Qualitative Environmental Navigation: Theory and Practice

Il-Pyung Park

CUCS-003-94

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences.

Columbia University
1994

ABSTRACT

Qualitative Environmental Navigation:
Theory and Practice

Il-Pyung Park

In this thesis we propose and investigate a new model for robot navigation in large unstructured environments. Current models which depend on metric information contain inherent mechanical and sensory uncertainties. Instead we supply the navigator with qualitative information. Our model consists of two parts, the map-maker and the navigator. Given a source and a goal, the map-maker derives a navigational path based on the topological relationships between landmarks. A navigational path is generated as a combination of "parkway" and "trajectory" paths, both of which are abstractions of the real world into topological data structures. Traversing within a parkway enables the navigator to follow visible landmarks. Traversing on a trajectory enables the navigator to move reliably into a homogeneous space, based on shapes formed by visible landmarks that are robust to positional and orientational errors. Reliability measures of parkway and trajectory traversals are defined by appropriate error models that account for the sensory errors of the navigator, the motor errors of the navigator, and the population of neighboring objects. Error detection and error recovery methods are also encoded into the generated path. The optimal path is further abstracted into a "custom map," which consists of a list of verbal directional instructions, the vocabulary of which is defined by our environmental description language. Based on the custom map generated by the map-maker, the navigating robot looks for events that are characterized by spatial properties of the environment. The map-maker and the navigator are implemented using two cameras, an IBM 7575 robot arm and PIPE (Pipelined Image Processing Engine.) Various experiments show the effectiveness of navigation "in the large" using the proposed methods.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincerest appreciation to my advisor and good friend Professor John R. Kender. During the past five years, he has been - in the truest sense - a role model for me. Without his brilliant insights, intriguing criticisms, unreserved kindness, and rock solid patience, I would not have been able to start, develop, continue, and finish this thesis.

I would like to thank Professor Peter Allen for generously providing me his laboratory facilities and "real-world" criticisms. I would like to thank Professor Terry Boult for always being there as an "oracle" that when given an input, never failed to come up with the optimal answer. Professor Shree Nayar deserves a special thanks for his very helpful comments, answers, and lots of encouragement. I would also like to thank Dr. Vijay Srinivasan and Dr. David Hoeltzel for serving as external members of my thesis defense committee.

David Yang collaborated with me to get my first project started, in which he developed the user interface program for the gridworld navigation. Billibon Yoshimi provided lots of valuable technical assistance. He programmed the "hot-zone" program in PIPE for my initial project, and in general was always available whenever I was stuck with "implementation details".

I would like to thank my parents for providing me with top quality education and for their financial support even when things were very tough for them. Finally, I would like to express my dearest appreciation to my wife Tina, who has given me incredible amounts of support, encouragement, and love. She made everything worth while.

This work was supported in part by the Advance Research Project Agency under Contract Number DACA 76-92-C-0007.

To my lovely daughter, Anne

Chapter 1

Introduction

1.1 Statement of problem

We are interested in navigation in a large and unstructured environment. An environment is regarded as large if at any instant the sensory and motor capacities of the navigating agent reside in a small subset of the whole terrain. Similar definitions can be found in [39]. An environment is unstructured if there are no external cues for the navigator to rely on, such as hallways or roads.

Navigation in a large unstructured environment requires different information and tools than that of navigation in a small structured environment. The information that is useful is mostly qualitative rather than quantitative. Therefore, we feel that high-level reasoning is necessary for the navigator to be able to handle ambiguities and errors. For example, "follow this road until you see a big white building on your right", is a good directional instruction, whereas "go straight for 200 meters and turn 40 degrees clock wise, ..." is not. One of the most important entities in giving directions, especially in a large environment, is the usage of landmarks. Without landmarks, it would be impossible to give "high-level" directional information. It would also be very hard for the navigator to verify that it has reached its destination or a subgoal. Topological relationship between landmark objects is another useful information because it tends to be invariant to metric errors.

In this dissertation, we provide a new framework for topological navigation in a large unstructured environment, that emphasizes the use of qualitative, high-level, landmark-based directional instructions that minimizes a navigator's efforts and prevents mistakes. Part of this dissertation is motivated by questions concerning landmarks, such as, (1) what is a good landmark? (2) what features of a landmark are important? (3) what sensors are to be used to recognize a landmark efficiently?[1] (4) how to describe a landmark to the navigator? (5) how to detect errors and (6) how to recover from them. In attempting to answer these questions, we suggest a dichotomy of roles in the navigation system: a superior direction giver (map-maker) and a dumb direction follower (navigator). Our qualitative environmental navigation system, which consists of the map-maker and the navigator, can be categorized as both a preplanned and reactive type [65]. It is preplanned because the desired path of the navigator is precomputed by the map-maker subsystem. But it is also reactive because the "custom-map" provided by the map-maker for the navigator does not hard code the desired path but rather "describes" what the navigator's sensors are supposed to look out for. Then, based on the activation of appropriate algorithm, the navigator reacts to the environment.

The rest of this chapter is as follows. First, in section 2, we briefly survey related work. This includes studies in psychology concerning cognitive maps and direction giving, and existing qualitative navigation systems. Then in section 3 we present an overview of this dissertation.

