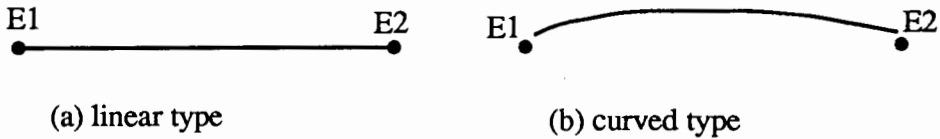


Figure 4. Standard shapes for a corridor.

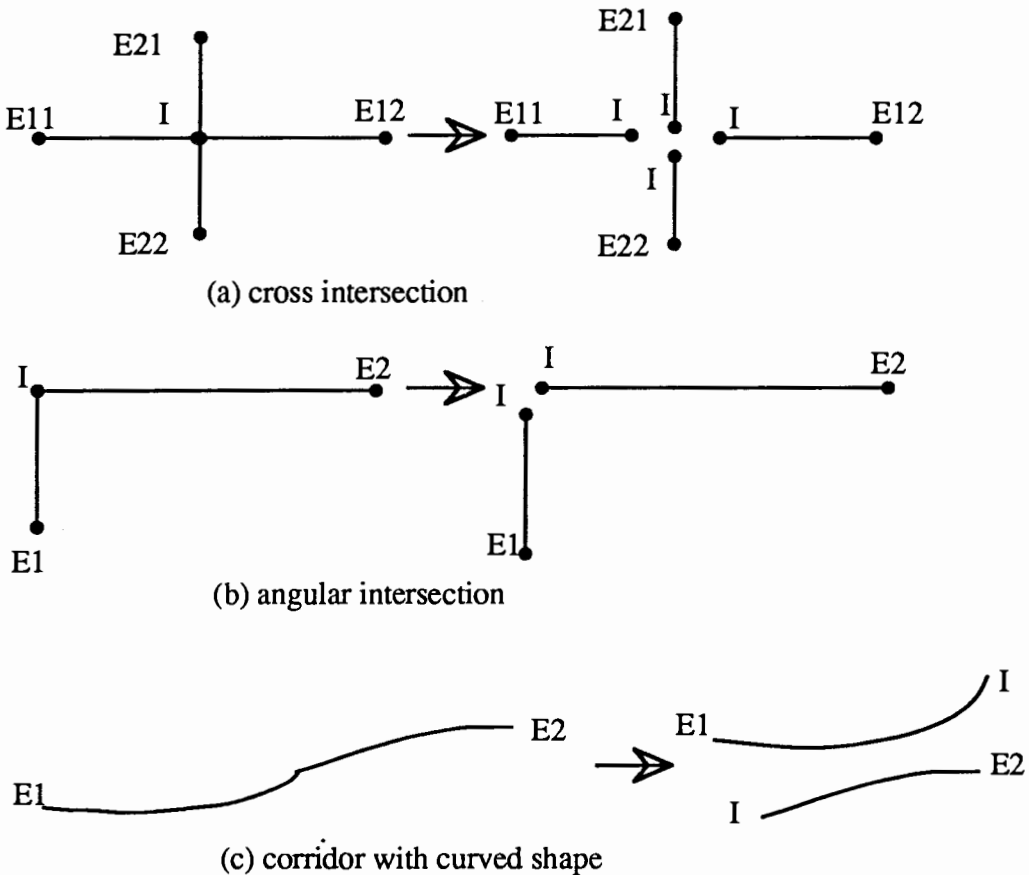


Figure 5. Normalization of the shapes of corridors.

Figure 6 shows an example of a hall in a building and how it is formulated in the

description of the map. In this formulation, the middle point of the hall (point I) has been ignored. Instead, the corridors have been taken on the sides of the hall. There is an inconvenience from this decomposition. Consider the problem to find the shortest path from point A to point B. It is obvious that the straight line A-I-B is the shortest path. However, the decomposition will give a path either A-r₁-C-B or A-r₂-B. This problem is not a major one. The most important problem at this point is to orient the wheelchair to follow an accurate route.

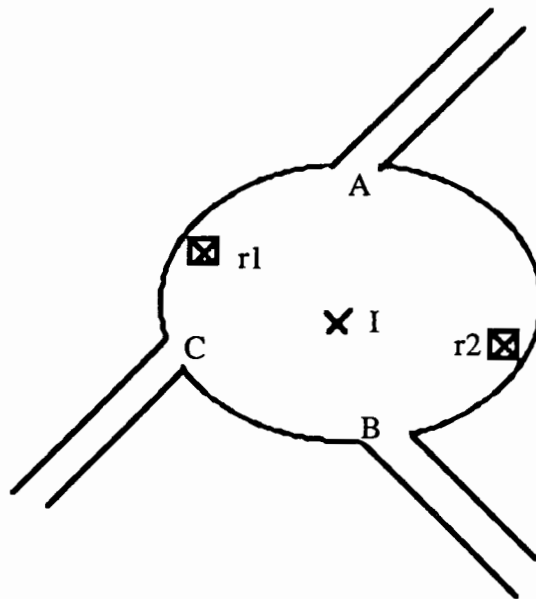


Figure 6. An example of a hall in a building.

II.3.4 A Simple Example of Floor Plan

Figure 7 shows a simple example of how the partitioning of corridors can be done for a regular and simple floor plan. It can be noted that the partitioning of corridors is such that, from each angle at a corner of a room, a perpendicular line is drawn to the opposite wall. An edged-graph of the floor is then drawn from this partitioning. It is however necessary to

point out that decompositions may not be unique. The partitioning of corridors should be carried out in a way to avoid ambiguity. For example, an intersection point of two corridors should appear as an end-point of each one of the two corridors.

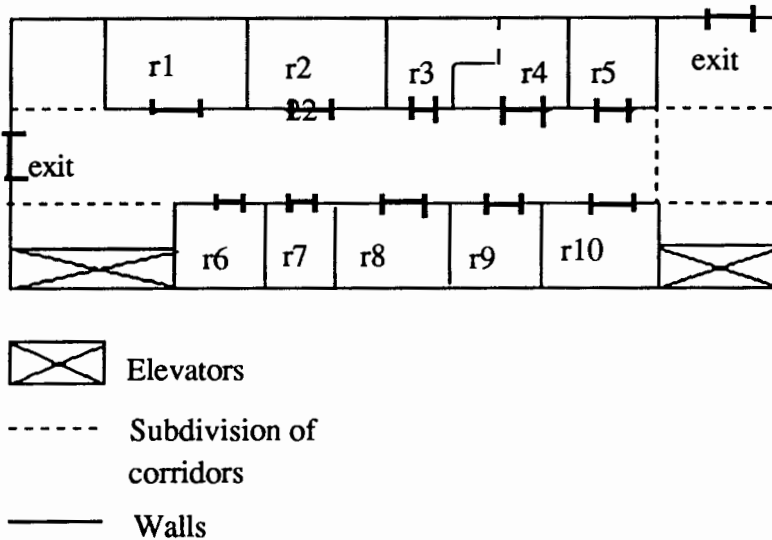


Figure 7. A simple example of a floor plan.

III.3.5 Model of the Inside of a Room

At the request of its user, the PSUBOT wheelchair will have to go to a designated room on the floor. Obviously, the goal for the user is to have some activity in the room, so that it is important to guide the wheelchair inside the room as well. The obstacles inside the room can be described in terms of forbidden areas that will be carefully avoided by the wheelchair's *Navigator Module* [2].

In this formulation, the inside of a room will be described only in terms of obstacles to avoid. In fact what motivates this choice is that more so than the corridor way, the inside of a room can have very diversified structure at the will and the needs of the room user(s). At this point it was not judged necessary to face the difficult problem of describing in detail the

inside of a room. Therefore the description was restricted only to the obstacles to be avoided in the room.

The room information comprises the main entrance door, the main exit door and obstacles in the room. The entrance door and the main exit door will appear on the list of objects on the corridor. By convention entrance door and exit door will be the same object if the room has only one door. An obstacle is represented as a rectangle in the Cartesian coordinates having the main entrance door as the origin of a coordinate system. Its x and y axes are parallel to the two dimensions of the smallest rectangle covering the room. The rectangle covering the room is considered because some rooms may have circular or L shape. The coordinates of the center of the obstacle rectangle will be entered as well as its dimensions (length and width, see Figure 8). An example of application is given in Appendix A.

Another approach would have been to use sonar to build a map of the inside of the room, but this method is obviously more costly in terms of information processing. However it may be necessary to add a sonar map to the simplified description of the room because the obstacles inside the room can frequently be displaced by the room user and there will be a necessity each time to update the information for each room of the building.

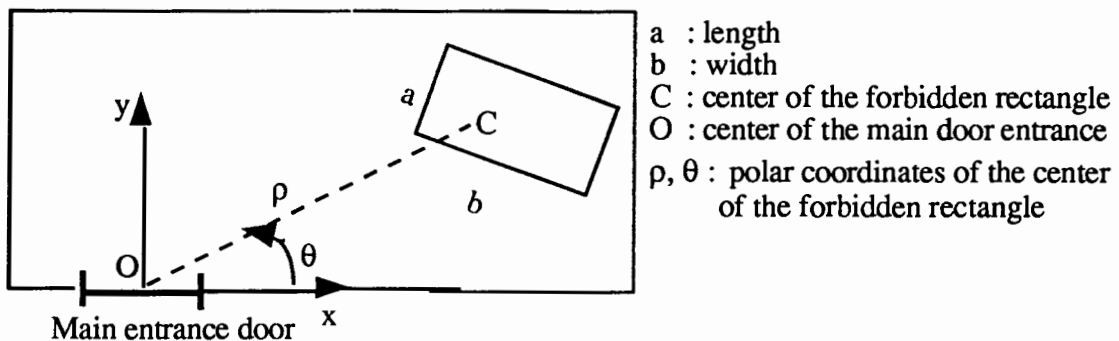


Figure 8. Simplified model of the interior of a room.

III.4 MODELS OF SELECTED PATTERNS IN THE INDOOR SCENE

The indoor scene of a building is so diversified that it is a presumption to claim to describe accurately and in detail the information involved. The primary goal in this approach is to recognize some locations of the building for the global localization of the wheelchair. Therefore, object recognition right now is a secondary interest in this work. The main problem to solve is to match an image of the scene with a template image of some location in the building and make a decision based on that match. It would be more interesting to describe the scene with some selected patterns that will be used as basic features in scene description.

The patterns include door frames, boxes, ceiling angles, corners, corridor ways, walls, windows, obstacles on corridor way, room entrances, and intersection of corridors. Each one of these patterns will have its own feature representing it and each model will be described (refer to Chapter VII). Let us point out, however, that in the hierarchical semantic approach, complex patterns can be described using the basic ones. For the current implementation and research, more emphasis is put on matching images for the purpose of localizing the wheelchair. At this stage, therefore, scene analysis and recognition are not a major subject of interest. Therefore, in this thesis, lines will be used as the only features to describe a scene. There is obviously a loss of information, especially on curved shapes because the feature extractor currently implemented [3] outputs only straight lines. However, it is possible to implement the general Hough Transform method to extract non linear line features, provided an analytical model of the nonlinear shape can be approximated.

The static map as it has been described, needs to be stored in memory, using the storage available on the wheelchair on-board computer. The information can be saved in simple files. However, the hierarchical structure of the map makes it more convenient to use a relational structure such as a relational database to store the information. It would also

be easier for efficient data management (update, retrieval, queries) to use a relational database structure to store the information on the map.

The purpose of this chapter was to define a model of the world for the wheelchair. This information is stored in the computer under the form of a relational database discussed in Chapter VI. The model proposed seems reasonable because it describes the floor plan of the building and gives information on the relative disposition of items on the floor plan (rooms, corridors, obstacles, etc..). The model proposed allows also to describe a set of neighboring buildings. However, we foresee a major inconvenience which is the abundance of data to enter into the computer for a multiple story building or a neighborhood of more than two buildings. The next chapter will discuss relational and intelligent databases.

CHAPTER IV

DATABASES, OBJECT BASES, AND KNOWLEDGE BASES

IV.1 INTRODUCTION

The primary definition of a database system is that of a computerized record-keeping system. The purpose of such a system is to maintain information and to make it available to the user on demand. Compared to a simple file system, a database system has two key advantages: reduction of redundancy of data and inconsistency. A database is operated by a Database Management System which is a layer of software that allows the user to access and manage the physical data residing in the storage support. A database management system (DBMS) [30] is an important type of programming system, used in computerized data management. Over the past years, as databases have been improved, so have the DBMSs. There are, however, new kinds of DBMS's [30] arising, dictated by new types of applications. Two particular classes arising are the "object-oriented" systems and "knowledge-based" systems. OODBMS's [30] support complex objects which mean that they have the ability to define data types with nesting structure and they support encapsulation, i.e, the ability to define procedures that apply only to objects of certain types. Those objects can be accessed only through such functions. An example is the operations PUSH and POP on stacks in C language. OODBMS's also support object identity, i.e, the ability of the system to distinguish two objects that look the same. This means that two objects, e.g, ("cor 9", "floor 1") represented by the same record structure, can be distinguished by such a system which is not the case for similar records in an

ordinary relational database. A knowledge-based management system (KBMS) [20, 31] has the capabilities of both the DBMS and the knowledge system.

The evolution of database systems started in the 1960's (see Table II). The early DBMS were based on network models and hierarchical data models. In the 1970's the relational systems were introduced first by Codd. Relational systems are declarative and value-oriented. Value-oriented or record-oriented support information under the form of record therefore they do not support object identity. The relational models had the disadvantage that they didn't allow for easy integration of DML and host languages.

In the 1980's the object-oriented DBMSs that support both object identity and abstract data types appeared. However they were not declarative. A declarative language is a language in which one can express, in a form of declaration, what one wants, without explaining exactly how the desired result should be computed. A procedural language is just the opposite.

In the 1990's, KBMSs are progressively rising. KBMSs have the built-in declarativeness and integration of DML/host language. They are inherently value-oriented and logic-based. The KBMSs are logic-oriented in the sense that they exploit and build logical relationships among data. For instance, deduction of new facts from existing facts in Prolog [32, 7], uses logical rules. TABLE II makes a summary view of the evolution of databases. Currently there are four main models of databases: hierarchical databases, relational databases, object-based and knowledge-based[31, 33, 34]. Their areas of application are wide: financial records, airline reservations, software engineering [35], and CAD[36]. In the area of robotics, especially that of autonomous mobile robots, databases and data structures are used to store the parameters of the robot system, the model of the world around the robot, and some intelligence primitives such as special functions. Modern database applications are numerous and are dictated by the type of activity. The types of applications of databases lead to the distinction between the DML and the host language

used for the application. Typical applications of the class of database management systems combining DML/host languages include VLSI design databases, CAD(Computer -Aided design databases) for solid modelling, databases for graphic data, and software engineering databases. Databases find increasing fields of application in the area of robotics, especially in the design of autonomous mobile robots [25, 11, 21, 12]. In this field, intelligent databases (knowledge-based) are used more and more to provide knowledge capabilities to the autonomous machines.

TABLE II
EVOLUTION OF DATABASE SYSTEMS

Decade	Systems	Orientation	Declarative?	DML/host
1960's	Network, hierarchical model	Object-oriented	No	Separate
1970's	Relational	Value-oriented	Yes	Separate
1980's	OO-DBMSs	Object-oriented	No	Integrated
1990's	KBMSs	Value-oriented	Yes	Integrated

Source: [31]

The orientation of the DBMSs differ with their category as summarized in the table above. Systems that support object identity are termed as “object-oriented”. Systems that do not support object identity are termed as “value-oriented” or “record-oriented”. For example all systems based on relational model and those based on logic are value-oriented.

IV.2 DATABASE MANAGEMENT SYSTEMS

IV.2.1. Function and Characteristics of DBMSs

Two main facts distinguish the database management systems from file management systems. First, there is a permanent physical database to store the data that the DBMS accesses and manages. Secondly, the DBMS has the ability to access large amount of data more efficiently than a regular file system. A regular file system does not provide help for access to arbitrary portions of data such as fields. A DBMS is capable to:

- support different data models,
- use high level languages,
- provide transaction management,
- protect security of data.

Support of many data models. The DBMS provides at least one abstract model of data. These models allow the user to grasp a better interpretation of the meaning of data. For example, a record is a data model or a data abstraction. The goal of supporting many data models has a major advantage in that it allows the user to define his own data models at the convenience of the application. It also allows the user to attach a meaningful interpretation of the data. In multi-user database systems, each user is given a window on the database and is allowed to choose his own data abstraction. For example, bank teller machines of different bank branches can share the same information on a customer, but each of them has a specific data record on the same customer. At a lower level, a DBMS allows to perceive data as a collection of files. A file of records is abstracted to a relation. An example of a record and a relation are given in Example 1 and Example 2, respectively.

Example 1: a record

record

```
corridor_name: char[10];
```



```

floor_name: char[10];
building_name: char[10];
type: int;
length: float;
traffic_freq: float;
obstacle_density: float;

```

end

Example 2: a relation

```

corridor(Corridor_name, Floor_name, Building_name, Type, length, Traffic_freq,
         Obstacle_density).

```

Support for high level languages. This feature allows the user to access the data more efficiently. For instance a DBMS should be able to access the file and specific record in a file, regardless of the length of that file. This is done by using query languages to allow making the access to file information easier to the user. Queries can be embedded in a DML such as PAL (Paradox Application language)[37, 38]. SQL (Simplified Query Language) is a very well known query language in database applications.

Example 3: a SQL query for the corridor table

```

SELECT [CNAME]
FROM corridor_table
WHERE [FNAME] = "First floor";

```

This query searches and finds all the names of corridors of floor "First floor".

Example 4: a PAL query for the corridor table

```

QUERY
corridor | CNAME   | FNAME   | BNAME   | TYPE | LENGTH | TFREQ | OBFREQ |
         | Check ~cor | Check ~flor | Check ~build | Check | Check   | Check | Check   |
ENDQUERY

```

DO_IT!

This query will retrieve information on the corridor name "cor" of the floor "flor" in the building "build." The bar signs which appear in this query are specific of the syntax of this language.

Transaction management. Another important capability of a DBMS is the ability to simultaneously manage a large number of transactions, which are procedures operating on the database. For instance, an airline reservation system database can be concurrently used at different locations.

Security of data. A DBMS must not only protect against loss of data when crashes occur, but also prevent unauthorized access. The DBMS must be able to give access privileges to data on files, fields and subsets of data. Access privileges are important to prevent misuse or override of data. For example, in multi-user databases, many users can access information at the same time. If common information is not protected, it may be destroyed by a user's application. Some confidential information need to be protected from the public.

IV.2.2 Basic Terminology for DBMSs

The first characteristic of a DBMS is the level of abstraction used to represent the data. Data abstraction levels range from the bit level, to byte level, record level, file level, and so forth. There are three levels of abstraction used in describing a database: physical , conceptual, and view level.

The Physical Database Level . At his level the information contained in the database is represented as a collection of files, storage structures, and indices used to store or access data more efficiently. The physical database resides on secondary storage devices.

The conceptual level . At this level, the information represents an abstraction of the real world for the user of the database. This allows the user to attach more meaning to the data that are behind the numbers or strings. A DBMS provides a data definition language (DDL)

to describe conceptual schemes and the implementation of conceptual scheme by physical scheme. The DDL allows the user to describe conceptual schemes in terms of data models. For instance in the relational model the data are perceived as tables. Another model is the network model, where nodes correspond to relations or files, and the arcs correspond to associations between these relations. The conceptual database is intended to be a unified entity, including all the data used by the organization. To build a conceptual scheme, some agreement about a unified structure must be reached. The process of doing so is called the database integration.

The view level . The view level or sub-scheme is a portion of the conceptual database, or an abstraction of a part of the conceptual database. Most database management systems provide a facility for declaring views, called sub-scheme data definition language, and a facility for expressing queries and operations on views, which is called a sub-scheme data manipulation language.

Other concepts attached to a database management system include scheme, instances and data independence. Data independence implies that the application program is independent of the storage structure and access strategies used. The concept of the scheme is used to designate the current contents of a database. In a well-defined database system, the physical scheme can be changed without altering the conceptual scheme or requiring re-definition of sub-scheme. This independence is defined as physical independence and is referred to as data independence. A view can be regarded as a virtual table derived from a base table. A view doesn't exist on its own, but appears so to the user. On contrary of a view, a base table is real in a sense that each of its rows resides in the physical data storage. To illustrate view, queries are requests that create a view of certain portions of data to the user.

IV.2.3 Database Languages, Host Languages, and Database General Architecture

Database languages. A database language consists of two parts: the data definition language and the data manipulation language. The conceptual scheme is specified in a language, provided as a part of the DBMS. The language is called the data definition language (DDL). This language is not procedural, but rather a notation to describe the types of entities, and the relations among the entities in terms of a particular data model.

Example 5: a PAL (Paradox Application Language) to create a table for a relation “corridor” used in the representation of map data.

```
CREATE corridor
  CNAME: char[10],
  FNAME: char[10],
  BNAME: CHAR[10],
  TYPE: INT,
  LENGTH: N,
  TFREQ:N,
  OBFREQ:N.
DO_IT!
```

The above statements are a request to create a table named “corridor” having as attributes CNAME, FNAME, BNAME (fields of 10 alphanumeric characters); TYPE is an attribute of type integer; LENGTH, TFREQ, OBFREQ are attributes of type real. The statement DO_IT! is used to validate the request.

The transactions on data(entry, update, retrieval) require a specialized language, called a data manipulation language (DML) or query language, in which query commands are expressed.

Example 6: a query in PAL given in Example 4

Host languages . Often, the manipulation of the database is done by an application program, written in advance to perform a certain task. It is usually necessary for this application program to do more than to manipulate the database. It must do some computations. Thus the programs that manipulate the database are commonly written in a host language, which is a conventional programming language such as C, COBOL, FORTRAN and so forth. It is understood then that there exist interfaces in most DBMS to convert data formats between the host language and the Application Language. The application program perceives data in a certain scheme that may vary with the DBMS, according to the interfacing used. Figure 9 gives a general idea of the data communication between the application program and the DBMS. The integration of the DML and the host language is of special interest to the user because it offers a ready interfacing between the two languages. The user doesn't have the task of designing an interface between the two languages. It allows the user to benefit from the facilities offered by both the DBMS and the Host language. In general, the DBMSs offer fast accessibility to files and data, while the host language offers more power in terms of computations.

Database system architecture (see Figure 10). The database system architecture is made up of different components and languages. The components include a Physical database, a File Manager, a Query Language Processor, a DDL compiler, Database Description Tables, Authorization Tables and Concurrent Access Tables. The database manager translates the commands given into operations on files, which are handled by the file manager. The next section recalls some terminology about data models for databases. A reader interested in more details is invited to consult the references [31] or [33] or any equivalent.

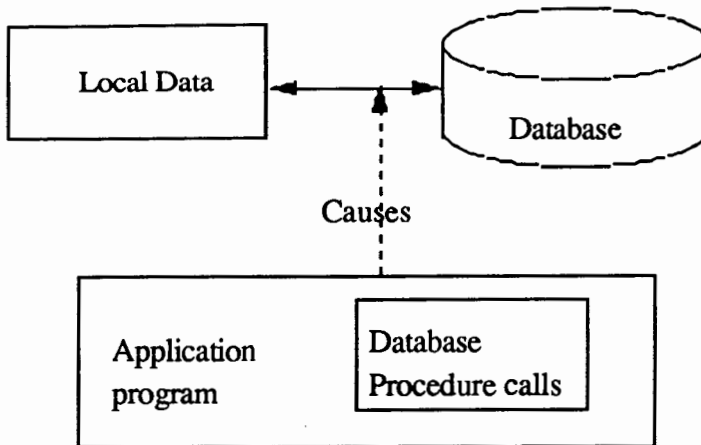


Figure 9. Communication between the DBMS and the application program.

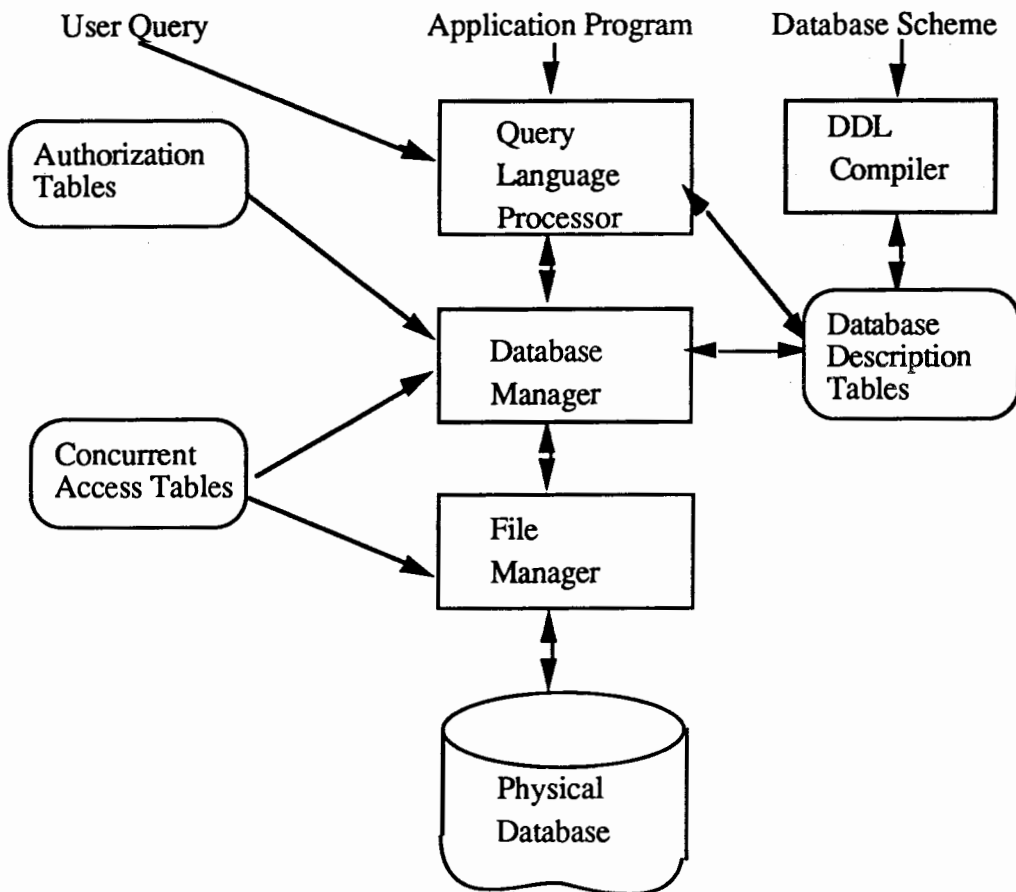


Figure 10. General structure of a database system.

IV.3 DATA MODELS FOR DATABASE SYSTEMS

A data model is a mathematical formalism with two components: an annotation for describing data and a set of operations used to manipulate data. The models for databases are: the entity-relationship, the relational model, the hierarchical model, the object model, and the network model [33].

IV.3.1 Specifications on Data Models

The entity-relationship model has the purpose to allow the description of the conceptual scheme of an enterprise to be written down without attention to the efficiency or database design, which are expected in most other models. In this model, the entity designates a thing that exists and is distinguishable. A group of “similar” entities forms an entity set. Entity sets have properties, called attributes, which associate with each entity in the set a value from a domain of values for that attribute. Each record in a table can be uniquely identified by the value of a given subset of its attributes. This is called a Key. For example, keys can be used to sort, update or search in tables. Keys in relations can be primary keys or borrowed keys. A borrowed key or foreign key for a relation, is a key which participates in another relation. An Example of illustration is given below.

Example 7:

BUILDINGS_TABLE

BNAME*	MNAME	NFLOORS	PBLC	ACCFAC

FLOORS_TABLE

FNAME*	BNAME*	FABOVE	ELVT

Consider the table BUILDINGS_TABLE (refer to Chapter VI). The attribute [BNAME] is the primary key of that relation. Similarly the primary key of the table FLOOS_TABLE is composite ([FNAME],[BNAME]). In this primary key, the key [BNAME] is a foreign key because it participates in the primary key of another relation which in this case is represented by the table BUILDINGS_TABLE.

- The network model is the entity-relationship model with all relations restricted to binary, many-one relations.
- The hierarchical model is a network that is a forest, i.e, a collection of binary trees in which all links point in the direction from child to parent.
- The object -oriented model supports object identity, complex objects, and type hierarchy.
- The relational model supports two kinds of notations for expressing operations on relations. The algebraic notation, called relational algebra, and the logical notation, called relational calculus [33]. They have limitations. First, in general, they cannot express the operation that takes binary relation and produces transitive closure of that relation. For example, a linked list cannot be stored explicitly in a table as a list. The technique that is often used is to represent it as a collection of nodes. Each node is made of an element and a pointer to its successor and/or its predecessor.

IV.3.2 Definitions

Definition (1) (Some operations in a relational model.) The relational model has a set of operands and operators for the relational algebra. The operands are either constant relations or variables denoting relations of fixed arity. There are five basic operations that serve to define relational algebra:

* Union: The union of relations R and S, $R \cup S$, is the set of tuples that are in R or S or both.

- * Set difference: The difference of relations R and S, $R-S$ is the set of tuples in R but not in S.
- * The Cartesian product of R and S is the set of all possible $(k_1 + k_2)$ tuples whose first k_1 components form a tuple in R and whose last k_2 components form a tuple in S.
- * Projection: The projection retrieves given fields from a table. It basically projects given value(s) of a field into a table. The result is a table whose graph is included in the main relation.
- * Selection selects from a table records that match a certain criterion.
- * Some additional operations such as join, Natural join, quotient, Semijoin [29].

Definitions (2) (relation, tuple, primary key, domain, atomic value, non-key attribute)

- a relation is represented in the conceptual view by a table.
- a tuple corresponds to a row (record) of such a table and an attribute to a column(field) of a table.
- a primary key is a unique identifier for the table, i.e, an attribute or combination of attributes with the property that any two rows of the table cannot contain the same value for that attribute (column).
- a domain is a pool of values from which one or more attributes take their values.
- an atomic value is a value that is not decomposable so far as the model is considered. A domain is a set of such values.
- A non-key attribute of a relation R is an attribute that doesn't participate in the primary key of R.

Definitions (3) (functional dependency of attributes.) Given a relation R, and attributes

X and Y of R:

- attribute Y is functionally dependent on attribute X noted $R.X \rightarrow R.Y$ if and only if each X-value in R has associated with it precisely one Y-value in R. Attributes X and Y may be composite.

- attribute Y is fully dependent on attribute X if it is functionally dependent on X and not functionally dependent on any proper subset of X (, i.e, there exists no proper subset Z of attribute constituting X, such that Y is functionally dependent on Z).

IV.4 RELATIONAL DATABASES

The relational database is a database in which the data model is relational. The characteristics of a relational data model have been defined above. Although not the data model used in the first database management system, the relational model, has grown slowly in importance since its exposition by E. Codd in 1970, to the point where it is generally the model of choice for the implementation of new databases. The most important reason for its popularity is that it supports powerful, yet simple and declarative languages with which operations on data are expressed. The relational model is value-oriented. A relation is a subset of the Cartesian product of sets specified by the list of domains. A domain is a set of values. The members of the relation are called tuples. Each relation is identified by its table and a given primary key that is used for such data manipulations as record identification, sorting, updating, etc. There are some interesting structures of relations that make them easy to use in queries, especially to avoid some problems such as redundancy of information in the database, loss of information etc. This is achieved by Normalization of relations.

IV.4.1 Normalization of Relations

Given a set of data to be represented in a database, there is a question of how to break these data into relations and what attributes should be chosen for these relations in order to optimize and add more efficiency to the query process and other manipulations on data. This is the problem of logical design of the database. The process of normalization of relations takes into account the concepts of domains, keys and functional dependency of attributes in the decomposition.

IV.4.2 Normal Forms

There are five categories of normal forms in relational database theory. The normal forms are defined as follows:

- (1) A relation is said to be in first normal form (1NF) if and only if it satisfies the constraint that it contains atomic values only. Every normalized relation is in 1NF (see Figure 11).
- (2) A relation is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent of the primary key.
- (3) A relation R is in third normal form (3NF) if and only if the non-key attributes of R (if any) are mutually independent and fully dependent on the primary key of R. In other words, a relation is in 3NF if and only if it is in 2NF, and every non-key attribute is non-transitively dependent on the primary key.

There are also other forms 4NF, 5NF, BCNF (Boyce-Codd normal form) that are seldom used. The 3NF is satisfactory for usual applications of relational databases.

Normalization of relations is very important for the designer of the logical database. If primary relations are not broken into well-normalized relations, there may be problems of redundancy of information and loss of information. Loss of information will occur when there is a need to reconstruct a parent relation from its children relations.

The hierarchy of normalization of relations is summarized in Figure 11. The abbreviations used in Figure 11 are explained bellow.

Abbreviations:

1NF: first normal form.

4NF: fourth normal form

2NF: second normal form

BCNF: Boyce-Codd normal form

3NF: third normal form

PJ/NF(5NF): fifth normal form

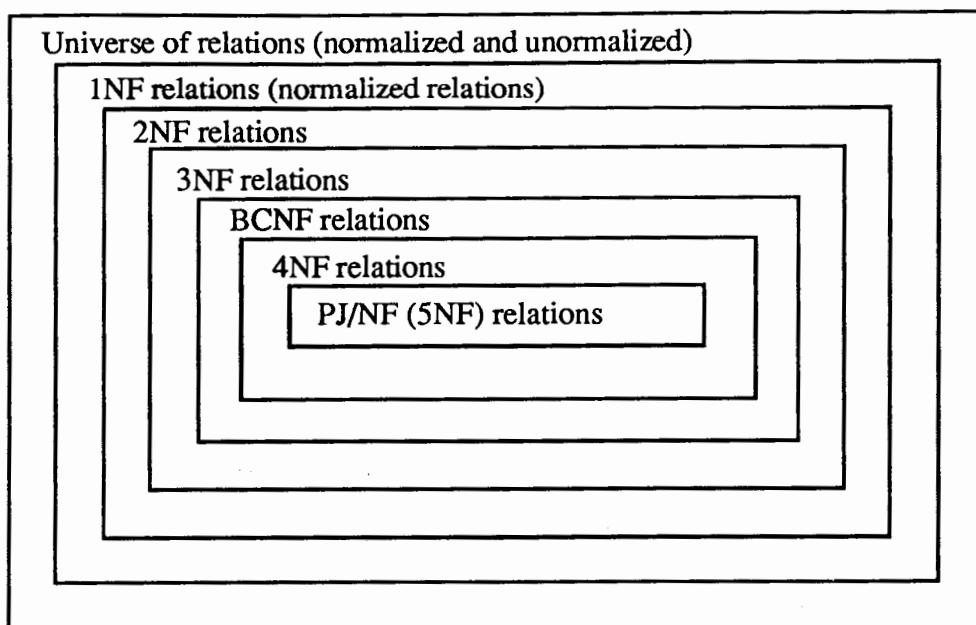


Figure 11. Diagram of the Hierarchy of Normalization.

IV.5 KNOWLEDGE BASES

IV.5.1 Basic Definitions

A knowledge base management system (KBMS) is a system that provides what a DBMS provides (support for efficient access, transaction management, etc.). It is also a system that provides a single, declarative language, to serve the roles played by both the data manipulation language and the host language in the DBMS. Conversely, a knowledge system is a system that supports only a declarative language, i.e, is a programming system with a declarative language. For example PROLOG [32, 39] has been, and still is the best-known knowledge system. A system such as PROLOG handles knowledge through its basic entities which are rules and facts.

- A fact is an atomic proposition. Basic facts of a knowledge base are always true (see Example 8).
- A rule is a logical statement. In this sense it can be understood as a complex logical proposition. In PROLOG, statements are composed of "atomic formulas." "Atomic

formulas" consist of predicates. As such, it can take arguments or not. The arguments called atoms are composed of constants, variables, and function symbols. Predicate symbols are boolean-valued functions, i.e, they return true or false as result. A rule is served by applying it as a goal.

- A goal is the expression of what the user wants (see Example 8).

Although PROLOG is much more declarative than procedural (such as C and Pascal for example), some kinds of simple computations such as basic operations(+,*,/,-) can be embedded into rules.

- Some conventions in PROLOG: constants begin with a lower-case letter. Variables must begin with capital letters. Logical statements, called rules, will usually be written as Horn clauses. These particular clauses are statements of the form,

" if A₁ and A₂ and ... A_n are true, then B is true".

The above statement can be written in PROLOG as:

B:- A₁&A₂&A₃....&A_n

The symbol ":-" stands for IF. The symbol "&" stands for AND and can be also represented by "," in some versions of PROLOG . The symbol ";" represents the logical OR. In the PROLOG syntax a rule is terminated by a period.

Example 8: a basic fact and a simple rule in PROLOG

PROGRAM

DATABASE

building(symbol,symbol,integer,symbol,symbol)

floor(symbol,symbol,symbol,symbol)

PREDICATES

Floors_of_same_build(symbol,symbol)

CLAUSES

Floors_of_same_build(F₁,F₂,B):-

$\text{floor}(F_1, B, F_{\text{above}_1}, \text{Elvt}_1), \text{floor}(F_2, B, F_{\text{above}_2}, \text{Elvt}_2).$

$\text{floor}(\text{"basement"}, \text{"pcat"}, \text{"firstfloor"}, \text{"y"}).$

$\text{floor}(\text{"firstfloor"}, \text{"pcat"}, \text{"firstfloor"}, \text{"y"}).$

$\text{building}(\text{"pcat"}, \text{"psu"}, 2, \text{"y"}, \text{"y"}).$

a basic fact

$\text{floor}(\text{"basement"}, \text{"pcat"}, \text{"first floor"}, \text{"y"}).$

This basic fact is like a record in the knowledge base. It keeps information on a floor named "basement" of a building "pcat". The remainder of the information is that the floor above this floor is named "first floor" and there is an elevator connecting the two floors (refer to Chapter 3: map data structure).

a rule

$\text{Floors_of_same_build}(F_1, F_2, B)$

The basic facts in this rule are $\text{floor}(F_1, B_1, F_{\text{above}_1}, \text{Elvt}_1)$ and $\text{floor}(F_2, B_2, F_{\text{above}_2}, \text{Elvt}_2)$. This rule means that two floors F_1 and F_2 are in the same building named B expressed as $\text{Floors_of_same_build}(F_1, F_2, B)$, if there exist facts $\text{floor}(F_1, B, F_{\text{above}_1}, \text{Elvt}_1)$ and $\text{floor}(F_2, B, F_{\text{above}_2}, \text{Elvt}_2)$. In the above rule, the basic facts $\text{floor}(F_1, B, F_{\text{above}_1}, \text{Elvt}_1)$ and $\text{floor}(F_2, B, F_{\text{above}_2}, \text{Elvt}_2)$ could have been expressed as $\text{floor}(F_1, B, _, _)$ and $\text{floor}(F_2, B, _, _)$. The " $_$ " are put in the place of variables in rule when the outcome of the result for this rule is independent of the value assigned to those variables. This concept is similar to a "projection" in a relational database relation. A "projection" will select some fields among the set of fields representing the attributes of a relation, namely those with values matching a certain criterion (see Example 9). In fact the above rule can be rephrased as "two floors F_1 and F_2 will be qualified as belonging to the same building B , if there exist two facts $\text{floor}(F_1, B, F_{\text{above}_1}, \text{Elvt}_1)$ and $\text{floor}(F_2, B, F_{\text{above}_2}, \text{Elvt}_2)$ in the knowledge base".

a goal

Floors_of_same_building("basement",F,"pcat").

The goal to satisfy in the query can be rephrased as " find the names of all the other floors of the building "pcat ". The satisfaction of the goal, i.e, the answer to this query will be False if no other floors are found in the knowledge base. The answer will be:

F = "first floor"

True

Example 9: : a projection in SQL

SELECT [FNAME]

FROM floors_table

WHERE [BNAME] = "pcat".

This SQL-like statement will create a view on the table *floors_table* consisting of a unique attribute [FNAME]. Practically this statement retrieves from the database, the names of all the floors of the building "pcat".

Backtracking is used in PROLOG to find all possible solutions to satisfy a goal. All the possible solutions are tried unless a cut of backtracking is intended. From a starting point, the search will be carried on until it fails or a solution is found. Then the search for a new solution is carried out from the starting point. Backtracking is one of the most powerful and interesting features of PROLOG, as well as its built-in database structure.

The major difference between SQL-like query languages and a knowledge language such as PROLOG is that the query languages lack the ability to compute transitive closures. A transitive closure of a binary relation r is the smallest relation s that includes r and is transitive, i.e, $s(X,Y)$ and $s(Y,Z)$ imply $s(X,Z)$ (see Example 10.a). A transitive closure is used to apply the transitivity of a relation to three variables. PROLOG computes transitive closure by unification. However, the user must signify the transitive closure to PROLOG by an appropriate choice of variables in the request(see Example 10.b). We will

now examine the rule support in PROLOG and the inference engine approach used in knowledge systems.

Example 10: a transition closure

Consider the binary relation R defined between floors of the same building by:

$R(F1,F2) \Leftrightarrow \exists$ a building B such that $(F1$ is a floor of $B) \wedge (F2$ is a floor of $B)$.

A trivial transition closure can be defined in this simple binary relation by:

$R(X,Y) \wedge R(Y,Z) \Rightarrow R(X,Z)$

Consider the relation R above and using the rule defined in Example 8. PROLOG is told of the transitive closure "of the same building" in this rule by repeating the same variable B in the two facts *floor()*.

Floors_of_same_build($F1,F2,\underline{B}$):-*floor*($F1,\underline{B},\text{Fabove1},\text{Elvt1}$),
floor($F2,\underline{B},\text{Fabove2},\text{Elvt2}$).

IV.5.2 Rule Support in PROLOG

Many applications, e.g., expert systems, maintain a knowledge base, i.e, a large, almost static set of rules which is used by the inference mechanism defined by the user to satisfy a given goal. In PROLOG-like systems the proofs are executed as rule-directed depth first search through a solution space, and are terminated when either no solution is found, or the data is found to satisfy the main goals and their sub-goals in the search tree. This process is sequential and often requires backtracking in order to satisfy a sub-goal having a new candidate solution.

- Classification of PROLOG rules. A fundamental unit for a PROLOG program is the literal. In Example 11, $X, Y, M, N,$ Jacob and Joseph are literals within the rule *father*.

Example 11:

father(Jacob, Joseph), $r(X,Y), M>N$.

The literal's predicate name corresponds to a procedure in a conventional programming language. A PROLOG program consists of sequence of of clauses. Clauses are logical

statements. A clause is made up of a *head* which represents the left hand side of the clause and a *body* representing the right hand side of the clause (see Example 12). The head usually is a single literal or is empty. The body consists of zero or more literals. *Goals* or *procedure calls* are the body literals.

The body consists of a sequence of zero or more literals. The body literals are called goals or procedure calls. A procedure p calls procedure q if a rule for p contains a goal whose predicate name is q . Procedure p references procedure q if either p calls q or p calls some procedure h and h references q .

A clause with an empty head is a query. A query is a statement of the form: $:- p_1, \dots, p_k$. The goals in the body of a clause are linked by the operator ‘,’ which can be interpreted as a conjunction. A clause with an empty body is called a *unit clause* or a *fact*.

A PROLOG *rule* is a non-unit clause. A PROLOG *procedure* is a collection of PROLOG rules all having the same predicate name in their head. Only *simple* variables are allowed in the head of a procedure, not functions. A variable appearing only once in a rule can be written as an anonymous variable denoted by the underline character “_”. It means that the value of that variable really doesn't affect the rule. A literal containing no variables is called a *ground literal*. A clause containing no variables is called a *ground clause*. A *base fact* is a ground unit clause (an assertion or a tuple in the relational database sense). A collection of base facts having the same head predicate is called a *base relation*. Unit clauses containing variables can be taken as rules by extension.