[1]This problem has been addressed by Kender et al. in [31], in which they present the computational complexity of selecting optimal set of sensors. Even in one-dimensional space, this proves to be an NP-complete problem.

1.2 Related work

1.2.1 Cognitive maps

A cognitive map is a general common-sense knowledge of space acquired by a human over time. Kuipers [35] adopted the concept of cognitive map in his TOUR model. He divides spatial knowledge into a hierarchy of four levels, namely, "sensorimotor", "procedural", "topological", and "metrical". In general, assimilation of knowledge (learning the map) proceeds from the lowest to the highest level, and the higher the level, the more complex cognitive computation becomes. Matching the spatial knowledge with the current environment (navigating) would involve the same semantic hierarchy and complexity. So, for example, trying to match topological knowledge with the topological features of current environment involves a greater cognitive complexity than trying to match sensorimotor knowledge with the features of current environment. Verbal direction is basically a sensorimotor or a procedural description, whereas a visual map is topological and metrical. When navigating using topological or metrical information, one has to somehow convert the information into sensorimotor or procedural knowledge because the latter information is easier to match with the environment that one is navigating. Spatial knowledge, in the TOUR model, is comprised of knowledge in five different categories : route descriptions, topological structures of street networks, relative positions and orientations, dividing boundaries, and regions. Knowledge of a particular environment is represented by description of each of these categories. Inference rules are used to update and manipulate knowledge about the environmental descriptions and current position information (you-are-here pointer) of the navigator. New information is assimilated into the existing knowledge structures by "walking" the current position pointer around the environment. Kuipers' model supports various degrees of the amount of knowledge being assimilated. For example, a complete route can be reconstructed from memory without having to travel if the knowledge accumulated has enough information. But if the memory lacks some details due to an interruption or some other failure, as long as critical information is preserved, it still can reconstruct the route by physically moving along the path. This is analogous to humans' ability to navigate in a not-so-familiar environment without being able to describe the route. So the performance of this model agrees with the property of representing a common sense knowledge which is to "degrade gracefully in situations with resource limitations". Kuipers goes further in analyzing how cognitive maps are formed in [38]. He tries to re-create an artificial cognitive map and compares it with cognitive maps of humans in the five different categories as follows: observations, route description, fixed features, topological relations, and metrical relations. He concludes that an alternate design of our cognitive map is not possible due to the environment that we live in.

Chase [8] conducted various laboratory tests and field tests on taxi drivers. His experiments showed that locations in people's minds are organized hierarchically by neighborhoods and regions. For example, two locations within same neighborhood seemed to be more closely bonded with each other then two locations in different neighborhoods. Results also showed that lesser known streets were mostly used by experts whereas novices basically stayed on the base network. One surprising result is that physical navigation gave better results than laboratory simulation. This means that performances of the drivers are often influenced by visual cues that are only present on-site. This result agrees with the Kuipers TOUR model in that a partial knowledge can be useful enough during the actual navigation.

Thorndyke [70] carried out an extensive experiments to compare the spatial knowledge acquired by studying maps versus spatial knowledge acquired by navigational experiences. The results of the experiments showed that generally, learning from maps gave better results than learning from navigation in estimating Euclidean distances (as-the-crow-fly) and locating places. On the other hand, learning from navigation proved to be superior than learning from maps in tasks such as orientation estimation and route distance estimation. However, the differences in performance between the two groups for estimating Euclidean distances and estimating locations disappeared as the navigator gained more experiences. The implication of this last phenomenon is that a long term exposure to an environment seems to give people not only a procedural knowledge of the routes but topographical knowledge of the area as well.

Streeter and Vitello [66] carried out an experiment to investigate the correlation between self-descriptions of each person's navigational skills and of various navigational proficiencies. The results showed that self-described good navigators indeed have higher spatial abilities than those who rated themselves to be poor navigators. They also noticed that the low spatial ability people rely heavily on landmarks for navigation, whereas the high spatial ability group used other spatial cues as well. A second experiment concerned route finding by computer algorithms and by humans. Various graph-search algorithms were compared with human algorithms. Methods used by each human differs considerably depending on how familiar he or she is with the environment and how much spatially proficient he or she is. The routes that they chose were radically different to the routes that computer algorithms chose, meaning that even though computer algorithms may produce the shortest path, it may be cognitively difficult for humans to use. They suggest a heuristic method that simulates human methods and indicate that future automatic route generating systems should produce a "customized" route that meets individual's navigational skills.

1.2.2 Direction giving

Direction giving is part of our daily life. We give and receive directions whenever we visit some places or invite someone. Figuring out how to get to an unfamiliar place is not an easy task. So most people rely on the directional instructions given to them by their hosts. In fact, many organizations provide the visitors with either customized maps or printed verbal directional instructions on how to get there. In this section, we survey various results in computational psychology with the focus on the following questions:

- What is the "best" route?
- How do you decide whether a directional instruction is clear and concise enough?
- What is the best medium in which to deliver the directional instructions to the driver.
- How do all these relate to the human spatial abilities?

The idea of giving directions in terms of "sensorimotor" or "procedural" information is supported by the result of the investigation by Streeter et al. [67]. They performed an experiment on a group of people to compare the effectiveness of four different types of navigational aids: vocal instructions, simplified maps, both together, and commercial maps. The result revealed the superiority of vocal instructions over the other three methods. This result is somewhat expected since driving is an intensive task which requires the driver's constant visual attention, whereas his auditory channel is less busy. It is interesting to note, however, that using both vocal instructions and simplified maps was less useful than using only vocal instructions. This indicates that too much information degraded the performance of the drivers. The investigators also emphasize that the qualities of voice instructions that they provided, which were derived from a careful study of navigation-related psychology, played an important role. The key point here is that it is very important not to over-burden the navigator with too much information. One drawback in using voice instructions is the lack of global view of the route traveled. Although they claim that this did not have any negative effect on the performances of the drivers, it could still be helpful to provide to the drivers, if done in a non-distracting way, an overall picture of the route. Based on this, one could envision having an on-board navigation system with a display screen which shows a digitized map with the driver's current location and the destination [17]. One important reason for people preferring verbal instructions rather than visual maps, that they neglected, is the computational complexity involved managing the hierarchy of spatial knowledge described in the previous section.

Elliott and Lesk [15] developed a computer program that generates driving directions using street maps and telephone books. They argue that the shortest distance route is not the best path to follow and introduce a criterion that penalizes each turn (the penalties for left and right turns are different because a right turn is easier than a left turn). So, a longer path with less number of turns could be a "better" path than a shorter path with many turns. This resulted in some interesting results, such as "going" an extra mile in order to avoid making turns. They also study various types of computer and human algorithms for finding the shortest path. Among computer algorithms, depth-first-search gave the most effective result. Typical human algorithms involved a hierarchical combination of divide-and-conquer and depth-first-search. (This is similar to Kuiper's intermediate goal in the TOUR model).

Mark [44] proposed "ease of description" (minimizing number of turns) as an important factor in choosing routes. To use a computer algorithm to find optimal route using this criterion, he represents route descriptions, which follows Riesbeck's model (described in the next paragraph), using a "frame". The number of slots in each frame needs gives a number representing the complexity of description. This number is scaled with appropriate weight constants and then it is added to current path length to produce the cost of current path. The A^* algorithm is applied to this current path cost function to derive an optimal route. The weight (scale factor) has a meaning that if it is set to 0, the algorithm will produce the shortest-distance-path, and if it is large the algorithm will favor a straight path to making any kind of turns (larger slot numbers). His experimental results with various weight constant showed that a person is willing to travel 0.36 km more in order to avoid a turn.

Riesbeck [62] suggested a computational method for evaluating the clarity of directions. He emphasizes that direction following, which involves a natural language analysis, does not require complete processing of the given instructions until the execution time. He emphasizes that the judgement of clarity is not to be based on whether or not the set of directions can be used to construct a meaningful map, but in fact it should be focused on the feasibility of the driver to follow the directions later. His program McMap, processes the text and using a set of inference rules it classifies the directions into three different components : motion, description, and comments. It then uses a set of domain-specific heuristic rules to determine the clarity and cruciality of each sentence and of the overall direction.

Mark states that [43] Kuiper's model of spatial knowledge implies (reversely) that a verbal description of the route is better than a visual display of the map. This is because procedural knowledge is lower level than topological or metrical ones. Mark also conducted an experiment in which 20 sets of invitational directions were analyzed, most of which consisted of customed maps and some of which consisted of verbal instructions [45]. The analysis indicated that most people consider orientation information as an important part of navigational aid. Distance information, however, did not prove to be as important because most people omitted the scale indications in their maps. One reason why there were more customized maps than verbal instructions was because the directions were intended for many people coming from various different places, so it was not possible to assume common origin. Another reason (more general) is because a map gives a global view, so that error recovery is easier. However, the relative weakness of procedural description shown in this particular study is inadequate because the directions used here were intended for many people. It would be a different story if directions are to be produced for each individual by an on-board computer system.

1.2.3 Other approaches

Traditional models in robot navigation

Traditional approaches to the robot navigation require accurate metric precision of the robot's paths. These methods include Configuration space by Lozano-Perez [42], Generalized Cones by Brooks [6], Segmented Model by Crowley [10], Grid-based Model by Moravec and Elfes [48], and Convex Cell model by Giralt et al. [22]. Both Configuration Space and Generalized Cones were developed as solutions to the well known find-path problem in which the objective is to find a collision free, continuous path for a navigator from an initial position and orientation to a goal position and orientation. The basic idea behind the Configuration Space [42] is that the navigator is "reduced" to a point while all the other objects are "enlarged" accordingly. The problem is then transformed into determining the free space that can be occupied by this reference point. In the Generalized Cone method [6], the free space is represented by overlapping generalized cones, producing "freeways" that represent the free space the navigator is allowed to occupy. In the Segment model used by Crowley [10] in his IMP (Intelligent Mobile Platform) project, the robot's navigational path is represented by a combination of linear segments. The robot's execution path segments have to positionally and orientationally match the planned segments. In the Grid Model developed by Moravec and Elfs [48], a 2 dimensional horizontal map is used to represent the navigable environment. Each cell of the $M \times N$ grid cells is probabilistically classified as empty, occupied, or uncertain. The HILARE system, a mobile robot project at LAAS, France, uses a Convex Cell model to represent the world. Here, the path planning is done by connecting the midpoints of adjacent convex cells that represent the free space.