Example 12 : rule, procedure, ground clause, basic fact, head, body of a clause

PROGRAM

DOMAINS

PREDICATES

q(real,symbol)

CLAUSES

$q(X,Y):- X<4, write(X),write(" ",Y).$

$q(X,Y):- write(Y," has less than"), write(X), write("money").$

The clause $q(X,Y)$ is a procedure because it has more than one instance. The head of the first instance of q is $q(X,Y):-$, the body is " $X<4, write(X),write(" ",Y).$ " Additionally, q is a procedure.

•Limiting backtracking. Two operators are used to limit backtracking in PROLOG: the *cut* operator denoted $!$ and the *braces* operator denoted $\{ \}$. The interpretation of the braces operator is the following. The left brace $\{$ always succeeds when the right bracket $\}$ is reached during backtracking, the backtracking continues at the goal immediately to the left of the left bracket $\{$. The braces operator may be nested (see Example 13). In order to tell PROLOG to explicitly list all the solutions of a goal, a strategy can be done by using the *fail* operator. The fail operator forces backtracking.

The cut operator is used to limit backtracking. Consider the following clause:

$Find_an_elevator(Elvt,F,B):- object(Elvt,_,F,B,3,_,_,_,_),!$

This means that as soon as one elevator is found the process is cut. There is no more need to find other solutions. If this goal is run, there will be no or only one solution

Another tool used to limit backtracking is the braces operator

Example 13:

Consider the rule $q:- \{x,\{y\},z\},w,\{t,u\},v.$

This rule is equivalent to:

$q:- q1,w,q2,v.$

$q1:- x,q3,z,!$

$q2:- t,u,!$

$q3:- y,!$

The fail operator is used to control backtracking usually when there is a need to list exhaustively all the solutions satisfying a particular condition. Consider the program in Example 14.

Example 14:

PROGRAM

DOMAINS

name = symbol

age = integer

PREDICATES

father(name,name)

everybody

parents_of(name,name,name)

aged(name,age)

is_sixty(name,age)

Young_father(name)

is_older_than(name,name)

have_same_age(name,name)

CLAUSES

father(leonard,catherine).

father(Malhusalem,jason).

mother(eva,catherine).

father(carl,marilyn).

aged(Malhusalem,60).

aged(catherine,6).

everybody:-

father(X,Y), write(X,"is", Y, "'s father\n"),fail.

parents_of(X,Y,Z):- father(Y,X),mother(Z,X).

is_sixty(X,N):- aged(X,N),N = 60,!.
 !.

Young_father(X):- father(X,Z), aged(X,N),N < 20.

repeat.

repeat:- repeat.

is_older_than(X,Y):- aged(X,N1),aged(Y,N2), N1> N2.

have_same_age(X,Y):- aged(X,N1),aged(Y,N2), N1 = N2.

If father(X,Y) is ran as an external goal (i.e, run from the window prompt) Turbo Prolog will list all the possible solutions in the way:

X =Y=

X =Y=

....solutions

If father(X,Y) is run as an internal goal, i.e, a clause inside a program, Turbo Prolog will continue with the next sub-goal once it has been satisfied, and will display only one solution. However, the predicate "everybody" uses the fail predicate to disturb the usual mechanism. Fail can never be satisfied, so Turbo prolog is forced to backtrack.

- Clause and basic fact. In the program above (see Example 14), "everybody" is a clause, "father(leonard,catherine)" is a basic fact. The clause "fail" is a ground literal and the clause "everybody" is a ground clause.

- A base relation. The following set of base facts about father()(see Example 14) constitute a base relation.

father(leonard,catherine).

father(carl,jason).

father(carl,marilyn).

A base relation is the equivalent of a table(of tuples) of a relation in a relational database.

IV.5.3 Some Definitions

Definition (1). A rule base is a set of rules. A rule is understood in this context as an "if-then" statement. Rule bases are for example used in expert systems. A good rule base in an expert system must be complete, i.e., enable the determination of the output parameters from the given data in all the cases. A rule base has a number of rules, external parameters, and internal parameters.

Definition (2). Goals that are not unified (in PROLOG sense) are called immediate goals, e.g. negation('not') , 'fail', $X \Theta C$, $X \Theta Y$ where X, Y are variables and C is a constant and Θ is a comparison operator.

Example 15:

$X \leq 4$ and $X = Y$. are immediate goals.

Definition (3). A goal in a body of a rule is semi-elementary in exactly one of the following cases:

- (1) The goal's predicate is the name of a base relation.
- (2) It is of the form $X = C$, where X is a variable and C is a constant.
- (3) It is of the form $X = Y$, where X or Y appear in a semi-elementary goal written prior to this one.
- (4) It is of the form $X \Theta C$, where X appears in a semi elementary goal written prior to this one and Θ is a comparison operator.
- (5) It is of the form $X \Theta Y$, where both X and Y appear in semi-elementary goals written prior to this one, and Θ is a comparison operator.
- (6) All rules whose head predicate is the same as the goal's predicate are semi-elementary.

Example 16 (refer to Example 14)

Consider the rule $\text{parents_of}(X,Y,Z)$. The goal $\text{father}(Y,Z)$ in the body of this rule is a

semi-elementary goal(case (1)), so are the goals $N = 60$ (case (2)). The goal *repeat* in the body of the recursive procedure *repeat* is also a semi-elementary goal(case (6)). The goals $N1 > N2$ (case (5)), $N < 20$ (case(4)) and $N1 = N2$ (case (3)) are also semi-elementary.

Definition (4). A rule is semi-elementary when it satisfies all the following requirements:

- (1) every head variable of the rule appears in a semi-elementary goal in the body of the rule.
- (2) Each of its goals is either semi elementary or is a negative goal of the form 'not(p(...))' where p matches a semi-elementary procedure.
- (3) Every head variable appearing within a negative goal must appear previously in a semi-elementary goal.

Example 17 (refer to Example 14)

The rule *is_sixty(X,N)* is a semi-elementary rule according to case (1) of the definition.

Definition (5). An immediate goal is a loose constraint in one of the following cases:

- (1) It is an inequality which involves a variable not appearing in any elementary goal of the rule written prior to this inequality.
- (2) It is the PROLOG axiom 'fail'
- (3) It is the predicate { }
- (4) It is a negative goal which involves head variables not appearing in any semi-elementary goal of the rule written prior to the negation.
- (5) It is of the form 'not(p(..))' where p is not a semi-elementary procedure.

Example 18

Consider the following rule:

```
Interchange_Copies((P1,P2,Day):- student(P1),student(P2),
                                class_mates(P1,P2),Day < 15.
```

This rule may be interpreted as " the classmates P1 and P2 will interchange copies on day Day if the day is before the 15th."

In this rule, the immediate goal $Day < 15$ is a loose constraint in the sense of definition (1). For example, this rule can be used to avoid compiler warnings or run time warnings that the variable *Day* is only used once in the clause. In fact, for some versions of Prolog, if a variable appears only once in a clause, there is either a compiler warning or a run time warning/error.

Definition (6). A rule is serviceable if after syntactically erasing it all "{" and "}" symbols and then discarding from it all cuts and loose constraints, the result is a semi-elementary rule. A procedure is serviceable if all its rules are serviceable.

Example 19 (refer to Example 14)

The rule *is_sixty(X,N)* is a serviceable rule.

Inference Engine methodology. Stengel [40] gives a definition of the inference engine as follows. The inference engine is a program that applies rules from the rule base to the knowledge in the database to infer new knowledge. A typical knowledge base is made of basic facts and deduced facts. Some rules can be defined between basic facts to derive new knowledge. Given the name of a parameter, the inference engine should be able to search the rule base for a list of rules that have this parameter as a variable. The rules are selected in such a way that when applied in a sequence, they infer a new knowledge, until the last rule provides a value for the desired parameter.

For the management of a rule base, a search method can be implemented using either forward-chaining or backward-chaining. The backward-chaining is used by the rule base for searching. It examines only those rules that effectively have a chance of giving a result. The forward-chaining tests all the rules in arbitrary order, until either no more rules exist or the value of the desired parameter has been found. The backward-chaining process starts with the desired parameter, and searches backwards to determine if all the rules necessary to produce the solution are satisfied with the chosen value of the parameter. The inference

engine is used in systems that use both knowledge bases. Such systems are, for example, expert systems.

IV.7 REPRESENTATION OF DATA IN MOBILE ROBOTS

Data representation in robotics has been generally not regarded as an issue until recently. Usually, conventional data structures are used for small applications. In large systems such as flexible manufacturing cells or automated plants, there is a need for a centralized database to keep all the information on the system (tools, machines, fabrication parameters, and system parameters) [22] and have the ready information that can be distributed to many posts in the infrastructure. In tools such as CAD systems, there is a need to have a centralized database to keep the models and the functions, as well as the interfaces to the whole system. Knowledge base and rule bases can also be included in such systems. Databases, knowledge bases and rule bases find an area of application in expert systems. Databases, and especially the distributed databases, are used extensively in business applications such as financial records and airline reservations.

For autonomous mobile robots systems equipped with vision sensing, vision (image processing) is the part of the system that usually deals with a large amount of data. However, there is a need of data structures or a small database to keep knowledge of the environment (map, behavior, description of the components of the environment etc.). In the literature, we have noticed that most autonomous mobile robots don't use an explicit database. In some systems, however, small relational databases are used to keep information on the model of the world [6, 41] or to store certain functions of the system [27]. Some systems use knowledge base [40].

R.C. Arkin [26] in his model AuRA, uses a database to store landmark templates and models, and another database to store motor schemas. R.C. Arkin defines motor schemas as a set of concurrent processes that operate in conjunction with associated perceptual

schemas and contribute independently to the overall concerted action of the autonomous vehicle. The motor schemas serve as the basic unit of behavior specification for the navigation of a mobile robot. A schema in this context can be defined as a pattern of action, or as a pattern of behavior for action.

Madarasz [4] in his model of autonomous wheelchair, uses a three-dimensional indexed array to represent the three levels of the map hierarchy (floor number is the first dimension, the second dimension is wall number, and the third one is the room number). Habib and Yuta [6] use a hierarchical data structure to represent their world model (static map with three levels of hierarchy). W. C. Collier [41] in his model of In-vehicle Route Guidance System (RGS) uses a database to store the information on the map. No mention is made of the type of the map database. However, it may be assumed that this database is a relational database.

The PSUBOT wheelchair system incorporates an "intelligent" database. This database has two components: a relational database to store the static world model, and the knowledge base representing the same information, but saved under the form of basic facts. The database stores also the images of landmark locations of the building under the form of files. Those files are indexed in the database. In fact, they are associated to the names of the landmark locations that they represent. When compared to other approaches, the PSUBOT database is more like a small expert system because it combines static data and dynamic knowledge base. However, the PSUBOT database doesn't follow a given canonical structure proper to an expert system. The structure of the PSUBOT database system will be proposed in Chapter VI.

CHAPTER V

PATH PLANNING FOR MOBILE ROBOTS

V.1 GENERAL CONSIDERATIONS

Autonomous mobile robots are intended to self navigate in a given environment that is known or unknown. To be able to navigate autonomously, just like the human-being with knowledge, the autonomous robot must possess some mechanisms that allow it to find a route between its present location and a desired destination. Along with finding a global path, the robot must navigate locally by avoiding obstacles on its way and plan some kind of strategy to move step-by-step from source to destination. The task of path planning for the autonomous robot is to find a global path and a local path. By a local path we mean a segment of the global path free of obstacles. A segment of global path is determined by the Navigator (the Navigator is the module which controls the navigation of the wheelchair). For example, it can correspond to a straight line portion of the path where the robot doesn't have to turn, or a corridor between two hallway intersections.

To cover the whole route, the wheelchair moves segment by segment. In this work, the tasks of local and global path planning are separated, although both are supervised by the Navigator. The Navigator, through the Pilot, is directly responsible for local path planning. Local path planning consists of guiding the wheelchair on the corridor or inside the room and avoiding obstacles. The database is responsible for global path planning. Global path planning consists of computing the global path. This path is the shortest and optimum path from a source point to a destination point. The global path is a list of locations (points) from source to destination point, in the order they appear physically on the floor map. The

Navigator sends a signal to the database to compute a global path. The optimum global path is computed and sent back to the Navigator, which then carries on with local path planning.

Some issues are yet to be met in the task of planning the path for the mobile robot. These issues are more concerned with sensor perception. Sensor perception of the world around is crucial for the robot, in order to move around safely, both for itself and other users of the environment. The robot has to know where it is, so that it will determine if the destination has been reached, to know if there is an immediate obstacle to avoid, and so forth. Sensory perception is indispensable for the autonomous robot to complete its path. The path planner will have to receive a continuous feedback from the sensors (vision and/or range sensors) to drive the robot. Some inherent difficulties are obvious. All the difficulties faced in sensor perception [24, 42] will be accumulated and path planning will be influenced. In fact, if there is a high degree of sonar inaccuracy, for example, the local path planner may have great difficulty in preventing the robot from bumping an immediate obstacle on its way. Similarly, if vision or other sensors used for localization fail, the planner will never know if the destination has been reached.

However, if we assume that sensors are doing their job well, a major question remains to estimate how fast sensor data is processed and whether it is fast enough to meet the expectation that the robot should navigate at a reasonable speed inside the building, such as if driven by a conscious disabled person with a joy stick.

Sensors are the eye, the ear and the skin of the autonomous robot. Therefore, if the information (perception of the world) that they provide, is not processed fast enough, the planner will never drive the robot normally and collisions may happen anytime. Also, the "useful" localization of the robot, i.e, that not created after the robot has already traversed the target location, will become impossible. Path planning strategies differ according to the world model and sensor perception used in the robot systems.

V.2 DIFFERENT APPROACHES TO PATH PLANNING

Generally, the systems which use partially modelled world (an a priori map)[4, 6, 26], implement two kinds of path planning: global path planning and local path planning. Global path planning has the task to find a route from a starting point to a destination point in the map. Local path planning is concerned with guiding the robot on its way by avoiding obstacles and keeping it on the track. The choice of the optimum path in problems such as this one may be dictated by certain cost criteria, such as the length of the path, the obstacles on the path and so forth.

A floor setting in a building can be described as corridors (hallways) connecting locations of that floor. The corridors intersect at some points. A description of the floor map such as formulated in this project can be represented by a simple planar graph. The graph is made simple by the assumption that between two consecutive rooms there exists only one corridor.

As mentioned in the first paragraph of this chapter, path planning methods differ from many perspectives. Systems that build a road map while navigating (built-map-based systems)[11, 13, 43], combine all the path planning into local path planning. There is no need to find a global path. However, the robot maintains a heading and an angular orientation. The heading is the compass orientation of the robot (angle and cardinal coordinate). The models of features to expect in the scene at the landmark locations, are stored in the knowledge base of the robot.

In the first approach to the a priori map-based navigation, the global path planning is justified to find an "optimum" path. For the second approach, there is almost no need for global path because this method is not based on a vertex-graph description of the world .

In the approach of R.C. Arkin [26], the Navigator accepts a starting point and a destination point. It searches for an optimum path using the A* algorithm. The input of the A* algorithm consists of the start and goal nodes and the space of midpoints (A*-1) or

triads (A*-3) of connecting adjacent meadows free of obstacles. In other words, a meadow map is a connectivity graph of free-space regions, i.e, meadows. The output is a coarse path consisting of a series of piecewise linear segments connecting the start point, the edges of bordering meadows and the goal point. The A* algorithm search has been implemented using a cost function based on terrain factors and traversability. The cost function is the sum of two components. One component is the measured cost of the path up to the current point and the other component is the evaluation (prediction) of the cost from the current point to the goal point. It takes into account a heuristic function. The path found is then passed to a path improvement module, which takes the terrain type into consideration.

In the model tested on the Stanford Cart and the CMU Rover, Crowley [20] finds a global path from a source point to a goal point. The path is a collision-free path without obstacles. A shortest path is computed using the Dijkstra's algorithm. Rotations are taken into account in the computation. The rotations are the movements of the wheelchair round its vertical axis. In those situations, the wheelchair turns almost at the same place without covering any distance on its path. However, since the wheels turn, distance is recorded.

Habib and Yuta [6] in their model of Yamabico.M-12 distinguish two cases of path planning: the in-room path planning and the corridor and building path planning. The in-room path planning is solved as a free space problem. They use PRA's (prime rectangular) methods to find a path through the room. A PRA is a rectangular area in the room, such that the optimum path between any two points in the PRA is included within the same PRA. So they search for the optimum path in terms of PRA sequence and specify the optimum path in each PRA. The corridor path planning is a planar graph problem of finding an optimum path between two nodes of the graph. They use a graph-model of the world. The building path planning is concerned with finding an optimum path made up of bridge(s) to go from one building to another in a campus environment. The output of global

path planning is a route map with information provided for each bloc included in the path. Local path planning is used to guide the robot on its route.

The *route runner* has the task of performing local path planning, i.e, to guide the robot on its local path. In this sense, the *route runner* plays the role of a pilot for the device. The main functions of this module are to control the wheels, to go straight, to turn and to use sensor perception (vision, sonar) of the environment to make navigational decisions.

Madarasz [4] in his model, uses global path planning to find the shortest path between two nodes in the map graph. In this model, path planning includes navigation between points of different floors. The output of global path planning is a sequence of commands to the motors of the wheelchair such as move, rotate etc. The commands are executed sequentially.

Closely examined, the problem of localization (scene recognition) seems to have common ground with the local path planning problem. The robot must sense its environment and recognize some features in the scene so that it can avoid obstacles, locate itself on the road, and locate itself globally as well (estimate its current location). These are important questions that directly touch the localization problem.

The PSUBOT wheelchair model uses global and local path planning. In this model, the *Pilot* [3] is responsible for local path planning, whereas the database is responsible for global path planning. The *Navigator* asks the database to locate the robot (find the current location), and issues a goal location. The database finds a global path in the vertex-graph representing the map (as a collection of corridors). The optimum path is computed with the cost criteria: length, traffic-frequency and obstacle-density. The optimum path is provided to the Navigator as a sequence of locations on the path from source to destination. The Pilot takes the task of guiding the robot on its road. The Pilot is basically responsible for road following, obstacle avoidance and orientation of the wheels. The Navigator receives sensor perception of the world through the database. The database combines sonar data and vision

data to estimate the location of the robot and to recognize some items on the scene. The Navigator is also responsible for in-room path planning. The method used for in-room path planning is similar to Habib's method [6]. The room is described as a list of forbidden rectangles. Each forbidden rectangle represents an obstacle in the room. Knowing the position of these rectangles, it is possible to find a path that uses the obstacle-free space. One major disadvantage of this method is that it is not flexible enough to dynamically update the information about the position of obstacles in the room. The disposition of obstacles in a room vary constantly at the will of the room user. So it will require frequent updating of obstacle information. The a priori knowledge will be simply a guide. The in-room navigation should be strongly assisted with sonar mapping [43]. New forbidden rectangles may be built and the information in the database can be updated. Basically, as the sonar scans the room, the database determines the forbidden areas quickly and the navigator has just to position and guide the robot amidst the obstacles.

V.3 OPTIMUM PATH PROBLEM

Global path planning raises the problem of finding an optimum path between two points. In the vertex-graph description approach, a floor can be described as a graph whose edges are corridors and the nodes(vertices) are the intersection points of corridors.

V.3.1 Definitions and Notation

The graph is made of collections of edges and vertices. A vertex is a node of the graph, whereas an edge is an arc connecting two vertices. The problem deals here with a directed graph $G = (N, A)$ where $n = |N|$ is the number of nodes (vertices) and $m = |A|$ is the number of arcs (edges); A is the set of undirected arcs. Each arc has a specified length a_{ij} and costs c^t_{ij} attached to it. A cost may be the length of an arc if distance is concerned or other measures, for example, the obstacle density. The arc's length may be positive or negative. A length can represent any given measure. In a flow graph, for instance, a negative length

may represent a flow in a direction opposite to the conventional direction of flow. An assumption is made that the graph is planar and contains no loops. A loop is defined in this formulation as an edge which has a single summit (the two summits are the same point). A loop can also be extended to designate any "double" edge between two adjacent nodes and such that the lengths of those edges are different.

A path is defined as a " simple," " loop-free" directed simple path. The length of the path is the sum of all the lengths of the arcs included in the path. The problem is to find the shortest path between two nodes of the graph. The shortest path is understood as a path of minimum "distance" from the starting point to the goal point. Finding the path between two nodes is a particular case of the general problem of finding all the shortest paths from a source node to all the other nodes of the network. For the PSUBOT wheelchair, we have selected distance as a suitable cost criteria. In fact, in order to go from one location to another, the wheelchair covers a physical distance. The shortest path between two nodes in such a graph is traversed arc by arc, i.e, from node to node, by selecting the arcs with minimum length. Finding the path between two nodes i and j requires the solution of Bellman's equations.

Bellman's equations. Let be u_j the length of the shortest path from the origin (node 1) to node j . Let be n the number of nodes and m the number of edges.

$$(1) u_1 = 0 \quad /* \text{ length of the shortest path from node 1 to node 1 } */$$

$$(2) u_j = \min \{ u_i + a_{ij} \} \quad j = 2, 3, \dots, n.$$

$$(i,j) \in A$$

There exists a directed tree from the origin node 1, such that the length of the unique path from node 1 to node j in the graph is equal to u_j if any such path exists. Such tree is called the shortest path tree. The problem is to solve the Bellman's equations((1) and (2)) and to construct the shortest path. Many algorithms exist in literature, of which one of the

most common is the Dijkstra's algorithm [44]. Some shortest path algorithms are improvements to the Dijkstra's algorithm.

V.3.2 The Dijkstra's Algorithm

With the notations and assumptions of section V.3.1, the Dijkstra's shortest path algorithm can be formulated as follows:

Step 0. Set $u_1 = 0$

Set $u_j = a_{1j}$ if node $(1,j) \in A$
 $= +\infty$, otherwise

Set $S = \{2,3, \dots, n\}$ /* set of nodes not yet visited */

Step 1. Find $k \in S$, where $u_k = \min_{j \in S} \{u_j\}$ /* take the node of the smallest distance to source */

If $u_k = +\infty$, stop; /* there are no paths to the nodes remaining in S */

Set $S = S - \{k\}$ /* node k has been visited */

If $S = \emptyset$, stop; /* the computation is completed */

Step 2. Set $u_j = \min \{ u_j, u_k + a_{ki} \}$, for all $\{k,j\} \in A$. /* update the distances to source*/

Go to Step 1.

Step 2 calls for $O(m)$ additions and comparisons overall. Step 1 calls for $O(n^2)$ comparisons overall if the values of u_j are maintained in a simple array. So the overall time is $O(n^2)$. ($n \leq m$). The time could be decreased by maintaining the u_j in a priority queue. Establishing the queue will cost $O(n)$ time and Step 1 will cost $O(n)$ time. Step 2 will require $O(\log(n))$ without the use of priority queues and $O(m \log(n))$ with the use of priority queues. Therefore, the algorithm can be implemented in $O(m \log(n))$ or in $O(n \log(n))$ time in a network(graph) with nonnegative arcs.

V.4 GLOBAL PATH PLANNING FOR PSUBOT

The problem to solve in this research is to find the shortest non cyclic path from the current location of the PSUBOT wheelchair to the location specified by the user. The map of the world in which the wheelchair is supposed to travel has been formulated in this project as hierarchical map (see Figure 2 of Chapter III). In this description, a floor is described as a collection of intersecting corridors. Along each corridor there are items representing room doors, access and exit doors, stairs and special landmark locations. We have formulated the solution of the global path planning in such a way to benefit from the hierarchical description of the map. One basic assumption has been made that between two locations there is only one direct arc, i.e, one corridor connecting them. This assumption yields a simple graph without bilateral arcs. Taking into account this consideration and the usual situation in buildings, we have assumed that the shortest and optimum path between two points on the same corridor belongs to the same corridor. The global path planning has been formulated as a hierarchical problem (see Figure 12). Two main cases are distinguished depending on whether the starting point and the destination points belong to the same building or to different buildings. Additionally, in the same building, a question is to know whether the starting and goal points belong to the same floor or to different floors in the building. Furthermore, it is necessary to know if, on the same floor, the two points belong to the same corridor or to different corridors. In the same corridor the task is to use deductive methods and inference to find the ordered list of locations and observation points between the starting and destination points. When the two points are located on different corridors of the same floor, an optimum path algorithm will be called to compute the global path. Following the hierarchical approach, the global path between two points will be the union of portions of paths (see Figure 13).

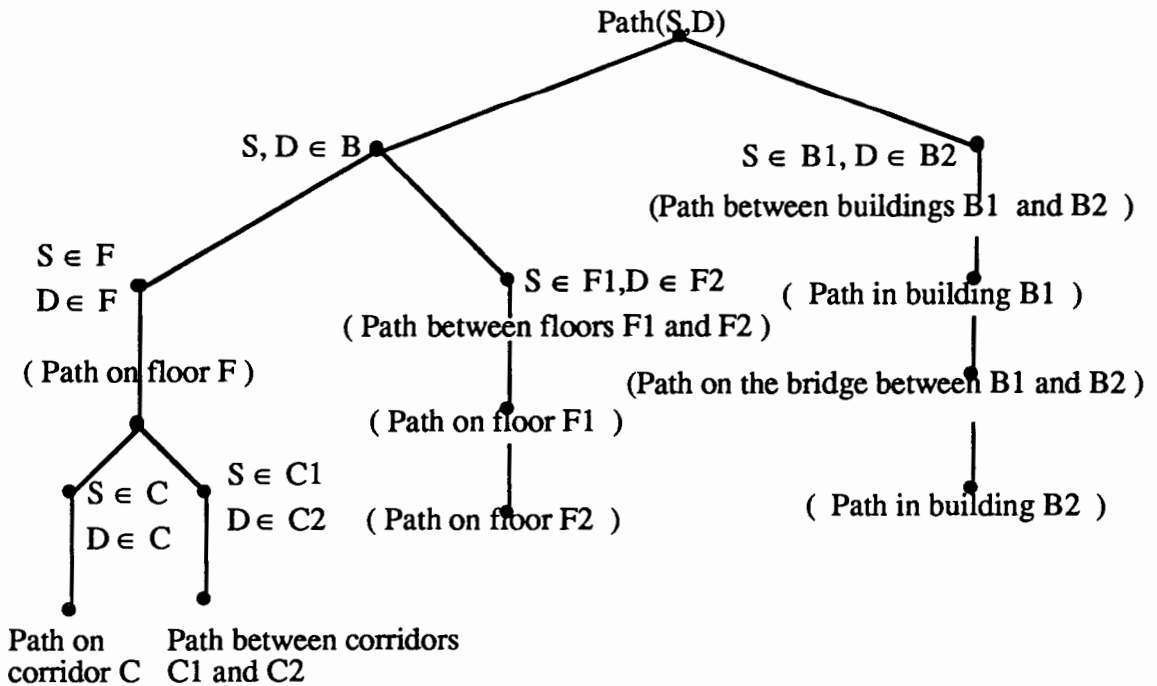
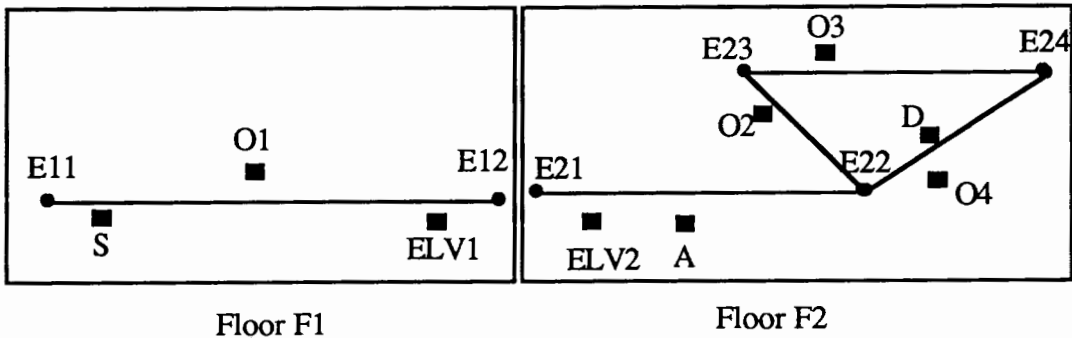


Figure 12. Hierarchical approach to global path planning.

Example 1:

In this example (see Figure 13), the problem to solve is to find an optimum path between S and D. Point S is located on corridor C1 of floor F1 and D is located on corridor C5 of floor F2 in the same building. The path will be computed from S to ELV1 (elevator point of floor F1) on the same corridor and from ELV2 (elevator point of floor F2) to point D. One portion of the total path is S-ELV1 and the other one is ELV2-D. The first part of the path is retrieved using the backtracking method to list all the locations between S and ELV1. This gives a portion of path S-O1-ELV1. The second part ELV2-D is computed as follows. An optimum (the shortest) path is computed between the nodes E21 of corridor C2 and E24 of corridor C5, giving a path E21-E22-E23-E24. Thereafter expert system deductive and inference approach is used to exclude from the final path the portions that are not physically included, giving a portion of path ELV2-A-E22-O2-E23-O3-D. We used this method because graph algorithms compute the path between nodes of the graph. The

requirement for the PSUBOT is to find an actual route as the list of all the locations and observation points between the starting and goal locations. In this example ELV1 and ELV2 are the access doors of the same elevator, but located on different floors.



$$\begin{aligned} C4 : E22-E24 \quad |E23E24| &= 0.8m \\ C5 : E23-E24 \quad |E23E24| &= 0.2m \end{aligned}$$

Figure 13. Example of a problem showing the hierarchical approach to global path planning.

The major contribution of this work is to combine the approach of using intelligence-like deductive and inference approach to add flexibility to global path planning. The claim that supports our statement is that powerful methods such as the A* methods give good results, but an approach such as ours saves time. If we take a problem where the starting and goal points are in the same corridor, a solution using the A* algorithm will compute the path by checking possible solutions in the whole graph. Our method will not go further. It will identify the category of the solution immediately and exclude other cases. In the implementation of the global path algorithm, we have broken the problem into eight categories of queries as follows.

V.4.1 Query Simplification

The request for computation of the global path between two points can be formulated as:

path([L1,F1,B1],[L2,F2,B2]).

L: location; F: floor ; B: building

The query grammar is composed of nine patterns of queries :

(0i) basic primitive: [L1,F1,B1]----[L2,F2,B2]

"Find path from location a L1 on a floor F1 of a building B1 to a location L2 on a floor F2 in a building B2."

(1i) (1) [L1,F1,xx]---[L2,F2,xx]; (2) [xx,xx,xx]---[L2,F2,xx]

Case(1): "Find path from a location L1 on a floor F1 to a location L2 on a floor F2 in this building."

Case(2): "Find path from HERE to a location L2 of a floor F2 in this building."

(2i) (1) [L1,xx,xx]---[L2,xx,xx]; (2) [xx,xx,xx]---[L2,xx,xx]

Case(1): "Find path from a location HERE (L1) to a location L2 in this building."

Case(2): "Find path from HERE to a location L2 in this building."

(3i) [L1,xx,xx]---[xx,F2,xx]

"Find a path from HERE (L1) to floor F2 in this building."

(4i) (1) [L1,F1,xx]---[L2,xx,B2]; (2) [xx,xx,xx]---[L2,xx,B2]

Case(1): "Find path from a location L1 on a floor F1 to a location L2 in a building B2."

Case(2): "Find path from HERE to a location L2 in a building B2."

(5i) (1) [L1,xx,B1]---[L2,xx,B2]; (2) [L1,xx,xx]---[L2,xx,B2]

Case(1): "Find path from a location L1 in a building B1 to a location L2 in a building B2."

Case(2): "Find path from a location L1 in this building to a location L2 in a

building B2."

(6i) (1) [L1,xx,B1]---[xx,xx,B2]; (2) [xx,xx,xx]---[xx,xx,B2]

Case(1): "Find path from a location L1 in a building B1 to a building B2."

Case(2): "Find path from HERE to a building B2."

(7i) (1) [xx,xx,xx]---[xx,F2,B2]; (2) [xx,F1,xx]---[xx,F2,B2]

Case(1): "Find path from HERE to floor F2 of the building B2."

Case(2): "Find path from floor F1 of this building to floor F2 of the building B2."

(8i) [xx,xx,xx]---[L,zz,zz]

"Find path from HERE to a location L on this map(building not indicated)."

Note: xx or zz means "not indicated." Each time the starting location of the path is not indicated, the program will try to find it either by retrieving the name of the building from the information on that starting point, or by LOCATING the current position.

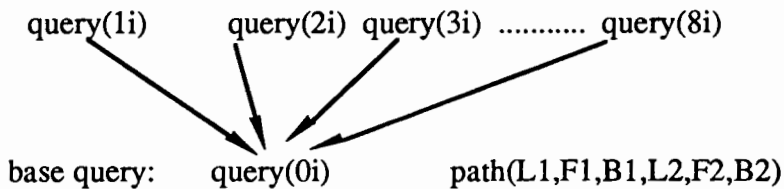


Figure 14. Query simplification.

Each of the the categories of queries from (1i) to (8i) can be simplified (see Figure 14) into the basic category of query (0i). Category of query (0i) computes the path between two points (L1,F1,B1) and (L2,F2,B2) where the variables are clearly identified (building, floor, location).

When the starting point and destination point of a portion of the path fall into the category " path between corridors," an optimum path algorithm using a graph method, such as the Dijkstra's shortest path algorithm, is invoked. Particularly in this project, the

shortest path algorithm used is the Dijkstra's algorithm. The problem in this case can be formulated in the following way:

The corridor is an edge(arc) between two vertices (points of the floor graph). To each corridor are attached supplementary cost functions: traffic frequency and obstacle density. The effect of these measures will be included in the length cost of the corridor as a correction factor(see equation (1) of section III.3.2). The problem is then that of a simple undirected graph with its edges having nonnegative lengths. The problem is to find a path of minimum value of the cost function. The cost function is the total length of the path between the source and destination. A correction factor has been added to the length of the corridor in order to take into account the obstacle density and the traffic frequency (see equation (1) of section III.3.2).

V.4.2 Pseudocode of the Path Finder Algorithm

The problem of finding the path in the map structure proposed can be solved as a hierarchical problem. The algorithm can be represented as a tree of cases. The leaves of the tree represent the two cases: path on corridor and path between corridors (see Figure 12). The path is found from nodes of the graph. It means that if the goal node is not on the current floor, a search will be done in the knowledge base to identify the corridors where it belongs and the problem will be reduced to the regular problem (see Figure 14). There are fourteen different types of queries in the implementation of the global path planner for the PSUBOT. Each of those queries can be reduced to the basic query where all the variables of the query are known (see Figure 14). The starting and destination nodes may not belong to the same graph. This situation can happen if the starting and goal points are located on different floors of the same building, or in different buildings. In those cases, a search will be made to identify the corridors they belong to, and the path will then be found between the two corridors. The final path will be the union of the paths.

The Dijkstra's algorithm computes an optimum path from one node to all the other nodes of the graph. For this reason, in our implementation, the destination point will be taken as the source point in the formulation of the Dijkstra's algorithm. Paths will be then computed from the destination point to all the other points of the graph.

(a) The MAIN program (declarative)

The pseudocode of the main program of the path finder is shown in APPENDIX B-2.

(b) DIJKSTRA's ALGORITHM (procedural)

The pseudocode of the implementation of the Dijkstra's algorithm is shown in APPENDIX B-3.

V.4.3 Other Considerations

This consideration is with the Navigation mode. When the starting point and the destination point of the portion of path to execute belong to the same corridor, there is a need to focus the attention of the sensory module (vision and sonar) to allow the wheelchair to recognize its destination location. This is what was mentioned as a "bootstrap" mode [23]. In this mode, images are taken at close distances in order to increase the chances to recognize the target destination. At this level the speed of the robot should be reduced to allow the processing to take place. When the robot is not travelling on the corridor containing the target destination, the navigation is in "feed- forward" mode [23]. Images are taken sparsely with the only purpose of recognizing the end of corridors, until the final corridor of the route is reached. Then the robot enters in a bootstrap navigation mode.

V.5 IMPLEMENTATION OF GLOBAL PATH PLANNING

The algorithm of the main program for the global path planner has been implemented in Turbo Prolog to take advantage of the artificial intelligence capability of Prolog and the declarativeness of the language as well as some features such as backtracking, which otherwise would require complex programming in a procedural language such as C. On the other hand, the Dijkstra's algorithm has been implemented in Turbo C++ to take advantage of the fast computation power that is offered in C. Turbo C or Turbo C++ are selected because we want to integrate the whole database on a PC. Therefore, using Turbo Prolog and Turbo C, both designed by the same company (Borland), provides a built-in interface between the two languages. Turbo C (ANSI C) could have been used instead of Turbo C++. These programs really don't use or need full use of the capabilities of C++ such as classes etc.. The reason why we have used C++ is in anticipation that in the future the whole system will be implemented in C++. The central control module of the system (the Navigator or other) will manage the other modules as objects in the concept of object-oriented programming. In this section we only present the results of the testing. The evaluation will be discussed in Chapter IX.

The choice of the Dijkstra's algorithm over other well performing ones such as the A* was dictated by :

- Simplicity -- the graph of a floor is not so large, therefore we judge that applying the A* search algorithm is a waste of resources. The A* algorithm gives better results but it is computationally intensive,
- the algorithm is fairly simple to implement,
- the goal is to have an algorithm, though not the optimum necessarily, fast enough to fit the time constraints.
- the Dijkstra's algorithm, as different from other similar algorithms, computes the path

from source to all the other nodes of the graph. Therefore, if the path between two points has been computed, there is no need to recompute if a subsequent request specifies the same destination. Therefore, before starting the global path planning between two nodes, a test will be performed to find out if a request has been made previously to find a path to the same goal location as presently. Since the destination is the same and knowing that the Dijkstra's algorithm computes the path from one point to all the other nodes, there is no need to compute an optimum path for a subsequent request if the goal point hasn't changed and the starting point is on the same floor as the goal point. Each time the algorithm is called, the path from each point to the goal point will be saved in memory until the destination point has been updated (new request for global path).

V.5.1 Testing and Results

We have tested the cases that cover most of the main categories of problems: path between points of the same building, path between points of different buildings, path between points of different floors, path between points of the same corridor. The results in Table III show the time it took to compute such paths.

Test Example 1. The example (see Figure 15) represents a floor plan used by Madarasz et al. to test their model. Our formulation of knowledge base facts of this example is shown in appendix A-2. This example was tested to find the path between two points of the floor. The points were chosen belonging to the same hallway or not. The stack size is 600 bytes. The database size is 6156 bytes and made up of 104 knowledge base facts. The results are shown in Table III.

Test Example #2. This example (refer to Figure 16) is intended for the evaluation of the speed of the path planning algorithm. The test results are summarized in Table IV.

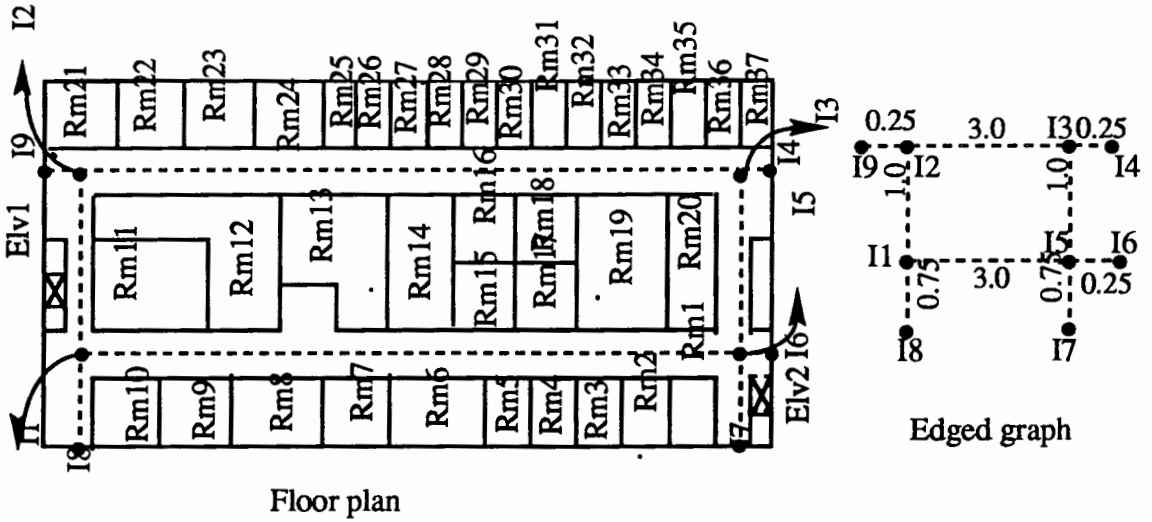


Figure 15. A problem from Madarasz's model [4].

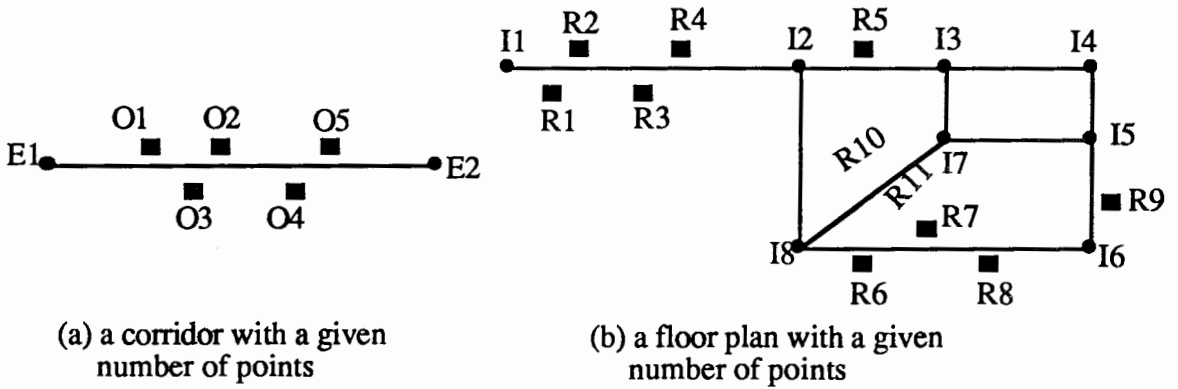


Figure 16. Evaluation of the speed of the path planning algorithm.

TABLE III

PATH ON CORRIDOR: COMPUTATION TIME AS FUNCTION OF THE NUMBER OF OBSERVATION POINTS

N	2	4	6	8	10	15	20	25
$\Delta\tau$	0.28s	0.22s	0.27s	0.27s	0.27s	0.33s	0.38s	0.39s

Note. Data from example in figure 16.a.

TABLE IV
PATH PLANNING RESULTS - EXAMPLE 1

Request	Computed path	Computation time (sec)
Path(Rm1, Rm9) (on same hallway)	2 -2.4ft Rm1-Rm2-Rm19-Rm3- Rm4-Rm17-Rm15-Rm5- Rm6-Rm14-Rm7-Rm13- Rm8-Rm12-Rm9	0.93s
Path(Rm9, Rm1) (see Note 1)	2 2.4ft Rm9-Rm12-Rm8-Rm13- Rm7-Rm14-Rm6-Rm5- Rm15-Rm17-Rm4-Rm3- Rm19-Rm2-Rm1	0.88s
Path(Rm12, Rm2)	2 1.81ft Rm12-Rm8-Rm13-Rm7- Rm14-Rm6-Rm5-Rm17- Rm4-Rm3-Rm19-Rm2	0.93s
Path(Rm2, Rm29) (on different hallways) (see Note 2)	2 0.6ft Rm2-Rm1-I5 1 1.0ft I5-I3 2 -1.9ft I3-Rm36-Rm35-Rm34- Rm33-Rm18-Rm32-Rm31- Rm16-Rm30-Rm29	2.64s
Path(Rm6, Rm30) (see Note 3)	2 -1.2ft Rm6-Rm14-Rm7-Rm13- Rm12-Rm9-Rm10-I1 1 1.0ft I1-I2 2 1.2ft I2-Rm22-Rm23-Rm24- Rm25	-----

Note 1. The path is computed portion by portion. Each portion is listed in the following way:

Navigation mode in the portion	length
List of locations (Location name, Floor, Building)	

The lengths are given in feet. The numeric values are testing values rather than values of a real application. A navigation mode of 2 means "bootstrap" mode. The wheelchair is navigating on a corridor where the starting point or destination point of the path are located at. There is a need to look at the scene more closely (images are taken more frequently in time).

A navigation mode of 1 means "feedforward" mode. The wheelchair is on a portion of path far away from the end-points of the desired route. Pictures need not be taken as frequently.

The length is computed negative if the direction of the traversed path is opposite to the orientation on the given corridor.

TABLE IV
 PATH PLANNING RESULTS - EXAMPLE 1
 (continued)

Since room Rm20 and Rm1 are directly opposite to each other they are assigned the same "coordinate" value. Consequently Rm20 is not listed. The path planner program uses internal backtracking to list the locations on the hallway. The locations are listed in the manner predecessor-successor. The next point is the point having the "coordinate" with the value immediately greater or smaller than the "coordinate" of the current point. To avoid the problem mentioned, two doors facing each other on a hallway should be given slightly different "coordinates".

The computed paths obtained from the requests Path(Rm1, Rm9) and Path(Rm9,Rm1) are strictly reversed compared to each other, which shows the reversibility of the path computation.

Note 2. This request computes the path between two points located on different hallways. The program first identifies the hallways on which the two points are located (C(I1,I5) for Rm2 and C(I3,I2) for Rm29), then finds the end-points closest to the starting and destination points (I5 for Rm2 and I2 for Rm29). Then, the program retrieves the edged graph of the floor as edges and vertices. A vertex represents a corridor end point. An edge represents the corridor. The cost assigned to an edge is the length of the corridor in feet. In this testing we have used the corrected lengths (taking into account the obstacle densities and traffic frequencies on the hallways).

After graph retrieval, the shortest path is computed between the points I5 and I3 of the edged graph by applying the Dijkstra's shortest path algorithm. The result is a path node-to-node described as a list of locations and the length of the path:

["I5", "I3", "I2"]
 4.0ft

This result is an intermediate one and is not listed in Table III.

The path improvement is called. At this step the portions of the path that are not physically included in the desired path are removed by applying knowledge method.

Note 3. For this request the program terminated before completion due to stack overflow. Even when we increased the stack size to a maximum of 4K, the problem could not be solved. The drawback is that the program uses recursion and backtracking to list the locations on the path. This becomes a limitation when the number of points on the physical path exceeds 16. Also the version of Turbo Prolog used doesn't support global variables. Therefore we had to pass the same variables each time between nested sub-routines, increasing the demand on the stack.

A future improvement could come from using a version of Turbo Prolog with global variable capability as well as the capability of releasing unnecessary variables at run time. Another precaution is to limit, as much as possible, the number of observation points on each corridor.

The location Elv1 was intentionally not listed in the path (I1, I2) because this path is navigated in "feedforward" mode. Only the nodes (intersection of corridors) are the points of interest in this mode.

In Table IV, N is the number of observation points along the corridor and $\Delta\tau$ is the path computation time. The curves on Figure 17 show that when the number of observation points on the corridor grows very big the computation time will reach a highest value of 2.3 seconds. This result leads us to conclude that, with this algorithm, the maximum time spent on computing a path between two points located on the same hallway is 2.3 seconds.

In the graphs of Figure 17 and Figure 18, N represents the total number of (observation) points of the floor graph. It can be concluded from the exponential curve fitting function that, when the number of points in the graph increases towards a big value, the time needed to compute a path between two points located on different hallways (resp. on the same hallway) asymptotically moves towards the value of 17.07 seconds (resp. 1.97 seconds). The results in this case are compatible with those of case 16.a. This estimation proves that the path planning algorithm implemented is fast enough on PC. Further discussion and evaluation of the path planning will be conducted in Chapter IX.

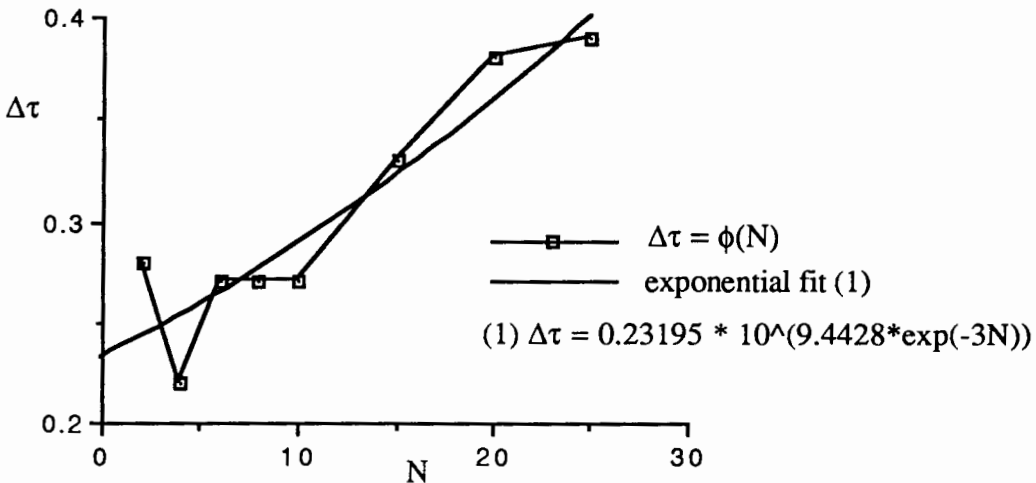


Figure 17. Computation time vs number of observation points on the corridor.

The results of the simulation are summarized in Table V. The emphasis on this simulation is to show that the computation time depends on the total number of observation points of the graph. The assumption which leads to this investigation is that the retrieval of the path is done through the backtracking process of Turbo Prolog.

TABLE V

PATH ON FLOOR: COMPUTATION TIME AS FUNCTION
OF THE NUMBER OF OBSERVATION POINTS OF THE GRAPH

Problem size	Path on corridor	Path between corridors
19 (the whole graph)	Request: Path(R1 , R4) $\Delta\tau : 0.48s$	Request: Path(R4 , R9) $\Delta\tau : 2.25s$
18 (graph I1-I2-I8-I7-I3-I2, I7-I5-I6-I8)	Request: Path(R1 , R4) $\Delta\tau : 0.38s$	Request: Path(R4 , R6) $\Delta\tau : 2.20s$
14 (graph I2-I8-I7-I3-I2, I3-I4-I5-I7, I5-I6-I8)	Request: Path(I8 , R8) $\Delta\tau : 0.38s$	Request: Path(R5 , R6) $\Delta\tau : 2.08s$
12 (graph I1-I2-I8-I7-I3-I2)	Request: Path(R1 , R4) $\Delta\tau : 0.28s$	Request: Path(R4 , R10) $\Delta\tau : 1.97s$
10 (graph I7-I5-I6-I8-I7)	Request: Path(I8 , R8) $\Delta\tau : 0.32s$	Request: Path(I7 , R8) $\Delta\tau : 2.03s$
7 (graph I1-I2-I3)	Request: Path(I1 , R4) $\Delta\tau : 0.27s$	Request: Path(I1 , R5) $\Delta\tau : 1.87s$

Note. Data from example in figure 16.b.

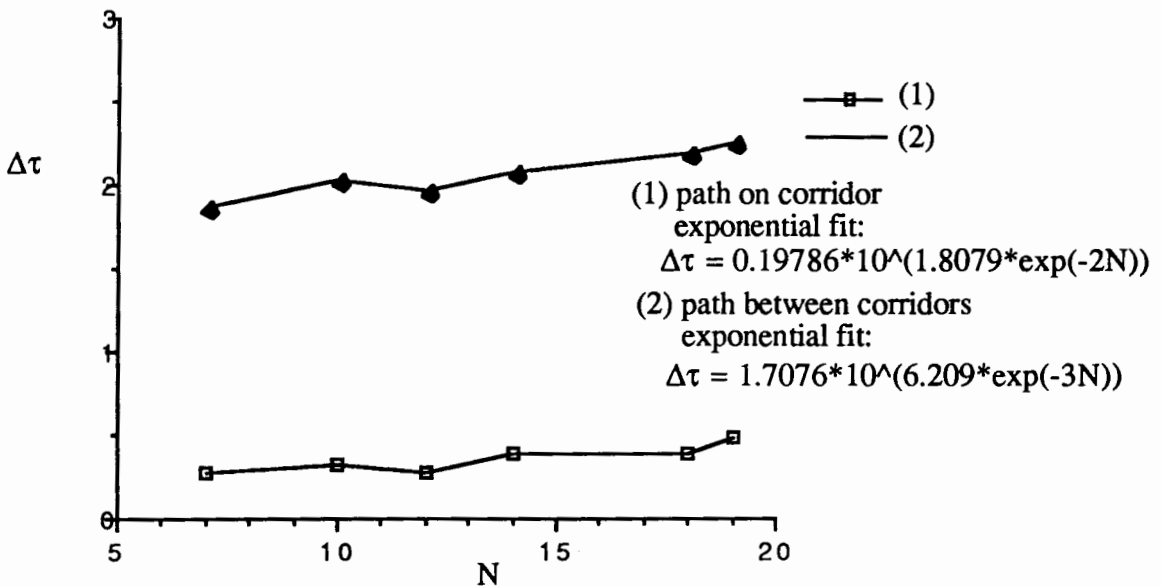


Figure 18. Computation time vs the problem size.

Test Example #3. The purpose of this test example is to answer the queries. The results are summarized in Table VI. The map is that of the complete example of a campus made of two buildings shown in APENDIX A-1. The size of the knowledge base is 260 facts occupying 16125 bytes of RAM.

Query (1) computes the path between two points located on the same hallway. Query (2) and (3) compute the path between points located on different hallways of the same floor. The result of query (3) is not optimal. The path should have been 128-I9A-I12-I2-I3-105-103-104-102.

The reason is that the Dijkstra's algorithm computes the optimum path from node to node. In this case the input of the algorithm was [128,103]. There was a choice between the initial edge I9A-I12 (14.4m) and I9A-I9 (2.4m). The edge with minimum length from the starting point I9A was therefore I9A-I9. Therefore the path was directed from I9A to I9 instead of I9A to I12. An improvement should be to check the length of all the edges connected to each edge issued from the starting point.

Query (4) computes the path between points located on different floors of the same building. An elevator (Elvt) is found connecting the two floors, otherwise the program stops the computation because there is not path for the wheelchair other than stairs (which is not a safe or feasible solution).

Query (5) computes the path between two buildings connected by a "bridge", i.e, a path that the wheelchair can travel on. The path is computed portion by portion from building B1 to building PCAT. Query (6) answers the same request. However, due to memory limitation and the total number of locations on the path, the program ended because of a stack overflow. However the portion of path computed was:

15-14 (building B1) I1-11-ExitB11 (building B1) ExitB11-Exitb1 (bridge)

Exitb1-Stairs-28 (basement PCAT) 28-27-26 (basement PCAT).

TABLE VI

PATH PLANNING RESULTS - EXAMPLE 3

Query	Computed path	Computation time (sec)
(1) Gpath("170G","firstfloor","PCAT", ,"170D","firstfloor","PCAT")	2 -5.2ft 170G-170A-170F-170B- 170E-170C-170D	1.32s
(2) Gpath("170E","firstfloor","PCAT", ,"102","firstfloor","PCAT")	2 -1.2ft 170E-170C-170D 2 32.4ft 170D-170H-1601-I4-I3 2 6.0ft I3-105-103-104-102	3.62s
(3) Gpath("128","firstfloor","PCAT", ,"102","firstfloor","PCAT")	2 0.0ft 128 1 79.6ft 128-I9A-I9-I7-146-I6-I5- 1601-I4-I3 2 6.0ft I3-105-103-104-102	3.54s
(4) Gpath("28","basement","PCAT", "1 46","firstfloor","PCAT")	2 -2.0ft 28-27-26-Elvt (basement) 2 1.2ft Elvt-124 (first floor) 1 32.8ft 124-128-I9A-I9-I7-146 2 0.0ft 146	3.62s
(5) Gpath("15","firstfloor","B1","28", ,"basement","PCAT")	2 -2.8ft (building B1) 15-14 1 17.8ft (building B1) I4-I2-I1 2 4.2ft (building B1) I1-11-ExitB11 1 20.0ft (bridge PCAT-B1) ExitB11-Exitb1 2 0.0ft (PCAT basement) Exitb1 1 5.0ft (PCAT basement) Exitb1-Stairs-28	3.2s
(6) Gpath("15","firstfloor","B1","146", ,"firstfloor","PCAT")	Path not completed because of stack overflow	-----

CHAPTER VI

PSUBOT DATABASE ORGANIZATION

VI.1 GENERAL CHARACTERISTICS AND ASSETS OF THE DATABASE APPROACH

In section IV.5 of Chapter IV, an investigation has been conducted to find the best representation of data for mobile robots. In almost all the approaches where a database has been used to keep data for the functioning of the robot, it has been noticed that the database was not by itself identified as a main module of the system and was used to store a priori data rather than for the knowledge support [18, 6, 26]. Most of the time the tasks are organized only around the *Navigator*, the *Path Planner* or the *Sensory Module*. In the approach used in this work, the database has been created as a stand alone module of the system. This method, far from being too original in general, seems quite interesting for PSUBOT for several reasons. First, we have mentioned in the previous chapters of this thesis that all the modules of the PSUBOT system can communicate through the database and hence the need for dedicated interfaces is eliminated. Secondly, all the information on the wheelchair operation is visible to each module of the system with the provision of data protection. Thirdly, the database will be a support of such mechanisms as knowledge management, and can serve also as a database for some functions such as the motor schemas used by R.C. Arkin [27]. The motor schemas can be understood as some behavioral primitives such as "move_left", "move_right", etc..

In general, the approach of building complex software systems around a database is not new. This approach is the current trend in CAD tool design. In the early stages of CAD

tools design, the designers would build a new application, then were forced to build a specific interface dedicated to interface the new application to all the existing applications of the system. With the development of database capability and database systems, researchers in this area have almost come to an unanimous conclusion that it is more beneficial to build a database first, and then build applications around the database. This approach is the same that is behind the concept of an intelligent database for a wheelchair. If the results of this work are good enough, the database will be the heart of the PSUBOT wheelchair in the sense that it will talk to all other modules of the system as objects.

One obvious problem is that for the autonomous robot, the database may be a bottleneck. Therefore, some bypass strategies should be implemented to overcome bottleneck situations. For example, the *Navigator* could ask for data directly from sonar readings if the processing in the database is timed out, or if there is an emergency situation such as an immediate obstacle on the wheelchair's way. The idea of object oriented approach mentioned by [2] in the implementation of the *Navigator* of the system is very suitable to eliminate the bottleneck problem and to insure stand-alone operations of some modules of the system. For instance, the centralized mechanism of the database can send a message to a module such as the *Navigator* to function in the stand-alone mode, just by having direct feedback from some sensors of the system. The database seems to be a suitable module to insure the function of a master for all the other modules, because in the current implementation the sensor data fusion and knowledge mechanisms are undertaken at the level of the database. The database is therefore the only module that communicates with all the other modules of the system. The next section presents the overview of the PSUBOT intelligent database system.

VI.2 OVERVIEW OF THE DATABASE SYSTEM

The PSUBOT database system has the primary goal of keeping all the necessary information for the operation of the autonomous wheelchair. Part of the information is the description of the world around the wheelchair. The second goal of the database is to provide the wheelchair with some kind of knowledge of how to use the static data and the sensor information to make intelligent inferences related to the wheelchair's operation. Therefore, the PSUBOT database system is the combination of a static relational database, a knowledge-base and management routines. Several researchers in the area of autonomous navigation of land vehicles have used the knowledge-base approach. They combine static or a priori information with a knowledge-based mechanism capable of making intelligent decisions based on the current state of the system (sensory feedback) and the static data required for the system's operations. R. Stengel and A. Niehaus [40] use the powerful knowledge base approach for automatic driving on a busy highway.

We would like to make clear that the PSUBOT database is not a DBMS Per Se. Instead, the relational part of the database has been implemented as an application. This relational database uses a commercial DBMS, the PARADOX3 and the PAL application language to make DDL and DML statements necessary to create and manage the data of the map.

The PSUBOT (see Figure 19) database has three major components:

- the *static relational database* ,
- the *management routines*,
- the *dynamic knowledge base*.

The static relational database keeps the data of the map. The management routines have various functions ranging from managing the map data, interfacing, monitoring of the whole database processes and communication with the other modules of the PSUBOT

system. The dynamic knowledge base builds information from the basic information provided by the static data base and the sensory information. There is a driver program that monitors the whole database. The schematic in Figure 19 shows the major parts of the database mentioned in this section.

The Map Manager and the Driver programs will be examined in section VI.5 of this chapter. The Interface routines are referenced in section VI.6 of this chapter. The Global Path Planner program has been referenced and reviewed in Chapter V. The Image Matcher will be referenced in Chapter VII.

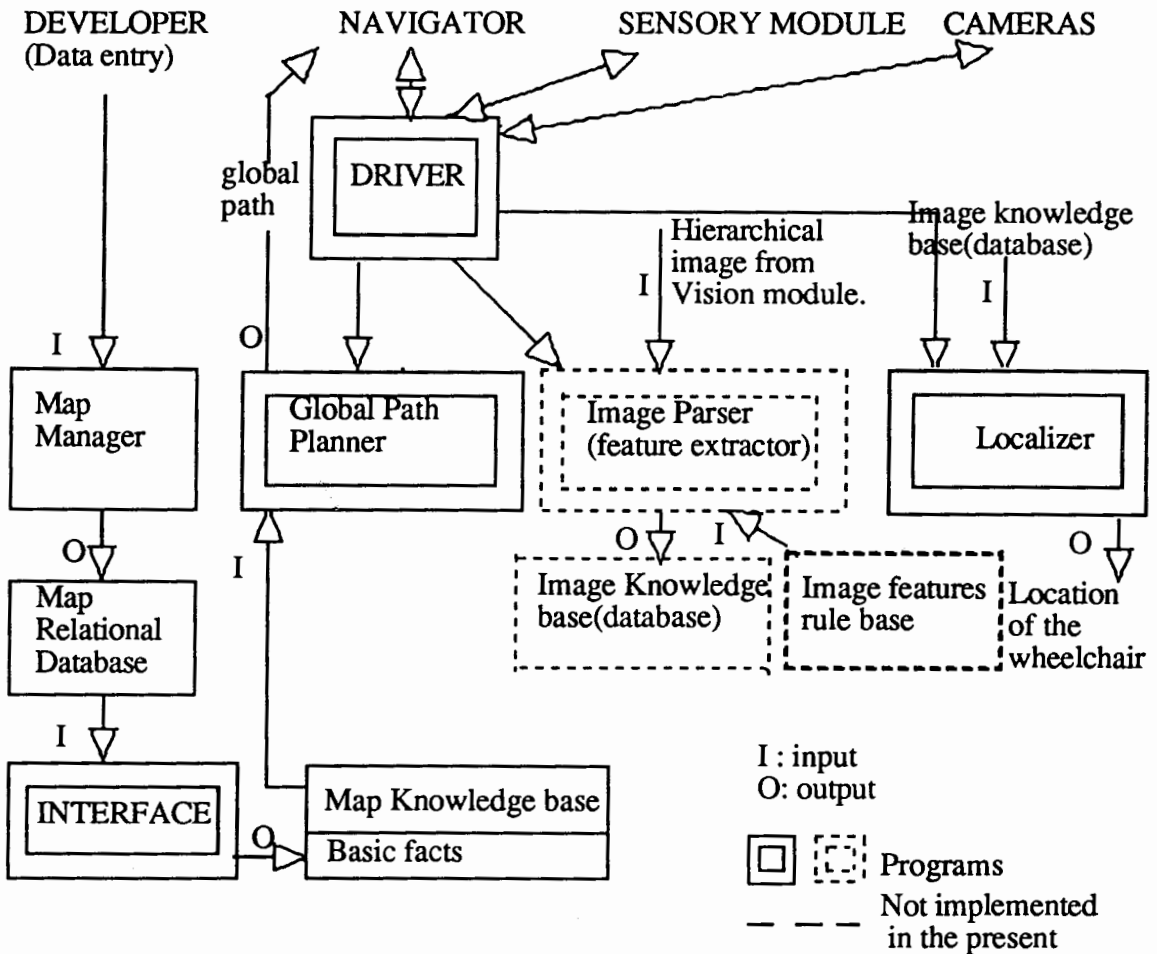


Figure 19. Structure of the PSUBOT database system.

VI.3 THE STATIC RELATIONAL DATABASE

As mentioned in the previous section, the static relational database has the role of keeping the information about the map of the environment in terms of the relative disposition of objects in the world. The static database is made up of base relations. These relations give information on corresponding levels of the map hierarchy: buildings table, bridges table, floors table, corridors table, corridor ends table, rooms table, objects table. There is a special table, the map table which keeps as a unique record the name of the map, and the names of all the seven tables of the map relational database.

Each relation will now be examined individually. The data entities represented by each table are explained. For each table the primary key will be indicated by an asterisk. All the relations in this simple database are in the 3rd Normal Form. The 3rd Normal Form has been defined in Chapter IV. To identify the components of a given map, the table names are built from the map name in this correspondence:

buildings_table = map name + "1"

bridges_table = map name + "2"

floors_table = map name + "3"

corridors_table = map name + "4"

corridor_ends_table = map name + "5"

rooms_table = map name + "6"

objects_table = map name + "7"

The "+" operation used in this formulation is the string concatenation operation. This particular coding is to allow the user to create and save many map applications in the same folder of the memory storage unit available. However, the table "map_table" will always have the information on the most recently entered or updated map. At each major operation on the map manager (enter, update, read, query) this table will be overwritten and saved with the information of the current map.

VI.3.1 Map Table Relation

The role of the table "map_table" is to carry out the name of the map for upcoming use. This table will be converted into a DOS file that will store the name of the current map. When the program creates or updates an existing map, the "map_table" table is created automatically. After the given operation, the interfacing of the PARADOX formats to the PROLOG formats is done automatically. This interfacing (INTERPAR.PRO routine) first opens the newly created "map_table" DOS file and retrieves the name of the map. The name of the map is used next to create the name of the dynamic PROLOG database to keep the knowledge base for the map. For example, if the map name is "psu", then the name of the map knowledge base file will be "psu.db". This strategy allows storage of several map applications in the disk storage.

TABLE VII
MAP_TABLE

Mname*	Btabl	Brtabl	Fltabl	Ctabl	Cetabl	Rtabl	Obtabl

Note. Mname: map name as a string of alphanumeric characters (7 max)

Btabl: buildings table name

Brtabl: bridges table name

Fltabl: floors table name

Ctabl: corridors table name

Cetabl: corridor ends table name

Obtabl: objects table name

Rtabl: rooms table name

VI.3.2 Relation "Building"

This relation keeps the information about a building of the map. The corresponding table is represented in Table VIII.

TABLE VIII
BUILDINGS_TABLE

BNAME*	MNAME	NFLOORS	PBLC	ACCFAC

Note. BNAME: name of the building as a string of alphanumeric characters (10 max).
MNAME: name of the map site as a string of alphanumeric characters (7 max).
NFLOORS: number of floors of the building (including underground floors) as a small integer number.
PBLC: "y" as yes if the building is a public building such as a Government building, a store or a school building and "n" as no if the building is not accessible to the public.
ACCFAC: "y" as yes if the building has access facilities accessible for handicapped such as ramps, automatic doors, elevators; "n" if not.

A building name in a map is unique. Therefore the attribute BNAME has been chosen to be the primary key.

VI.3.3 Relation "Bridge"

This relation keeps information on bridges of the map.

TABLE IX
BRIDGES_TABLE

BRNAME*	MNAME	BNAME1	EXIT1	BNAME2	EXIT2	LENGTH	TFREQ	OBDENS

Note. BRNAME: name of the bridge as a string of alphanumeric characters (10 max).
MNAME: name of the map site as a string of alphanumeric characters (7 max).
BNAME1: name of the of end #1 building as a string of alphanumeric characters (10 max).
EXIT1: name of the of end #1 of the bridge from BNAME1 as a string of alphanumeric characters (10 max).
BNAME2: name of the of end #2 building as a string of alphanumeric characters (10 max).
EXIT2: name of the of end #2 of the bridge from BNAME2 as a string of alphanumeric characters (10 max).
LENGTH: length of the bridge in meters as a real number floating point type.
TFREQ: traffic frequency on the bridge as a real number floating point type.
OBDENS: obstacle density on the bridge (number of fixed obstacles per unit length) as a real number floating point type.

The traffic frequency, obstacle density and map site terminology have been defined in Chapter III. The data consistency requirement is that BNAME1 and BNAME2 be distinct .

VI.3.4 Relation "Floor"

This relation keeps information on floors of a building.

TABLE X
FLOORS_TABLE

FNAME*	BNAME*	FABOVE	ELVT

Note. FNAME: name of the floor as a string of alphanumeric characters (10 max).

BNAME: name of the building as a string of alphanumeric characters (10 max).

FABOVE: name of the floor above as a string of alphanumeric characters (10 max)
will be the same as FNAME if NFLOORS = 1.

ELVT: "y" if elevator connecting FNAME and FABOVE; "n" if not.

A floor is unique in a building and a building is unique in a map, but floors usually are identified as "first floor", "second floor" etc. Therefore, a floor is identified by its name and the building name. The primary key of this table is a composite made of the attributes FNAME and BNAME. The data consistency requirement is that the attributes FNAME and FABOVE on one hand should have the same value and ELVT should have the value "y" if the number of floors in the building is 1.

VI.3.5 Relation "Room"

This relation keeps information on rooms of a floor in a building.

TABLE XI
ROOMS_TABLE

RNAME*	FNAME*	BNAME*	ENT-DOOR	EXIT-DOOR	TYPE	LENGTH	WIDTH	OBST-DISPO

Note. RNAME: name of the room as a string of alphanumeric characters (10 max).

FNAME: name of the floor as a string of alphanumeric characters (10 max).

BNAME: name of the building as a string of alphanumeric characters (10 max).

ENTDOOR: name of the main entrance door as a string of alphanumeric characters (10 max).

EXITDOOR: name of the main exit door as a string of alphanumeric characters (10 max).

TYPE: shape of the room as a small integer (1 if rectangular, 2 if L shape).

LENGTH: the larger of the two dimensions of the room or the enveloping rectangle of the room.

WIDTH: the smaller of the two dimensions of the room or the enveloping rectangle of the room.

OBSTDISPO: string of alphanumeric characters preset to "[obst(____)]" representing the list of forbidden rectangles inside the room; this list represents the obstacles in the room.

A room name is unique on a floor, a floor is unique in a building and a building is unique in a map. The primary key has been chosen to be composite (RNAME,FNAME,BNAME). The data consistency requirement is that the room name RNAME as well as ENTDOOR and EXITDOOR should figure in the list of objects (points) along a corridor of the floor. ENTDOOR and EXITDOOR will be the same if the room has only one access door. The symbol "[obst(.,.,.,.)]" represents an empty list of obstacles in a room (see section VI.4). A list of obstacles in a room is represented as the following:
 [obst(rho1,theta1,length1,width1),obst(rho2,theta2,length2,width2),....].

The symbol *obst(rho,theta,length,width)* represents an obstacle represented as a forbidden rectangle(see Chapter III). At the data entry(at the level of the relational database) a list of obstacle is entered in a table as a list [obst(.,.,.,.)] . The information is entered later with another routine (ADOBST.PRO).

VI.3.6 Relation "Corridor"

This relation keeps information on corridors of a floor in a building.

TABLE XII
CORRIDORS_TABLE

CNAME*	FNAME*	BNAME*	TYPE	LENGTH	TFREQ	OBDENS

Note. CNAME: name of the corridor as a string of alphanumeric characters (10 max).

FNAME: name of the floor as a string of alphanumeric characters (10 max).

BNAME: name of the building as a string of alphanumeric characters (10 max).

TYPE: shape of the corridor as a small integer 1 if linear, 2 if circular.

LENGTH: length of the corridor in meters as a real number floating point type.

TFREQ: traffic frequency on the corridor in meters as a real number floating point type.

OBDENS: obstacle density on the corridor (number of fixed obstacles per unit length) as a real number floating point type.

The primary key for this relation is (CNAME, FNAME, BNAME).

VI.3.7 Relation "Corrend"

This relation keeps information on the end points of corridors of a floor in a building.

TABLE XIII
CORRIDOR_ENDS_TABLE

CNAME*	FNAME*	BNAME*	END1- NAME	THETA1	ORIENT1	END2- NAME	THETA2	ORIENT2

Note. CNAME: name of the corridor as a string of alphanumeric characters (10 max).

FNAME: name of the floor as a string of alphanumeric characters (10 max).

BNAME: name of the building as a string of alphanumeric characters (10 max).

END1NAME: name of the reference end of the corridor as a string of alphanumeric characters (10 max).

THETA1: angle of the corridor at that end with reference to the horizontal axis of the floor reference - real number floating point precision.

ORIENT1: cardinal orientation of the end point (N,S,E,W and composite).

END2NAME: name of the other end of the corridor as a string of alphanumeric characters (10 max).

THETA2: angle of the corridor at that end with reference to the horizontal axis of the floor reference - real number floating point precision.

ORIENT2: cardinal orientation of the end point (N,S,E,W and composite).

The primary key is (CNAME,FNAME,BNAME). The data consistency requirement is that END1NAME and END2NAME be distinct and they should belong to the list of objects along the corridor CNAME.

VI.3.8 Relation "Object"

This relation (see Table XIV) keeps information on objects or some specific points along a corridor. The data consistency requires that an object with no successor should be assigned itself as successor on the ordered list of objects on the corresponding side of the corridor. In the database table, this means that the attribute NEXTOB will be assigned the same value as OBNAME. Items along a corridor can be seen as a list. Therefore, the item at the end of the list is assigned itself as successor, by convention in our formulation. The TYPE will help to identify the objects. For instance, an exit door to the outside may be dangerous for the paraplegic user. Also, there is a need to locate exit-to-the-outside doors to go from one building of another building of the campus environment.

TABLE XIV
OBJECTS_TABLE

OB-NAME*	CNAME*	FNAME*	BNAME*	TYPE	SIDE-ON	AT-DIST	CHAR1,2 ,3 , 4	NEXT-OB

Note. OBNAME: name of the object as a string of alphanumeric characters (10 max).

CNAME: name of the corridor as a string of alphanumeric characters (10 max).

FNAME: name of the floor as a string of alphanumeric characters (10 max).

BNAME: name of the building as a string of alphanumeric characters (10 max).

TYPE: small integer 0 to 8; 1 (room door), 2 (corridor door), 3 (elevator), 4 (stairs), 5 (fountain), 6 (fixed obstacle), 7 (exit to outside door), 8 (ramp), 0 (other).

SIDEON: side of corridor on; small integer from 1 to 4 ; 1 (left) 2 (right) 3 (end-point 1 of the corridor) 4 (end-point 2 of the corridor).

ATDIST: distance to the reference end of the corridor; real number floating point precision; ATDIST = 0 if the point is an end-point of a corridor.

CHAR1: 1 (automatic), 2 (manual), 0 (other).

CHAR2: opening mode for a door; 1 (open in), 2 (open out), 3 (open in and out), 4 (slide), 0(other).

CHAR3: 1 (dangerous), 2 (non dangerous).

CHAR4: 1 (if observation point) 0 (otherwise).

NEXTOB: next object on right as a string of alphanumeric characters (10 max).

The object attributes CHAR1,CHAR2,CHAR3,CHAR4 reveal important information on the object. For instance, if a corridor door is manual, there is a need to issue a help signal from the wheelchair so that a person nearby can help to open the door. Another case is the stairs down. They are extremely dangerous for the wheelchair user who doesn't have control of his/her motions. S. Rao and R. Kuc in [6] use a smart ultrasonic sensor for the INCH prototype to detect drop-offs. In the approach used in this project, the intelligent system knows the characteristics of the objects and when a dangerous object like stairs down is recognized and located, the system will send a signal to the *Navigator* to take some action. For instance, the message can be to stop and change the path. The type for an access door on a corridor seems to be contradictory. However, it takes into account in the knowledge-base that an access door on a corridor as well as stairs are considered as obstacles.

To end this section there is a remark that a redundancy of information about floor name and building name has been purposely carried out. The reason for this redundancy is that it allows the user programmer to enter two floors with same names, but belonging to different buildings. It also allows two corridors of different floors to have the same name. This problem may have been eased by the primary key feature of the high level relational database system used, but it would create problems in front for the knowledge base syntax and operations. Additionally, in a building floors are often referenced as "first floor", "second floor" or by numbers. With this formulation, it is possible to distinguish the two records in the floors_table:

```
(1,pcat,2,y) /* floor 1 of building pcat having floor 2 above and an elevator
            between */
```

```
(1,lhall,2,y) /* floor 1 of building lhall having floor 2 above and an elevator
            between */
```

This explains the use of a composite primary key for most of the base relations instead of a simple primary key.

VI.3.9 Image Data

Some locations of observation exist on the floor. These locations help the robot to locate itself in the building. At the learning stage, the picture of each of these locations is taken and processed through the low-to-medium image processing [3] and stored in memory. The concern in this section is to indicate how these pictures are named in the database. An observation location can be either an end point of a corridor or a room door seen from the corridor or other specific locations such as public (sitting) places in the building. Therefore, the name of each observation location should figure in the list of items qualified as "objects" in the database or as corridor end.

The names of objects are 7 characters (alphanumeric) long at most. The rule used to find the name of the image file representing a location is the following.

If the name of the location is "xxxxxxx", then the name of the template image file is "xxxxxxx.IM".

Consider for example, a location named PCAT28 on the map. Then its template image file, if stored, will be named PCAT28.IM.

While the wheelchair navigates, an image of the environment is captured by the TV camera. That image is processed and stored in memory. The recognition module matches this image with the template images in order to recognize the location of the robot. The name of the current image will be "CURRENT.IM".

VI.4 THE DYNAMIC KNOWLEDGE BASE

The goal of the dynamic knowledge base is to generate more complex information from basic knowledge that is provided by static data or basic facts. The definition of the concepts fact and rule, have been made in Chapter IV. The basic facts for the knowledge base are directly obtained by the information from the static database tables. The basic facts are listed in the next paragraph. The basic knowledge-base facts are the following:

building(Bname,Mname,Nfloors,Pblc,Accfac),

bridge(Bname,Mname,Bname1,Exit1,Bname2,Exit2,Length,Tfreq,Obdens),

floor(Fname,Bname,Fabove,Elvt),

room(Rname,Fname,Bname,Entdoor,Exitdoor,Type,Length,Width,Obstdispo),

corridor(Cname,Fname,Bname,Type,Length,Tfreq,Obdens),

corrend(Cname,Fname,Bname,End1name,Theta1,Orient1,End2name,Theta2,Orient2),

object(Obname,Fname,Bname,Type,Sideon,Atdist,Char1,Char2,Char3,Char4,Nextob).

The variables appearing in each basic fact have a meaning (see base relation tables in section VI.3). Higher-order facts can be deduced from the basic facts. Some rules can also be established to create relations among basic facts.

Example 1

This rule will establish a relationship between two objects because they are on the same side of the same corridor.

(1) On_same_side(O1,O2):-

object(O1,Cname,Fname,Bname,_,Side,_,_,_,_),

object(O2,Cname,Fname,Bname,_,Side,_,_,_,_).

This means that objects O1 and O2 share the same information about the corridor they are on: floor, building and side. The Don't Cares (dashes) mean that the corresponding information is not necessary for the rule. This rule means that objects O1 and O2 are on the same side if they belong to the same corridor Cname of a floor Fname in a building Bname and they are both on the side Side of the corridor, regardless of their type, their characteristics (Char1,...Char4) and the object next to them.

Example 2

This rule determines if a floor belongs to a building.

(2) Is_a_floor_ofBuild(Fname,Bname):-

floor(Fname,Bname,_,_).

The facts in the dynamic knowledge base can be updated if necessary. For instance the in-room information is entered after the map has been entered. An example of application is given in the APPENDIX A.

Example 3: application (see APPENDIX A)

From the example in APPENDIX A, the building "PCAT" has two floors named

"basement" and "first floor". The "objects" "PCAT104" and "PCAT105" represent the doors of the rooms bearing the same name. Those two objects are on the left side of the corridor "C102" leading to "PCAT 102" (EE office).

The basic facts are:

object("PCAT104", "C102", "first floor", "PCAT",1,1,2m,2,1,2, 1, "PCAT105").

object("PCAT105", "C102", "first floor", "PCAT",1,1,4m,2,1,2, 1, "I6").

floor("basement", "PCAT", "first floor", "y").

floor("first floor", "PCAT", "first floor", "y").

Rule (1) of example 1 is satisfied with the couple of values (PCAT104, PCAT105). The objects PCAT104 and PCAT105 are both on the left side of the same corridor C102 of the first floor. Therefore the fact `On_same_side(PCAT104,PCAT105)` is true and can be used as a deduced knowledge about the map. Similarly rule (2) is satisfied with the couple of values (basement, firstfloor). Both floors are in the same building PCAT. Rule (1) may be enforced by adding the variable representing the corridor in the rule. It becomes `On_same_side(O1,O2,C)`, which will be read, "objects O1 and O2 are on the same side of corridor C." The same thing applies to rule (2) depending on the level of precision that is intended.

VI.5 THE MANAGEMENT ROUTINES

The PSUBOT database system, like any other database application system, has management routines. The primary goal for management routines is to manage data(input, output, retrieve, and update). The management routines of the PSUBOT database are grouped into three categories (see Figure 19): the MAP MANAGER, the DRIVER routines (that insure interconnection and communication of the PSUBOT database and the other modules of the system), the INTERFACE routines, the IMAGE MATCHER and the LOCALIZER. The map manager routines with the extension .SC are written in

PAL(BORLAND PARADOX3 database management application language)[37, 38]. We had two choices: either to design a database management system from scratch for our application, or to use an existing DBMS on the market. We have chosen the second alternative for two main reasons. First, building a relational database management system is not the main objective in this project. Secondly, it doesn't make much sense to design a new database management system when there are already good performing ones in the market. The criteria was to save time, energy and use efficient tools.

VI.5.1 The MAP MANAGER

The map manager comprises a procedure library and seven modules. The function of each module will be succinctly described.

LIBRARY.SC. This script is used to create the procedure library file MAP.LIB. The procedure library is composed of routines that are common to different modules of the map manager. The procedure library is used in PAL to insure best memory management (makes the procedure swapping less costly in terms of time and allows more procedures to be used in the same script).

TUTORIAL.SC. This script provides the user of the Map Manager with general help for the execution of the routines. This script uses a text file TUTORIAL.HLP to display help information.

MAP.SC. This script is the main menu script of the Map Manager. All the other modules are called in a recursive loop by this script. At the beginning of the execution of this script, the procedure library is created and compiled into RAM. When the user quits the application, the tables are converted into ASCII for future needs to build the knowledge base. The algorithm of the map manager's main program MAP.SC can be abbreviated by the following pseudocode:

ALGORITHM #2

BEGIN:

WHILE (NOT EXIT; GET MENU CHOICE)

LOOP:CASE (1) : EXECUTE PROCEDURE TUTORIAL.SCCASE (2) : EXECUTE PROCEDURE ENTER.SCCASE (3) : EXECUTE PROCEDURE READ.SCCASE (4) : EXECUTE PROCEDURE QUERY.SCCASE (5) : EXECUTE PROCEDURE UPDATE.SC

CASE (6) : EXIT

ENDLOOP

ENDWHILE

END

ENTER.SC. This module is used to enter the data of a new map. It creates new tables for the new map. The map table is updated. The tables are saved progressively and automatically converted into ASCII data format when the user quits the map manager application. When the user programmer quits the map manager application, the table files are converted from PARADOX3 formats to text formats for further use in the knowledge base.

READ.SC. The script helps the user programmer to read the data of a previously entered map. This script also allows the user to print a report on any of the seven relations of the Map Manager relational database. This routine allows the user to globally read information on items of the map hierarchy (buildings, bridges, floors, corridors, corridor ends, rooms, objects), one table at a time.

QUERY.SC. This module allows the user to ask simple queries to the relational database. The queries concern the main levels of the map hierarchy: buildings, floors, corridors. Queries are, for instance: information on a building, a floor, a corridor, list of items of a corridor, a floor or a building. The types of queries have been dictated by the common questions that may be asked about a building: number of floors, list of floors, list of rooms and corridors of a floor, information on a corridor, printout all information on a given floor or a given corridor. The user of the Map Manager can check his/her data entry by running a query. For example, it may be necessary to make sure that all the information on a given floor has been entered. Therefore, a query can be run to list all the items of the floor. The program will list all the items (rooms, corridors, objects) that are related to the floor.

Example 4

(i) List all the floors of a building Build_name

The SQL (structure query language) equivalent of this query can be stated as follows

```
SELECT * (get all the record of the table meeting the criteria)
```

```
FROM floors_table;
```

```
WHERE BNAME = Build_name ;
```

(ii) Information on items of a floor Floor_name of a building Build_name

- corridors of the floor:

```
SELECT * (get all the records of the table meeting the criteria)
```

```
FROM corridors_table; /* corridor data(length, traffic frequency etc.) */
```

```
WHERE BNAME = Build_name
```

```
AND FNAME = Floor_name;
```

```
SELECT * (get all the records of the table meeting the criteria)
```

```
FROM corridor_ends_table; /* information on corridor end-points */
```

```
WHERE BNAME = Build_name
AND FNAME = Floor_name;
```

- rooms of the floor:

```
SELECT * (get all the record of the table meeting the criteria)
FROM rooms_table; /*data on rooms */
WHERE BNAME = Build_name
AND FNAME = Floor_name;
```

The meaning of variables used in the records mentioned in the queries have been defined for each table. The reader is invited to refer to the tables representing the relations involved.

UPDATE.SC. The script UPDATE.SC is used to update (insert, delete) information on a previously entered map. This module also uses the same type of queries as the module READ.SC. For instance, if a request is made to remove information concerning a given floor of a building, information concerning rooms and corridors of that floor should be deleted. Similarly, deleting information on a corridor requires the deletion of information on every object located on that corridor.

Example 5

Suppose the query 'Remove from the database all all information on corridor "C1" of floor "F11" of the building "PCAT" . An Embedded SQL¹ equivalent to solve this query is the following:

```
/* delete information on the parameters of the corridor */
EXEC DELETE
```

¹ SQL is both an interactive query language and a database programming language. As such the term "Embedded SQL" is used to designate a version of SQL that can be used as an application language. This language is able to execute the DBMS commands inside an application program as the DBMS itself would do when statements are issued at a terminal.

```

FROM corridors_table;
WHERE CNAME = "C1"
AND FNAME = "F11" AND BNAME = "PCAT";
    /* delete information on the end-points of the corridor */
EXEC DELETE
FROM corridor_ends_table;
WHERE CNAME = "C1"
AND FNAME = "F11" AND BNAME = "PCAT";
    /* delete information on all the objects of the corridor */
EXEC DELETE
FROM objects_table;
WHERE CNAME = "C1"
AND FNAME = "F11" AND BNAME = "PCAT";

```

ADOBST.PRO. This routine is written in BORLAND Turbo Prolog language. It allows the user of the Map Manager to enter information about the inside of each room. The information is concerned mainly with the disposition of obstacles (objects) in the room. It has been mentioned in Chapter III that an obstacle in a room is represented as a forbidden rectangle. Globally, all the obstacles identified in a room can be described in a list. The input of this routine is the knowledge base file "xxxxxxx.db", where "xxxxxxx" represents the name of the map site.