Qualitative navigation

Most traditional methods, including the ones introduced in the previous section, perform reasonably well only in small environments and fail to be useful in large unstructured environments [39]. The reason for this is because the metric information is inaccurate due to the low mechanical accuracy and sensory errors [69]. Kuipers was one of the first to acknowledge this. He used "qualitative" approaches in his robot navigation projects, namely, the TOUR model [35] and the NX Robot [39]. As stated earlier in previous section, the TOUR model adopts the concept of cognitive map in which the spatial knowledge is divided into four levels: "sensorimotor", "procedural", "topological", and "metrical". The high level reasonings to derive paths are applied to the "topological" and "metrical" levels, whereas the low level commands to steer the navigator through the environments use "sensorimotor" and "procedural" levels of the hierarchy. NX is a simulated robot that explores and builds map of a structured indoor environment. It uses sonar sensors for local environmental experiences, and a compass for global orientation. NX represents the world in three levels of procedural, topological, and metric maps. A distinctive place (DP) is defined to be a position in the environment that is the local maximum in the robot's sensory system. It is identified by the robot by a hill-climbing search in the procedural level. In the topological level, a DP of the environment is represented as a node of the graph data structure. Metric map contains the local geometric information of the place.

Levitt et al. proposed qualitative methods for navigation in large scale space [40] in their Qualnav model. The simulated robot uses a global map in its spatial memory to indicate each landmark's estimated direction and distance for path planning. In the topological (qualitative) level, Qualnav divides the environment into discrete regions called orientation regions, created by landmark pair boundaries (LBP). In the metric (quantitative) level, the world is represented as "viewframes" which encodes the observable landmark information that includes angles between landmarks and range estimates. A path is a sequence of locations represented in the qualitative or the quantitative map.

Sutherland and Thompson [69] use similar model for navigation. Using their "inexact navigation" model, they show that configuration of visible landmarks is very important for accurately localizing the navigator. The "area of uncertainty", in which the navigator can self-locate within a given error range depends on the choice of landmarks. Dai and Lawton [11] suggest qualitative navigation algorithms to work complete at the topological level. They distinguish the difference between "distinctive" and "non-distinctive" landmarks. A distinctive landmark is the one which can be recognized from different points of view [2]. In their simulated experiments, they show that the more distinctive landmarks are available, the efficiency of navigation increases.

Zheng et al. [78] present a unique method in long distance navigation. They use *Panoramic View* (PV), which is a continuous image of sideways view taken from the mobile robot, to represent the navigable path. From the PV, they extract low-level (brightness, hue, texture, etc.), mid-level (area, perimeter, etc.), and high-level (structural properties) features. These features are then used to identify "distinctiveness" of the scene to serve as landmarks. Since this framework assumes that the navigation is done in a structured environment, such as roads, it is not clear how well it can be adopted for use in an unstructured environment that doesn't provide external guidelines.

Most of the work in qualitative navigation, including the ones describe above, emphasize the importance of landmarks. And their navigation depends heavily on the usage of visual landmarks. However, none provide a formal answer to the question "what is a landmark?" They assume that landmarks can be readily identified by their intrinsic qualities. Otherwise, they use the general definition of "distinctiveness" of features in the sensory readings to indicate landmarks. We

[2] Similar definition can be seen in [31] [32] presented in a more formal way.

argue that without detailed criteria for defining and selecting good landmarks, topological navigation is not possible. In this thesis, we present qualitative methods to identify landmarks, as well as present means to navigate using the selected landmarks.

1.3 Overview of the thesis

This dissertation is organized as follows. In the next chapter, we present some of our previous work of landmark based navigation in one-dimensional (lineland) and grid-based two-dimensional (level helicopter world) environments. Kender and Leff [31] formalized the basic concept of the map-maker's relationship to the world in different levels of abstractions, and the navigator's custom map dependency. They discovered large range of computationally difficult problems that can be solved only using heuristics, even in a simplified one-dimensional space. Kender and Park [32] used this framework to implement a landmark based navigator in grid-based two-dimensional space using a camera and a robot arm. They developed useful criteria for creating a custom map based on sensing cost, communication cost, reliability, landmark distinctiveness, and travel time. Chapter 3 provides basic definitions and assumptions to be used throughout the rest of the dissertation. We explain the need for multi-level abstraction of the world, and describe the roles of the map-maker and the navigator. In chapter 4, we introduce two fundamental graph-like abstract data structures, created and used by the map-maker for representing the environment mainly in the topological level of the cognitive map layer [35]. Parkways represent viable paths that are based on visible landmarks, trajectories provide "junctions" between parkways. Various types of parkways and trajectories are introduced. Chapter 5 describes another level of abstraction of the world, the qualitative description language for efficient and error minimizing communication between the map-maker and the navigator. A basic vocabulary to identify landmarks is introduced, some of which are implemented in the thesis. In chapter 6, we analyze various sources of error and model them analytically and statistically. These error models are incorporated into the cost matrix for the generation of reliable paths. We also present a general framework for error detection and error recovery. Details in the error detection and recovery algorithms which use the Quadrant Occupancy Count (QOC) are then discussed. In chapter 7, we discuss various issues in path planning, which includes specification on how parkway and trajectory paths are represented in the transition and cost matrices, how different types of cost measures affect the path generation algorithm, and what kinds of heuristics to be used. We also present the formal grammar for the custom map language and give examples of how parkway and trajectory paths are further abstracted into more manageable forms of custom maps. Chapter 8 describes the setup of the experimental hardware for the map-maker and the navigator, which are implemented using CCD cameras, an IBM robot arm, and a Sun workstation. Experiments using this hardware setup are also reported in this chapter. Finally, in chapter 9, we present the conclusions of this dissertation and discuss the future work.