QUERY.PRO. This routine allows the user to ask simple queries on the knowledge base. Such queries are, for example, to display the list of corridors, rooms or 'objects' of a floor. The input of this module is the knowledge base file.

ROOM.PRO. This routine allows to get or update information about a single room in a building. This routine has been added specifically to update or read information on the

disposition of obstacles inside a room at the request of the Navigator. The routine reads the knowledge database and retrieves the information on the room or updates the information on the desired room.

Pseudocode of the map manager tasks. The Map Manager "command" file can be abbreviated as follows.

BEGIN:

EXECUTE PROCEDURE LIBRARY.SC /* create procedure library this file eats up memory. It can be created each time the map manager is called and deleted after use.*/

EXECUTE PROCEDURE MAP.SC

END

VI.5.2 The DRIVER

The tasks of this routine are management mechanism for the database and the communication protocols with the other modules of the PSUBOT wheelchair. The tasks of these routines and their algorithms will be defined in section VI.7 of this chapter. The purpose of this program is to drive the whole database. It is the main program that will be calling other routines. This program will be written in TC++. This program receives requests from the Navigator module of the wheelchair.

The algorithm of the DRIVER is formulated as follows:

PSEUDOCODE

Program DBASEDR(.C)

BEGIN:

. CALL *INIT* /* initialize I/O ports */

LOOP:

. POLL port until status has changed

```

.DISABLE port for writing
.IF signal_Compute_global_path is high /* request to compute global path */
    THEN . CALL GPATH /* compute optimum path: program GPATH.PRO */
        . set signal_Global_path_ready
ELSE()
ENDIF

.IF signal_Locate_me is high /* the request is to compute the current position */
    THEN . CALL LOCATE /* Identify the current location : program
LOCATE.PRO*/
        . set Position_found /* the result is in the file LOCATION.DAT */
ELSE()
ENDIF

.IF signal_Room_info_read is high /* provide information on a room. */
    THEN . CALL ROOMINFO /* Retrieve information on the given room:
        program ROOMINFO.PRO*/
        . set Room_info_ready
    .IF signal_Room_info_update /* request to update the information on a room. */
        THEN . CALL ROOMUPD /* Retrieve information on the given room:
            program ROOMUPD.PRO*/
            . set Room_info_updated
    .IF signal_What_do_I_see_is is high
        THEN . CALL WHATISEE /* scene analysis: program WHATISEE.PRO*/
            . set What_You_see_is
ELSE()
ENDIF
.GOTO LOOP

```

END DBASEDR

VI.5.3 The Global Path Finder GPATH.PRO

The global path finder program is responsible for computing the shortest path between two locations in the map. This program uses as main input data files: the knowledge base file, the path file "PATH.DAT". The file PATH.DAT contains the name of the map site and the starting and destination locations. This information is provided by the Navigator. The output of the program is the computed path. The global path is in the file "PATH.LOG". The program GPATH.PRO combines edged graph shortest path techniques with expert system-like intelligence method (Prolog) to find the global path.

VI.5.4 The IMAGE MATCHER : MATCH.C

The role of the image matcher is to match two images of the scene described by straight lines. The image matcher is examined in details in Chapter VII. This routine will be used by the LOCALIZER which is responsible for identifying the current location of the wheelchair. The current image taken with the camera is processed and matched with template images. The template images candidate of the matching, are chosen in a neighborhood in which the wheelchair is most likely to be located. The input of this program are the files containing the current image from vision (CURRENT.IM) and the template image of a location.

VI.5.5 The LOCALIZER

The LOCALIZER is the routine responsible for localization of the wheelchair. Its algorithm and will be the subject of Chapter VIII.

VI.6 THE INTERFACE ROUTINES

The goal of the interface routines is to serve as data interfaces between the database host system used for the map manager and the high level languages used in the other parts of the PSUBOT data base system, namely, the knowledge base support part of the database and the computational intensive part of the database management. The routine to interface PARADOX3, a relational database management system to Turbo Prolog, is INTERPAR.PRO. This routine converts the PARADOX3 records in each of the base tables into Turbo Prolog facts. This interface program is a dedicated one and is used for the special purpose of serving as a data interface between the Turbo Prolog and PARADOX3 applications. Such interfaces are needed [7] because Turbo Prolog works with facts and the relational database with records and tables.

VI.7 INTERCONNECTION WITH THE OTHER MODULES OF THE PSUBOT SYSTEM

The PSUBOT wheelchair system is made up of three main parts: the *Navigator* [2], the *Intelligent Database*, the *Sensory Module* (VISION [3], SONAR and other sensors). The database is the link between the *Navigator* and the sensory module. In this sense there should be established some communication protocols between the database and the *Navigator* on the one hand, and the database and the sensory module on the other. The sensory module can be qualified as "the eye and the skin" of the intelligent wheelchair. Therefore, the communication protocols and the information treatment by the database should be efficient and fast enough so that the database will be not a bottleneck of the system. This is a major problem that we want to avoid in this work because vision is already very computational intensive and also needs to be faster. Of course, the operations could be pipelined, but such an architecture is not implemented on the personal computer used in this project. The *Navigator* sends a message to the database. The database reads

the message and starts servicing the requests of the *Navigator*. In its turn, the database will send some message to the sensory module if there is a need for sensory information for the database to perform the task requested by the *Navigator*. Basically, the database will ask the vision sub-module to process a new image of the scene. It will also request a new reading from the sonar sensor. The database will make a decision based on these two sensory data packets and will send a feedback message to the *Navigator*. This explains why it has been mentioned earlier in this work that the database needs very efficient communication protocols and fast routines. The algorithms of the communication protocols are shown in APPENDIX B. The diagram in Figure 20 gives an idea of the communication protocols between the different parts of the PSUBOT wheelchair system.

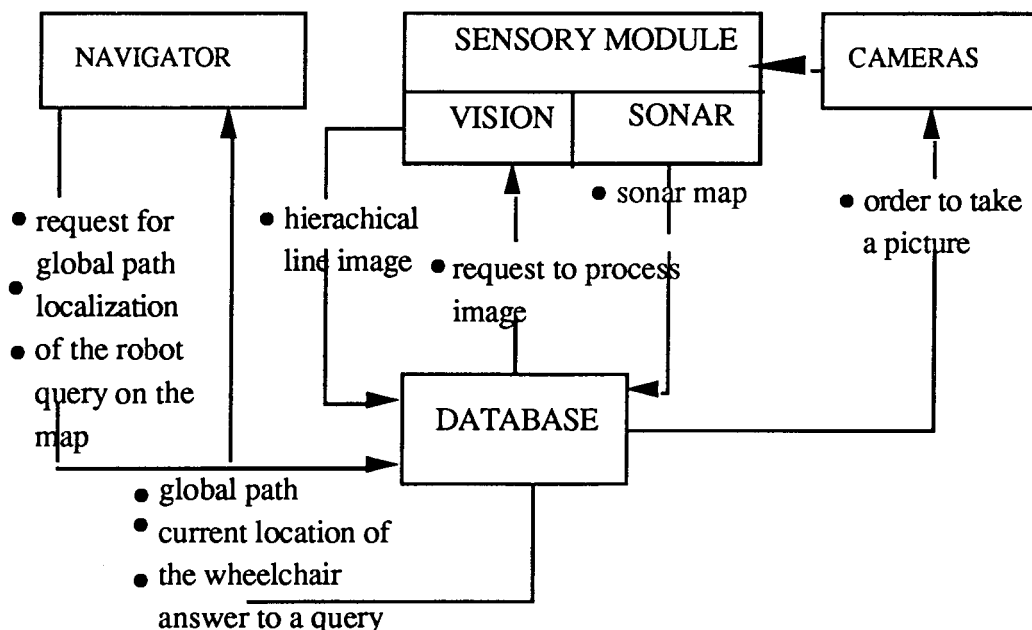


Figure 20. Communication between the database and the other modules of the PSUBOT system.

CHAPTER VII

MATCHING TWO IMAGES

VII.1 DEFINITION OF THE PROBLEM

The primary goal of the research on PSUBOT is to build an autonomous wheelchair that can navigate inside a building or a campus to help a handicapped person to go from one location to another. Like a human being, the PSUBOT must have a mechanism to recognize at least certain details in the building such as specific locations and specific objects such as door frames etc.. Recognition of these details is necessary for this intelligent wheelchair to know, for example, that it has reached a destination location or to detect an obstacle. For human beings, the simplest and common method of recognizing a detail is to match the image of the perceived scene (vision, touch, etc.) with a mental representation of the detail. For example, in our human knowledge of a bird, we know that all birds have two wings. Therefore, if a human sees a living creature with two wings, then the first thought and the start point of recognition is to say: "it may be a bird." Based on this simple example, we can explain why there is a need for template matching for the PSUBOT model. The PSUBOT model as has been mentioned in previous chapters, is a model-directed autonomous wheelchair [18]. This is to say, that the robot has a model of the world (geometric disposition of objects, properties and description of those objects). This model helps the robot to "recognize" given points on the floor. For example, when a request to go to a given room is issued, the robot must move and constantly "look" at the details on its path until it finds a detail or group of details that have been entered in the knowledge of this machine as representing the location where the robot is supposed to go.

The PSUBOT wheelchair now has two means of sensing its environment: vision and sonar. A picture of the environment is taken as an image and the information is processed. Previously, at a learning stage, pictures of the template and locations of interest have been taken and stored in the knowledge base of the machine. The image matching in this case is concerned with matching the current image taken, with the images previously stored at the learning stage. The purpose is to recognize a location.

There are many ways to match two images. However, two kinds of problems of interest for our application are image-image matching and image-model matching. For the first method, the image is globally matched to another image, following certain criteria. In the second method, some specific features or details in the image are extracted and matched to a given model. This method is more oriented towards the pattern/object recognition approach [41, 22], than the general matching approach. To come back to our example of the bird, let us take a system to recognize birds in an image. For those systems, the image would search for wings (details) to issue a decision. In robotics such methods are used [45] for the recognition of industrial parts by machines. These industrial robots will take a picture of the scene, then focus on specific forms that they will compare to models stored in their knowledge base. Then a decision will be made, based on knowledge of whether or not a specific part is present on the table of the robot or machine [10].

For the applications of autonomous mobile robots, both methods have been used: Image-image matching or Image feature-template feature matching. For example, Fennema et al.[18] tried to find a correspondence between the features of the perceived image and models of specific features expected at certain locations. The features used by Fennema in the mobile robot HARVEY are walls, baseboards, moldings, pillars and door ways. They don't include details such as sockets and posters.

Bergevin and Levine [45] are more for object recognition than autonomous navigation. In the system PARVO (*Primal Access Recognition of Visual Objects*), the models are 2D

line drawings of 3D objects representing various details in the scene. Classes of 3-D features are defined. For example, those with symmetry, those with parallelism, those qualified as vertices etc..

Hebert [11]'s system has been implemented on the Martin Marietta vehicle ALV. The system is a map-based navigation system. The image captured from a reflectance ERIM laser range finder is processed, and features such as road edges are extracted from the input image. The intent is to build a map from a sequence of consecutive images. To do so, the relative positions of features observed from different observation points are computed in order to merge them into a consistent map expressed in a single coordinate system. The matching of geometric features, such as road boundaries, is done from one image to another. The best estimate of the current position of the vehicle from dead reckoning (a recorded route obtained from joining segments of the visited path) is made to compute the relative positions.

V. Capellini et. al.[9] use an object oriented approach for object recognition and classification. Their research is targeted towards recognition and tracking of moving objects such as cars, for traffic monitoring application. In the object-oriented approach, data can be modelled by identifying both the belonging to a most general type and the structural relations among the top level type and its sub-parts.

Sanderson and Foster [28] use an attributed image matching technique which implements minimum representation. In robotics applications the interpretation of complex data is fundamental. Therefore, the matching of stored model structures to observed data from sensors is an important approach to the problem. Their minimum representation size criterion helps to reduce the complexity of a model and to facilitate identification of model's structure and parameters. The minimum representation is based on a principle of minimum complexity of a program which explicitly generates observed data. Their method matches noisy-gray level images to attributed graph. Each input image is represented as a set of

features with attributes, and each object model is represented in similar manner for a given view of the object. The image matching requires the identification of the correspondence between features and an associated geometric transformation which aligns the image with the object model. Alignment of the image with the model just means a close correspondence between the two. In other words the matching could be considered similar to the projection (in the geometric sense of it) of the model into the image itself.

Crowley [20] uses a local model, which is described as a list of directed segments. Each line segment has some information (angle, length, state, type). The matching of line segments of the composite local model and the sensor model is done by establishing a one-to-one correspondence between the lines of the model image and the sensor image. This is correspondence matching strategy. The sensor model is matched to the local model in two stages. At the first stage, the best correspondence is found for each line segment in the sensor model by making a call to a function named CORRESPOND. The list of segment lines in the composite local model (template) is searched to find the line segment that corresponds the best to the given line segment in the sensor model. In the second stage the list is scanned to determine the sensor model line that has the best correspondence to each sensor model line in the composite local mode. This second correspondence list is used to update the local composite model. The correspondence function uses a sequence of tests to find the difference in angle between the two lines, and the difference between other attributes of the two lines.

The matching of two images for a system such as PSUBOT involves trade-offs between several concepts. The difficulties and trade-offs that need to be addressed are: the definition of models for features, the extraction of features, the learning stage, the speed of processing, consideration of uncertainty and noise in the images, and the scaling.

- Definition of Models for Features: The models of the features have to be clearly defined. These models should be quantitatively and qualitatively representative of the real object on

the scene. By this, we mean that the number of models should be adequate enough to provide the kind of information expected to be extracted from the reconstruction of the image after processing. If the number of features is very small, then less information can be retrieved. For example, an image where only the floor and wall models are expressed doesn't tell much about the shape of the doors. The quality of the models is important for computations and extraction. A complex model will require more computation resources (time , software) to be carried out. Also, a model that is not clearly defined will cause ambiguity. These considerations are crucial for a case like the PSUBOT system, where the image features are extracted after the whole low-to medium image processing has been carried out. It is difficult and complex to reconstruct a scene from bare line features. However, a good approach may be to identify key features and formulate their description as simple as possible.

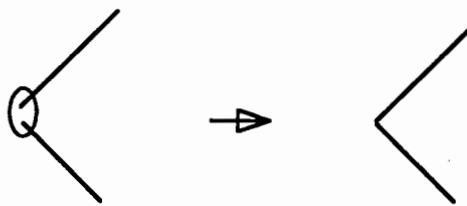
- The Extraction of Features : This step can be taken as a preprocessing stage for the matching of two images. This step should be fast enough not to be a bottleneck. Also, it should take the fuzziness into account in order to counter the effect of noise and loss of information from the early low-to-medium image processing (see Figure 21 and Figure 22). In Figures 21 and 22 it can be noticed that some lines have been broken and some have been removed. Some information about shape (example: curved lines) is missing. In this example, the hierarchical line extraction using the Hough Transform [3] produces lines only at the output, as opposed to more powerful tools such as general Hough Transform [46] and Extended Kalman filtering method [17] which extract curved lines as well.
- The Learning Stage: A learning stage may be necessary to take the images of the templates, process them (extract the features) and finally, store them in memory.
- The Speed of Processing: The extraction of the features and the matching of higher level features should be fast enough so that this step will not be a bottleneck for the localization

process that makes use of matching of images. Also, for the case of PSUBOT, the algorithms must be optimized so as to make the process fast enough on a PC.

- **The Uncertainty and Noise:** The image processing is inherently attached with noise. Each additional image processing from edge extraction to line extraction adds noise to the interpreted image. The Hierarchical Line Extraction Scheme [3] using the Hough Transform method in particular, discards some lines that are too short, what may result in the loss of some information in the image. Therefore, fuzziness and uncertainty [26, 13] should be considered in the representation and the extraction of the higher level line features (see Figure 21). This aspect of the problem is one of the expectations for future improvement that the intelligent database is intended to bring to the PSUBOT system.
- **The Scaling:** The scaling and orientation of the image may vary from the template to the current image. In fact, there is no guarantee that the position of the camera and the angle of capture will be the same. This uncertainty should be taken into account by the use of structural and statistical matching, rather than exact matching. The figures 22, 23 show the outputs of the different steps of line extraction using the Hierarchical Hough Transform [3].

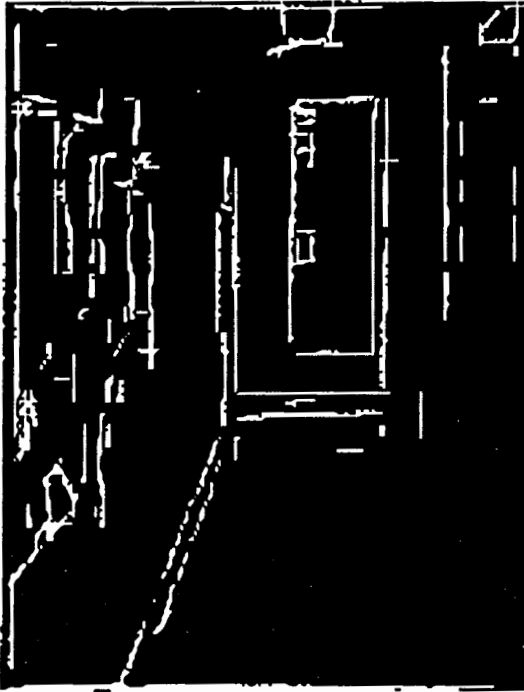


(a) line merging

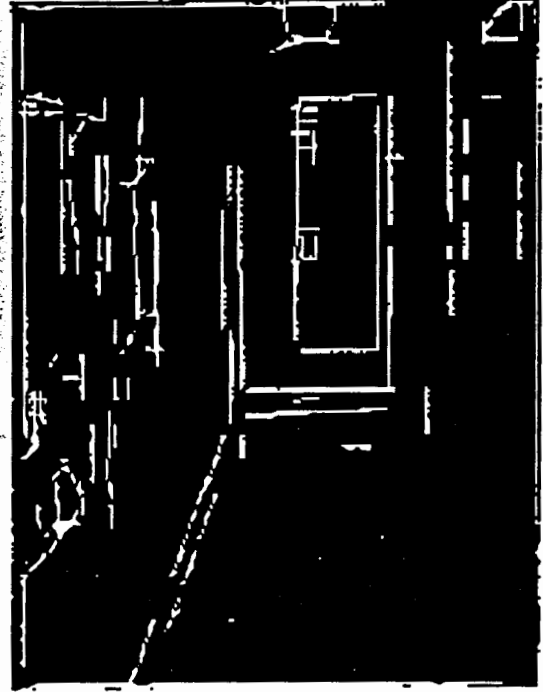


(b) extraction of a corner

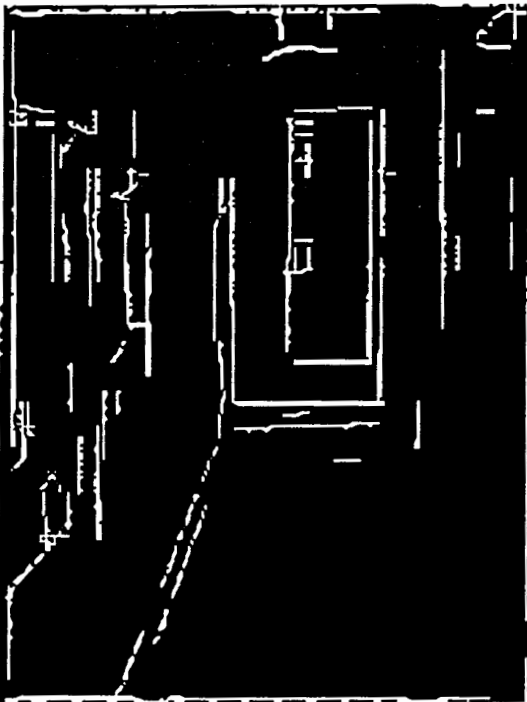
Figure 21. Adding uncertainty to image matching and feature extraction.



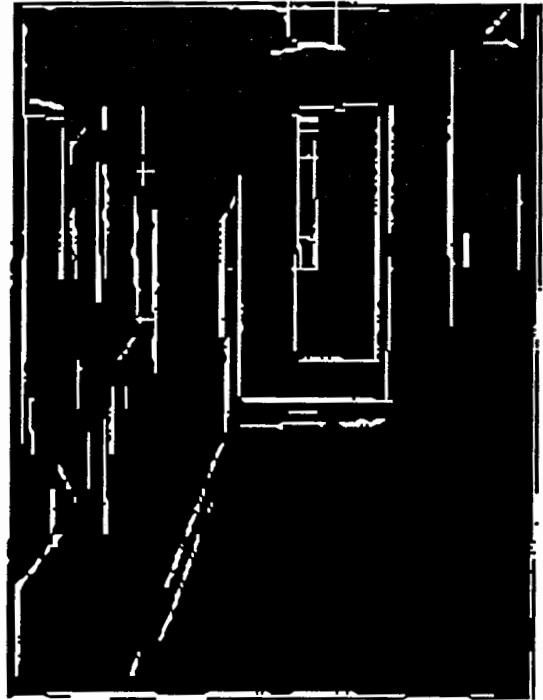
(a) level 1



(b) level 2

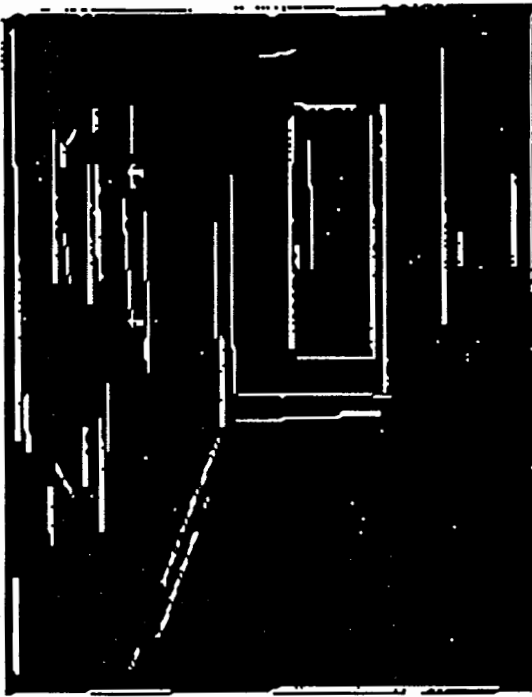


(c) level 3

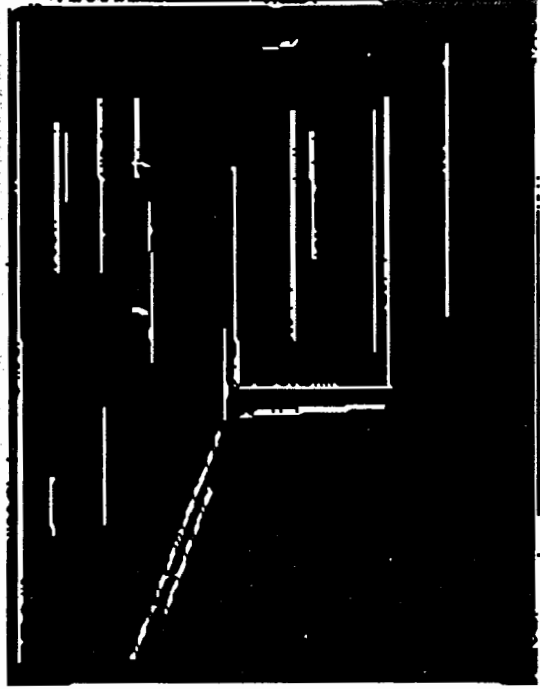


(d) level 4

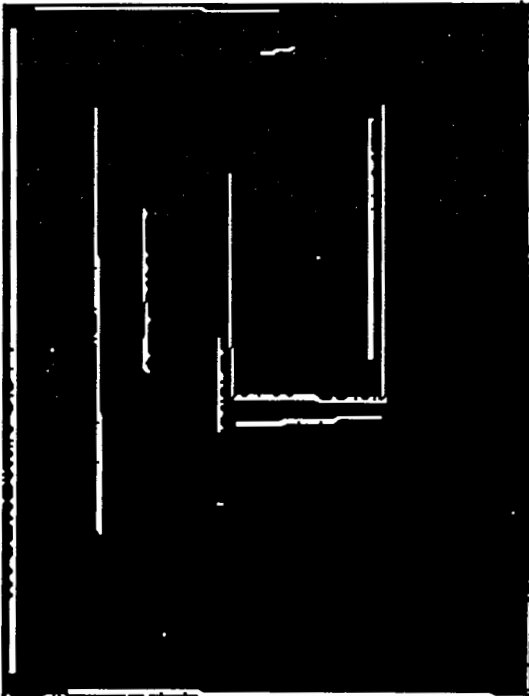
Figure 22. An output image from low-to-medium image processing [3] (level 1 to 4).



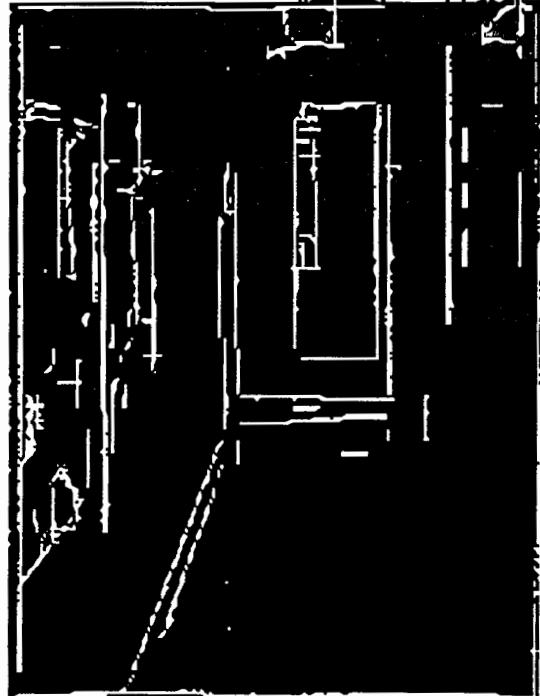
(a) level 5



(b) level 6



(c) level 7



(d) all

Figure 23. An output image from low-to-medium image processing [3] (level 5 to 8).

VII.2 DIFFERENT APPROACHES TO THE MATCHING PROBLEM

In the previous sections, some models have been referenced. This section will concentrate on examining some difficulties inherent to each of these approaches. Hardware consideration is important for the computation aspect of the implementation of autonomous mobile robots such as the PSUBOT. An implementation on a mainframe computer has some advantages, such as speed and memory capacity, but also some drawbacks for the system. A system using a mainframe needs wires to connect the mobile robot to the mainframe computer, therefore, the system cannot navigate freely in the building. A model such as PSUBOT is intended to be self-contained in that it can navigate freely from one location to another on the same floor, on a different floor, or in a different building. The PC is by far slower than a mainframe computer, and the memory capacity and management is not as enhanced as on a mainframe. We will then examine the hardware used for each system. The method of matching two images will be investigated and some conclusions will be drawn.

VII.2.1 Sanderson [28]

Their matching algorithm could be considered statistical because they make decision based on correct matches and failures. However, the matching algorithm that they use is a graph matching algorithm. They match images as set of features by finding a correspondence and transformation between features of the image and the model image. The computer system they used was not mentioned. However, a reference of the use of Monte Carlo techniques for numerical optimization, suggests that, because of the computational demands of those methods, they were using a mainframe computer.

Image matching is difficult to achieve with sufficient generality for many reasons, of which two are obvious. First, the number of models of the features of interest is small, therefore, many details in the image are discarded. Secondly, image interpretation from

higher level line features itself is a complex problem. In the case of PSUBOT, higher level line features could be extracted from a collection of lines. The lines are extracted from the original image after low-level image processing and Hierarchical Hough transform [3].

Image matching is also complex from a computational point of view, because the number of matches to perform grows exponentially with the number of features. Good image matching must be able to add uncertainty to feature extraction and matching, by including missing and spurious data, extra features and noisy attributes. Including uncertainty and robustness has been a major problem in this study, because their implementation does not include the treatment of missing data or extra data. Sanderson uses a graph method to match two images described as sets of features. Each feature has a label and a set of attributes. A correspondence function finds a match between the subset of features of the image and the subset of features of the template model. A match is defined by an injunctive correspondence and a transformation.

A solution to the problem seems to add intelligence methods to the matching process. For instance, features of the same kind can be matched primarily and the matching can be improved later. The problem of extra features in the matching could be solved by attaching a predictability weight to some features. This weight is a number assigned to features that are the most likely to be encountered in the scene. For example, if the robot is moving in the hallway of a public building, most probable features expected are the wall-floor intersection lines, humans, door and window frames. At the time of the matching images, features with a weight less than a certain threshold can be ignored.

Parallel implementation has been suggested in order make the matching faster. Extensions of this approach include solving matching problems with three dimensions, problems with multi-sensor data fusion, and problems with moving objects.

VII.2.2 Hebert [11]

Hebert describes images as sets of features. Consider two images I_1 and I_2 described as sets of features F_i^1 and F_j^2 . Matching the two images is to find a transformation T and a set of pairs $C_k = (F_{ik}^1, F_{jk}^2)$ such that $F_{jk}^2 = T(F_{ik}^1)$ where $T(F)$ denotes the transformed by T of a feature F . The algorithm is as follows.

First, for each feature F_i^1 a set of candidates that could match this feature is computed given an initial estimate T_0 of the displacement

$$E = \sum d(F_{ik}^1, T(F_{ij}^2))_k. \quad (1)$$

The F_{ij}^2 should lie in a prediction region centered at $T_0(F_i^1)$. The prediction region is defined as a set of features located at a Cartesian distance lower than a given threshold δ . The angular distance of such features should be lower than a threshold value ϵ from $T_0(F_i^2)$. The features in each prediction region are sorted according to some feature distance $d(F_i^1, T_0(F_{ij}^2))$. The feature distance depends on the type of features. When the prediction regions have been built, a search for matches between the two images is carried out. The search proceeds by matching the features F_{ik}^1 to the features F_{ij}^2 that are in their prediction region, starting at the most important feature. The result is a set of possible matchings, each of which is a set of pairs $S = (F_{ik}^1, F_{jk}^2)_k$ between the two sets of features. The transformation T is estimated by minimizing an error function of the form (1). The matching S that realizes the minimum E is reported as the final match between the two images. The system has been implemented using two separate processors (Sun3). As it appears, this method is computationally intensive because a transformation has to be computed which would certainly be a bottleneck for a PC-based system such as PSUBOT.

VII.2.3 Crowley [20]

Crowley uses a correspondence function to match a sensor image described as set of 2-D lines to a local model described with the same type of features. This function sequentially compares the attributes of the two images such as the angular orientation and the length.

The best correspondence is searched for and determined among the candidate lines. The matching of the two images is not an exact match, but rather the best match. There was no mention of the architecture used for this robot system.

A problem that may be at issue is the speed. Each line feature of the sensor image is matched to all the other line features of the template image to determine a best match. However, the function used is a sequential function that tests all the parameters sequentially. Consider a problem with a sensor image composed of m lines and a model image with n features. To find a best match for each line of the sensor image requires n correspondence matches. Therefore the whole process will take $m*n*t$ time to complete. Where t is the time for the correspond function to compute a correspondence between two lines, a structural matching of lines defined as objects with the same kind of features may be faster. For example in Prolog, each line could be defined as a fact with attributes. This approach could eliminate some of the tests and exploit the backtracking mechanism to find solutions. This method uses lines as the only features participating in the matching. Therefore, it doesn't include any object recognition. This is the main future improvement that the PSUBOT database is intended to bring to the PSUBOT system.

VII.2.4 Fennema [18]

The matching is done between the landmark models and the sensor image data. A landmark is an observation point to serve as a guide to the robot for recognizing its path. The emphasis is placed on determining the best of the imperfect matches. The matching is posed in terms of optimization over possible matches. The correspondence problem is combinatorial and generally one landmark line is mapped to many data lines. In order to measure the quality of a given model-to-data correspondence, the best 2-D position of the model with respect to the data must be determined. They call that a spatial fitting problem. Hence a match involves both the model-data correspondence and the associated best-fit

position given that correspondence. There exists three components to the 2-D correspondence. First, the search space is defined. The search space may be viewed as a graph in which the nodes correspond to a particular model-to-data correspondence (a state) and the arcs to state transitions between the correspondences. Secondly, a method for determining the optimal spatial fit for a given 2-D correspondence is used. The spatial fit minimizes the quadratic error of the fit. Thirdly, an objective match measure for evaluating the spatial fit for the given 2-D correspondence is executed.

The system was implemented on a time-shared Vax 11-750. This implementation is computationally intensive. The process could be made faster by making a semantic description of the model of features and using correspondence between the features defined clearly with attributes. About 30% of the time is spent on computing the best matching between the image and the template landmarks. The model matching is a computational complex operation in the system. Even written in C, it is possible for the program to run for hours if the search is not focussed. This handicap could be improved by adding deductive and inference support embedded in the search, rather than applying a purely graph or heuristic search.

After this review, a common ground can be found with regard to three points. First, most of these approaches use correspondence matching of features to determine the best match between the two images. Secondly, most of these systems only match two images, rather than interpreting what is seen in those images as well. Thirdly, most of these methods don't explicitly take consideration of uncertainty and loss of information present in the images.

VII.3 IMAGE MATCHING FOR THE PSUBOT WHEELCHAIR: PROPOSED APPROACH

VII.3.1 Input Image Format

Two dimensional image data can be represented in many ways [47]. The most common data representation for digital images is the Pixel form. In this representation, the amplitude of the digital image is quantized to 256 levels. Each level is denoted by an integer with 255 corresponding to the brightest and 0 to the darkest. Each point in the image space is called a pixel. Other representations of images use medium or higher level features such as edges (after edge extraction), or lines and curves, or derived features. Straight Line features can be extracted after edge detection and medium processing, using, for example, a Hough Transform method [3]. Some versions of the Hough Transform are able to extract curved line features as well [48]. From line features, which make the medium level of representation of an image, higher level features can be extracted. Each high level feature represents an object to see in the scene. For example, a door frame feature can be built from two couples of parallel lines. The input format of images for this stage of processing is a hierarchical description with straight lines [3]. An example of description is given in Example 1.

Example 1: A hierarchical description of an image with two levels.

This is the output format of the image from the line extraction processing [3].

F Pyramid numlevs size maxlength numH numV numN numT

F L_evel levnumber maxlength numH numV numN numT

F HVN ys xs rhos thetas length ywidth xwidth

F C_onversionfactors rhodel thetadel(for rhos,thetas)

C 0.400000 0.003608

P 2 256 256 13 20 9 42

L	5	256	1	1	1	3	
N	26.5	133.5	233	503	211	5.00	211.00
V	111.50	80.00	-309	0	144	256.00	0.00
H	238.00	127.50	-296	512	256	0.00	256.00
L	4	127	1	1	1	3	
N	1.00	47.50	290	502	95	2.00	95.00
V	55.50	148.00	510	64	256	0.00	0.00
H	94.00	47.50	63	512	96	0.00	256.00

The explanation of the terminology is as follows.

F: is an indicator for a comment line

size: size of the image in terms of the bigger dimension

P: is an indicator for the pyramid maxlength: length of the longest line of the pyramid

C: entry which gives the sizes of the numH: number of horizontal lines

quantization interval (rho, theta) numV: number of vertical lines

numlevs: number of levels numN: number of slanting lines (not vertical nor
horizontal)

V N H : respectively vertical, numT: overall total number of lines for the the level
or slanting and horizontal lines the pyramid

VII.3.2 Correspondence Matching Algorithm

Correspondence matching strategy is used. Each feature of the current image is matched to all the other features of the target image and a 'best' correspondence is found. However, with the hierarchical description of the image, the two images are matched level by level (see Figure 24). Each level is described as a set of line features. To each level is assigned a weight. The weight relates to the importance of the level in the matching. Referring to Figure 21 and 22, it can be concluded that information on the image is lost when moving

from lower levels to upper levels. In the hierarchical line extraction approach [3], level $i+1$ is obtained by merging shorter lines from level i and eliminating lines whose length is less than a given threshold value. Therefore, the higher levels are made of longer lines. However, the inconvenience is that many shorter lines are eliminated from the picture, which constitutes a loss of information. We would then conclude that information is more accurate for lower levels than for higher levels of the hierarchy. That is why we have assigned lower weights to higher levels. The weight assignment is practically the following: level i is assigned a weight $0.1*(8-i)$. Level 7 which is made of the longest lines and is the highest level to which the weight 0.1 is assigned. Level 1 is assigned the weight 0.7.

The quantities h_i represent the partial hit ratios of the matching of lines of level i_k of image I1 with lines of image I2.

Formulation. Consider two images, I1 and I2. I1 is the image to match to the template image I2.

$$I1 = \{Level_i\}_{i \in \{1,2,\dots,7\}} \quad Level_i = \{F_{ik}\}_{k \in \{1, 2, \dots, n\}}$$

$$\omega_i = 0.1*(8 - i) \text{ weight assigned to level } i$$

Finding the best match is to find a correspondence function T between I1 and I2 such that the cost function

$$C = (\sum_k \omega_k * c_k) / \sum_k \omega_k$$

is maximized.

c_k is the hit ratio of the best match of level k of image i with levels of I2. The best match is defined in this context as the match with the highest score.

Using a pseudocode, the algorithm of correspondence matching proposed can be formulated as follows.

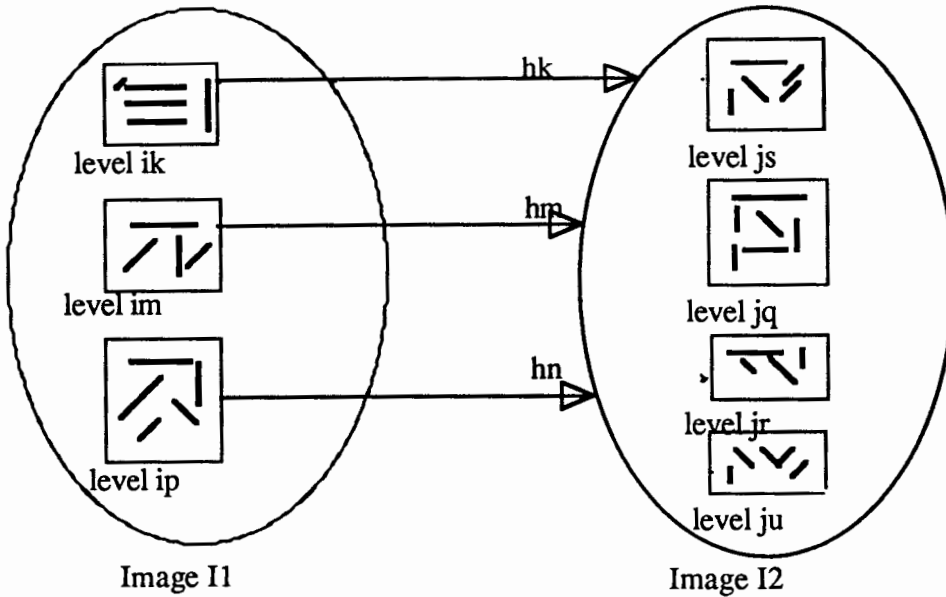


Figure 24. Correspondence matching of two images.

BEGIN:

* Read image I1

* Read Image I2

* For level $i \in$ Set of levels of I1 DO

* - match level i (score) /* find a match for lines of level i */

Endfor

* calculate the cost $C = (\sum_k \omega_k * c_k) / \sum_k \omega_k$ where $c_k = \text{best_score}(k)$ and

$\omega_k = 0.1 * (8 - k)$

* if $C \geq 0.0$ then I2 is the most probable image that matches I1

END

It is clear that the value of C lies between 0 and 1.0, because the coefficients ω_k are less than unity. Their sum in any case is less than 2.8.

The matching takes fuzziness and loss of data into consideration. We will explain the criteria for the matching of two lines in the next paragraph.

The description of a line takes into account the following main parameters (see Figure 25):

$\rho, \theta, x_s, y_s, \text{length}, x_w, y_w.$

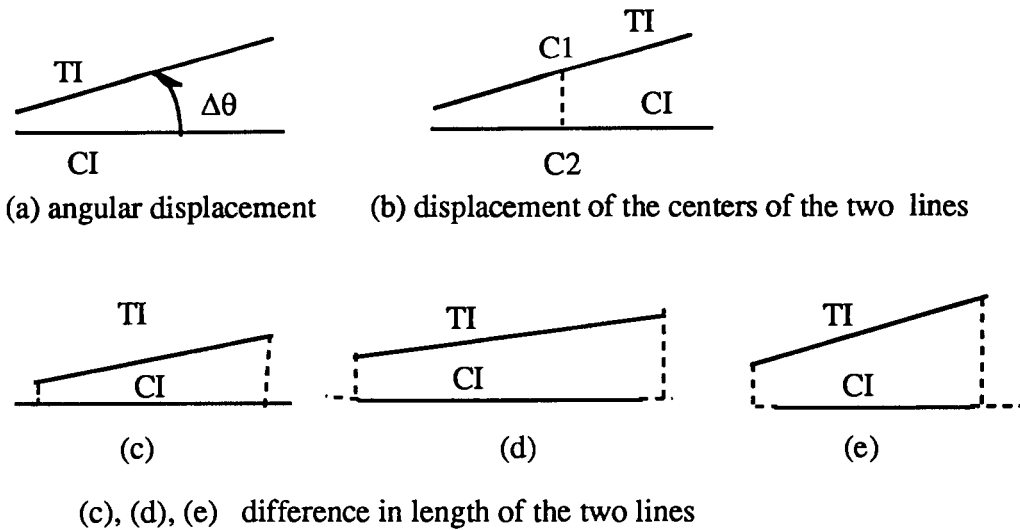
The interpretation of the cases in Figure 25 is the following. Two lines from the two images are considered to represent the same information, if the following conditions are met.

- (1) The difference in angle is less than a certain threshold value $\mathcal{E}_{\Delta\theta}$: case (a) of Figure 25.
- (2) The distance of their centers is less than a certain threshold value \mathcal{E}_d : case (b).
- (3) The difference between their lengths is less than a certain threshold $\mathcal{E}_{\Delta l}$: case (c),(d) or (e).

Choice of the threshold values. The experiment is to find the values of \mathcal{E}_d , $\mathcal{E}_{\Delta l}$ and $\mathcal{E}_{\Delta\theta}$ such that a picture of the same scene taken with an angular displacement of the camera within 10 degrees (reasonable limit) can be recognized, i.e, matches the initial image. It is assumed that the camera axis rotates on a horizontal plane only (see Figure 26). An initial picture of the scene is taken and processed through the image processing. Thereafter other pictures of the same scene are taken by slightly rotating the camera axis around the vertical axis.

Initially the values \mathcal{E}_d and $\mathcal{E}_{\Delta l}$ have been estimated to 2.0 ($\mathcal{E}_d = \mathcal{E}_{\Delta l}$). The value $\mathcal{E}_{\Delta\theta}$ has been chosen as the same angle difference threshold value used in the extraction of line features [3].

In the extraction of line features $\Delta\theta$ and $\Delta\rho$ are the discretization errors introduced by the ρ - θ parametrization. In that formulation, $\Delta\rho$ is taken to be 1.414 and $\Delta\theta$ is $0.5\pi/4L$. L is the sub-image size used in the pyramid. We have chosen to use the sub-image size 4 which experimentally gives better results. We have maximized the error in length (difference of lengths and difference of distances) to the square of $\Delta\rho$, i.e, 2.0. In the matching of two lines, the value of $\mathcal{E}_{\Delta l}$ has been chosen to be the smallest of the lengths of the two lines.



TI: a line of the
template image
CI: a line of the
current image

Figure 25. Criteria for matching of two lines.

VII.3.3 Experiments

The experiments conducted in this section of the thesis involved image matching. Two categories of experiments were conducted: (1) matching two images and (2) matching a given image to a set of candidate images in order to recognize a location inside the room.

Experiment #1. This experiment was conducted inside a room 4mx4m large. Some places in the room were selected as scenes to film. The camera was placed horizontally on a platform at a distance of 1m above the ground and a distance of about 4m from the scene. The picture of the scene at the given place was taken. Other pictures of the same scene were taken by moving the camera axis horizontally around the initial position (see Figure 26). Each of the images was processed and matched to initial image and the similarity measure was calculated. The average time of the computation was recorded as well. Statistics were recorded for each image: number of levels (NL), total number of lines (NT), total number of slanting lines (NN), total number of vertical lines (NV), total number of horizontal lines

(NH), the length of the longest line (Maxlen), the average computation time for the matching with the original image (AvT), the average similarity measure function value (AvSM).

- **Image #1:** The image of the location P201 of the room was taken. Figure 27 and Figure 28 show the steps of the image processing to extract the line features in the image (level 1 to level 7). The other images of location P201 of the room were taken in the clockwise direction (all images labeled with +) and counterclockwise (images labeled with -). The next step consisted of matching each of the images with the original image. For each matching, a similarity measure was calculated. The similarity measure is a mean of estimation of the resemblance or likeliness of the two scenes represented by the images to match. The expression of the similarity ratio was given in section VII.3.2. This quantity is a function of the partial hit ratios obtained for each level of the image to match to the template image. Another quantity measured was the computation time of the matching. The dependency relation between the computation time and the product of the number of lines of both images to match was estimated. The results are shown in Table XV and Table XVI.

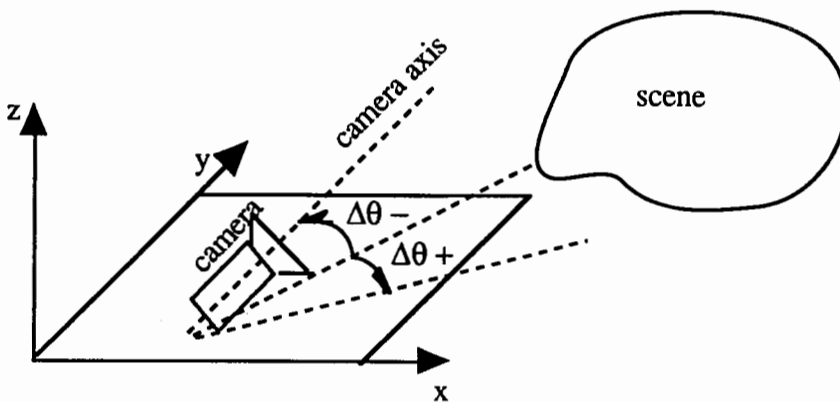
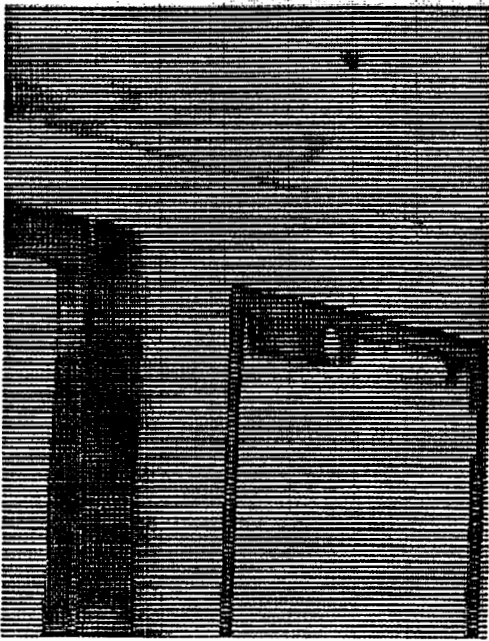
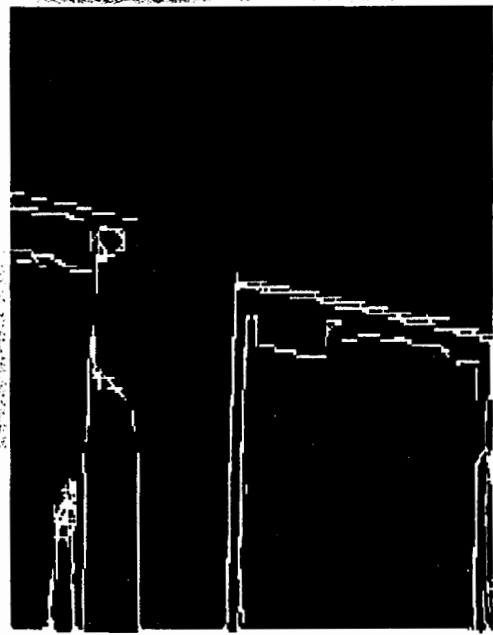


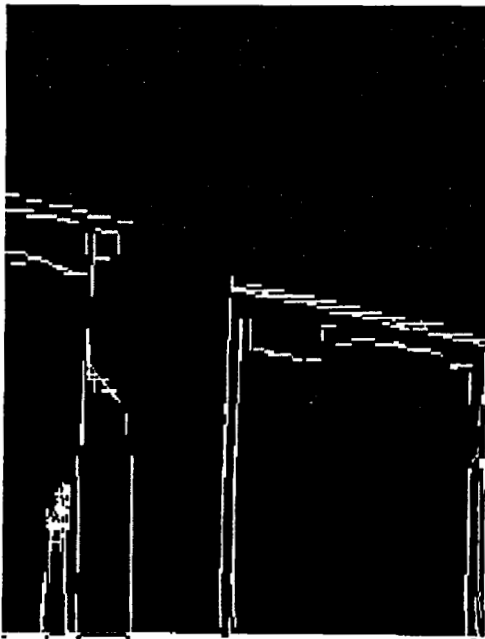
Figure 26. Experiment 1: changing the angle of the camera.



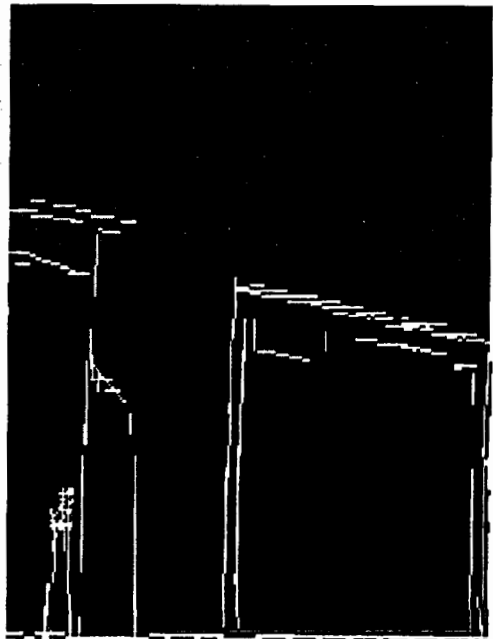
original image
corner P201



level 1 line features

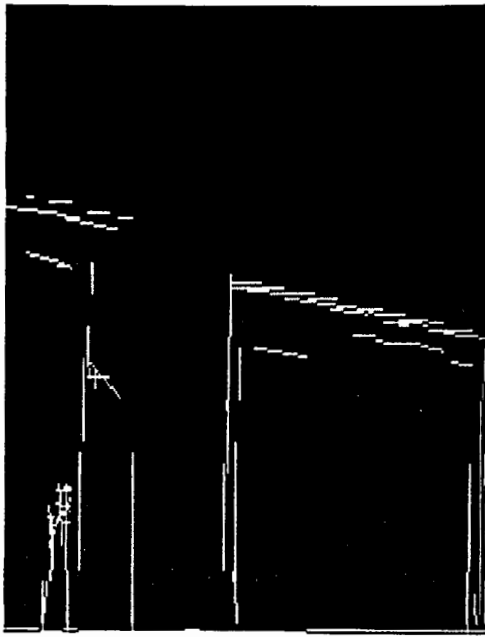


level 2

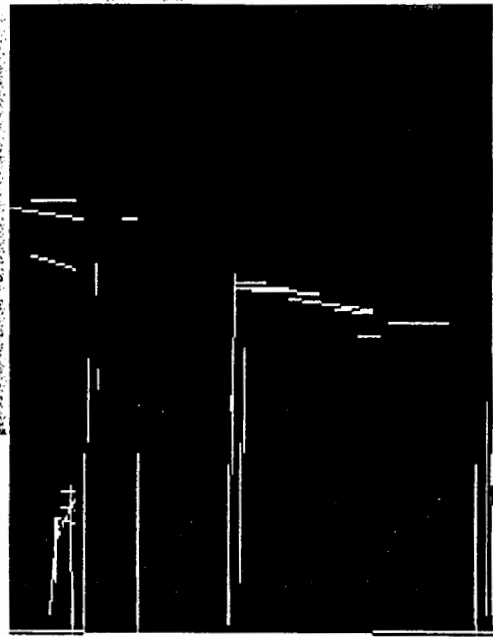


level 3

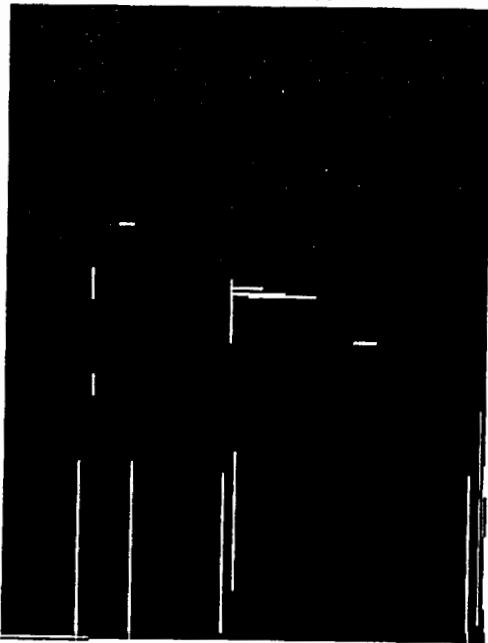
Figure 27. Line extraction process for image P201 (level 1 to 3).



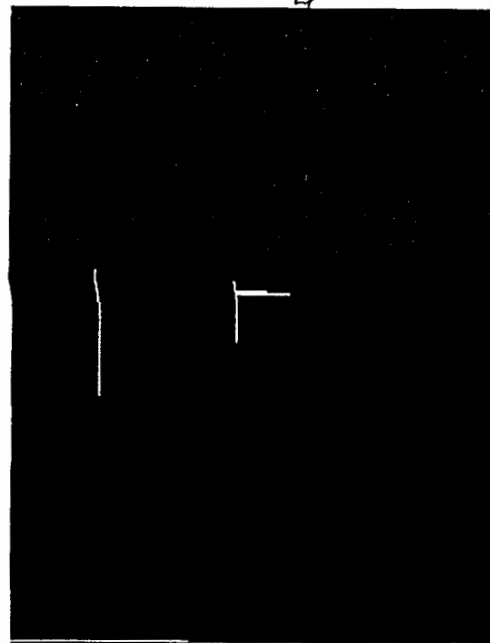
level 4



level 5

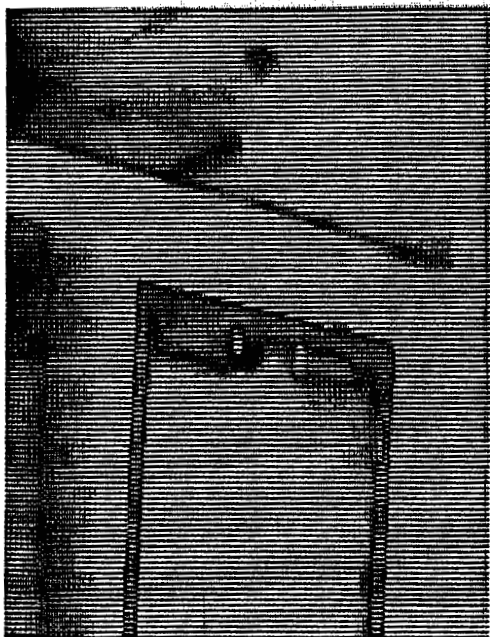


level 6

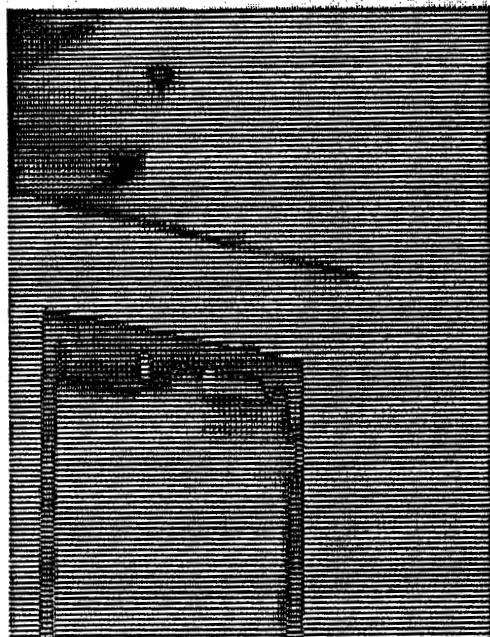


level 7

Figure 28. Line extraction process for image P201 (level 4 to 7).



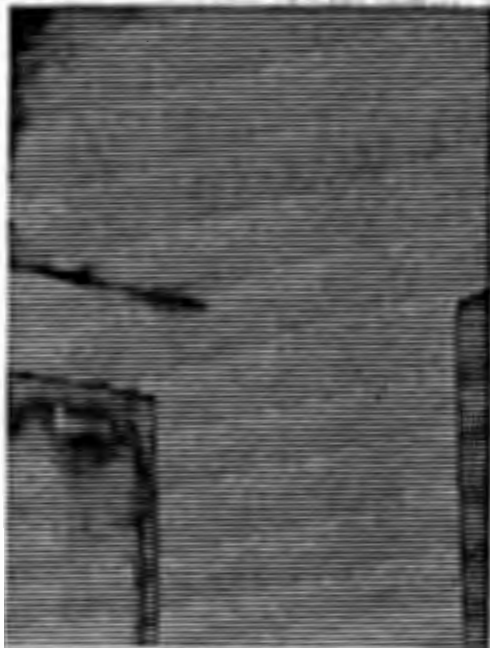
+ 3 degrees



+ 6 degrees



+ 8 degrees

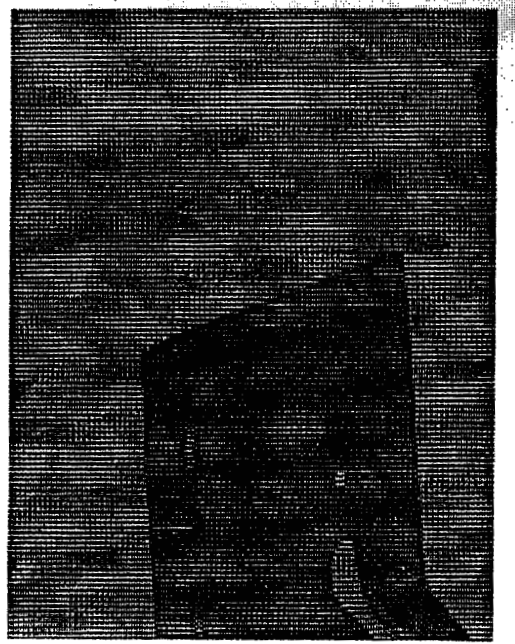


+ 10 degrees

Figure 29. Template Images of location P201
(+3 degrees to +10 degrees).



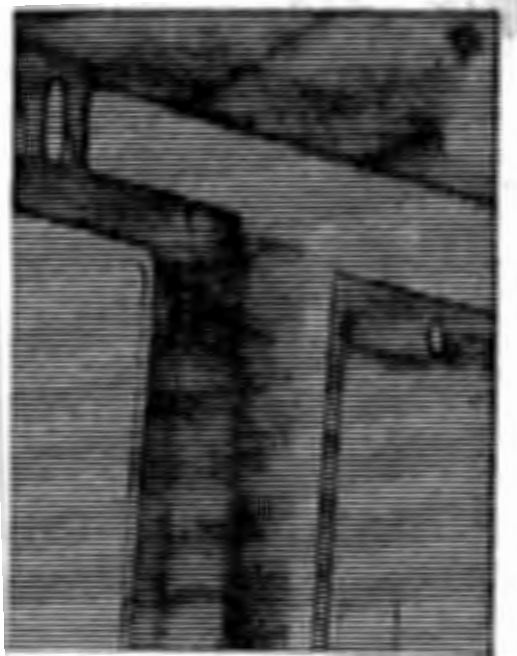
+ 15 degrees



+ 20 degrees

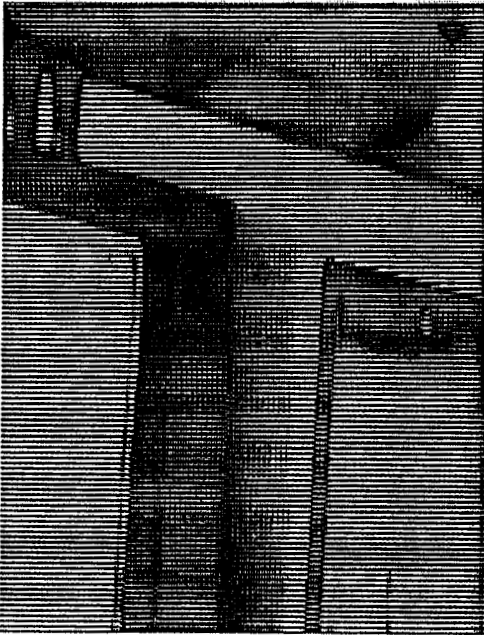


- 3 degrees

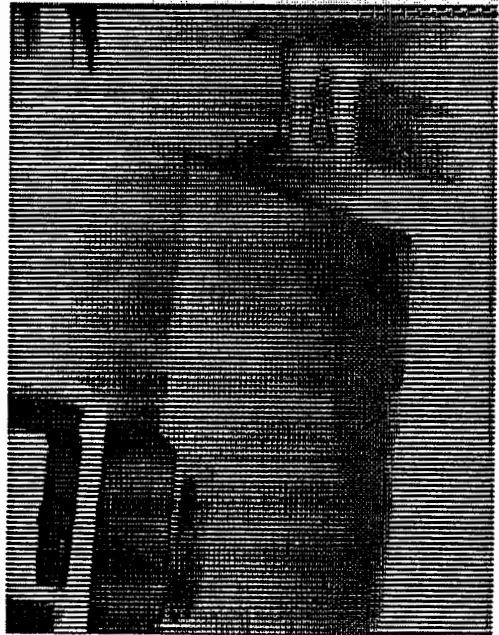


- 6 degrees

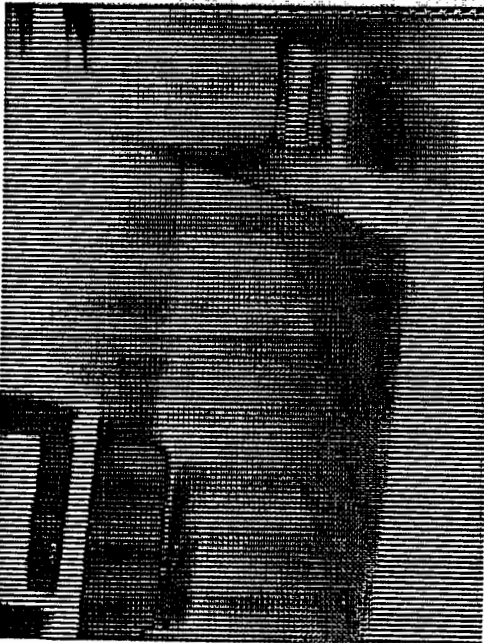
Figure 30. Template Images of location P201
(+15, +20, -3, -6 degrees).



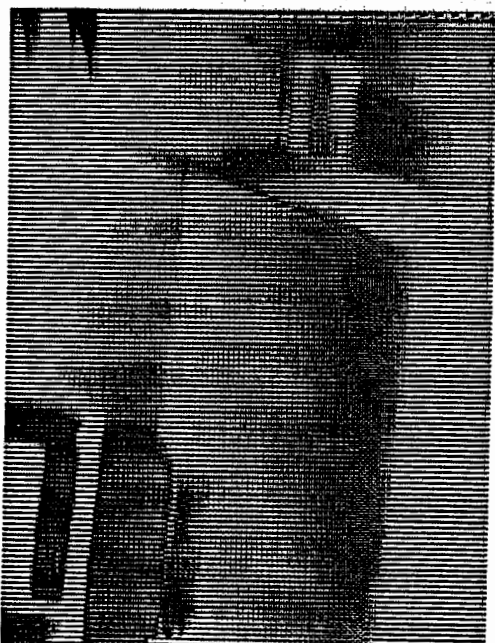
- 8 degrees



- 10 degrees



- 15 degrees



- 20 degrees

Figure 31. Template Images of location P201 (-8 degrees to -20 degrees).

TABLE XV
IMAGE MATCHING RESULTS - IMAGE P201

Scene	NL	NT	NN	NV	NH	Maxlen	AvT (s)	AvSM
P201	7	188	112	34	42	256	--	--
+3 o	7	185	106	35	44	256	0.09	0.49
+6 o	7	107	50	27	30	243	0.06	0.36
+8 o	7	80	39	14	27	255	0.04	0.35
+10 o	7	75	44	6	25	255	0.06	0.06
+15 o	7	75	44	6	25	255	0.06	0.07
+20 o	7	114	56	23	35	256	0.06	0.28
-3 o	7	200	100	46	54	256	0.07	0.51
-6 o	7	200	100	46	54	256	0.08	0.51
-8 o	7	200	100	46	46	256	0.08	0.51
-10 o	7	74	21	18	35	255	0.06	0.06
-15 o	7	195	83	70	42	240	0.18	0.06
-20 o	7	195	83	70	42	240	0.16	0.06

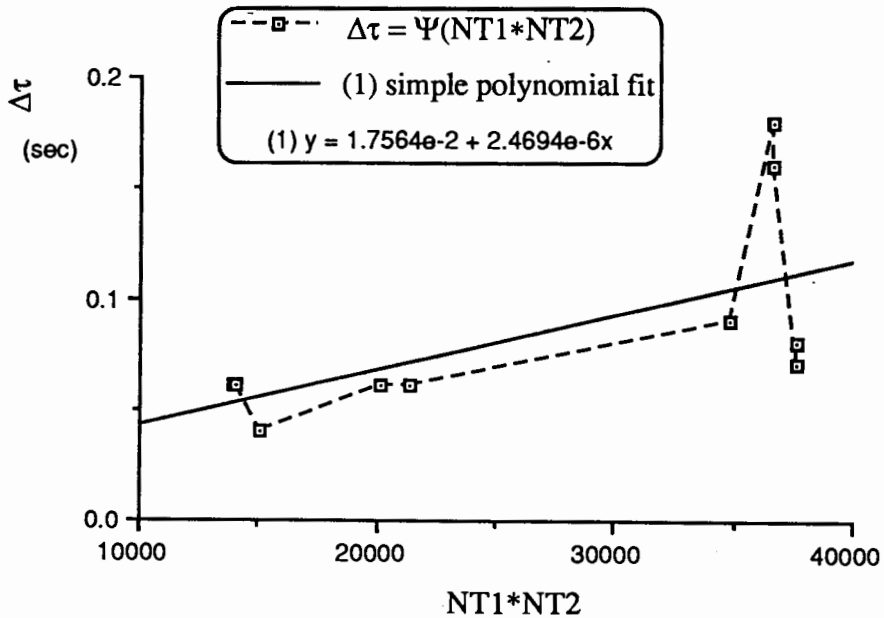


Figure 32. Computation time versus total number of lines, image P201.

- Image #2: Similar experiment as the previous one was conducted with another image representing a location P203 inside the room. Figure 33 shows the location of P203 within the room. The data were collected in Table XVI.

TABLE XVI
IMAGE MATCHING RESULTS - IMAGE P203

Scene	NL	NT	NN	NV	NH	Maxlen	AvT (s)	AvSM
P203	7	151	83	29	39	256	--	--
+3 o	7	171	91	31	49	255	0.06	0.2
+6 o	7	188	99	31	58	255	0.04	0.33
+8 o	7	188	99	31	58	255	0.06	0.33
+10 o	7	188	99	31	58	255	0.06	0.33
+15 o	7	188	99	31	58	255	0.06	0.33
+20 o	7	188	99	31	58	255	0.06	0.33
-3 o	7	188	99	32	58	255	0.07	0.32
-6 o	7	205	110	55	40	256	0.2	0.21
-8 o	7	205	110	55	40	256	0.15	0.21
-10 o	7	185	101	53	31	256	0.12	0.10
-15 o	7	185	101	53	31	256	0.13	0.10
-20 o	7	185	101	53	31	256	0.12	0.10

The data in Table XVI globally show that the similarity measure AvSM decreases as the angle of the capture increases in one direction. The case of positive angles show a constant similarity measure. We think that this is due to the quality of images from capture to line extraction. As will be discussed later, the quality of the image taken with the video camera is very important for the low-to-medium image processing software used; a poor image will virtually produce few or no lines at all after the image processing.

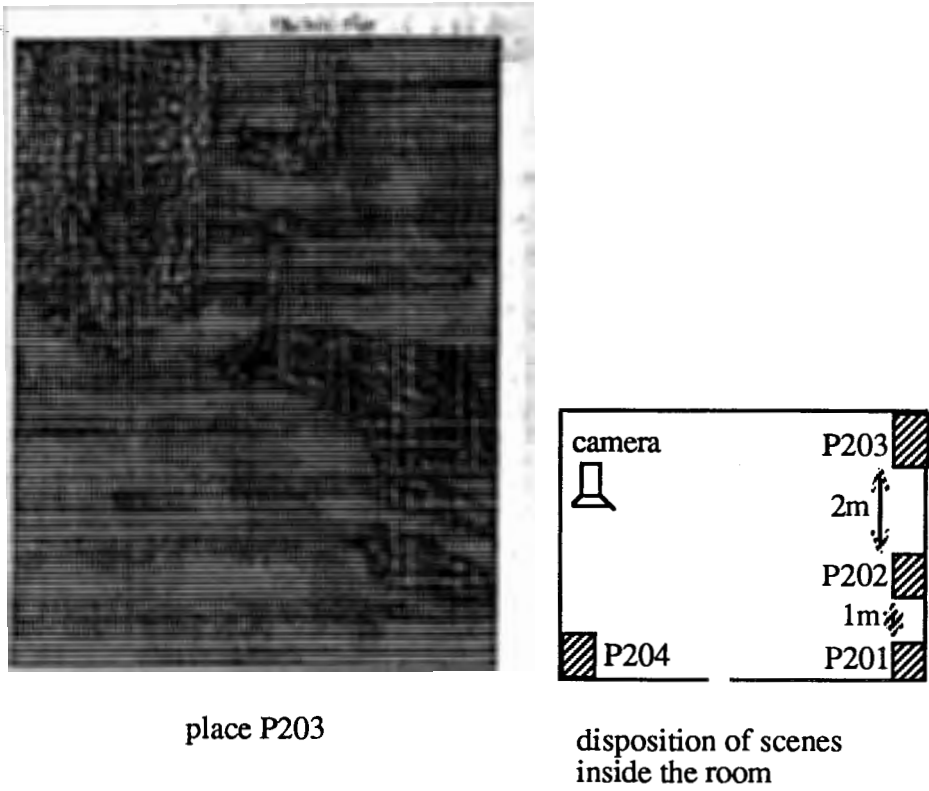


Figure 33. Image P203 and disposition of locations in the room.

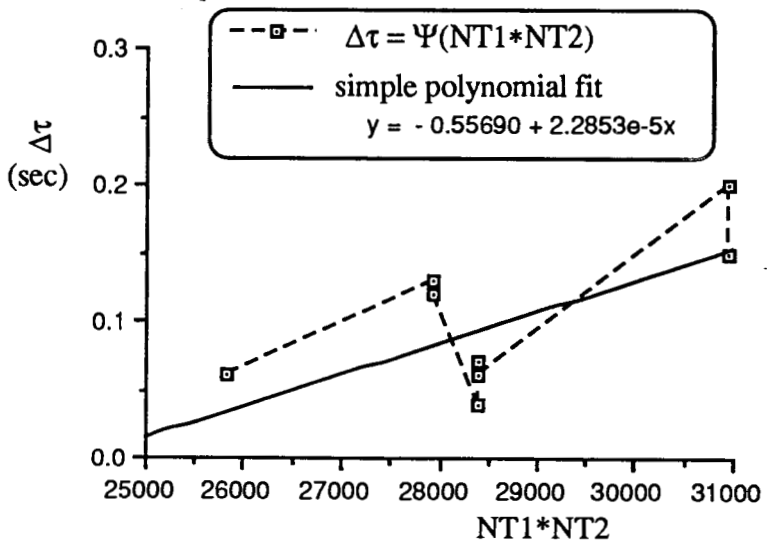


Figure 34. Computation time versus total number of lines, image P203.

- Other images: Images P202 and P204 represent other locations inside the room (see Figure 33)

- Image P204

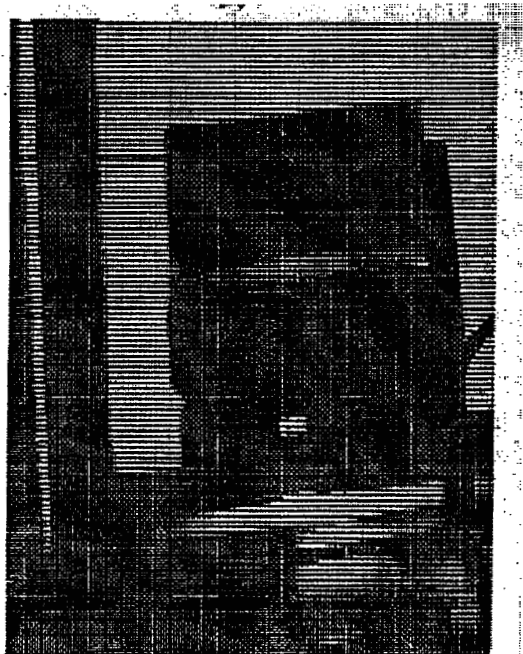
NL = 7, NT = 6, NN = 2, NV = 2, NH = 2, Maxlen = 256

- Image P202

NL = 7, NT = 251, NN = 124, NV = 68, NH = 59, Maxlen = 240



place P202



place P204

Figure 35. Locations P202 and P204.

VII.3.4 Simulation of localization

Simulation of localization is conducted. The goal is to recognize a location among a set of candidate locations in the neighborhood. Figure 33 shows the disposition of locations in the room. The pictures were taken such that the clockwise angle views of P203 are on the side of P202 and the counterclockwise angle views of P201 are on the side of P202. The template position of location P201 was taken as a current image. The program was run in each case to guess the most probable location represented by the current image. The results are shown in Table XVII.

TABLE XVII
RESULTS - SIMULATION OF LOCALIZATION

Camera image	AvSM	AvT(s)	Location recognized
P201	0.71	1.87	P201
P202	0.71	2.08	P202
P203	0.71	1.92	P203
P204	0.71	1.16	P204
P201(+3)	0.32	1.76	P202
P201(+6)	0.52	1.7	P201
P201(+8)	0.71	1.38	P202
P201(+10)	0.43	1.43	P202
P201(+15)	0.32	1.37	P202
P201(+20)	0.57	1.59	P201
P201(-3)	0.40	1.87	P202
P201(-6)	0.40	1.86	P202
P201(-8)	0.60	1.98	P202
P201(-10)	0.60	1.43	P202
P201(-15)	0.63	1.98	P202
P201(-20)	0.63	2.03	P202
P203(+3)	0.40	2.09	P202
P203(+6)	0.71	1.6	P203
P203(+8)	0.71	1.71	P203
P203(+10)	0.71	1.64	P203
P203(+15)	0.71	1.65	P203
P203(+20)	0.71	1.65	P203
P203(-3)	0.71	1.6	P203
P203(-6)	0.32	2.36	P203
P203(-8)	0.32	2.09	P203
P203(-10)	0.0	2.52	could not recognize any
P203(-15)	0.0	2.52	could not recognize any
P203(-20)	0.0	2.47	could not recognize any

TABLE XVII
RESULTS - SIMULATION OF LOCALIZATION
(continued)

Note. P201(+n) designates the picture of the scene P201 taken with an angular displacement of the camera of n degrees clockwise. P201(-n) designates the picture of the scene P201 taken with an angular displacement of the camera of n degrees counterclockwise.

The results show that the algorithm is fast. The upper bound of the computation time is proportional to the product of the number of lines of both images. Views of the same location were recognized successfully and some were not. For example, the location P202 was recognized when P201(-3) was taken as the current image. An explanation can be given that the location P201(-3) is on the side of P202. Therefore details of image P202 were seen on image P201(-3). The major problem is that this method cannot be accurate enough. It takes the best match as criteria. However, there is still a fear that two images could match at the best even though they represent totally different scenes. P203(-20) could not be matched (the similarity measure was 0) because of poor illumination on that image. There were not enough lines extracted after binarization and thinning. This problem is encountered when the scene is poorly illuminated. The problem of scene illumination will be discussed in Chapter IX.

VII.4 A STRUCTURAL APPROACH TO IMAGE MATCHING:
PROPOSED METHODOLOGY

In this section we propose a methodology for image matching that will include recognition. The focus of this thesis was primarily to match two images successfully. The matching concentrated on finding the best correspondence, i.e., the correspondence that gives the highest value of similarity measure.

Such a method of correspondence matching doesn't include the problem of recognition needed by the PSUBOT wheelchair. The PSUBOT needs to know when it has reached the given destination requested by the handicapped user. Therefore, sensor data (vision, sonar

and other) must be analyzed to recognize locations in the building, as well as some details to guide the robot. For example, the robot should be able to recognize the intersection of corridors and the entrance to a room seen from the corridor in order to be able to focus its attention on recognizing the given room. The following is a proposal of such an approach that can combine both matching of images and recognition of some details in the building. This approach has not been implemented in this thesis.

The methodology is simple and can be summarized in three steps as follows. At the first step, the image from low-to-medium image is preprocessed. At this stage, consideration of uncertainty and missing data is taken into account. Two lines that are in continuation (having same angle) with each other and which have two of their ends very close, may represent the same line in the real scene. In step two, higher level image features are extracted from line features. The output image at this step is described as a database of facts. Each fact represents a higher level image feature. The third step deals with the matching of two images (the current image of the environment) and a template image of a location previously processed and stored in memory. The matching can be summarized as finding the best match between the set of features of one image with the set of features of the other image.

Each of the two images reveals meaningful information on the scene because each fact represents a model of object or detail in the real scene. For example, a rectangular long box can represent a window frame or a door frame. A model for a corridor is given so that the knowledge base of the wheelchair (database) can make deductions on the nature of details perceived by the TV camera.

However, vision is not the only sensor intended for the PSUBOT. Sonar is also an important one. Combining vision and sonar is a problem of multi-sensor data fusioning [42]. In the next paragraph a picture of the whole processing is given. Also presented are some higher level features of interest and the rules needed for their extraction.

VII.4.1 Overall Matching Process Proposed

Figure 36 shows the conceptual process of the matching of images.

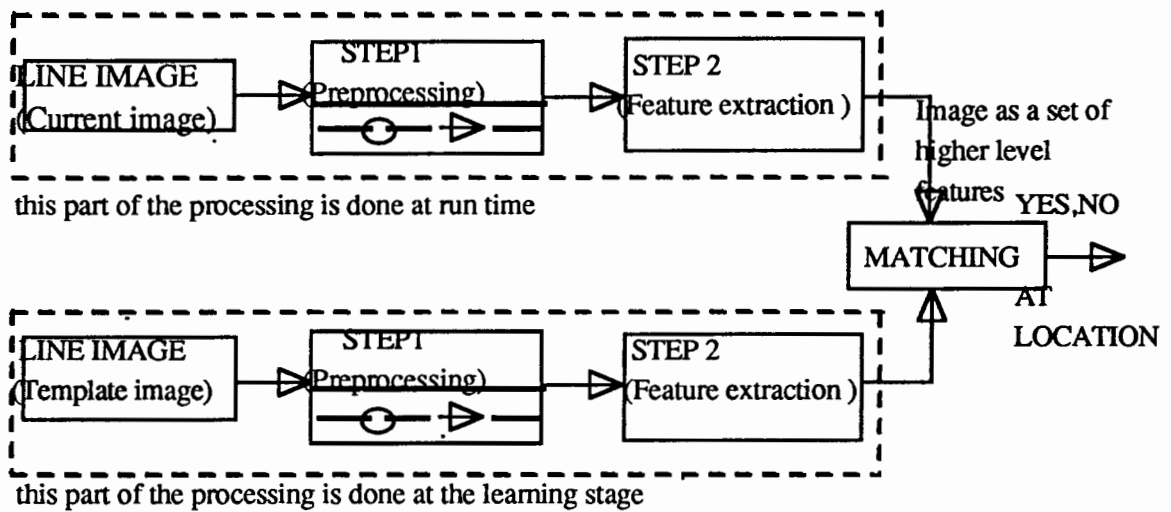


Figure 36. Image matching approach using higher level features.

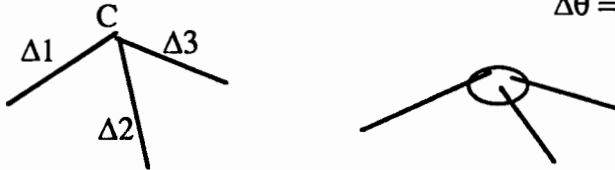
VII.4.2 Models of Higher Level Features

The models of higher level features derived from straight lines are chosen to be representative of key information necessary for the navigation and localization of the wheelchair. The details inside a building are numerous and varied, so only the most interesting ones are chosen for this proposal. The models described can be defined as Prolog facts for later use in the image database. In this selection, the corner is the basic element after the line. The idea we would like to express here is that a rectangle, for example, can be built from four corners that can be connected couple by couple. Similarly, a triangle can be obtained from two connected corners. We will propose a structural definition of a corner. A corner is identified by the lines (the two sides), the center (intersection point of the two lines) and the angle difference between the two lines. The intersection point, if it exists, is the physical intersection of the two straight lines. The rule to extract a corner can be formulated in Prolog as follows.

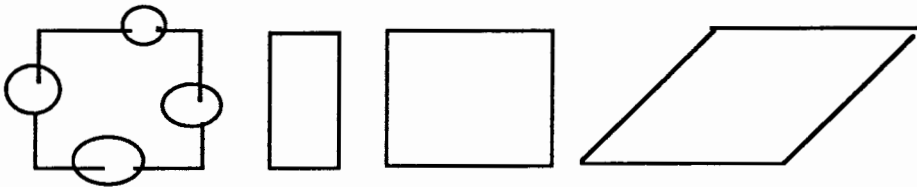


(a) $\text{line}(\rho, \theta, \text{length}, \text{center}(x_c, y_c))$

(b) $\text{corner}(\Delta_1, \Delta_2, C(x, y), \Delta\theta)$
 rectangular corner if
 $\Delta\theta = (2k+1)\pi/2$

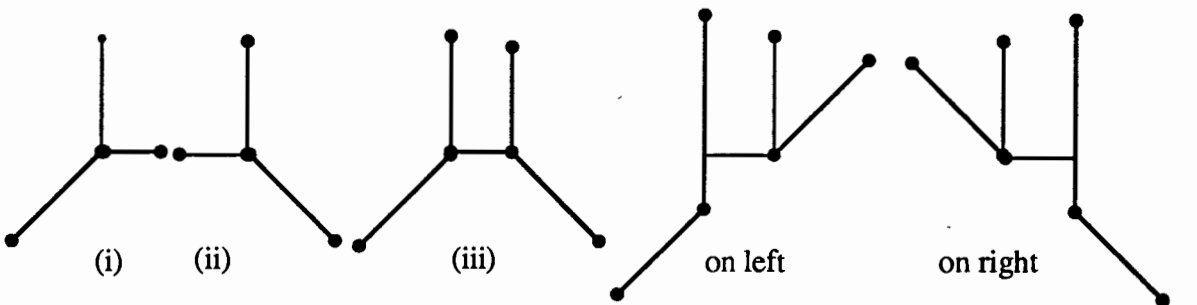


(c) a "ceiling corner" $(\Delta_1, \Delta_2, \Delta_3, C(x, y))$



(d) boxes: reactangle long, wide; square, parallelogram

Figure 37. Models of higher level features:
 corners, rectangles, parallelogram.



(e) a corridor

(f) an entrance (door, corridor intersection)
 seen from the corridor

Figure 38. Models of higher level features,
 corridor.

$\text{corner}(\Delta_1, \Delta_2, C(x, y), \Delta\theta) :- /* \text{ case (1) see Figure 38 */$

$\text{line}(\Delta_1, \theta_1), \text{line}(\Delta_2, \theta_2), \Delta\theta = \theta_2 - \theta_1, \Delta\theta \neq (k\pi),$

$\text{Intersect}(\Delta_1, \Delta_2, C(x, y)), \text{End_of_line}(C(x, y), \Delta_1),$

End_of_line($C(x,y)$, $\Delta 2$).

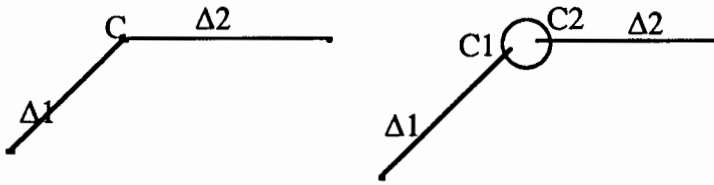


Figure 39. Extraction of a simple corner.

corner($\Delta 1, \Delta 2, C1(x,y), \Delta \theta$):- /* case (2) see Figure 39 */

line($\Delta 1, \theta 1$), line($\Delta 2, \theta 2$), $\Delta \theta = \theta 2 - \theta 1$, $\Delta \theta \neq (k\pi)$,
 not(Intersect($\Delta 1, \Delta 2, C(x,y)$)), End_of_line($C1(x1,y1)$),
 End_of_line($C2(x2,y2)$), $d(C1, C2) \leq \epsilon$.

The predicates Intersect($\Delta 1, \Delta 2, C(x,y)$) mean respectively that $\Delta 1$ and $\Delta 2$ intersect at point $C(x,y)$. The predicate End_of_line($C(x,y), \Delta 1$) mean that point $C(x,y)$ is a physical end of line $\Delta 1$. It is important to recall that we are dealing with finite lines instead of infinite ones. The other higher level facts can be extracted from the 'corner' and 'line' facts with similar rules. The image database will then consist of facts of the kind: line, corner, ceiling corner, square, rectangle, parallelogram, corridor, entrance on corridor.

The approach briefly presented can be used later as an improvement of the database capabilities. However, we recognize that this proposal takes into account vision data only. A more complete and reliable approach will combine sonar data (sonar map of the environment) to build a representation of the scene that can be interpreted.

CHAPTER VIII

LOCALIZATION OF THE WHEELCHAIR

VIII.1 PROBLEM STATEMENT

As mentioned earlier in this thesis, an autonomous mobile robot as an autonomous device should be able to navigate by itself in the environment and make decisions based on the perception of the world around it. An autonomous mobile robot is intended to carry on specific kind of tasks such as carrying materials from one place to another. For the case of the PSUBOT wheelchair, the main objective is to carry a handicapped person from one location to another location in the building. To accomplish this task successfully, the robot needs to be able to locate itself within the building, i.e, to know its current location. Localization of the PSUBOT is important for three reasons. First, the robot needs to follow a global path (keep track of locations visited). Secondly, the robot needs to recognize locations in order to know if it has reached the destination. Thirdly, if the robot has lost track of its route, it should be able to relocate itself by identifying its current location.