1.4 Summary

The purpose of this chapter was three-fold. Firstly, we discussed the problem of navigation in large unstructured environment. Secondly, we presented a brief survey on the related work, including topics in the domains of human psychology and previous work in the area of qualitative navigation. Lastly, we provided an outline of this thesis.

Chapter 2

Earlier Work

In this chapter, we present some of our previous work of topological navigation in one and two dimensional spaces. Section 1 discusses the model of "Lineland", in which a theoretical framework for topological navigation in one-dimensional space is presented [31][1]. Navigation is based on matching landmarks to partial object descriptions, provided by the direction giver. Even in a very simple and error-free one-dimensional environment, navigation proves to be a very hard problem (NP-complete.)

In section 2, we talk about our experimentations with the "level-flying helicopter" navigational model, in which we formalized and implemented a two-dimensional grid-based topological visual navigation system. The navigator's movement is restricted to the four compass directions of *north, east, south, and west*, and its sensing range is assumed to be very limited. Depending on the type of cost (sensor, distance, or communication.) to minimized, we specified different ways in which visual landmarks are to be selected.

The common denominator of these work (and also this dissertation) is in the

[1]The author did not have any direct contributions to the Lineland project. However, some fundamental concepts introduced in the Lineland are employed in this thesis. Therefore, it is introduced here.

emphasis of creating efficient directional instructions, which we call custom maps. This is in contrast to much of existing work on robot navigation, which emphasizes spatial learning of metric information through direct experience.

2.1 Linear navigation

Kender and Leff [31] formalized a model of topological navigation in one-dimensional space, such as corridors and transportation routes, which they named "Lineland". They developed the concept of guiding a navigator by "custom maps", and discussed the importance of using different representations of the world perceived by the direction giver and by the direction follower. It turns out that even in a simplest one-dimensional space, and with an error free navigator, direction giving is a very hard problem. In fact, the computational complexity of optimizing the sensor selection and the sequencing of sensor modalities are proved to be NP-complete.

2.1.1 Multi-level abstraction of the world

Many navigational environments, such as corridors, highways, and subways, are in a sense one-dimensional because the navigating agent has only one degree of freedom. The environment can be abstracted according to three different perceptions of the world. First is the one-dimensional world as it exists which contains a lot of extraneous information, such as multi-featured objects with spatial extent, distance measure, continuity, and empty spaces. It is the "zeroth" level of abstraction. The second is the perceptions of the map-maker (we redefine this term in Chapter 3), which filters out the most unimportant information. The abstracted world is a sequence of feature-tuples. For example, a Lineland world model that contains 10 objects, and each object is described by two feature tuple, taken from the domains {r, g, b} for color, and {s, l} for size, is given by: (gs)-(rs)-(gl)-(rl)-(gs)-(rl)-(rs)-(rs)-(gl). This is the first abstraction level. Thirdly, for the next level of abstraction, the map-make communicates even less of this sequence to the navigator using a "custom map" of the form <*start, direction*^{D}, *goal*> where start and goal are Lineland sequence indices, and D is the length of the custom map. Each entry in the custom map, indicated by "direction" above, consists of a heading ("-" or "+"), and a partial object description (POD) that describe the object to be sought in terms of a feature tuple. For example, one of many possible custom maps to guide the robot from the left-most object to the right-most object is : <1, +rl+gl ,9>.

2.1.2 Creation of custom map

The custom map is created by using two abstract data types, the transition graph and the cost graph. The graphs record information about the most efficient direction (heading plus object description), if any, that takes a navigator from any given node in the world model graph to any other node. If there is such a unit direction, the transition graph records it, and the cost graph records its cost. The process of finding the optimal graph is done in two stages.

First Stage: Formal Definition of "Landmark" In the first stage the transition graph is filled with unit directions of the most inefficient kind, namely the full object description of the unit direction's goal. For example, in the example one-dimensional world model (gs)–(rs)–(gl)–..., the unit transition connecting nodes 1 and 3 would initially be given as $+gl$, even though either $+g$ or $+l$ would suffice. In the most general case in two dimensions, creating the world planar graph, and labeling the arcs between two reachable nodes is a complex procedure, highly dependent on the vocabulary of headings and objects, and it may be non-trivial to select the unit directions between nodes in an efficient manner. In special cases,

	gs	rs	gl	rl	gs	rl	rs	rs	gl
gs		$+rs$	$+gl$	$+rl$	$+gs$				
rs	$-gs$		$+gl$	$+rl$	$+gs$		$+rs$		
gl	$-gs$	$-rs$		$+rl$	$+gs$		$+rs$		$+gl$
rl	$-gs$	$-rs$	$-gl$		$+gs$	$+rl$	$+rs$		$+gl$
gs	$-gs$	$-rs$	$-gl$	$-rl$		$+rl$	$+rs$		$+gl$
rl		$-rs$	$-gl$	$-rl$	$-gs$		$+rs$		$+gl$
rs		$-rs$	$-gl$		$-gs$	$-rl$		$+rs$	$+gl$
rs			$-gl$		$-gs$	$-rl$	$-rs$		$+gl$
gl			$-gl$		$-gs$	$-rl$		$-rs$	