Important problems are related to localization. First, localization needs analysis and recognition of locations, i.e, scene analysis. This is where sensor data are read to interpret the world seen around. For the PSUBOT, an image of the scene is taken by a TV camera. The image is processed and matched with template images taken at the learning stage. Template images are images taken at the observation points. The observation points are special locations of the building and they serve as landmarks for the wheelchair. The second problem linked with localization is navigation. If the robot navigates faster than it should, there will not be enough time for the sensor data to be processed and analyzed to

recognize the environment. Thirdly, the accuracy of sensor perception and the matching of two images is crucial for PSUBOT.

Research about localization of autonomous robots is ongoing, because autonomous mobile robots need to make decisions by themselves and sense the world around them. Madarasz et al.[4] locate the wheelchair in the hallway by tracking the wall-floor intersection. When the robot is disoriented, their strategy is for the vehicle to realign itself relative to the wall and then proceed until a recognizable landmark is met. However the robot should be able to know its position in the building at any time. The position can be a relative position, it is not an x-y position for this kind of robot. In Krotkov's [12] approach, similar to Sugihara's [48], the robot's position and orientation is determined by establishing the correspondence between landmark directions and points in the map. The landmark directions are determined from the image of the scene. The image is matched with images of locations (points) taken at the learning stage. These locations are on the map of the floor (environment).

VIII.2 LOCALIZATION STRATEGY FOR PSUBOT

The problem of localization of the PSUBOT wheelchair is not the emphasis of the present work. In this work, we are proposing an algorithm that illustrates the concepts developed in this thesis and particularly matching, which can be improved later for future work. In fact, localization should include recognition of the scene. For example, the robot can track the wall-floor intersection by extracting the wall-floor intersection information from the current image. The current image will be matched to template images in order to locate the robot. Localization of the wheelchair necessitates adjustment of the speed to the knowledge and data processing. Actually, the vision processing is slow, therefore the robot needs to slow down, even stop, while the image processing and localization are being done. Another aspect to be taken into consideration is that locations are not contiguous,

rather they are along the hallway. The robot may miss them because pictures are not taken continuously.

In this section we propose an algorithm for localization of the wheelchair. This algorithm is just for the purpose of simulation. Localization of the wheelchair is done in three situations. Firstly, the Navigator will request the database to locate the robot in the building when the robot is lost. Secondly, during the navigation, an image of the current location is taken. This image is matched with templates of locations on the global path. When a location has been successfully recognized, it is put in the list of traversed locations. The process is repeated until the destination location is reached successfully. The current position of the wheelchair in the map is constantly computed in order to travel the global route. Thirdly, when the request for computing the global path is made by the Navigator, the starting point of the path needs to be known. In case it is not given explicitly, the current position of the wheelchair needs to be computed.

The localization strategy implemented in this work takes vision into account as the only sensor. We propose two algorithms for localization, one for the navigation mode and the other for finding the position in the building when the robot is lost, i.e, doesn't know its current position.

Algorithm to find the current position of the wheelchair

PSEUDOCODE of the program LOCATE.PRO

INPUT:

The most recently visited location (Ls,Fs,Bs) in the DOS file "PATH.HIS." This location is identified by L (location name) F (floor name) B (building name). This is the starting heuristic for the search. If the search fails, it is assumed that the position of the robot is still at this location.

OUTPUT:

The computed position of the wheelchair. The resulting location (Lh,Fh,Bh) is written in the DOS file "LOCATION.DAT."

BEGIN:

- * Read the last visited location (Ls,Fs,Bs)
- * Send signal to the Navigator to stop the wheelchair.
- * Send signal to cameras to take a picture
- * Send signal to Vision to process the image
- * match image to template image of location (Ls,Fs,Bs)
- * IF match successful

THEN return values (Ls,Fs,Bs)

END PROCEDURE POSITION.PRO

* ELSE

- * find corridor C such that the location (Ls,Fs,Bs) is on that corridor

* CALL MATCH_ON_CORRIDOR(C)

* read Success from file TEMP.DAT

* IF Success = 1

THEN read(L,F,B); return(L,F,B)

ELSE

/* a list of corridors sharing at least one end with C*/

* build neighborhood of corridor C

* WHILE X is in the neighborhood of C and match not successful DO

CALL MATCH_ON_CORRIDOR(X)

read Success from file TEMP.DAT

ENDWHILE

* IF Success = 1

THEN read (L,F,B) from TEMP.DAT

```

        write (L,F,B) in file LOCATION.DAT
        EXIT /* END PROCEDURE */

    ELSE

        write (Ls,Fs,Bs) in file LOCATION.DAT

        EXIT /* END PROCEDURE */

END PROCEDURE

SUBROUTINE MATCH_ON_CORRIDOR(X)

/* match the current image to templates of locations on corridor X*/

* WHILE list not covered DO and match not successful

    read a location (L,F,B) on corridor X

    match the current image to the template representing the position (L,F,B)

    IF match successful

        THEN write 1(Success) and (L,F,B) in file TEMP.DAT

        BREAK WHILE LOOP

    ELSE ()

    ENDIF

ENDWHILE

IF match not successful

    THEN write 0(Failure) in file TEMP.DAT

ELSE()

ENDIF

RETURN

```

Algorithm for navigation mode of the wheelchair (Route runner)

The Route Runner plays the same role as the DRIVER for the database in the full

navigation mode. In this mode, the database must continuously keep track of the current position of the wheelchair inside the building. The wheelchair navigates at a regular speed. However, time must be allocated for image processing, therefore the robot must reduce the speed during the processing time. The overall processing will be decided when all the modules of the system will be integrated.

PSEUDOCODE of the subroutine NAVIG

INPUT: global path in file PATH.LOG /* no empty lines */

the starting point of the path (Ls,Fs,Bs) in file PATH.DAT

OUTPUT:

the last visited location in PATH.HIS, continuously updated

the list of locations visited

BEGIN:

* IF not exist DOS file PATH.LOG

THEN EXIT /* path not yet computed */

* read the starting point (Ls,Fs,Bs) from file PATH.DAT

* write the starting point (Ls,Fs,Bs) in file PATH.HIS

* send signal to the Navigator to move the wheelchair ahead

/* picture will be periodically taken in time and in space */

* WHILE destination not reached DO

Identify current corridor X from most recently visited position

/* in fact the program should keep track of the most recently visited

location */

CALL PATH_ON_CORRIDOR(X)

* ENDWHILE

RETURN

The two proposed algorithms need knowledge-based method to build information such as the neighborhood of a corridor, and fast methods for computation. We suggest that future implementation use both a procedural (and fast) language (Turbo C++) with a declarative (knowledge-based) language (Turbo Prolog).

One problem at issue for the Route Runner is to know how periodic a picture of the scene should be taken with the camera. If the pictures are taken with too much time ellapsed between frames, there is a chance that locations will be missed. If pictures are taken continuously, there is also a great risk that Vision will not process them all because the image processing algorithms are slow right now; they require an average processing time of 4 minutes. However, with the advent of faster processors and the use of parallel processing or special image processing/DSP chips, the image processing routines can be made faster, hence eliminate the bottleneck.

CHAPTER IX

TESTING AND EVALUATION OF THE DATABASE SYSTEM

IX.1 JUSTIFICATION OF THE CHOICE OF LANGUAGES

As discussed in previous chapters, the tasks of the intelligent database include keeping a map of the world, finding a global path for the wheelchair, and matching images for the purpose of localization of the device. Each of these tasks has certain requirements for the power and speed of computations and knowledge (data) handling.

As discussed in Chapter III and Chapter VI, the hierarchical nature of the description adopted for a building has led us to choose a relational database management system in order to store the information of the map. The choice was either to build a small relational database system from scratch or to use an existing DBMS. The second solution has been chosen and makes more sense. However, the choice of the DBMS was driven by the requirement that the data formats must be easily interfaced to languages such as Turbo C and Turbo Prolog formats. We have then chosen PARADOX3, a relational database system from BORLAND company. PARADOX3 has an application language PAL which we used to implement the data management routines needed to create, update and retrieve the map data. This DBMS along with Turbo C, Turbo C++ and Turbo Prolog are designed by the same company and interfacing of data with these languages is simple.

The task of computing a global path requires knowledge-based method of inference, backtracking and deduction to retrieve the path. The formulation of the path planning using a traditional optimum path algorithm such as the Dijkstra's shortest path algorithm or the A*, for example, is not well suitable for the map described. The map, described

hierarchically can be retrieved as a simple graph where each building is described floor by floor. However retrieving such a graph would be a burden for the limited memory capacity. Therefore applying knowledge-based method is useful.

We have chosen knowledge-based method to retrieve the path portion by portion. For example, retrieving a path between two points located on the same hallway is a simple problem of knowledge-based retrieval of information. However, retrieving the path between points located on different hallways of the same floor needs both knowledge-based retrieval and shortest path method. Therefore, we have mixed knowledge-based method (declarative) and procedural method in the path planning. Another task requiring knowledge-based method is localization and pattern matching. We have chosen Turbo Prolog as a knowledge-based language for these tasks.

As mentioned in Chapter V, we have used Turbo C++ to implement Dijkstra's algorithm and image matching which both require fast and intensive computations. Turbo Prolog doesn't support intensive numeric computations and certain types of data structures. The implementation and testing of the global path planning and localization were done on a PC-386SX with 640K of base memory and a clock resolution of 20MHz.

IX.2 EVALUATION OF THE DATABASE

As mentioned in the introduction of this thesis, we will conduct the evaluation of the database sub-part by sub-part. Thereafter, the overall performance of the database will be judged. The localization program has not been implemented in this thesis.

IX.2.1 The Relational Database

The relational database has been implemented in PAL (Paradox3 application language). The management routines are user friendly and have been designed such that the user doesn't need any knowledge of Paradox3 to run these applications. The tasks implemented are: enter a new map, update an existing map, read and query a map.

The map manager programs are not used on-line, i.e, when the robot is running. As many other DBMS applications, they run slowly and require quite an amount of memory. After new data have been entered, the interface routine is called which automatically converts the records of the relational tables into Turbo Prolog knowledge-based facts.

The routines implemented enforce data consistency. For example, the end-points of a corridor should figure in the list of observation points of the map. This type of data consistency is needed for our application.

The data entry could have been made easier just by editing the knowledge base file. However, it is important to keep the format of knowledge base facts needed for subsequent applications such as global planning and localization.

We have implemented another routine to query the knowledge base after it has been created. This routine is named QUERY.PRO and is written in Turbo Prolog.

IX.2.2 Global Path Planning

The task of the global path planner is to find an optimum path for the wheelchair. At the request of the Navigator, the optimum path is computed between two locations. We have implemented the optimum path planning in a way to match the formulation of the map as close as possible and to reduce the demand on memory. Therefore, the path is computed portion by portion. For example, if a request is to compute the path between two points located on different floors, the path is computed on the first floor from the starting point to an elevator (if it exists) and the remainder of the path is computed from the elevator point on the second floor to the destination point. The graph of the floor is retrieved only when the portion of the path requires computation of the path between two points located on different hallways, in which case the Dijkstra's algorithm is called after the edged graph has been retrieved.

The knowledge database is loaded in memory during the length of the computations to make the processing of facts faster. The knowledge base could have been kept on file.

However, it is very slow. When compiled, the global path planner GPATH.EXE and the program DJIK.EXE (implementation of the Dijkstra's algorithm) occupy respectively 41K and 40K of RAM. We have run GEOBASE.INC, a geographic program of the USA. This program contains information on all the cities, roads, rivers, lakes and mountains of the USA, assembled in a knowledge base of facts. The database file of this program occupies 71K of RAM. When both programs are loaded in memory (GPATH.EXE and DJIK.EXE) and assuming a maximum stack size of 4K there remains theoretically 484K of free RAM. It is possible then to load a quite big knowledge base file in RAM.

Up to sixteen types of queries have been implemented (refer to Chapter V), each answering one kind of request for global path planning. All these types of queries have been extensively tested on various examples shown in Chapter V and give good results. The program can successfully retrieve the path between two points located on the same corridor, different corridors, different floors and different buildings. However problems arose when the portion of the path to retrieve contained more than 16 observation points to list. There was a stack overflow error. Even when we increased the stack size to 4K, we could not solve the problem. The reason is that program uses recursion and backtracking. Also, the version of Turbo Prolog used doesn't support global variables. We had to pass the same variables each time between nested modules which unnecessarily occupies the stack. A solution would be to reduce the number of observation points on each corridor. A more robust solution is to use a version of Turbo Prolog that implements global variables.

The results of Test Example 2 of Chapter V show that it takes an average of 0.48s to compute a path between two points located on the same hallway and 2.25s for points located on different hallways. Similar results on a floor plan of an existing building (PCAT) show that it takes an average of 1.5s (including loading of the database) to compute the path between two points located on the same hallway and 3.5s between two points located on different hallways of the same floor.

The exponential fits show in Test Example 2 of Chapter V that when the size of the database (total number of observation points or facts) tends towards infinite, it takes a maximum of 1.97s to 2.3s to compute a path between two points located on the same corridor, and a maximum of 17s to compute the path between two points located on different hallways.

For a medium size problem such as the one shown in Example 3 of Chapter V, the speed of the algorithm is approximately 1.5s for points located on the same hallway and 3.5s for points located on different hallways.

The main evaluation criteria is to see if the path is correctly retrieved and if the computations are fast enough to be carried out on line, i.e, when the wheelchair is running. The database receives request of global path planning from the Navigator module and should give the results back fast enough. Our evaluation of the speed is based on the assumption that the path should be computed within a distance of 1m when the wheelchair is running at a reasonable speed inside the building. From there we will estimate the speed of the wheelchair compatible with the speed of knowledge retrieval by the database. For each computation time, we will estimate the speed of the wheelchair, so that the path can be computed within a distance of 1m when the wheelchair is running.

The speed of the wheelchair was estimated. The results are shown in Table XVIII. The speed of computation so obtained helps support the argument that the path can be computed when the wheelchair is running. The wheelchair may not need to stop in order to compute the path. The speeds estimated are reasonable for normal operation of a wheelchair inside a crowded building. Therefore, our judgement is that an intelligent database is not a bottleneck to the whole system, since it has been proven that the speed of computations is acceptable. The main problem to deal with at this point is to make the image processing fast enough so that the database can provide information to the Navigator without a big delay in time.

TABLE XVIII
ESTIMATION OF THE SPEED OF THE WHEELCHAIR

Δt (seconds)	$L = 1m = V * \Delta t$	V (speed of the wheelchair)
1.97s	---	1.13mile/hr
2.3s	---	0.96mile/hr
3.5s	---	0.63mile/hr
17s	---	0.13mile/hr

Madarasz et al.[4] claim path planning computation time of 1 to 2s. However, in their model, distance between observation points is not recorded. Their criteria is that an optimum path is in the direction which contains the fewer intervening rooms from source to destination. We can then infer that their path planning doesn't make as many computations as we do. We have run their test example with our program. It took 0.88s to 0.93s to compute the path between two rooms located on the same corridor and 2.64s between two rooms located on different corridors. The times include loading of the database in RAM. We think that our results show an improvement compared to their model because in addition, our model includes distance and shortest path (graph) algorithm in the path retrieval

Habib and Yuta [6] in their model have implemented global path planning to retrieve a shortest path on corridor as a sequence of portions of corridors from source to destination point. They retrieve an edged floor graph and compute a shortest path between nodes of this graph. Nodes represent corridor intersections. The final route includes only corridor intersections. Their model of the world is very similar to the model that we have proposed for PSUBOT. However, their model doesn't include the floor as a level of the map hierarchy which leads us to suspect that their model is limited to one story buildings. However, our path retrieval lists all the observation points on the physical route. For their test example, it took 5sec to generate a global path when the starting and goal points are

located on different buildings, 3.5sec when located in the same building but different corridors and 1.5sec when the two points are located on the same corridor. A typical test run on our Example 3 in Chapter V gives the following results. For points located on the same hallway it took an average of 1.5sec, 3.5sec when points are located on different hallways of the same floor, 3.62sec when they are located on different floors, and 3.2 to 5sec when that are are located on different buildings. These times as shown in Example 2 can change when the problem size increases. The empirical formula can be estimated from the equations in Figure 18 of Chapter V, to give the approximate time. We judge that our results are as good as theirs. We consider our approach as an improvement because in our path retrieval, we list all the observation points on the path. We would have claimed much faster computation times if we limited the path only to the nodes of the graph. Also, our model is more complete because it includes many-story buildings.

IX.2.3 Image Matching

The image matching method implemented in this thesis is correspondence matching. The matching methodology was explained in Chapter VII. Each line in the current image is matched to a line in the template image and a success score is recorded. The overall similarity measure of the two images was computed as a function of hit ratios obtained for each level of the image. The matching is the best match, i.e, the one with the higher strictly positive value of the similarity measure.

The upper bound of the duration time of the matching of two images is proportional to $n*m$, where n and m represent the total number of lines in the first and the second image respectively. A curve fit shows in Figures 28 and 30 shows that the average of the computation time increases with the value of the product of the number of lines in both images. Our experiment involved images with seven levels of hierarchy. Over the experiment it took an average of less than 0.3 seconds to match two images. This results prove that the algorithm is fast enough. At this speed, up to three images could be matched

within one second, i.e, practically an attempt to recognize three locations can be carried out while the robot is running.

We simulated localization by trying to recognize a location in a neighborhood of places. The target location was listed in the neighborhood made of a list candidate locations. The current image (simulating the actually perceived image of the scene) was taken to be the image of the target location taken with an angular displacement. The results were satisfying. The target location was recognized in most cases. In one case a location just next to the scene of interest was recognized. It took an average of 2.5 seconds to recognize a location among a "neighborhood" of four candidate locations. Making an assumption of proportionality of the computation time with the size of images and assuming images of approximately same size, it will take an average of 6 seconds to recognize a location among ten candidates, which is the case of a corridor with ten observation points (real case of a corridor with eight rooms along it for example). Assuming a constant speed of the wheelchair and if the information is to be processed within a distance of 1m when the wheelchair is running, it will require a speed of approximately 0.4 mile/hr. This estimation shows that the database can process the knowledge fast enough for on-line requests, i.e, when the wheelchair is running.

The matching of images is not a bottleneck as was feared. Compared to the image processing which takes an average of 4 minutes to process the image, matching (localization) takes less than 10 seconds making it nearly 40 times as fast. This improvement places the database in a position to process the information fast enough to give a feedback to the Navigator module of the wheelchair.

We faced many problems with image matching. Firstly, the vision processing system is not fully automatic, rather it needs the intervention of the programmer to take the picture and to launch the processing. It needs to be fully integrated such that from the application program, an order to take a new picture can be issued and the processing of the image

carried out to produce the output image for the matching. For example, the capture of the image is manual. The "snap" option of the "acquire" routine of SIMPP2 needs user intervention to take the picture and save the quadrant of the picture judged good enough. Also it required sometimes more than two retries to capture an image into the frame grabber. The process could be made autonomous by presetting the "snap" option and the quadrant of the image to save. However, the question is to know 1) if the quadrant chosen was the best portion to represent the scene, 2) if the picture has been really taken (because we had sometimes to try twice to snap a picture), 3) if the image of the template location was taken with the same parameters. These questions need to be answered at run time in order to have an autonomous vision system.

The second serious problem that we faced was illumination of the scene. We have noticed that when illumination of the scene is not adequate, pictures are either dark or too bright. In the first case, after binarization-threshold determination and thinning almost all the edges are wiped out and the line extraction produces only few spots (short lines), almost meaningless for matching. In the second case, the line extraction produces too many lines causing the program to crash due to memory allocation failure. We solved the problem by using a special UV light projector. However, such a light has bad effect on human eyes especially for frequent or prolonged exposure to it.

A question to be addressed with is the meaning of the matching of images described as sets of lines. We have noticed that two lines may match in both images although they belong to totally different details in the physical scene. Since a successful match is defined in terms of the best value of the similarity measure between the current image and the candidate template image, the meaning of the match could be questioned in some cases. Two images could match at the best although they represent totally unrelated scenes. A solution to this problem seems to adopt the methodology of structural matching of images, i.e, to extract higher level features made up of lines as discussed in Chapter VII. However,

this process may be a bottleneck because it takes time to extract those features. The ambiguity is not removed however even with the higher level features. For example, a rectangular long box extracted on the scene can represent either the back cover of a book seen from a short distance, an open door seen from afar, or other rectangular object. How to know exactly what it is? We think that the structural matching would be useful for confined scenes such as the desk of an industrial machine. In this situation, specific template objects are recognized. The components of the scene are very limited in number and clearly distinguishable. So it is easier to guess from the feature extracted on the scene which specific object is seen.

After this discussion, we think that, vision should not be used on this wheelchair for recognition of many details in the environment. Vision should be limited to recognize specific and useful details such as the edges of the corridor materialized by the wall-floor intersection lines or specific landmarks. The feature extractor could concentrate on tracking these details to keep the robot running parallel to the two walls. Objects lying on the hallway could be identified by convolution of the extracted detail with the model detail of the particular location. In this research, other authors [4, 6] have used vision for the purpose of tracking specific landmarks only. Specific landmarks attached to rooms, for example, will bring a little change to the environment. However, we think that this is not an issue, because putting a special sign doesn't change much the interior of the building. It is also a matter of public policy to allow special signs for the disabled in public buildings. This remark is not even a question for a private house. The only problem will be the nuisance caused to other users of the building if an UV projector is used for illumination of the scene.

The method of matching also uses a lot of template images representing the observation points. The limited memory on PC makes it virtually impossible to store the images of all the observation points (such as rooms) in memory. However, if for example semantic

description is used to store information on a landmark of a given place, the problem will be reduced to extract such a feature in the perceived scene in order to come up to the decision that the robot is at that given location.

CHAPTER X

CONCLUSIONS AND FUTURE WORK

X.1 CONCLUSIONS

An intelligent database has been implemented in this thesis to be integrated to the PSUBOT wheelchair system. The intelligent database is so called because it uses knowledge-based method to perform its tasks. An autonomous device such as desired for the PSUBOT, needs a central thinking unit which can make intelligent reasoning based on learned facts and perceived information. In this thesis, the main tasks assigned to the database were map management, global path planning and intelligent localization of the wheelchair inside the building. All the routines were fully implemented at the exception of the localization routine. However, the image matching used by this routine has been implemented and tested. All the routines have been implemented such that they can communicate with other modules of the system through file sharing and interrupts. The main constraint of this development was that the computations and knowledge retrieval be fast enough so that they could be carried on when the wheelchair is running at a reasonable speed.

The thesis started with the definition of a world model. The model proposed was a hierarchical description of a group of neighboring buildings. The main levels of the hierarchy of the description of a building were the floor, the corridor, the room and the object (observation point) on the corridor. This model maintained information on the geometric disposition of items on the floor and other specific information on those items. The map was subsequently used in finding a global route for the wheelchair. A relational

database and a set of management routines were implemented to keep the map data.

The next step of the development was the implementation of global path planning. The task was to compute a global optimum route for the wheelchair between two points of the map. A major contribution of this work has been to combine knowledge-based method to a regular graph method to add more flexibility and knowledge power to the process, and to seek a speedup of the computations on PC. The methodology well suited the formulation of the problem and gave good results. The program was able to retrieve the path fast enough. An estimation of the speed of the wheelchair compatible with the speed of global path planning allowed us to conclude that, at a speed of 0.13 to 1.13 mile/hr, the path could be computed within a distance of 1m when the wheelchair is running.

Image matching for localization was conducted with the method of correspondence matching of images described as sets of straight lines. The results were satisfactory. It took an average of 0.3 second to match two images with seven levels of hierarchy, and 2.5 seconds to recognize a location among four candidate locations. However, we have noticed that matching of lines is ambiguous. Matching lines only can be meaningless when applied to real scenes because two images representing unrelated scenes could match if the matching relies only on the best similarity measure between the two images. Along with this problem, template matching of images as implemented in this project will yield a lot of memory to store the images.

The results obtained have proven that knowledge-based method can improve the speed of global path planning and can help to integrate more knowledge capability to the PSUBOT system through the database. For example, intelligent localization of the wheelchair can be implemented. Simple queries judged useful for the user of the device could be asked to the database. Such questions, could be, for example, "list of the room on this hallway". The voice control will translate the questions into queries addressed to the database and the result will be translated into sounds or special signals back to the user.

We judge that the small system implemented is satisfactory for this stage of development and has met our expectations for the speed of processing of information.

X.2 FUTURE WORK

The present work was the beginning of integration of an intelligent database to the PSUBOT system. Many questions have been raised, for example, image matching for localization. Future work in this topic will include three main directions. Firstly, the localization routine should be implemented and tested. The localization algorithm, which we suggest to be knowledge-based, should make use of the structural matching of images. A model of landmark will be chosen for each location as well as for the corridor. The recognition of a location will be conducted by recognizing its template landmark in the perceived image. The second horizon of extension of the intelligent database is to integrate the management of voice or signal queries from the user of the device. The last horizon we foresee is the solution of the stack limitation problem (encountered on PC) through special software techniques in order to handle more recursion and backtracking.

REFERENCES

- [1] A. M. Legate, EE406 Project Report submitted to Dr. Marek A. Perkowski, Department of Electrical Engineering, Portland State University, Spring 1989.
- [2] K. Stanton, "PSUBOT: System Overview and Path Planning," *Proc. of Northcon 91*, pp. 285-290, Oct. 1-3, 1991.
- [3] C. Espinoza, "A Low-to-medium Image Processing System for a Mobile Robot," Master's thesis, May 1991, Portland State University.
- [4] R. L. Madarasz, L. C Heiny, R. F. Crompt and N. M Mazur, "The Design of an Autonomous Vehicle for the Disabled," *IEEE J. Robotics Automat.*, vol. RA-2, no. 3, pp. 117-126, Sept. 1986.
- [5] S. Rao and R. Kuc, "INCH: an Intelligent Wheelchair Prototype," *IEEE 15th Annual Bioengineering Conference*, pp.35-36, Aug. 1989.
- [6] M. K. Habib and S. Yuta, "Development and Implementation of Navigation System for an Autonomous Mobile Robot Working in a Building Environment with its Real Time Application," University of Tsukuba, Inst. of Information Science and Electronics, Intelligent Robot Lab., Tennoudai, Tsukuba, Ibaraki 305, Japan, 1989.
- [7] G. Marque-Pucheu, J. Marting-Gallaussiaux and G. Jomier, "Interfacing Prolog and Relational Database Management Systems," New Applications of Databases, G. Gardarin and E. Gelenbe, Academic Press, 1984.
- [8] K.S. Fu, "Syntactic Methods in Pattern Recognition," *Mathematics in Science and Engineering* , Vol. 112, Academic Press, 1974.
- [9] V. Capellini, A. Del Bimbo and A. Mecocci, "An Object Oriented Approach for Object Recognition and Classification," *ICASSP*, vol.3, pp 1723-1726, 1989.
- [10] W. Eric and L. Grimson, "On Recognition of Parametrized Objects," MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, Mass. 02139.
- [11] M. Hebert, "Building and Navigating Maps of Road Scenes Using an Active Sensor," *ROBOT*, vol. 2, pp 1136-1142, 1989.
- [12] E. Krotkov, "Mobile Robot Localization Using a Single Image," *ROBOT*, vol. 2, pp 984-989, 1989.

- [13] N. Ayache and O. D. Faugeras, "Building, Registering, and Fusing Noisy Visual Maps," *The International J. Robotics Research*, vol. 7, no. 6, pp. 45-65, Dec. 1988.
- [14] G. A. Boy, "Man-Machine Distributed Intelligence," *IFAC Proceedings Series*, No. 4, pp. 279-285, 1989.
- [15] A. Kasinski and A. Hanczak, "Robot Control System with Distributed Intelligent Functions," *IFAC Proceedings Series*, No. 4, pp. 337 - 341, 1989.
- [16] G. N. Saridis, "Intelligent Machines: Distributed vs Hierarchical Intelligence," *IFAC Proceedings Series*, No. 4, pp. 29 - 34, 1989.
- [17] N. Ayache and O. D. Faugeras, "Maintaining Representations of the Environment of a Mobile Robot," *IEEE T. Robotics Automat.*, vol. 5, no. 6, pp. 804-819, Dec. 1989.
- [18] C. Fennema, A. Hanson, E. Riseman, J. R. Beveridge and K. Kumar, "Model-Directed Mobile Robot Navigation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 6, Nov./Dec. 1990.
- [19] J.M Jansen and F.W Sijstermans, "Template Matching with an MIMD Computer," *Proc. ICASSP*, vol.3, pp 1727-1730, 1989.
- [20] J. L. Crowley, "Navigation for an Intelligent Mobile Robot," *IEEE J. Robotics Automat.*, vol. RA-1, no. 1, pp. 31-41, Mar. 1985.
- [21] O. Shmueli, S. Tsur and H. Zfira, "Rule Support In Prolog," *Expert Database Systems, Proceedings from the First International Workshop*, pp. 247-269, Edited by Larry Kerschberg, 1986.
- [22] L. E. De Heer, "Plant Scale Process Monitoring and Control Systems: Eighteen Years and Counting," *Computer Aided Process Operations*, Edited by G. V. Reklaitis and H. D. Spriggs, Published by CACHE - ELSEVIER, 1987.
- [23] L. M. Waxman, J. J. LeMoigne, L. S. Davis, B. Srinivasan, T. R. Kushner, E. Liang and T. Siddalingaiah, "A Visual Navigation System for Autonomous Land Vehicles," *IEEE J. Robotics Automat.*, vol. RA-3, no. 2, pp. 124-141, Apr. 1987.
- [24] T. O. Binford, "Key Issues in Robot Vision," Artificial Intelligence Laboratory, Stanford University, Stanford, California 94305, USA., 1989.
- [25] R. C. Arkin and R. R. Murphy, "Autonomous Navigation in a Manufacturing Environment," *IEEE T. Robotics Automat.*, vol. 6, no. 4, pp. 445-454, Aug. 1990.
- [26] R. C. Arkin, "Navigational Path Planning for a Vision-based Mobile Robot," *Robotica*, vol. 7, pp 49-63, 1989.
- [27] R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *IEEE International Conference on Robotics and Automation*, pp. 264-271, 1987.

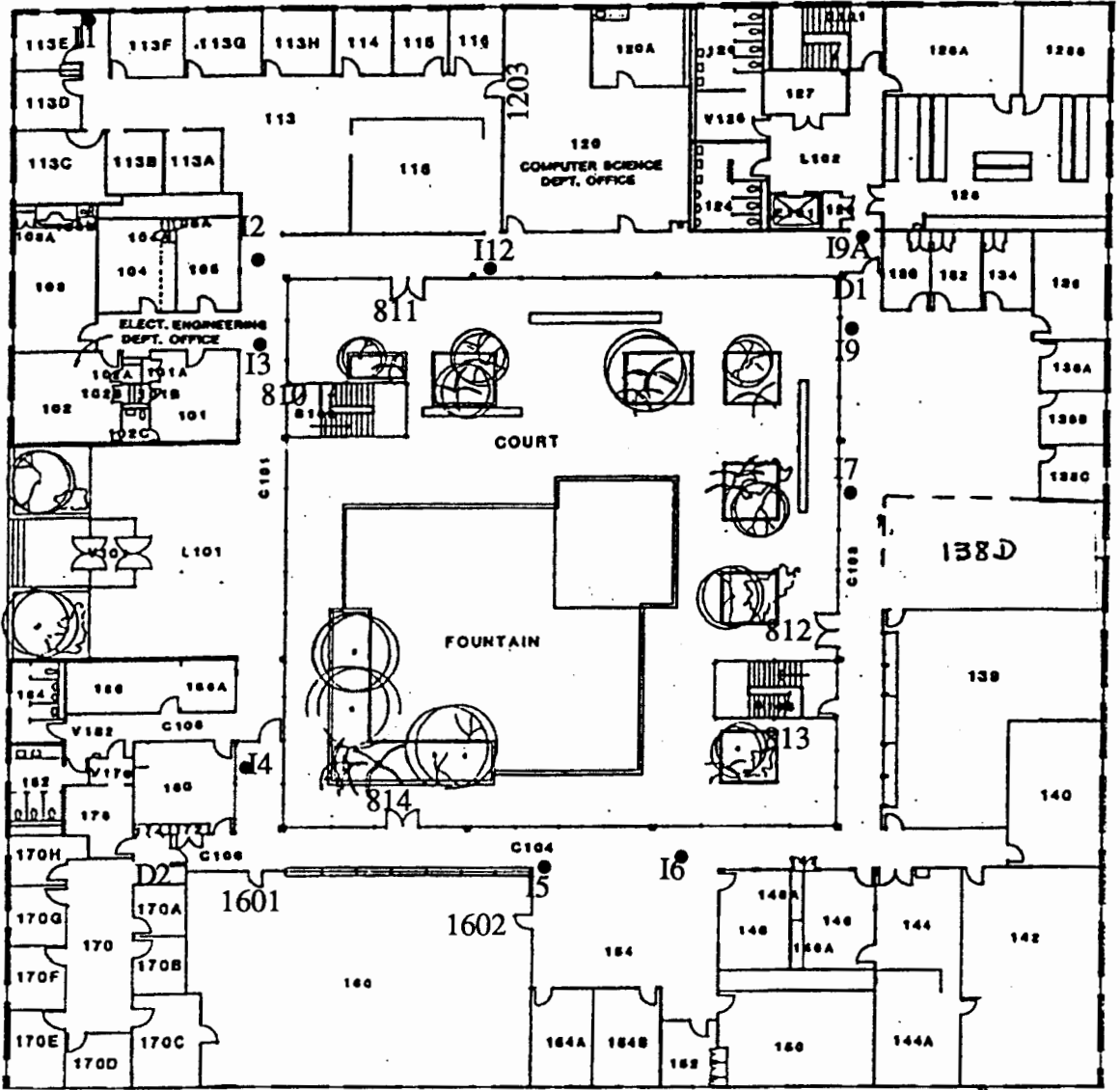
- [28] A. Sanderson and N. J. Foster, "Attributed Image Matching Using a Minimum Representation Size Criterion," *ROBOT*, vol. 1, pp 360-365, 1989.
- [29] R. K. Brail, J. W. Hughes and C. A. Arthur, "Transportation Services for the Disabled and the Elderly," Published by Center for Urban Policy Research, New Brunswick, New Jersey, 1976.
- [30] M. Missikoff and G. Wiederhold, "Towards A Unified Approach For Expert And Database Systems," *Expert Database Systems, Proceedings from the First International Workshop*, pp. 383-399, Editor Larry Kerschberg, 1986.
- [31] J. D. Ullman, "Principles Of Database And Knowledge-Base Systems," vol. I, *Computer Science Press Inc.*, 1988.
- [32] I. Bratko, "Prolog Programming for Artificial Intelligence," *International Computer Science series*, Addison-Wesley Publishing Company, 1987.
- [33] C.J. Date, "An Introduction To Database Systems," Vol. I, Fourth Edition, *Addison-Wesley Publishing Company*, Reprinted June 1987 from Copyright 1986.
- [34] K. Parsaye, M. Chignell, S. Khoshafian and H. Wong, "Intelligent Databases: Object-oriented, Deductive, Hypermedia Technologies," Published by John Wiley & Sons Inc., 1989.
- [35] R. H. Katz, "Transaction Management in the Design Environment," New Applications of Databases, G. Gardarin and E. Gelenbe, Academic Press, 1984.
- [36] M. Gray, "Databases for Computer-Aided Design," New Applications of Databases, G. Gardarin and E. Gelenbe, Academic Press, 1984.
- [37] BORLAND International Inc. "PARADOX: Immediate Database Power," Guide to The Paradox Personal Programmer, Ed. BORLAND, 1988.
- [38] BORLAND International Inc. "PARADOX: Immediate Database Power," PAL User's Guide, Ed. BORLAND 1988.
- [39] BORLAND International Inc. "Turbo Prolog The Natural Language of Artificial Intelligence," Ed. BORLAND 1986
- [40] R. Stengel and A. Niehaus, "Intelligent Guidance Headway and Lane Control," *ACC*, vol. 2, pp. 1059-1064, 1989.
- [41] W. Clay Collier, "In-vehicle Route Guidance Systems Using Map Matched Dead Reckoning," *IEEE PLANS: 90 Position Location and Navigation Symposium 1990*, vol.4, pp. 359-363, 1990.
- [42] J. M. Richardson and K. A. Marsh, "Fusion of Multisensor Data," *The International J. Robotics Research*, vol. 7, no. 6, pp. 79-96, Dec. 1988.
- [43] A. Elfes, "Sonar-based Real World Mapping and Navigation," *IEEE J. Robotics Automat.*, vol. RA-3, no. 3, pp. 249-265, June 1987.

- [44] E. L. Lawler, "Shortest Path and Network Flow Algorithms," "Discrete Optimization I", *Annals of Discrete Mathematics*, vol.4, P.L Hammer, E.L.Johnson and B.H. Korte, North-Holland Publishing Company - Amsterdam, New York, Oxford, 1979.
- [45] R. Bergevin and M. D. Levine, "Extraction of Line Drawing Features for Object Recognition," *Proc. 10th IEEE Intern. Conf. On Recognition*, vol. I, pp. 496-501, 1990.
- [46] O. R. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and curves in Pictures", *Communications of the ACM*, Vol. 15, No.1, pp. 11-15 , January , 1972.
- [47] J. S. Lim, "Two Dimensional Signal and Image Processing," *Prentice Hall Signal Processing Series*, Prentice Hall, Englewood Cliffs, New Jersey, Ed. 1989.
- [48] K. Sugihara, "Location of a Robot Using Sparse Visual Vnformation," *IEEE 15th Annual Bioengineering Conference*, pp.613-622, Mar. 1989.

EXAMPLES OF APPLICATION

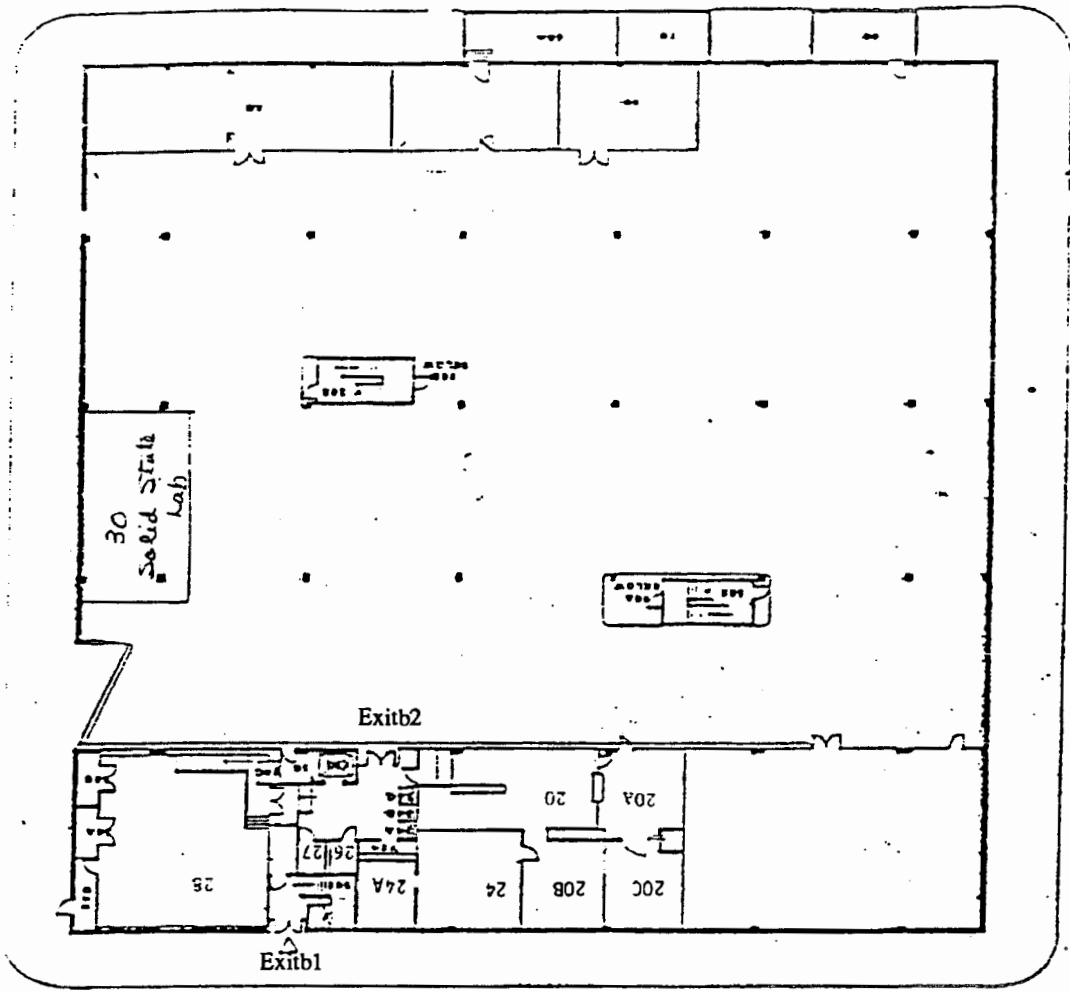
APPENDIX A

A-1 AN EXAMPLE OF A MAP WITH TWO BUILDINGS



PCAT: first floor

Figure 40. Floor plan of the first floor of the building PCAT



PCAT: basement

Figure 41. Floor plan of the basement of the building PCAT.

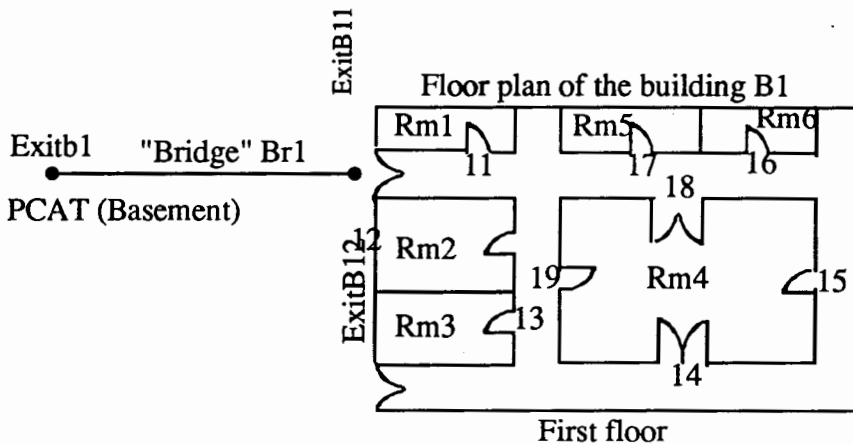


Figure 42. Floor plan of the first floor of the building B1.

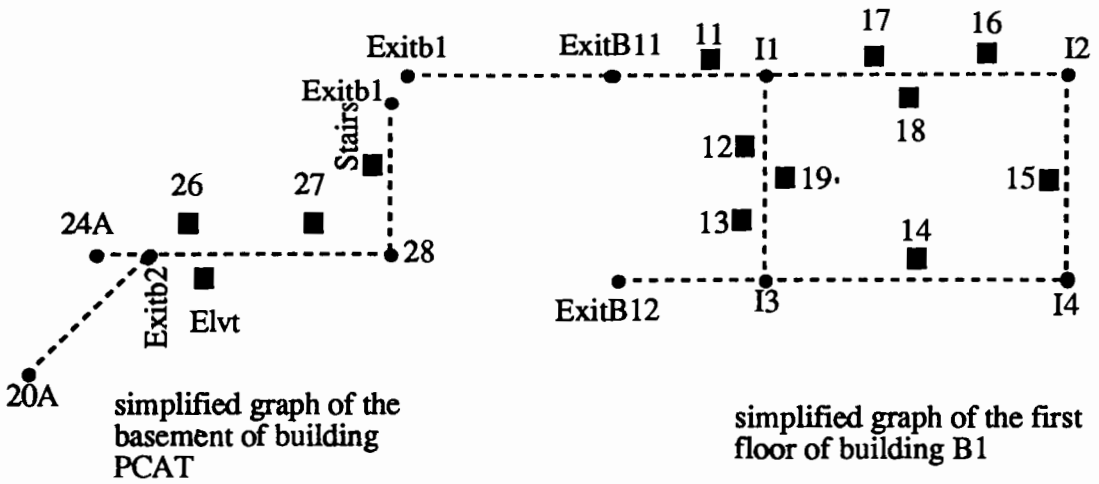


Figure 43. Problem formulation as an edged graph (basement of PCAT and first floor of B1).