Table 2.1: First stage transition matrix

such as the one-dimensional world given above, or in the Cartesian two-dimensional world actually implemented, clever encodings do yield good efficiencies of both processing and storage. For the example, the full first stage transition graph is shown is Table 2.1, where empty entries are considered to be filled with sentinel values.

In the most general case, the transition graph will be indexed by node number, and the entry for a unit transition between node i and node j will record the heading in its vocabulary and the full object description of j. However, what is most significant is that in general, if navigation is in-the-large, by definition there will be a large number of objects with the same object descriptions; since only the first (that is, the nearest) object is attainable by a given unit direction, the transition graph will always be sparse, and it may be compressible by the usual techniques.

We now note that one definition of a "landmark" would be an object whose column in the transition graph is very nearly filled, that is, an object that can be obtained under a unit direction from very nearly everywhere. In the above example, the gl object is an optimal landmark. One can easily define a concept such as the "landmark radius" of an object in the obvious way. Because landmarks can be used to navigate over great numbers of objects, they will tend to appear regularly in custom maps. However, landmarks need not be "reversible"; for example, the fourth object in the above example is the first of two such objects close together, and is only valuable as a landmark while navigating to the right; it is perceptually "shadowed" by the sixth object when navigating to the left. In other words, traveling left from most places it is impossible to reach it in one step.

Second Stage In the second stage of transition and cost graph construction, the transition graph is optimized by selecting the minimal object description that maintains object uniqueness. The cost of this optimal unit direction is recorded in the cost graph; sentinel values in the transition graph giving rise to infinities in the cost graph. In general, the second stage is more complex, conceptually and computationally, for several reasons. First, there may be more than one optimal partial object model that attains the optimal cost: in the example, the unit direction from 1 to 3 can be $+g$, or $+l$. Depending on the overall model of map goodness they may all have to be recorded. Secondly, depending on regularities in the world, or the vocabularies of the navigator directions, the optimization of the entire graph may be done in a more efficient way than by optimizing each entry separately. But third and most importantly, the optimization of even a single unit transition is NP-complete in the number of partial models. For detail, see [31]

Selecting the optimal partial model can be obtained by heuristic search, with the most effective heuristic appearing to be the one that says "select the partial model that conflicts most with all the other objects along the path to node j". In short, the topological visual navigation problem subsumes the problem of minimal object description, except that it provides a well-defined context: the context of the path to the object.

Applying optimization to the example, we find the following full second stage transition and cost graphs, in Table 2.2 and Table 2.3, where cost is given as sensing cost over the intervening objects.

At this point, the problem is no longer a vision or a robotics one; standard

	gs	rs	gl	rl	gs	rl	rs	rs	gl
gs		$+r$	$+g$	$+rl$	$+gs$				
rs	$-g$		$+g$	$+r$	$+s$		$+rs$		
gl	$-g$	$-r$		$+r$	$+g$		$+rs$		$+gl$
rl	$-gs$	$-r$	$-g$		$+g$	$+r$	$+rs$		$+gl$
gs	$-gs$	$-s$	$-g$	$-r$		$+r$	$+s$		$+g$
rl		$-rs$	$-gl$	$-r$	$-g$		$+r$		$+g$
rs		$-rs$	$-gl$		$-g$	$-r$		$+r$	$+g$
rs			$-gl$		$-g$	$-l$	$-r$		$+g$
gl			$-gl$		$-g$	$-l$		$-r$	

Table 2.2: Second stage transition matrix

	gs	rs	gl	rl	gs	rl	rs	rs	gl
gs		1	2	5	6				
rs	1		1	2	3		7		
gl	2	1		1	2		6		8
rl	5	2	1		1	2	5		7
gs	6	3	2	1		1	2		4
rl		6	5	2	1		1		3
rs		7	6		2	1		1	2
rs			7		3	2	1		1
gl			8		4	3		1	

Table 2.3: Cost matrix

graph search algorithms such as Dijkstra or A^* over the transition and cost graph can be used to find the least costly path from a given node to another. The resulting sequence of unit directions is the custom map.

2.1.3 Map quality measures

The goodness of the custom map is measured by two independent criteria. A custom map is "good" if, among all such maps that attain the goal from the start position, it does so with (1) the least amount of directions or (2) least amount of sensing. These criteria can be used in conjunction as well.