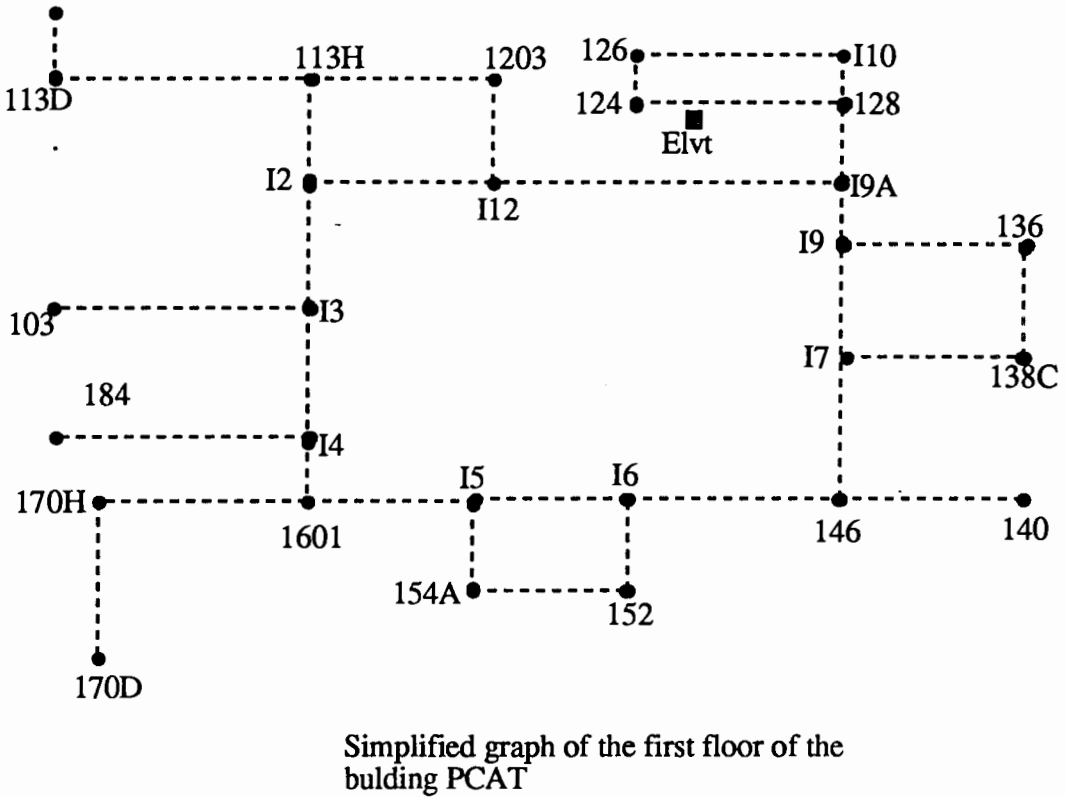


Figure 44. Problem formulation as an edged graph (first floor of PCAT).

A-2 KNOWLEDGE BASE FACTS

(of the problem in Example 1 of Chapter V, page 93)

```

building("house", "village", 1, "y", "y")
floor("fl1", "house", "fl1", "y")
corridor("c1", "fl1", "house", 1,
          0.75, 0.3, 0.04)
corridor("c2", "fl1", "house", 1,
          1.0, 0.3, 0.04)
corridor("c3", "fl1", "house", 1,
          0.25, 0.3, 0.04)
corridor("c4", "fl1", "house", 1,
          3.0, 0.3, 0.04)
corridor("c5", "fl1", "house", 1,
          0.25, 0.3, 0.04)
corridor("c6", "fl1", "house", 1,
          1.0, 0.3, 0.04)
corridor("c7", "fl1", "house", 1,
          0.25, 0.3, 0.04)
corridor("c8", "fl1", "house", 1,
          0.75, 0.3, 0.04)
corridor("c9", "fl1", "house", 1,
          3.0, 0.3, 0.04)
corrend("c1", "fl1", "house", "i8",
         0.0, "ne", "i1", 0.0, "se")
corrend("c2", "fl1", "house", "i1",
         0.0, "ne", "i2", 0.0, "se")
corrend("c3", "fl1", "house", "i2",
         0.0, "se", "i9", 0.0, "ne")
corrend("c4", "fl1", "house", "i3",
         0.0, "ne", "i2", 0.0, "se")
corrend("c5", "fl1", "house", "i3",
         0.0, "ne", "i4", 0.0, "se")
corrend("c6", "fl1", "house", "i3",
         0.0, "ne", "i5", 0.0, "se")
corrend("c7", "fl1", "house", "i5",
         0.0, "ne", "i6", 0.0, "se")
object("Rm2", "c9", "fl1", "house", 1, 1,
        0.6, 1, 2, 3, 4, "Rm3")
object("Rm3", "c9", "fl1", "house", 1, 1,
        0.9, 1, 2, 3, 4, "Rm4")
object("Rm4", "c9", "fl1", "house", 1, 1,
        1.2, 1, 2, 3, 4, "Rm5")
object("Rm5", "c9", "fl1", "house", 1, 1,
        1.58, 1, 2, 3, 4, "Rm6")
object("Rm6", "c9", "fl1", "house", 1, 1,
        1.8, 1, 2, 3, 4, "Rm7")
object("Rm7", "c9", "fl1", "house", 1, 1,
        2.1, 1, 2, 3, 4, "Rm8")
object("Rm8", "c9", "fl1", "house", 1, 1,
        2.4, 1, 2, 3, 4, "Rm9")
object("Rm9", "c9", "fl1", "house", 1, 1,
        2.7, 1, 2, 3, 4, "Rm10")
object("Rm10", "c9", "fl1", "house", 1, 1,
        2.98, 1, 2, 3, 4, "Rm10")
object("Rm11", "c9", "fl1", "house", 1, 2,
        2.98, 1, 2, 3, 4, "Rm12")
object("Rm12", "c9", "fl1", "house", 1, 2,
        2.41, 1, 2, 3, 4, "Rm13")
object("Rm13", "c9", "fl1", "house", 1, 2,
        2.11, 1, 2, 3, 4, "Rm14")
object("Rm14", "c9", "fl1", "house", 1, 2,
        1.81, 1, 2, 3, 4, "Rm15")
object("Rm15", "c9", "fl1", "house", 1, 2,
        1.51, 1, 2, 3, 4, "Rm17")
object("Rm17", "c9", "fl1", "house", 1, 2,
        1.25, 1, 2, 3, 4, "Rm19")
object("Rm19", "c9", "fl1", "house", 1, 2,
        0.65, 1, 2, 3, 4, "Rm20")
object("Rm20", "c9", "fl1", "house", 1, 2,
        0.3, 1, 2, 3, 4, "Rm20")

```

```

object("Rm28","c4","fl1","house",1,2,
      2.2,1,2,3,4,"Rm29")
object("Rm29","c4","fl1","house",1,2,
      1.9,1,2,3,4,"Rm30")
object("Rm30","c4","fl1","house",1,2,
      1.6,1,2,3,4,"Rm31")
object("Rm31","c4","fl1","house",1,2,
      1.4,1,2,3,4,"Rm32")
object("Rm32","c4","fl1","house",1,2,
      1.3,1,2,3,4,"Rm33")
object("Rm33","c4","fl1","house",1,2,
      1.0,1,2,3,4,"Rm34")
object("Rm34","c4","fl1","house",1,2,
      0.7,1,2,3,4,"Rm35")
object("Rm35","c4","fl1","house",1,2,
      0.4,1,2,3,4,"Rm36")
object("Rm36","c4","fl1","house",1,2,
      0.1,1,2,3,4,"i3")
object("Rm37","c5","fl1","house",1,2,
      0.2,1,2,3,4,"i3")
room("Rm1","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm2","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,
     [obst(2.5,1.57,1.2,1.5),
      obst(1.5,2.096,1.2,1.5)])
room("Rm3","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm4","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm7","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm8","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm9","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm10","fl1","house","Rm1",
     "Rm1",1,0.45,0.2,0.5,[])
room("Rm19","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm20","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm21","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm22","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm23","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm24","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm25","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm26","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm27","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm28","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm29","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm30","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm31","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm32","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm33","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm34","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm5","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm6","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])
room("Rm14","fl1","house","Rm1",
     "Rm1",1,0.45,0.25,[])

```

```

corrend("c8","fl1","house","i5",
0.0,"ne","i7",0.0,"se")
corrend("c9","fl1","house","i5",
0.0,"ne","i1",0.0,"se")
object("i1","c1","fl1","house",0,
2,0.0,1,2,3,1,"i1")
object("i2","c2","fl1","house",
0,2,0.0,1,2,3,1,"i2")
object("i3","c4","fl1","house",
0,2,0.0,1,2,3,1,"i3")
object("i4","c5","fl1","house",
0,2,0.0,1,2,3,1,"i4")
object("i5","c6","fl1","house",
0,2,0.0,1,2,3,1,"i5")
object("i6","c7","fl1","house",
0,2,0.0,1,2,3,1,"i6")
object("i7","c8","fl1","house",
0,2,0.0,1,2,3,1,"i7")
room("Rm12","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
room("Rm13","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
room("Rm16","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
room("Rm17","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
room("Rm18","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
object("Rm1","c9","fl1","house",1,1,
0.3,1,2,3,4,"Rm2")
object("Rm16","c4","fl1","house",1,2,
1.5,1,2,3,4,"Rm18")
object("Rm18","c4","fl1","house",1,2,
1.2,1,2,3,4,"i4")
object("Elv1","c2","fl1","house",3,2,
0.6,1,2,3,4,"i2")
object("Elv2","c8","fl1","house",3,2,
0.5,1,2,3,4,"i6")
object("Rm21","c3","fl1","house",1,2,
0.2,1,2,3,4,"Rm21")
object("Rm22","c4","fl1","house",1,2,
2.98,1,2,3,4,"Rm23")
object("Rm23","c4","fl1","house",1,2,
2.9,1,2,3,4,"Rm24")
object("Rm24","c4","fl1","house",1,2,
2.8,1,2,3,4,"Rm25")
object("Rm25","c4","fl1","house",1,2,
2.7,1,2,3,4,"Rm26")
object("Rm26","c4","fl1","house",1,2,
2.5,1,2,3,4,"Rm27")
room("Rm15","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
room("Rm35","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
room("Rm36","fl1","house","Rm1",
"Rm1",1,0.45,0.25,
[obst(3.0,1.047,1.8,1.9)])
room("Rm37","fl1","house","Rm1",
"Rm1",1,0.45,0.25,[])
object("i9","c3","fl1","house",0,2,
0.0,1,2,3,1,"i9")

```

PSEUDOCODES OF PROGRAMS

APPENDIX B

B-1 THE GLOBAL PATH PLANNER

MAIN PROGRAM (declarative)

INPUT:

- "path.dat" is the data file from the Navigator. The contents of this file are organized as follows.

First record line: name of the map site

Second record line LOC(L1,F1,B1)-starting point of the path

Third line LOC(L2,F2,B2) - ending point of the path

Note: L is the name of the location; F is the floor name; B is the building name and LOC is the name of the domain entity that represents a location in the knowledge base.

- "*.dba" is the map knowledge base file. The "*" stands for the name of the map site. This file contains all the information on the map described as a collection of facts (see Chapter VI of the thesis document).

OUTPUT:

- "graph.fl" and "vertices.fl" are both files that contain respectively the information on the edges and the vertices of the edged-graph of the floor. The graph is retrieved when there is a need to compute the path between two points which belong to different corridors (the Djistra's algorithm is used to compute the optimum path).

- "path.log" is a file containing the computed path. This file is sent back to the Navigator.

run:-

* open file "section.dat"

* read information:

(1) name of the map site Map

(2) names of the starting and destination points of the path to compute - L1 and L2.

* exist DOS file for the map database: Mbase = Map + ".dba"

* consult map database Mbase.

* CALL Gpath(S,D). /* compute the path between L1 and L2 */

ROUTINE Gpath: compute the global path

INPUT:

- starting point (L1,F1,B1) and ending point (L2,F2,B2)

OUTPUT:

- computed path

/ case (1i) */*

Gpath(L1,F1,"xx",L2,F2,"xx"):- */* sub-case (1) */*
 * EXISTS facts object(L1,F1,B) and object(L2,F2,B),
 * CALL path(L1,F1,B,L2,F2,B).

Gpath("xx","xx","xx",L2,F2,"xx"):- */* sub-case (2) */*

* CALL Locate_me(L1,F1,B,1) */* compute the current location of the robot */*
 * EXISTS a fact object(L2,F2,B)
 * CALL path(L1,F1,B,L2,F2,B).

/ case (2i) */*

Gpath(L1,"xx","xx",L2,"xx","xx"):- */* sub-case (1) */*
 * EXISTS facts object(L1,F1,B) and object(L2,F2,B)
 * CALL path(L1,F1,B,L2,F2,B).

Gpath("xx","xx","xx",L2,"xx","xx"):- */* sub-case (2) */*

* CALL Locate_me(L1,F1,B,1)
 * EXISTS fact object(L2,F2,B)
 * CALL path(L1,F1,B,L2,F2,B).

/ case (3i) */*

Gpath(L1,"xx","xx","xx",F2,"xx"):-
 * EXISTS fact object(L1,F1,B) and
/ find a point at an elevator */*
 * EXISTS fact object(Elvt,F2,B) which is an elevator
 * CALL path(L1,F1,B,Elvt,F2,B).

/ case (4i) */*

Gpath(L1,F1,"xx",L2,"xx",B2):- */* sub-case (1) */*
 * EXISTS facts object(L1,F1,B1) and object(L2,F2,B2)
 * CALL path(L1,F1,B,L2,F2,B).

Gpath("xx","xx","xx",L2,"xx",B2):- */* sub-case (2) */*

* CALL Locate_me(L1,F1,B1,1),
 * EXISTS fact object(L2,F2,B2),
 * CALL path(L1,F1,B1,L2,F2,B2).

/ case (5i) */*

Gpath(L1,"xx",B1,L2,"xx",B2):- */* sub-case (1) */*
 * EXISTS facts object(L1,F1,B1) and object(L2,F2,B2)
 * CALL path(L1,F1,B1,L2,F2,B2).

Gpath(L1,"xx","xx",L2,"xx",B2):- */* sub-case (2) */*

* EXISTS facts object(L1,F1,B1) and object(L2,F2,B2)
 * CALL path(L1,F1,B1,L2,F2,B2).

/ case (6i) */*

Gpath(L1,"xx",B1,"xx","xx",B2):- */* sub-case (1) */*
 * EXISTS facts object(L1,F1,B1)
 * EXISTS a fact on bridge bridge(B1,B2,Br),
 * find the end points of the bridge (X1 and Y1)
 * EXISTS fact object(L2,F2,B2)
 * CALL path(L1,F1,B1,L2,F2,B2).


```

Gpath("xx","xx","xx","xx","xx",B2):- /* sub-case (2) */
    * CALL Locate_me(L1,F1,B1,1),
    * A_bridge(B1,B2,Br),!,
    * EXISTS a fact on bridge bridge(B1,B2,Br)
    * find the end points of the bridge (X1 and Y1)
    * EXISTS fact object(Y1,F2,B2)
    * CALL path(L1,F1,B1,Y1,F2,B2).

/* case (7i) */
Gpath("xx","xx","xx","xx",F2,B2):- /* sub-case (1) */
    * CALL Locate_me(L1,F1,B1,1),
    * EXISTS fact object(L1,F1,B1)
    * EXISTS fact object(Elvt,F2,B2) which is an elevator
    * CALL path(L1,F1,B,Elvt,F2,B2).
Gpath("xx",F1,"xx","xx",F2,B2):- /* sub-case (2) */
    * EXISTS object(Elvt1,F1,B1),/*find a point at an elevator on floor F1*/
    * EXISTS object(Elvt2,F2,B2),/*find a point at an elevator on floor F2*/
    * CALL path(Elvt1,F1,B1,Elvt2,F2,B2).

/* case (8i) */
Gpath("xx","xx","xx",L,"zz","zz"):- /* sub-case (1) */
    * CALL Locate_me(L1,F1,B1,1),
    * EXISTS fact object(L1,F1,B1)
    * EXISTS fact object(L,F2,B2)/* find any object named L */
    * CALL path(L1,F1,B1,L,F2,B2).

/* case (0i) */
Gpath(L1,F1,B1,L2,F2,B2):-
    * CALL path(L1,F1,B1,L2,F2,B2).

/* ---PRIMITIVE PATH(S,F1,B1,D,F2,B2)
   S and D designate the source and destination points respectively ---
   * ----- *
   Level (1) : Top level of the algorithm - check if the points belong to the same building.
   * ----- */

path(S,F1,B1,D,F2,B2):- /* S and D are in different buildings */
    * EXISTS facts object(S,F1,B1) and object(D,F2,B2) such that B1 <> B2
    * CALL path_between_buildings(S,B1,D,B2)
path(S,F1,B1,D,F2,B2):- /* S and D are in the same building */
    * EXISTS facts object(S,F1,B1) and object(D,F2,B2)
    * CALL path_in_building(S,D,B1) /* B1 = B2 */
path(S,F1,B1,D,F2,B2):-
    * other case write message( Error or Data missing) /* error or data missing */

/* ----- *
   Level (2) : Path between points located in different buildings
   * ----- */

path_between_buildings(X,B1,Y,B2):-
    * EXISTS a bridge bridge(B1,B2,Br)

```

```

* find the end points of bridge Br (X1 and Y1)
  /* compute path between the points X and X1 in building B1 */
* CALL path_in_building(X,X1,B1)
* CALL path_on_bridge(X1,Y1,B1,B2)
  /* compute path between the points X and X1 in building B1 */
* CALL path_in_building(Y1, Y,B2)

```

```

path_between_buildings(X,B1,Y,B2):- /* no bridge between B1 and B2 */
  * write message "A: There is not a path connecting the buildings"
  * EXIT

```

```

path_on_bridge(S,D,B1,B2):- /* S and D are the end-points of the same bridge */
  * check that S and D are the end points of a bridge between B1 and B2
  * retrieve the path between S and D

```

```

/* ----- */
Level (2) : Path between two points located in the same building
/* ----- */

```

```

/* case (i): S and D are in same building but on different floors */

```

```

path_in_building(S,D,B):-
  * EXISTS a fact object(S,F1,B) /* identify the floor on which S ia located */
  * EXISTS a fact object(D,,F2,B) /* identify the floor on which S ia located */
  * F1 <> F2
  * CALL path_between_floors(S,F1,D,F2,B).

```

```

/* case(ii): S and D are on same floor */

```

```

path_in_building(S,D,B):-
  * EXISTS a fact object(S,F1,B) /* identify the floor on which S ia located */
  * EXISTS a fact object(D,,F2,B) /* identify the floor on which S ia located */
  * F1 = F2
  * CALL path_on_floor(S,D,F,B).

```

```

/* ----- */
Level (3) : Path between two points located on different floors
/* ----- */

```

```

path_between_floors(X,F1,Y,F2,B):-
  /* not possible if no elevator connecting them */
  * not a working elevator between F1 and F2
  * write message : "A: No mean to go from the present floor F1 to floor F2.

```

```

path_between_floors(X,F1,Y,F2,B):-
  * X belongs to corridor C1 of floor F1
  * Y belongs to corridor C2 of floor F2
  * EXISTS a fact object(ElvF1,CE1,F1,B), /* ElvF1 is elevator on floor F1 */
  * EXISTS a fact object(ElvF2,CE2,F2,B), /* ElvF22 is elevator on floor F2 */
  * CALL path_on_floor(X,ElvF1,F1,B) /* compute the section of path on F1 */
  * CALL path_on_floor(ElvF2,Y,F2,B)./* compute the section of path on F2
*/

```

```
/* ----- *
Level (3) : Path between two points located on the same floor
* ----- */
```

```
/* case (ii) : S and D are located on the same hallway */
path_on_floor(S,D,F,B):- /* are end-points of the same corridor */
    * S and D are both the end points of a corridor C of floor F in building B.
    * CALL path_on_corridor(S,D,C,F,B)
path_on_floor(S,D,F,B):-
    * S belongs to corridor C1 of floor F
    * D belongs to corridor C2 of floor F
    * C1 = C2,
    * CALL path_on_corridor(S,D,C1,F,B)
```

```
/* case (i) : S and D are located on different hallways */
path_on_floor(S,D,F,B):-
    * S belongs to corridor C1 of floor F
    * D belongs to corridor C2 of floor F
    * C1  $\diamond$  C2,
    * CALL path_between_corridors(S,C1,D,C2,F,B).
```

```
/* ----- *
Level (4) : computing the path
* ----- */
```

```
/* case (i): compute path on same corridor by simply listing all the
locations between S and D */
path_on_corridor(X,Y,C,F,B):-
```

```
    /* X and Y are the located on the same hallway */
    * RETRIEVE all the locations that are between S and D
```

```
/* case (ii): compute path between two corridors by applying the Djisktra's
algorithm to find the shortest path between the end-points of the
two corridors then applying intelligence to eliminate the locations on the
starting and ending corridors which are not on the physical path.*/
```

```
path_between_corridors(S,C1,D,C2,F,B):-
    /* Path between corridors C1 C2 using the Djisktra's algorithm */
    * EXISTS facts corrend(C1,F,B,E11, E12) /* find the end points of corridor C1
    */
    * EXISTS facts corrend(C2,F,B,E21, E22) /* find the end points of corridor
    C22 */
    * CALL GRAPH(F,B) /* build the edged-graph of the floor F */
    * CALL DJIK(E11,E22) /* apply the djikstra's algorithm to find the shortest
    path */
    * CALL Refine_path /* refine the path */
```

- Implementation of the DJIKSTRA's shortest path algorithm

```

/*****
* PROGRAM DJIK.C
* CALLED by the main program GPATH
* Function to achieve: Find an optimum path between a given point of the
*                       vertex-graph using the Dijkstra's shortest path algorithm.
*****/

```

INPUT: DATA FILES

```

"graph.fl",          /* data file for graph */
"vertices.fl",      /* data file for vertices */
"section.dat",      /* data file source containing the name of source and destination
points*/

```

OUTPUT: DATA FILES

```

"section.log",      /* data file containing the path between source and destination; this
file is
                    sent back to the main program GPATH */

```

DATA STRUCTURES

```

/* an edge of the graph */
STRUCTURE {
    char vertex1[10]; /* end1 of an edge */
    char vertex2[10]; /* end2 of an edge */
    float length;     /* length of edge */
    float obfreq;     /* obstacle frequency */
    float tfreq;      /* travel frequency */
} an_edge;

/* a vertex */
STRUCTURE {
    char vertex_name[10];
    int number; /* has been added to identify vertex at step 1*/
    float dist_source; /* distance to start point */
    short visited; /* 1 if has been visited 0 if not */
    int pathto[20]; /* pat from here to source node */
} a_vertex;

an_edge edge[100];
a_vertex vertex[200];
int number_of_edges = 0 ;
int number_of_vertices = 0 ;
int IS, IT; /* the label of the source and target nodes respectively */
char source[10], target[10]; /* source node and target node names */
int IKL ; /* the latest ending point with dist_source < infinite
*/

```

MAIN_PROGRAM

BEGIN:

```

* CALL Init()      /* read data from file into data structures */
* CALL step0();   /* step 0 of the method */
* CALL step1();   /* step 1 of the method */
END MAIN_PROGRAM

```

```
/* INIT: read data from files into structures and variables */
```

```
PROCEDURE Init()
```

```
BEGIN:
```

```
* READ read source node and target node from file "section.dat"
```

```
* READ edges of the file "graph.fl" into the edges structure
  - count the number of edges
```

```
* READ vertices of the file "vertices.fl" into the vertices structure
  - count the number of vertices
  - initialize the path and distance to source for each vertex
```

```
ENDPROCEDURE
```

```
/* ***** STEP 0 *****
step 0 of the algorithm:
```

here the distances of all the other nodes. The source node in this algorithm is the destination point of the global path from the main program GPATH from the source node are calculated. If an edge links a node with the source node its distance is finite otherwise it is a big number */

```
PROCEDURE step0()
```

```
BEGIN:
```

```
* find the labels of the source and target nodes (resp. Is and It )
```

```
* mark the source node as visited
  (visited means here that the path from source to that node has been searched)
```

```
* now fill in the distances of other edges to the source node according to the
algorithm
```

```
- distance will be infinite if the node is not directly connected to the source by an
edge.
```

```
- distance will be equal to the measure of the edge connecting the node and source
point if they are connected by an edge.
```

```
* For each vertex of the graph set the source node as the start of the path from source to
that node
```

```
ENDPROCEDURE
```

```
/* ***** STEP 1 *****
PROCEDURE step1() /* find the optimum path node by node */
```

```
BEGIN
```

```
* set distance_min = 1e+6 ( this is the value to represent infinite )
```

```
* find ikmin the vertex with minimum distance to source point: min_value
```

```

* if (min_value == 1e+6) then
    // no path to remaining nodes of the graph
    // END OF COMPUTATIONS
    - print_optimum_path(); // print the computed optimum path in the "section.log" file
    - END DJIKSTRA
* else
    - for all the vertices connected to imin, copy the path from source to imin
      into their path
* if all the vertices have been visited then
    //END OF COMPUTATIONS
    - print_optimum_path(); // print the computed optimum path in the "section.log" file
    - END DJIKSTRA
else compute Step 1

ENDPROCEDURE

/* ***** STEP 2 ***** */
STEP 2 AND STEP 1 HAVE BEEN MERGED

PROCEDURE check_set() /* check if all the nodes of the graph have been visited */

BEGIN:
    * SCAN the list of vertices
    * IF any one is not visited
      THEN return 0
      ELSE return 1

ENDPROCEDURE

/* Results of the DJIKSTRA' s Algorithm: write the optimum path distance and
route in the communication file */

PROCEDURE print_paths()

FILE "section.log " , /* this is the communication file with the navigator */

BEGIN:
    * write the source and target nodes in the "section.log" file.
    * write the length of the path from source point to target point into "path.log" file
    * write the path from source node to destination (target) node as an ordered list of
      node names. The order is node and its following in the path.
    * close "section.log"
ENDPROCEDURE

```

B- 2 ALGORITHM FOR THE DRIVER PROGRAM

The purpose of this program is to drive the whole database. It is the main program that will be calling other routines. This program will be written in assembler(i3806) or TC++. This program receives requests from the Navigator

module of the wheelchair.

Program DBASEDR(.C)

BEGIN:

```

    . CALL INIT /* initialize I/O ports */
LOOP:
    . POLL port until status has changed
    . DISABLE port for writing
    . IF signal Compute_global_path is high /* request to compute global path */
      THEN . CALL GPATH /* compute optimum path: program GPATH.PRO */
        . set signal_Global_path_ready
    ELSE()
    ENDIF
    . IF signal Locate_me is high /* the request is to compute the current position */
      THEN . CALL LOCATE /* Identify the current location : program
LOCATE.PRO*/
        . set Position_found /* the result is in the file LOCATION.DAT */
    ELSE()
    ENDIF
    . IF signal Room_info_read is high /* provide information on a room. */
      THEN . CALL ROOMINFO /* Retrieve information on the given room:
        program ROOMINFO.PRO*/
        . set Room_info_ready

    . IF signal Room_info_update /* request to update the information on a room. */
      THEN . CALL ROOMUPD /* Retrieve information on the given room:
        program ROOMUPD.PRO*/
        . set Room_info_updated

    . IF signal What do I see is is high
      THEN . CALL WHATISEE /* scene analysis: program WHATISEE.PRO*/
        . set What_You_see_is
    ELSE()
    ENDIF
    . GOTO LOOP
END DBASEDR

```

B- 3 - ALGORITHM FOR THE MATCHING PROGRAM

Program MATCH.C

Computes the best correspondence matching between two images described as sets of straight lines.

INPUT: file CANDIDAT.DAT contains the list of candidate locations for the matching. This file is created in the program LOCATE used to find the current position of the wheelchair inside the building

OUTPUT: file LOCATION.DAT contains the name of the location successfully recognized
 1st line: status ---> 1 if a location has been recognized($\geq 65\%$)
 2 otherwise
 2nd line: name of the location (if matching successful)

BEGIN:

```

* Read the data of the sensor image (current image of the scene) into the data structures
* WHILE (not end of file CANDIDAT.DAT ) DO
    read the name of a candidate location from file
    read its image data file (template image) into the data structures
    CALL MATCH_IMAGES() /* match the current image and the template
                        image */
    IF value_of_matching  $\geq$  threshold_value
        THEN swap (threshold_value, value_of_matching)
    ELSE ()
ENDWHILE
IF (value_of_matching  $\geq 65\%$  ) /* a best match is acceptable if the hit ratio is greater
of equal to a threshold of 65% */
    THEN print the name of the location and the status "success" in the file
        LOCATION.DAT
ELSE
    print status "failure" in the file LOCATION.DAT
END MATCH.C

```

```

/* ----- */
Routine MATCH_IMAGES() /* match two images */

```

INPUT: the image data for both images
 OUTPUT: global_hit_ratio

BEGIN:

```

* read N , number of levels in image CI (current image)
* read M , number of levels in image TI (template image)
* I = 0; /* counter for the number of levels in image CI */
* WHILE ( I < N) DO
    best_hit(I) = 0; /* initialize the best hit ratio pointer */
    J = 0; /* counter for the number of levels in image TI */

    * WHILE ( J < N) DO
        score = match_levels(I,J) /* the score is the correspondence matching hit
ratio:each line in level I is matched to all the lines in level J
to determine a corresponding line if successful
score = 100 * matches/ (misses + matches) */
        IF (score < best_hit(I) ) THEN swap (score, best_hit(I))
        ELSE ()
    ENDWHILE
ENDWHILE
ENDWHILE
* global_hit_ratio = [  $\sum_I$  ( best_hit(I) *  $\omega(I)$  ) ] /  $\sum_I \omega(I)$ 

```


$\omega(I) = 0.1 * (8 - I)$ is the weight assigned to level I

* return (global_hit_ratio)
RETURN

B- 4 - ALGORITHM FOR THE LOCALIZATION PROGRAM

The purpose of this program is to find the current location of the wheelchair using information provided by vision image and sonar data as well as using artificial intelligence methods .This program will be written in Turbo Prolog. This program is called by the driver program of the database.

The input of the program is data about: the most recently visited location, the time out parameter, the bypass parameter. The time out indicates the time after which the program will be relinquished if the task is not completed. The bypass parameter is used to bypass the location mechanism. If bypass is set to 1, the program assumes that the current location is the same as the most recently visited location. Otherwise if it is 0, there will be a need to locate the robot.

INPUT:

LASTLOC(Place,Floor,Building) CULOC(L,F,B) Time_out Bypass

Program LOCATE(.PRO)

BEGIN:

 . IF *Bypass* != 1 /* initialize I/O ports */
 THEN . CALL *INIT*

 . start timer (initialized for the value of *Time_out*)

 . send signal *STOP* = 1 to the Navigator /* the navigator has to stop the wheelchair a bit to allow processing of information*/

 . send signal *CAMERA* = 1 to the the camera(s) to snap

 . read *LASTLOC* to identify the most probable current corridor, floor and building.

 . CALL *VISION (Time_out2) & SONAR*
 (set signal *Vision* =1 & *sonar* =1),

 LOOP:

 WHILE *Time_out* not expired DO

 IF *EndVision* != 1 & *Time_out2* expired

 THEN QUITLOOP

 Message (Fail)

 EXIT

 IF *EndVision* != 1 & *Time_out2* not expired /* ideal

```
case */
  THEN CALL EXTRACT /* Feature extractor */
    CALL MATCHING /* match the image
      with locations in the neighborhood */
    Message (Success)
    EXIT
  Check Time_out
ENDLOOP
ELSE CURLOC := LASTLOC
ENDIF
END LOCATE
```

SOURCE CODES OF PROGRAMS¹

APPENDIX C

C-1 THE DRIVER ROUTINE

```

/ * ****
* PROGRAM DBASEDR.C
* VERSION: TURBO C++
* FUNCTION: This program is the driver program for the database
*           It receives requests from the Navigator and sends
*           messages to the Vision sub-module.
* ****
port 0 is the input port
    bit 0: signal Compute_global_path; the request is to
            compute the global path for the wheelchair.
    bit 1: signal Locate_me; the request is to compute the current position
            of the wheelchair.
    bit 2: signal Room_info_read; provide information on a room.
    bit 3: signal Room_info_update; update information on a room.
    bit 4: signal What_do_I_see; describe the scene

port 1 is the output port
    bit 0: signal Global_path_ready;
    bit 1: signal Position_found; the result is in the file LOCATION.DAT
    bit 2: signal Room_info_ready
    bit 3: signal Room_info_updated
    bit 4: signal What_You_see_is; response to What_do_I_see.
*/
#include <stdio.h>
#include <dos.h>

main()
{
    int in_port = 0; // serial port 0
    int out_port = 0; // serial port 1
    unsigned message = 0x0000;

    while(){ // loop for ever
        outputb(1,0x0000) ; // reset output port to 0x0
        message = inport(); // read port 0 value i.e message from Navigator module
        if (( message & 0x0001) == 0x0001){ // bit 0 set
            system("GPATH"); // call Gpath - compute global path
            outputb(1,0x0001); // send signal path ready
        }
        if (( message & 0x0010) == 0x0010){ // bit 1 set
            system("LOCATE"); // call LOCATE - compute position
            outputb(1,0x0010); // send signal position computed
        }
        if (( message & 0x0011) == 0x0011){ // bit 2 set
            system("ROOMINFO"); // call ROOMINFO - find information on a room
            outputb(1,0x0011); // send signal Room_info_ready
        }
    }

    if (( message & 0x0100) == 0x0100){ // bit 3 set

```

```

        system("ROOMUPD"); // call ROOMUPD - update information on a room
        outputb(1,0x0100); // send signal Room_info_ready
    } if (( message & 0x0101) == 0x0101){ // bit 4 set
        system("SCENE"); // call SCENE - scene recognition
        outputb(1,0x0101) ; // signal What_you_see_is
    }
}
}
/* NOTE: toggling on and off may not work in a loop. The other processor may see
the signal off because it was busy at the time the signal was high; a solution
is to reset the output port to 0000 at the beginning of each cycle.
***** */

```

C-2 THE MAP MANAGER MAIN PROGRAM

```

; *****
; *   FILE           : MAP.SC           *
; *   VERSION        : 3.0             *
; *   DESIGNED BY    : Dieudonne Mayi  *
; *   Function       : Master's student in EE *
; *   Date           : June 1, 1991    *
; *   Project        : Design of a database for the PSUBOT *
; *   Supervisor     : M. Perkowski    *
; *   Institution    : PORTLAND STATE UNIVERSITY *
; *   Function to achieve : MANAGE MAP DATA *
; *   Language       : PAL (BORLAND PARADOX 3.0) *
; *****
;----- are in the procedure library map.lib -----
; Convert(table_in, table_out) , Quitmap(), Setprivedir(), Confirm()

AUTOLIB = "map" ; map library to be autoloaded

;----- This is the main program to enter the map. -----
PROC MapManager()
CANVAS OFF
CLEARALL CLEAR
PAINTCANVAS ATTRIBUTE 30 0,0,24,79
CANVAS ON
@1,30 ?? "WELCOME TO THE PSUBOT \"MAP MANAGER\" "
@2,30 ?? "*****"
@3,10
SETMARGIN 10
TEXT
*****
*           MENU           *
*****
* Tutorial           * Useful information about the program *
*****
* Enter map         * Enter new map *
*****

```

```

* Query map          * Ask a simple query on a map          *
*****
* Read map           * Retrieve information on a map          *
*****
* Update map         * Add/delete information on a map       *
*****
* Privatedir         * Set default directory (to do first !) *
*****
* ToDOS              * Go to DOS Shell to issue command      *
*****
* Quit               * Exit the map manager                  *
*****

```

```
ENDTEXTConfirm()
```

```
WHILE TRUE
```

```
  CLEARALL CLEAR
```

```
  SHOWMENU
```

```
  "Privatedir" : "Set working directory, DO IT FIRST !",
```

```
  "Tutorial"   : "Useful information about the program",
```

```
  "Enter map"  : "Enter new map data",
```

```
  "Query map"  : "Ask a simple query on a map",
```

```
  "Read map"   : "Read/report data on an existing map",
```

```
  "Update map" : "Update an existing map",
```

```
  "ToDOS"      : "Go to DOS Shell to issue DOS command",
```

```
  "Quit"       : "or Press Esc to quit the map manager "
```

```
; the conversion of data to prolog will
```

```
; be done automatically
```

```
TO Choice_menu
```

```
SWITCH
```

```
  CASE Choice_menu = "Enter map" :
```

```
    @10,20 ?? "Please wait ..."
```

```
    PLAY "enter"
```

```
  CASE Choice_menu = "Update map":
```

```
    @10,20 ?? "Please wait ..."
```

```
    PLAY "update"
```

```
  CASE Choice_menu = "Read map" :
```

```
    @10,20 ?? "Please wait ..."
```

```
    PLAY "read"
```

```
  CASE Choice_menu = "Query map" :
```

```
    @10,20 ?? "Please wait ..."
```

```
    PLAY "query"
```

```
  CASE Choice_menu = "Tutorial" :
```

```
    @10,20 ?? "Please wait ..."
```

```
    PLAY "tutorial"
```

```
  CASE Choice_menu = "Privatedir":
```

```
    Setprivedir()
```

```
  CASE Choice_menu = "ToDOS"   :
```

```
    @10,20 ?? "Please wait ..."
```

```
    DOSBIG
```

```
  CASE Choice_menu = "Quit"    :
```

```
    Quitmap()
```

```
    QUITLOOP
```

```
  CASE Choice_menu = "Esc"     :
```

```

Quitmap()
QUITLOOP
OTHERWISE : LOOP
ENDSWITCH
ENDWHILE
ENDPROC
@10,20 ?? "Please wait ..."
;----- main script -----
PLAY "library"
MapManager()
CLEARALL CLEAR
@10,20 ?? "Leaving the PSUBOT\'MAP MANAGER\' ..."
SLEEP 4000
BEEP BEEP
QUIT

```

C-3 THE GLOBAL PATH PLANNER PROGRAMS

- DATABASE (KNOWLEDGE BASE) DOMAINS DECLARATION AND UTILITY CLAUSES

```

/ *****
* PROGRAM UTIL.PRO *
* VERSION : BORLAND TURBOPROLOG 1.1 *
* Date : Sept 9, 1991 *
* Function : database domain declarations and utilities *
* routines *
*****/

```

DOMAINS

```

LISTOFREAL = REAL* /* A list of real numbers */
LISTOFSYMBOL = SYMBOL* /* A list of symbols */
LISTOFINTEGER = INTEGER* /* A list of integers */

```

```

obst = obst(REAL,REAL,REAL,REAL) /* An obstacle in a room */
LISTOFOBST = obst* /* A list of obstacles */
LOC = LOC(SYMBOL,SYMBOL,SYMBOL) /* A location in the building */
LISTOFLOC = LOC* /* A list of locations */

```

```

FILE = thefile; thelog; thepath;the_edges; the_vertices; thedat; the_djik

```

DATABASE

```

/* The database predicates for the items of the map: bridge, building, floor, corridor, room,
corrend, object. These predicates will maintain the information on the map. */

```

```

/* bridge(BRNAME,MNAME,BNAME1,EXIT1,BNAME2,EXIT2,LENGTH,TFREQ,
OBFREQ) */
bridge(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,REAL,

```

```

REAL,REAL)
/* building(BNAME,MNAME,NFLOORS,PBLC,ACFAC) */
building(SYMBOL,SYMBOL,INTEGER,SYMBOL,SYMBOL)

/* floor(FNAME,BNAME,FABOVE,ELVT) */
floor(SYMBOL,SYMBOL,SYMBOL,SYMBOL)

/* corridor(CNAME,FNAME,BNAME,TYPE,LENGTH,TFREQ,OBFFREQ) */
corridor(SYMBOL,SYMBOL,SYMBOL,INTEGER,REAL,REAL,REAL)

/* corrend(CNAME,FNAME,BNAME,END1NAME,THETA1,ORIENT1,END2NAME,
THETA2,ORIENT2) */
corrend(SYMBOL,SYMBOL,SYMBOL,SYMBOL,REAL,SYMBOL,SYMBOL,
REAL,SYMBOL)

/* object(OBNAME,CNAME,FNAME,BNAME,TYPE,SIDEON,ATDIST,CHAR1,
CHAR2, CHAR3,CHAR4,NEXTOB)*/
object(SYMBOL,SYMBOL,SYMBOL,SYMBOL,INTEGER,INTEGER,REAL,
INTEGER,INTEGER,INTEGER,INTEGER,SYMBOL)

/* room(RNAME,FNAME,BNAME,ENTDOOR,EXITDOOR,TYPE,LENGTH,WIDTH,
OBSTDISPO) */
room(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,INTEGER,REAL,
REAL,LISTOFOBST)

```

PREDICATES

```

moredata(FILE)
writeoblist(LISTOFOBST)
write_list(LISTOFSYMBOL)
member(SYMBOL,LISTOFSYMBOL)
append(LISTOFSYMBOL,LISTOFSYMBOL,LISTOFSYMBOL)

```

CLAUSES

```

/* read in-room information : [obst(x1,x2,x3,x4),obst(y1,y2,y3,y4),....] */
writeoblist([]):- write("obst(-1,-1,-1,-1))." /* obst(-1,-1,-1,-1) is a dummy obstacle just
for the sake of the program */
writeoblist([obst(X1,X2,X3,X4)|T]):-
write("obst(",X1,',',X2,',',X3,',',X4,',','),",writeoblist(T).

```

```

moredata(_).
moredata(File):-not(eof(File)),moredata(File).

```

```

/* DEALING WITH LISTS OF SYMBOLS
Similar logic for other types of lists */

```

```

write_list([]). /* writes a list of symbols separated by spaces*/
write_list([H|T]):- write(H," "),write_list(T).