2.2 Two-dimensional grid world navigation

In this section, we describe the work on the topological visual navigation in a two-dimensional grid world, as presented in [32]. The domain we explored is a formalized variant of the visual world seen by a level-flying helicopter whose perception is limited to a narrow field of view, and whose visual axis is normal to the world below it. Thus, occlusions are not yet an issue, but object identity and confusability are, and so are issues of communicative, perceptive, and motive economy.

This work was ultimately motivated by the questions of what is a good map, what is a good landmark, and what is a good visual environment for navigation. It addresses the abstract problem of how to optimally choose visual landmarks and their sensory features in order to describe and discriminate objects, and how to create short or efficient sequences of such descriptions for low cost (however defined) qualitative navigation from a given place to another.

2.2.1 The world model

The initial application of these ideas is to a highly idealized two-dimensional world. Nevertheless, it serves as an arena for the exploration of significant computational and representational problems. The world is divided into M rows and N columns, and the navigator's vocabulary of headings is is restricted to traveling in one of the four directions: *north, east, south*, and *west*. Although severely restricted, this world still adequately models much navigation in urban areas, like taxi-cabs in New York, or within man-made structures, like mail delivery in offices. Restricting the world to rows and columns still leaves room for a larger vocabulary of heading strategies: various space-filling searches, for example.

An object, or interchangeably a landmark, can be placed in any cell of this grid.

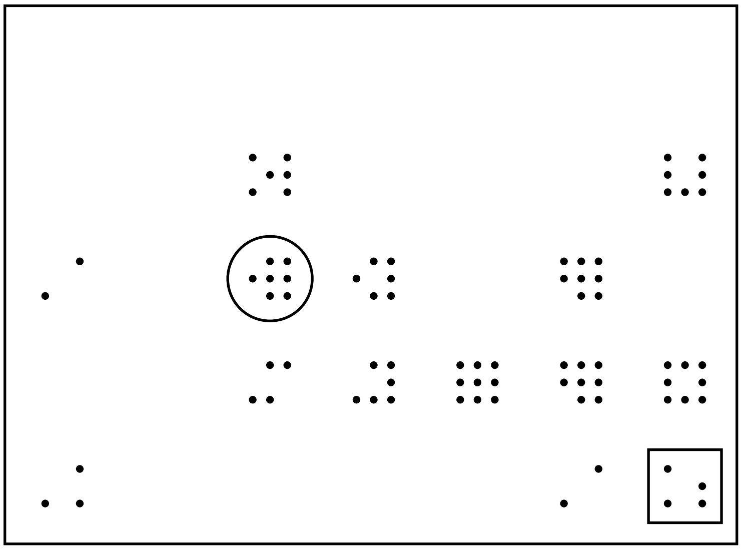


Figure 2.1: A sample dots world

Given the current limits on processing speed, we define as our object model a 3×3 matrix of dots; this allows 511 possible objects, but far fewer if rotation of objects are not considered distinct. There are no object symmetries due to rotations in this world, since the camera is room-oriented, not path-oriented: that is, as our robot arm changes direction, the camera does not rotate. (This is roughly equivalent to a human being's navigating by turning his road map on his lap as his car turns.) We further impose a restriction such that all objects have at least one dot on each of the four edges of the matrix. This simplifies the *adjust* phase of the navigation which is to be described in a later section, and avoids the epistemological issues such as how many objects there are that have a single matrix dot (one or nine?). Figure 2.1 shows an example of the map-maker's view of our experimental world of size 7×5. The landmark enclosed by the square is the start point and the landmark enclosed by the circle is the goal. Because there are no occlusions, objects appear the same from all viewing directions; this simplifies custom map creation. In general, however, object models must address the characteristic view problem, the occlusion problem, and simultaneously be flexible enough to accommodate multiple kinds of descriptive features that the sensors of the navigator can identify: size, color, texture, shape, number of components, topological relations between components, etc.

2.2.2 Navigator model

We used a CCD array camera mounted vertically on a robot arm as the navigator's means of observing its environment. The height of camera was fixed above the world model and it could only move in a plane parallel to it. This obviated collision detection algorithms. Because viewing was normal to the ground plane and fixed, there were no occlusions nor even choices of field of view. Field of view was a fixed window, and very small relative to the navigational environment. At most two landmarks were visible to the navigator at any instant; nearly always there was a fraction of one or less.

The navigator is capable of operating in two different phases: *seek phase*, and *adjust phase*. It is assumed to have no metric capabilities outside its view window, and within its view window it perceives by a type of symmetry calculation. During seeking, the navigator continuously looks for any object coming into its viewing window. When an object is spotted on the boundary of the window, the navigator adjusts its position so that the object is centered in the view window. Adjustment not only counteracts errorful drift; it may be strictly necessary on turns to properly align the next heading. Other than seeking and adjusting, the navigator is not intelligent and makes no decisions; the map-maker has made them all. (The opposite extreme is that of a navigator with no *map* at all: everything depends on the intelligence of the navigator's search.) Navigation succeeds if the navigator attains the final object description.