```

```

/* membership to a list of symbols */

```



```
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```

```
/* append one list to another list */
append([],List,List).
append([X|L1],List2,[X|L3]):- append(L1,List2,L3).
```

- MAIN PROGRAM

```
*****
* PROGRAM GPATH.PRO *
* VERSION : BORLAND TURBOPROLOG 1.1 *
* DESIGNER : Dieudonne Mayi *
* Date : Oct. 28, 1991 *
* Project : Design of a database for a PSUBOT *
* Function : Compute global path for the PSUBOT *
* wheelchair *
*****
```

----- PERFORMANCES -----

This program answers requests of the kind: path([L1,F1,B1],[L2,F2,B2]).

L: location; F: floor ; B: building

The query grammar is composed of nine patterns of queries :

- (0i) basic primitive: [L1,F1,B1]---[L2,F2,B2]
 "Find path from location a L1 on a floor F1 of a building B1 to a location L2 on a floor F2 in a building B2."
- (1i) (1) [L1,F1,xx]---[L2,F2,xx]; (2) [xx,xx,xx]---[L2,F2,xx]
 Case(1): "Find path from a location L1 on a floor F1 to a location L2 on a floor F2 in this building."
 Case(2): "Find path from HERE to a location L2 of a floor F2 in this building."
- (2i) (1) [L1,xx,xx]---[L2,xx,xx]; (2) [xx,xx,xx]---[L2,xx,xx]
 Case(1): "Find path from a location HERE(L1,xx,xx) to a location L2 in this building."
 Case(2): "Find path from HERE to a location L2 in this building."
- (3i) [L1,xx,xx]---[xx,F2,xx]
 "Find a path from HERE(L1,xx,xx) to floor F2 in this building."
- (4i) (1) [L1,F1,xx]---[L2,xx,B2]; (2) [xx,xx,xx]---[L2,xx,B2]
 Case(1): "Find path from a location L1 on a floor F1 to a location L2 in a building B2."
 Case(2): "Find path from HERE to a location L2 in a building B2."
- (5i) (1) [L1,xx,B1]---[L2,xx,B2]; (2) [L1,xx,xx]---[L2,xx,B2]
 Case(1): "Find path from a location L1 in a building B1 to a location L2 in a building B2."
 Case(2): "Find path from a location L1 in this building to a location L2 in a building B2."
- (6i) (1) [L1,xx,B1]---[xx,xx,B2]; (2) [xx,xx,xx]---[xx,xx,B2]
 Case(1): "Find path from a location L1 in a building B1 to a building B2."
 Case(2): "Find path from HERE to a building B2."
- (7i) (1) [xx,xx,xx]---[xx,F2,B2]; (2) [xx,F1,xx]---[xx,F2,B2]
 Case(1): "Find path from HERE to floor F2 of the building B2."

Case(2): "Find path from floor F1 of this building to floor F2 of the building B2."

(8i) [xx,xx,xx]---[L,zz,zz]

"Find path from HERE to a location L on this map(building not indicated)."

Note: xx or zz means "not indicated". Each time the building of the starting location of the path is not indicated, the program will try to find it either by retrieving the name of the building from the information on that starting point or by LOCATING the current position.

----- */

NOWARNINGS

CODE = 2500

INCLUDE "UTIL.PRO" /* Utilities and database declarations */

PREDICATES

Gpath(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

path(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

Locate_me(SYMBOL,SYMBOL,SYMBOL)

path_on_bridge(SYMBOL,SYMBOL,SYMBOL,SYMBOL)

path_between_buildings(SYMBOL,SYMBOL,SYMBOL,SYMBOL)

path_in_building(SYMBOL,SYMBOL,SYMBOL)

path_on_floor(SYMBOL,SYMBOL,SYMBOL,SYMBOL)

path_between_floors(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

path_on_corridor(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

path_between_corridors(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

between(REAL,REAL,SYMBOL,SYMBOL,SYMBOL)

The_next(REAL,SYMBOL,REAL,SYMBOL,SYMBOL,SYMBOL)

exist_a_path_between(SYMBOL,SYMBOL,SYMBOL)

Elevator(SYMBOL,LISTOFSYMBOL) has_elevator(SYMBOL,SYMBOL)

route_from_djik(LISTOFSYMBOL)

belongs_to_cor(SYMBOL,SYMBOL,SYMBOL,SYMBOL,REAL)

Ends_of_a_corridor(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,REAL)

Ends_of_a_bridge(SYMBOL,SYMBOL,SYMBOL)

A_bridge(SYMBOL,SYMBOL,SYMBOL)

Refine_path(LISTOFSYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,
SYMBOL)

The_other_end(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

The_nearest_end_from(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

GRAPH(SYMBOL,SYMBOL)

write_vertex(LISTOFSYMBOL)

write_list1(LISTOFSYMBOL,SYMBOL,SYMBOL)

substract(REAL,REAL,REAL)

Continue_job(SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL,SYMBOL)

run

GOAL run.

CLAUSES

```
run:- openread(thedat,"path.dat"),readdevice(thedat),readln(Map_name),
      readterm(loc,loc(L1,F1,B1)), readterm(loc,loc(L2,F2,B2)), closeFile(thedat),
      concat(Map_name,".dba",MapDatabase), existfile(MapDatabase),
      Continue_job(MapDatabase,L1,F1,B1,L2,F2,B2),!.
```

```
/* The structure of "path.dat" the communication file from Navigator
First record line: name of the map site
Second record line: loc(L1,F1,B1) loc(L2,F2,B2) - starting and ending points
of the global path to compute. */
```

```
run:- beep,closeFile(thelog),openwrite(thelog,"path.log"),writedevicethelog,
      write(3),nl,write("A: ERROR map database not found!"), nl,closeFile(thelog).
```

```
Continue_job(Mbase,L1,F1,B1,L2,F2,B2):-
  consult(Mbase). /* load database in RAM */
Continue_job(Mbase,L1,F1,B1,L2,F2,B2):-
  openwrite(thelog,"path.log"),writedevicethelog,
  Gpath(L1,F1,B1,L2,F2,B2)./*continue: compute path */
```

```
/* ----- MAIN ROUTINE : Gpath -----
```

This routine answers the nine types of queries mentioned in the performances of the program. This routine takes the parameters on the starting and destination points of the desired path and calls the primitive "path" which carries on the computation of the path.

```
case (1i) */
Gpath(L1,F1,"xx",L2,F2,"xx"):- /* sub-case (1) */
  object(L1,_,F1,B,_,_,_,_,_,_),object(L2,_,F2,B,_,_,_,_,_,_),
  path(L1,F1,B,L2,F2,B).
```

```
Gpath("xx","xx","xx",L2,F2,"xx"):- /* sub-case (2) */
  Locate_me(L1,F1,B),object(L2,_,F2,B,_,_,_,_,_,_),
  path(L1,F1,B,L2,F2,B).
```

```
/* case (2i) */
Gpath(L1,"xx","xx",L2,"xx","xx"):- /* sub-case (1) */
  object(L1,_,F1,B,_,_,_,_,_,_),object(L2,_,F2,B,_,_,_,_,_,_),
  path(L1,F1,B,L2,F2,B).
```

```
Gpath("xx","xx","xx",L2,"xx","xx"):- /* sub-case (2) */
  Locate_me(L1,F1,B),object(L2,_,F2,B,_,_,_,_,_,_),
  path(L1,F1,B,L2,F2,B).
```

```
/* case (3i) */
Gpath(L1,"xx","xx","xx",F2,"xx"):-
  object(L1,_,F1,B,_,_,_,_,_,_),
  object(Elvt,_,F2,B,3,_,_,_,_,_,_)/* find a point at an elevator */
  path(L1,F1,B,Elvt,F2,B).
```

```

/* case (4i) */
Gpath(L1,F1,"xx",L2,"xx",B2):- /* sub-case (1) */
    object(L1,_F1,B1,_,_,_,_,_,_,_),object(L2,_F2,B2,_,_,_,_,_,_),
    path(L1,F1,B1,L2,F2,B2).
Gpath("xx","xx","xx",L2,"xx",B2):- /* sub-case (2) */
    Locate_me(L1,F1,B1),object(L2,_F2,B2,_,_,_,_,_,_),
    path(L1,F1,B1,L2,F2,B2).

/* case (5i) */
Gpath(L1,"xx",B1,L2,"xx",B2):- /* sub-case (1) */
    object(L1,_F1,B1,_,_,_,_,_,_,_),object(L2,_F2,B2,_,_,_,_,_,_),
    path(L1,F1,B1,L2,F2,B2).
Gpath(L1,"xx","xx",L2,"xx",B2):- /* sub-case (2) */
    object(L1,_F1,B1,_,_,_,_,_,_,_),object(L2,_F2,B2,_,_,_,_,_,_),
    path(L1,F1,B1,L2,F2,B2).

/* case (6i) */
Gpath(L1,"xx",B1,"xx","xx",B2):- /* sub-case (1) */
    object(L1,_F1,B1,_,_,_,_,_,_,_),
    A_bridge(B1,B2,Br),!,Ends_of_a_bridge(X1,Y1,Br),
    object(Y1,_F2,B2,_,_,_,_,_,_,_),path(L1,F1,B1,Y1,F2,B2).
Gpath("xx","xx","xx","xx","xx",B2):- /* sub-case (2) */
    Locate_me(L1,F1,B1), A_bridge(B1,B2,Br),!,
    Ends_of_a_bridge(X1,Y1,Br),
    object(Y1,_F2,B2,_,_,_,_,_,_,_),path(L1,F1,B1,Y1,F2,B2).

/* case (7i) */
Gpath("xx","xx","xx","xx",F2,B2):- /* sub-case (1) */
    Locate_me(L1,F1,B1),object(L1,_F1,B1,_,_,_,_,_,_),
    object(Elvt,_F2,B2,3,_,_,_,_,_,_)/* find a point at an elevator */
    path(L1,F1,B,Elvt,F2,B2).
Gpath("xx",F1,"xx","xx",F2,B2):- /* sub-case (2) */
    object(Elvt1,_F1,B1,3,_,_,_,_,_,_)/*find a point at an elevator */
    object(Elvt2,_F2,B2,3,_,_,_,_,_,_)/*find a point at an elevator */
    path(Elvt1,F1,B1,Elvt2,F2,B2).

/* case (8i) */
Gpath("xx","xx","xx",L,"zz","zz"):- /* sub-case (1) */
    Locate_me(L1,F1,B1),object(L1,_F1,B1,_,_,_,_,_,_),
    object(L,_F2,B2,_,_,_,_,_,_),!/* find any object named L */
    path(L1,F1,B1,L,F2,B2).

/* case (0i) */
Gpath(L1,F1,B1,L2,F2,B2):- path(L1,F1,B1,L2,F2,B2).

```

/* ---PRIMITIVE path(L1,F1,B1,L2,F2,B2)

This is the base query to compute the path between two locations of the map. The locations have been clearly identified(parameters) by the caller routine Gpath(). The base query path() computes the path using the hierarchical approach. Two main cases are distinguished: "path_in_building()" and "path_between_buildings()". The first one "path_in_building()" is called when the two locations are inside the same building and "path_between_buildings()" if not. L1,F1,B1 --> starting location

parameters L2,F2,B2 --> destination location parameters

```
* ----- *
Level (1) : Top level of the algorithm - check if the points belong
to the same building.
* ----- */
```

```
path(S,F1,B1,D,F2,B2):- /* S and D are in different buildings */
    bound(B1),bound(B2),
    B1 <> B2,path_between_buildings(S,B1,D,B2),nl,closeFile(thelog).
path(S,F1,B1,D,F2,B2):- /* S and D are in the same building */
    object(S,_,F1,B1,_,_,_,_,_,_,_),
    object(D,_,F2,B2,_,_,_,_,_,_,_),
    path_in_building(S,D,B1),nl,closeFile(thelog). /* B1 = B2 */
```

```
path(S,F1,B1,D,F2,B2):-
    bound(B1),
    beep,beep,nl,write(3),nl,write("A: Error or Data missing !"),
    closeFile(thelog).
```

```
/* ----- *
Level (2) : Path between points located in different buildings
* ----- */
```

```
path_between_buildings(X,B1,Y,B2):-
    Ends_of_a_bridge(X1,B1,Y1,B2,Br,_),nl,
    nl,path_in_building(X,X1,B1),nl,path_on_bridge(X1,Y1,B1,B2),nl,
    path_in_building(Y1,Y,B2).
```

```
path_between_buildings(X,B1,Y,B2):-
    beep,beep,write(3),nl,
    write("A: There is not a path connecting the buildings <<"),
    write(B1,">> and <<","B2,">>").
```

```
path_on_bridge(S,D,B1,B2):-/* S and D are the end-points of the same bridge */
    Ends_of_a_bridge(S,B1,D,B2,BR,L),object(S,_,F1,B1,_,_,_,_,_,_,_),
    object(D,_,F2,B2,_,_,_,_,_,_,_),nl,write(1," ",L),nl,
    write(S," ",F1," ",B1),nl,
    write(D," ",F2," ",B2 ),nl,
    write("***" " " "***" " " "***"),nl.
/* end of the prtion of path */
```

```
/* ----- *
Level (2) : Path between two points located in the same building
* ----- */
```

```
/* case (i): S and D are in same building but on different floors */
path_in_building(S,D,B):-
    object(S,C1,F1,B,_,_,_,_,_,_,_),object(D,C2,F2,B,_,_,_,_,_,_),
    F1<>F2, path_between_floors(S,F1,D,F2,B).
```

```
/* case(ii): S and D are on same floor */
```

```
path_in_building(S,D,B):-
    object(S,C1,F,B,_,_,_,_,_,_,_),object(D,C2,F,B,_,_,_,_,_,_),
    path_on_floor(S,D,F,B).
```

```
/* ----- *
   Level (3) : Path between two points located on different floors
  * ----- */
```

```
path_between_floors(X,F1,Y,F2,B):-
    /* not possible if no elevator connecting them */
    not(exist_a_path_between(F1,F2,B)),beep,beep,nl,write(3),nl,
    write("A: No mean to go from the present floor <<"F1,">> to floor <<"",
    write(F2,">> ").
```

```
path_between_floors(X,F1,Y,F2,B):-
    belongs_to_cor(X,C1,F1,B,_),belongs_to_cor(Y,C2,F2,B,_),
    /* ElvF1 is elevator on floor F1 */
    object(ElvF1,CE1,F1,B,3,_,_,_,_,_,_),
    /* ElvF2 is elevator on floor F2 */
    object(ElvF2,CE2,F2,B,3,_,_,_,_,_,_),
    path_on_floor(X,ElvF1,F1,B),nl, path_on_floor(ElvF2,Y,F2,B).
```

```
/* ----- *
   Level (3) : Path between two points located on the same floor
  * ----- */
```

```
/* case (ii) : S and D are located on the same hallway */
path_on_floor(S,D,F,B):- /* are end-points of the same corridor */
    Ends_of_a_corridor(S,D,C,F,B),path_on_corridor(S,D,C,F,B),!
```

```
path_on_floor(S,D,F,B):-
    belongs_to_cor(S,C1,F,B,_),belongs_to_cor(D,C2,F,B,_),C1 = C2,
    path_on_corridor(S,D,C1,F,B),!
```

```
/* case (i) : S and D are located on different hallways */
```

```
path_on_floor(S,D,F,B):-
    belongs_to_cor(S,C1,F,B,_),belongs_to_cor(D,C2,F,B,_),
    C1 <> C2, path_between_corridors(S,C1,D,C2,F,B).
```

```
/* ----- *
   Level (4) : computing the path
  * ----- */
```

```
/* case (i): compute path on same corridor by simply listing all the
   locations between S and D */
```

```
path_on_corridor(X,Y,C,F,B):-
    corrend(C,F,B,X,_,_,Y,_,_),corridor(C,F,B,_,L,_,_),
    nl,write(2," ",L),nl,write(X," ",F," ",B),nl,
    The_next(0.0,X,L,C,F,B),write("***", " ", "***", " ", "***"),nl.
```

```
path_on_corridor(X,Y,C,F,B):-
    corrend(C,F,B,Y,_,_,X,_,_),corridor(C,F,B,_,L,_,_),
```

```

nl,write(2," ",L),nl,write(X," ",F," ",B),nl,nl,
The_next(L,X,0.0,C,F,B),write("***", " ", "***", " ", "***"),nl.
/* X and Y are not both end-points of the same corridor */
path_on_corridor(X,Y,C,F,B):-
  belongs_to_cor(X,C,F,B,Dist1),belongs_to_cor(Y,C,F,B,Dist2),
  subtract(Dist1,Dist2,L),nl,write(2," ",L),nl,write(X," ",F," ",B),nl,
  The_next(Dist1,X,Dist2,C,F,B),write("***", " ", "***", " ", "***"),nl.

/* case (ii): compute path between two corridors by applying the Djisktra's
algorithm to find the path between the end-points of the two corridors.
knowledge-based inference is applied to eliminate the locations on the starting
and ending corridors which are not on the physical path. The Djisktra's algorithm
is implemented in BORLAND TURBO C++ */

path_between_corridors(S,C1,D,C2,F,B):-
  /* Path between corridors C1 C2 using the Djisktra's algorithm */
  The_nearest_end_from(S,E1,C1,F,B),
  The_nearest_end_from(D,E2,C2,F,B),
  openwrite(the_djik,"section.dat"),
  writedevicethe_djik,write(E1," ",E2),nl,closeFile(the_djik),
  /*retrieve graph then apply DJIK */
  GRAPH(F,B),writedevicethe_log,
  system("DJIK"), path_on_corridor(S,E1,C1,F,B),
  route_from_djik(Roughpath,L),The_other_end(Ex,E2,C2,F,B),
  Refine_path(Roughpath,L,Ex,E2,D,C2,F,B).

/* ***** UTILITIES ***** */

/* find the end of the corridor nearest to the point X */
The_nearest_end_from(X,E1,C,F,B):-
  belongs_to_cor(X,C,F,B,AtDist), corrend(C,F,B,E1,_,_,E2,_,_),
  corridor(C,F,B,_,L,_,_),AtDist < L/2,!

The_nearest_end_from(X,E2,C,F,B):-
  belongs_to_cor(X,C,F,B,AtDist), corrend(C,F,B,E1,_,_,E2,_,_),
  corridor(C,F,B,_,L,_,_).

/* find the other end Ex of a corridor C2 knowing one of them E2 */
The_other_end(Ex,E2,C2,F,B):-
  corrend(C2,F,B,Ex,_,_,E2,_,_),!.

The_other_end(Ex,E2,C2,F,B):-
  corrend(C2,F,B,E2,_,_,Ex,_,_).

Refine_path(Roughpath,L,Ex,E2,D,C2,F,B):-
  member(Ex,Roughpath),append(Path1,[E2],Roughpath),
  write(1," ",L),nl,write_list1(Path1,F,B),Path_on_corridor(Ex,D,C2,F,B).

Refine_path(Roughpath,L,Ex,E2,D,C2,F,B):-
  write(1," ",L),nl,write_list1(Roughpath,F,B),Path_on_corridor(E2,D,C2,F,B).

/* - READ the path computed with the DJIKSTRA'S algorithm in TC++ -- */
route_from_djik(Route,L):-

```

```

    openread(thepath,"section.log"),
    readdevice(thepath),readterm(LISTOFSYMBOL,Route),
    readreal(L),closeFile(thepath).
route_from_djik([]):-
    closeFile(thepath),openread(thepath,"section.log"),readdevice(thepath),
    readln(RAW),closeFile(thepath),frontstr(5,RAW,RAW1,RAW2),!,
    RAW1 = "ERROR",nl,write(3),nl,
    write("A:ERROR, could not compute the path\n"), closeFile(thelog),exit.

```

```

/* ----- Belonging of one point to a corridor -----

```

```

(i) the point is an end-point to that corridor

```

```

(ii) the point is an object on that corridor

```

```

The distance to the first end-point is AtDist */

```

```

belongs_to_cor(X,C,F,B,AtDist):- /* case (i) */
    object(X,C,F,B,_,_,AtDist,_,_,_,_),AtDist <>0.
belongs_to_cor(X,C,F,B,0):- /* case (ii) first end-point */
    corrend(C,F,B,X,_,_,_,_).
belongs_to_cor(X,C,F,B,L):- /* case (ii) second end-point */
    corrend(C,F,B,_,_,_,X,_,_),corridor(C,F,B,_,L,_,_).

```

```

/* Test existence of a path between two floors Exists path between floors F1 and
F2 of building B possible path between two floors if they both have an elevator
*/

```

```

exist_a_path_between(F1,F2,B):-
    floor(F1,B,_,_),floor(F2,B,_,_),Elevator(B,L), member(F1,L),
    member(F2,L).

```

```

Elevator(B,List_of_floors):- /* finds the list of floors having an elevator facility */
    findall(F1,has_elevator(F1,B),List_of_floors).

```

```

/* Note: this may appear not complete but it is because usually in a building two
elevators connect the sme floors. Wings of different levels in a physical building
are considered as separate buildings */

```

```

has_elevator(F1,B):- /* there is a working elevator on the floor F1 of building B */
    floor(F1,B,_, "y").
has_elevator(F1,B):-
    floor(F1,B,_, "Y").

```

```

/* ----- FINDING END-POINTS OF a corridor or a bridge ----- */

```

```

Ends_of_a_corridor(E1,E2,C,F,B):- /* endpoints of a corridor */

```

```

    corrend(C,F,B,E2,_,_,E1,_,_),!.

```

```

Ends_of_a_corridor(E1,E2,C,F,B):-

```

```

    corrend(C,F,B,E1,_,_,E2,_,_),!.

```

```

Ends_of_a_bridge(X,B1,Y,B2,Brname,Length):-

```

```

/* endpoints of a bridge between two buildings */

```

```

    bridge(Brname,Mname,B1,X,B2,Y,Length,_,_).

```

```

Ends_of_a_bridge(X,B1,Y,B2,Brname,Length):-

```

```

    bridge(Brname,Mname,B2,Y,B1,X,Length,_,_).

```



```

A_bridge(B1,B2,Brname):- /* identifying a bridge */
    bridge(Brname,Mname,B1,_,B2,_,_,_).
A_bridge(B1,B2,Brname):-
    bridge(Brname,Mname,B2,_,B1,_,_,_).

/* ----- RETRIEVE the list of locations within distance interval [Dx,Dy] */
The_next(Dx,Here,Dy,C,F,B):- /* ascending order of coordinate */
    Dx < Dy,belongs_to_cor(Here,C,F,B,Dh),Dh < Dy,
    belongs_to_cor(Z,C,F,B,Dz), Dh < Dz,not(between(Dz,Dh,C,F,B)),
    write(Z," ",F," ",B),nl, The_next(Dz,Z,Dy,C,F,B).

The_next(Dx,Here,Dy,C,F,B):-
    Dx < Dy,belongs_to_cor(Here,C,F,B,Dh),Dh < Dy,
    belongs_to_cor(Z,C,F,B,Dz), Dh = Dz,write(Here," ",F," ",B),nl,
    write(Z," ",F," ",B),nl, The_next(Dz,Z,Dy,C,F,B).
The_next(Dx,Here,Dy,C,F,B):- /* descending order of coordinate */
    Dx > Dy,belongs_to_cor(Here,C,F,B,Dh),Dh > Dy,
    belongs_to_cor(Z,C,F,B,Dz),Dh > Dz,not(between(Dh,Dz,C,F,B)), write(Z,"
",F," ",B),nl, The_next(Dz,Z,Dy,C,F,B).

The_next(Dx,Here,Dy,C,F,B):-
    Dx > Dy,belongs_to_cor(Here,C,F,B,Dh),Dh > Dy,
    belongs_to_cor(Z,C,F,B,Dz),Dh = Dz, write(Here," ",F," ",B),nl,
    write(Z," ",F," ",B),nl, The_next(Dz,Z,Dy,C,F,B).

The_next(Dx,Here,Dy,C,F,B):- write(" "). /* else */

/* Finds if there is any object in the distance interval [D1,D2] */
between(D1,D2,C,F,B):-
    object(Ob,C,F,B,_,_,D,_,_,_,_),D1 < D, D < D2,!.
between(D1,D2,C,F,B):-
    object(Ob,C,F,B,_,_,D,_,_,_,_),D2 < D, D < D1.

/* this predicate is used to locate the wheelchair. It calls an extrenal
program LOCATE. This program finds the current position of the robot in the
building. The program reads the name of the last location visited from the file
"PATH.HIS" then uses it as a heuristic in the search. If the current location has
not been found, then it is assumed that the robot has not moved at all from the last
position. The current position found for the robot is in the file LOCATION.DAT.
The location is given as loc(L,F,B) example: loc("i1","f1","pcat")
*/
Locate_me(L,F,B):-
    system("LOCATE"),openread(thedat,"location.dat"),readdevice(thedat),
    readterm(loc,loc(L,F,B)),closeFile(thedat).

/* routine to retrieve the graph for a floor */
GRAPH(F,B):-
    openwrite(the_edges,"edges.fl"), openwrite(thefile,"temp.fl"),
    writedevicethefile,write('['), corridor(C,F,B,_,L,Tf,Obf),
    corrend(C,F,B,E1,_,E2,_,_), writedevicethedges,
    write(E1," ",E2," ",L," ",Tf," ",Obf),nl,

```

```

        writedevice(thefile),write('"'',E1,'"',';','"',E2,'"',';'),fail.
GRAPH(F,B):- writedevice(the_edges),
write("***", " ", "***", " ", 0.0, " ", 0.0, " ", 0.0), closeFile(the_edges),
writedevice(thefile),write('"'', "***", '"', ']'), closeFile(thefile),
openread(thefile, "temp.fl"), readdevice(thefile),
readterm(LISTOFSYMBOL, Liste),
openwrite(the_vertices, "vertices.fl"),
writedevice(the_vertices), write_vertex(Liste).

write_vertex([]):- /* write the vertex graph in the files "vertices.fl" and "edges.fl" */
write("***"), closefile(the_vertices), closeFile(thefile), deletefile("temp.fl").
write_vertex([X|T]):-
not(member(X, T)), X <> "***", write(X, " "), write_vertex(T).
write_vertex([X|T]):-
write_vertex(T).
/* write the computed path (from Dijkstra's algorithm into PATH.LOG file */
write_list1([], F, B).
write_list1([_], F, B).
write_list1([H1|T], F, B):-
belongs_to_cor(H1, C, F, B, _), belongs_to_cor(H2, C, F, B, _),
member(H2, T), path_on_corridor(H1, H2, C, F, B), nl, write_list1(T, F, B).

```

- Implementation of the DJIKSTRA's shortest path algorithm

```

/ *****
* PROGRAM DJIK.C *
* VERSION: TURBO C++ *
* Function to achieve: Find an optimum path between a given *
* point of the floor to another point of the floor. The path algorithm *
* implemented is the Dijkstra's shortest path algorithm *
*****/

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>

// an edge of the graph
typedef struct {
    char vertex1[10]; // end1 of edge
    char vertex2[10]; // end2 of edge
    float length; // length of edge
    float obfreq; // obstacle frequency
    float tfreq; // travel frequency
} an_edge;
// a vertex of the graph
typedef
struct {
    char vertex_name[10];
    short int number; // has been added to identify vertex at step 1
    float dist_source; // distance to start point

```

```

    short visited ; // 1 if has been visited 0 if not
    short int pathto[20]; // path from here to source node
} a_vertex;

an_edge edge[100];
a_vertex vertex[200];
short int number_of_edges = 0 ;
short int number_of_vertices = 0 ;
short int is, it; // the label of the source and target nodes respectively
char source[10], target[10]; // source node and target node names

FILE *pathfile,*pathdat;
// "section.dat" and "section.log" are the the communication files
// with the program GPATH
// structure of "section.log" file:
// 1rst line: computed path as a list of locations exple ["i1","i2"]
// 2nd line: total length of the path

void main()
{
    void Init(), step0(), step1(); void
    print_optimum_path();

    pathfile = fopen("section.log","w");//file: holds path back to the Navigator
    pathdat = fopen("section.dat","r");// file: source target of path

    fscanf(pathdat,"%s%s",source,target);
    fclose(pathdat);
    Init(); // read data from file into structure
    step0(); // step 0 of the method
    step1(); // step 1 of the method
}

// INIT: read data from files into structures and variables
void Init()
{
    short int i = 0 ;
    float x1 = 0.0, x2 = 0.0, x3 = 0.0;
    char x[10], y[10];

    FILE *the_edges,*the_vertices;
// (the_edges,"edges.fl"): data file for edges of the graph of the floor
// (the_vertices,"vertices.fl"): data file for vertices of the graph of the floor

// check that the graph files exist; exit if one at least is missing
if((the_edges = fopen("edges.fl","r")) == NULL ||
    (the_vertices = fopen("vertices.fl","r")) == NULL){
    fprintf(pathfile,"ERROR, data file missing");
    fclose(pathfile);fclose(the_edges);fclose(the_vertices);exit(0);
}

the_edges = fopen("edges.fl","r");
the_vertices = fopen("vertices.fl","r");

```

```

// read edges of the graph from the data file
while ( feof(the_edges) == 0 ) {
    fscanf(the_edges,"%s %s %f %f %f",x,y,&x1,&x2,&x3);
    edge[i].length = x1;
    edge[i].obfreq = x2;
    edge[i].tfreq = x3;
    strcpy(edge[i].vertex1,x);
    strcpy(edge[i].vertex2,y);
    if (strncmp(x,"**",2) == 0){ break ; }
    else { ++i; }
}
number_of_edges = i ;

// read vertices of the graph
i = 0;
while ( feof(the_vertices) == 0 ) {
    fscanf(the_vertices, "%s", vertex[i].vertex_name);
    vertex[i].number = i;
    vertex[i].dist_source = 1e+6;
    vertex[i].visited = 0;
    vertex[i].pathto[0] = -1;
    vertex[i].pathto[1] = -1;
    if (strncmp(vertex[i].vertex_name,"**",2) == 0){ break ; }
    else { ++i; }
}
number_of_vertices = i ;
fclose(the_edges); fclose(the_vertices);
}
// ***** STEP 0 *****
// step 0 of the algorithm : here the distances of all the other nodes
// from the source node are calculated. If an edge links a node with
// the source node its distance is finite otherwise it is a big number.

void step0()
{ short int i = 0,j,found = 0;
  // find source and target nodes indexes
  for (i = 0; i < number_of_vertices ; ++ i){
    if (strcmp(vertex[i].vertex_name, source) == 0 ) {
        is = vertex[i].number; // the label of the source node
    }
    if (strcmp(vertex[i].vertex_name, target) == 0 ) {
        it = vertex[i].number; // the label of the target node
    }
  }
}

// the source point is considered to have been visited now
vertex[is].visited = 1;
vertex[is].dist_source = 0; // now fill in the distances of other edges to the source node

for (j = 0; j < number_of_vertices; ++j){
// for each vertex insert the source node as the start of

```

```

//the path to that vertex.
    vertex[j].pathto[0] = 2; // the beginning of the path list
    vertex[j].pathto[1] = is; // beginning of the path
    vertex[j].pathto[2] = -1; // end of the path is -1

if ( vertex[is].number == vertex[j].number ) {;}
else {
    i = 0;
    found = 0;
    while (i < number_of_edges && found != 1){
        if(((strcmp(vertex[is].vertex_name,edge[i].vertex1) == 0) &&
            (strcmp(vertex[j].vertex_name,edge[i].vertex2) == 0 )) ||
            ((strcmp(vertex[is].vertex_name,edge[i].vertex2) == 0) &&
            (strcmp(vertex[j].vertex_name,edge[i].vertex1) == 0 ))) {
            // source vertex and vertex j belong to edge i
            vertex[j].dist_source = edge[i].length;
            found = 1;
        }
        ++i;
    }
}
}
}

// ***** STEP 1 *****
void step1()
{
    short int imin, i, k, ptpt, j, test = 0;
    float minvalue = 1e+6, Old_distance ; void
    print_optimum_path(); short int check_set();

// find the vertex with minimum distance
for(i = 0; i < number_of_vertices ; ++i) {
    if ( vertex[i].visited == 1) {;}
    else {
        if (vertex[i].dist_source <= minvalue){
            imin = i;
            minvalue = vertex[i].dist_source;
        }
    }
}

if (minvalue == 1e+6){
// no path to remaining nodes of the graph
// END OF COMPUTATIONS
    print_optimum_path(); // print the computed optimum path in the
                        // "section.log" file
    exit(0);
}
else {
    vertex[imin].visited = 1;
    ptpt = vertex[imin].pathto[0]; // path tail pointer
}
}

```

```

vertex[imin].pathto[ptpt] = imin;
vertex[imin].pathto[ptpt + 1] = -1;
vertex[imin].pathto[0] = ptpt + 1 ;
// for all the vertices connected to imin, copy the path from source to imin
// into their path
for (j = 0; j < number_of_vertices; ++j){
    if ( vertex[j].visited == 1){;}
    else {
        i = 0;test = 0;

        while (i < number_of_edges && test != 1 ){
            if (((strcmp(vertex[imin].vertex_name,edge[i].vertex1) == 0) &&
                (strcmp(vertex[j].vertex_name,edge[i].vertex2) == 0 )) ||
                ((strcmp(vertex[imin].vertex_name,edge[i].vertex2) == 0) &&
                (strcmp(vertex[j].vertex_name,edge[i].vertex1) == 0 ))) {
                test = 1; // vertex j and vertex ik belong to edge i
                //(i) update distance to source
                Old_distance = vertex[j].dist_source;
                vertex[j].dist_source = min(Old_distance,minvalue + edge[i].length);
                // if the following test is not made the retrieved path will be incorrect
                // although the distance to the source point will be correct
                if (Old_distance != vertex[j].dist_source){
                    //(ii) copy path of ik into their path
                    for (k = 0; k < 20; ++k){
                        vertex[j].pathto[k] = vertex[imin].pathto[k];
                    }
                }
            }
            ++i;
        }
    }
}
if (check_set() == 1){
// all the nodes have been visited
// END OF COMPUTATIONS
    print_optimum_path(); // print the computed optimum path in the "section.log" file
    exit(1);
}
else { step1(); }
}

// check if all the vertices have been covered
// if yes this function will return 1 else it will return 0

short int check_set()
{ short int test = 1, i;

    for (i = 0; i < number_of_vertices ; ++i ) {
        if (vertex[i].visited == 0){
            return 0;
        }
    }
}

```

```

        else { test = test * 1 ; }
    }
    return(test);
}

// Results of the DJKSTRA' s Algorithm: write the optimum path
// distance and route in the communication file "section.log"

void print_optimum_path()
{
    short int i ,k;

    // write the length of the path from source point to target point
    // write path from source node to destination(target) node

    fprintf(pathfile, "%c%c%s%c", '[', "", vertex[vertex[it].pathto[1]].vertex_name, "");
    for (k = 2; vertex[it].pathto[k] != -1; ++k){
        fprintf(pathfile, "%c%c%s%c", ',', "", vertex[vertex[it].pathto[k]].vertex_name, "");
    }
    fprintf(pathfile, "%c", ']');
    // write the length of the path from source point to target point
    fprintf(pathfile, "\n%.4f\n", vertex[it].dist_source);
    fclose(pathfile);
}

```

C-4 THE MATCHING PROGRAM

```

/*****
* match.c
*
* Matching of two images. One of the two images represent the current
* image captured by sensors(cameras). The other one is a template image
* taken and stored at the learning statge.
* Each image is described as a set of lines. The image is described as
* a hierarchy with many levels, up to seven(maximum).
* Line segments are extracted using the Hierarchical Hough Transform *
* CALLED BY: LOCATE (program used to locate the robot by matching the
* current image to a sequence of images in the neighborhood
*
* INPUT : current image file "CURRENT.IM", and a file containing the names of
* the locations candidate of the matching "CANDIDAT.DAT"
* OUTPUT: a location in file "LOCATION.DAT"
*****/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define epsi_theta 0.26 /* 15 degrees */

/* data structure for global information on an image described

```

```

as a hierarchy */

typedef struct {
    float Crho; /* rho conversion factor for the hierarchy */
    float Ctheta; /* theta conversion factor for the hierarchy */
    int numlevs; /* number of levels */
    int Maxlen; /* length of the longest line in the hierarchy */
    int Size; /* size of the image */
    int numT; /* total number of lines in the hierarchy */
    int numH; /* total number of horizontal lines in the hierarchy */
    int numV; /* total number of vertical lines in the hierarchy */
    int numN; /* total number of slanting lines in the hierarchy */
}a_pyramid;

/* data structure for global information on level of the image
hierarchy */
typedef struct {
    int levnum; /* level number */
    int Maxlen; /* length of the longest line of the level */
    int numT; /* total number of lines of the level */
    int numH; /* total number of horizontal lines in the level */
    int numV; /* total number of vertical lines in the level */
    int numN; /* total number of slanting lines in the level */
    float score; /* score of the matching(sensor image) */
}a_level;

/*data structure for a line of the image*/
typedef struct {
    char Type[2]; /* type of the line(N,V,H) */
    int levnum; /* level number the line belongs to */
    float ys ; /* y-coordinate of the center of the line */
    float xs ; /* x-coordinate of the center of the line */
    float rhos; /* rho : length of the normal vector of the line */
    float thetas; /*theta : angle of the line with the x-axis */
    float length; /* length of the line */
    float yw; /* length of the smallest rectangle containing the line */
    float xw; /* width of the smallest rectangle containing the line */
}a_line;

/* "current"(resp. "template"): current image hierarchy(resp. template image
hierarchy). "curlevs[7]"(resp. "templevs[7]"): table to keep the information on the levels
of the current(resp. template) image hierarchy.
" curlines[200]"(resp. "templines[200]"): tables to keep information on the lines of the
images.
*/
a_pyramid current, template;
a_level curlevs[7],templevs[7];
a_line curlines[200],templines[200];

void main()
{
    FILE *locations, *place;

```



```

char location[10], the_most_probable[10];
void READ_CURRENT_IMAGE(),READ_TEMPLATE_IMAGE();
float MATCH_IMAGES(),threshold = 0.0, its_score,match_val;

/* input file contains the list of candidate locations
   for the matching. This file is created in the program LOCATE used to
   find the current position of the robot inside the building */

locations = fopen("candidat.dat","r");

/* output file contains the name of the location recognized
   1rst line: status ---> 1 if a location has been recognized(>= 65%)
   2 if not 2nd line: location if matching successful
*/
place = fopen("location.dat","w");

READ_CURRENT_IMAGE(); /* read sensor image in memory */
while ( !feof(locations)){
fscanf(locations,"%s",location);
  if (strcmp(location,"**") == 0){break;}
  else {
    READ_TEMPLATE_IMAGE(location);/* read template image in memory*/
    match_val = MATCH_IMAGES();
    if (match_val >= threshold){
      threshold = match_val;
      strcpy(the_most_probable,location);
      its_score = match_val;
    }
  }
}
fclose(locations);
if (its_score >= 0.0){ /* the most probable */
  fprintf(place,"%hd\n",1); /* indicator for success */
  fprintf(place,"%s",the_most_probable);
}
else {
  fprintf(place,"%hd\n",2); /* failure to find a matching location */
}
fclose(place);
}

/* read the sensor image from files */

void READ_CURRENT_IMAGE()
{ FILE *curim;

float rhos,thetas,ys,xs,yw,xw,length,crho,ctheta; int numlevs,
levnum,numH,numV,numN,numT; int
i,j,level,Size,Maxlen,maxlen;
char Type[2],dum00,dum01[2],dum02[2];
char dum1[126],dum2[126],dum3[126],dum4[126];

```

```

curim = fopen("current.im","r");

/* read sensor image from file */
fgets(dum1,126,curim);fgets(dum2,126,curim);
fgets(dum3,126,curim);fgets(dum4,126,curim);
dum00 = fgetc(curim);
fscanf(curim,"%f %f",&current.Crho,&current.Ctheta);
fscanf(curim,"%s%d%d%d%d%d%d",dum01,&current.numlevs,&current.Size,
&current.Maxlen,&current.numH,&current.numV,&current.numN,&current.numT);

for (i = 0; i < current.numlevs; ++i){
    fscanf(curim,"%2s%d%d%d%d%d",dum02,&curlevs[i].levnum,&curlevs[i].Maxlen,
&curlevs[i].numH,&curlevs[i].numV,&curlevs[i].numN,&curlevs[i].numT);

    level = curlevs[i].levnum; /* level number */
    for(j = 0; j < curlevs[i].numT; ++j){
        curlines[j].levnum = level; fscanf(curim,"%2s%f%f%f%f%f%f",curlines[j].Type,
&curlines[j].ys, &curlines[j].xs,&curlines[j].rhos,&curlines[j].thetas,
&curlines[j].length,&curlines[j].yw,&curlines[j].xw);

        curlines[j].rhos = curlines[j].rhos * current.Crho;
        curlines[j].thetas = curlines[j].thetas * current.Ctheta;
    }
}
fclose(curim);
}

/* Read template image from file */

void READ_TEMPLATE_IMAGE(char observ)
{ FILE *templim;

float rhos,thetas,ys,xs,yw,xw,length,crho,ctheta; int numlevs,
levnum,numH,numV,numN,numT; int i,j,level,Size,Maxlen,maxlen;
char Type[2],dum00,dum01[2],dum02[2], template_file[13];
char dum1[126],dum2[126],dum3[126],dum4[126];

/* copy template image file into temporary file */

strcpy(template_file,observ);
strncat(template_file,".im",3);

templim = fopen(template_file,"r");

fgets(dum1,126,templim);fgets(dum2,126,templim); fgets(dum3,126,templim);
fgets(dum4,126,templim);
dum00 = fgetc(templim);
fscanf(templim,"%f%f",&template.Crho,&template.Ctheta);
fscanf(templim,"%2s%d%d%d%d%d%d",dum01,&template.numlevs,&template.Size,

```

```

&template.Maxlen,&template.numH,&template.numV,&template.numN,&template.numT);
for (i = 0; i < template.numlevs; ++i) {
    fscanf(templim,"%2s%d%d%d%d%d",dum02,&templevs[i].levnum,
        &templevs[i].Maxlen, &templevs[i].numH,&templevs[i].numV,&templevs[i].numN,
        &templevs[i].numT);
    level = templevs[i].levnum; /* level number */
    for(j = 0; j < templevs[i].numT; ++j) {
        templines[j].levnum = level; fscanf(templim,"%2s%f%f%f%f%f%f",
            templines[j].Type,&templines[j].ys, &templines[j].xs,&templines[j].rhos,
            &templines[j].thetas, &templines[j].length,&templines[j].yw,&templines[j].xw);

        templines[j].rhos = templines[j].rhos * template.Crho;
        templines[j].thetas = templines[j].thetas * template.Ctheta;
    }
}
fclose(templim);
}

```

******* module to match two images *******
 Each level of the current image is matched to a level in the template image.
 A decision is made after each level has gone through the matching process.
 Two levels will match if there can be found a satisfactory correspondence
 between the lines of that level and the lines of the candidate level. Each level
 has attached to it a weight. The weight represents the importance of the level in
 the matching process. The weights will be taken into account to estimate the overall
 cost function of the matching.

*******/**

```

float MATCH_IMAGES()
{
    int i,j;
    float cost_function = 0.0,c1, sum_of_weights = 0.0;
    float match_levels(), partial_hit_ratio = 0.0, weight = 0.0;

    for (i = 0; i < current.numlevs; ++i) {
        curlevs[i].score = match_levels(i); /*match lines of level i*/
        weight = 0.8 - (float) 0.1*curlevs[i].levnum;
        sum_of_weights += weight;
        cost_function += curlevs[i].score * weight;
    }
    c1 = cost_function;
    if (sum_of_weights == 0) {return (0.0);}
    cost_function = c1 / sum_of_weights ;
    return(cost_function);
}
}

```

/* ----- match two levels -----*/

```

float match_levels(l)
int l;
{
short int match_lines();

```

```

int misses = 0, matches = 0;
int i,j,bool=0;
float hit_ratio;

for (i = 0; i < curlevs[l].numT; ++ i){
    if (curlines[i].levnum != l) {;}
    else {
        bool = 0;
        for (j = 0; j < templines.numT; ++ j){
            if (match_lines(i,j) == 1){
                bool = 1; break;
            }
        }
        if (bool == 1){ matches += 1;}
        else {misses += 1;}
    }
}

if (curlevs[l].numT == 0) {return (1.0);}
else {
    hit_ratio = (float) matches/curlevs[l].numT;
    return(hit_ratio);
}

/* ----- match two lines -----*/
short int match_lines(i,j)
int i,j;
{
float delta_theta = 0.0,test, d1,delta_dist_centers = 0.0;
float delta_dist_ends_1 = 0.0, delta_dist_ends_2 = 0.0;

epsi_centers = min (curlines[i].length,templines[j].length);
epsi_ends = epsi_centers;
delta_theta = curlines[i].thetas - templines[j].thetas;
if (fabs(delta_theta) > epsi_theta) { return(0);}
else{
    delta_dist_centers = pow(fabs(curlines[i].xs - templines[j].xs),(double)2.0) +
    pow(fabs(curlines[i].ys - templines[j].ys),(double)2.0 );
    delta_dist_centers = (float) pow((double)delta_dist_centers,(double)0.5);
    delta_dist_ends = (float) fabs((float)templines[i].length - curlines[j].length );
    if (delta_dist_centers <= epsi_centers ) {
        if (delta_dist_ends <= epsi_ends){
            return(1);
        }
        else {return(0);}
    }
    else {return(0);}
}
}
}

```

¹ Only source codes of few programs have been listed in this appendix. All the source codes of the database programs are listed in the User's Manual.