2.2.3 Map-maker model

In the level helicopter world, each direction in the custom map has the format (*heading, description*). Headings are the four compass points, and the partial object description is simply the leading edge of the 3-by-3 matrix of dots, in the direction of travel. For example, (*north*, '...'), (*west*, ' .') is interpreted as travel north until you see three dots in a row, then travel west until you see a single dot on the right (i.e., the north). There does not appear to be any advantage in encoding the description further, either in the custom map, or in any of the intermediate data structures used by the map-maker. The object itself, though impoverished, has four different views of three tokens each, and they are trivial to obtain; the object itself has no more than nine tokens. Depending on the ease with which characteristic views can be obtained from objects in general, it may happen that the best description of an object is the object itself. The computational cost of obtaining and storing all views, most of which by definition will not be visible or useful in navigation-in-the-large would be even more excessive than in the special case here.

2.2.4 Definitions of custom map quality

Since there are typically multiple custom maps to travel between any two landmarks in the grid world, it is necessary to define a quantitative model for evaluating the quality of a custom map. There are a number of metrics to measure the cost of a path. Each metric may lead to a distinct path, so each metric may lead to its own heuristics. The choice of metrics depends on the goals of the map-maker and the navigator, and on the characteristics of the grid world. Two possible goals are minimizing travel time and maximizing the probability of reaching the goal.

One obvious cost metric is distance, measured by the Manhattan metric. A second is sensor cost. In our model, this is the number of times along the path that an object is encountered, which reflects the number of times the navigator needs to stop and check if it has reached the next specified object in its set of instructions. In general, the robot may have different sensors with different costs, or the cost to verify an object may vary depending on the landmark. Time of travel is a third possible metric. Relevant data would include speed limits, obstructions, and sensing times for the available landmarks. Speed limits are generally dictated by law or by roughness of the terrain. The average waiting time for certain obstructions, like traffic lights, may vary depending on the direction of travel. A fourth metric is map length, which is the number of directions that the map-maker gives to the navigator to get from one point to another. This metric measures map transmission costs and memory requirements. A fifth gives preference to straight paths; Elliott and Lesk [15] noted that the number of turns and the direction of the turn also affect the time of travel, and therefore counted right turns as 1/8 mile and left turns as 1/4 mile.

All of these metrics tradeoff with reliability, which generally decreases with distance traveled. This suggests a class of non-monotonic reliability metrics, which would penalize both very short and very long unit direction distances. A second class counts as its cost ¯some function of the confusability of the objects along or nearby the specified path: some objects are not only unique, they are unique in robust ways. A third class gives preference to "rear view mirror" navigation, that is, prefers landmarks that are distinctive on both sides, so that landmarks can be verified once obtained.

Of course, all of these metrics can be combined in numerous ways. And some or all of these metrics may cause the navigator to back up. For example, suppose the navigator is due*west* of the destination, all landmarks between the navigator and the destination, including the destination landmark, are red, and a green landmark is found due *east* of the destination. It is easier to specify the instructions to the destination as "go *east* until you reach the green landmark, then go *west* to the

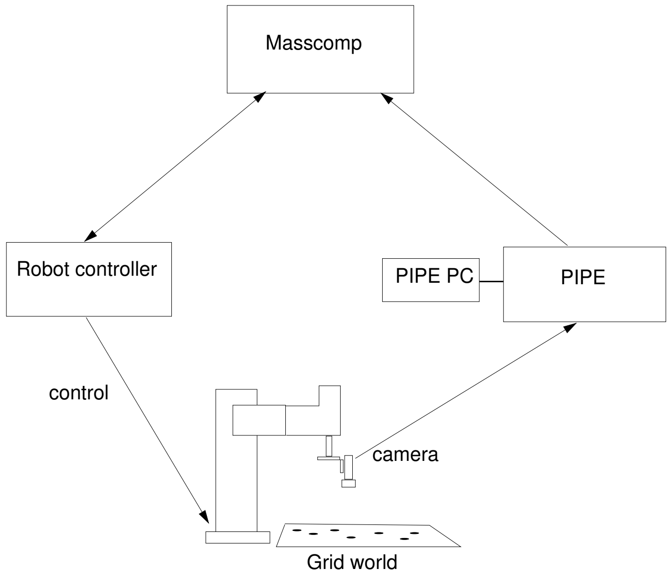


Figure 2.2: Configuration of the experimental equipment

red landmark."

2.2.5 Implementation

The world, map-maker, and navigator were implemented. A Masscomp hosted the map-maker, and the high level control of the navigator. Basic image processing was performed using a PIPE (Pipelined Image Processing Engine) system [4]. The navigator was a camera with a 12.5mm lens mounted on an IBM 7575 robot arm [27] programmed using IBM's AML/2 programming language [28] (Figure 2.2). The logical components of our system consists of three main modules. The first, an environment creator, has a user-friendly interface for entering object descriptions and object positions. The second, the navigator module, reads the set of commands generated by the map-maker program and performs the navigation. During the seek phase, the leading edge of the image is monitored by the PIPE. Upon detection, it adjusts and attempts to verify object identity by its dot signature. Some calibration was necessary to ensure that during the monitoring movement was not fast enough to "lose" a dot. The third component, the map-maker, is the most complex. Much of its efficiency came from the exploitation of the observation that unit directions could only travel within rows or within columns of the world. Being in the same row/column is a necessary but not sufficient condition for an object to be reachable; it also could not be "shadowed" by an object that had an identical leading edge. Note also that reachability is not symmetric.

When every pair of landmarks has been examined for reachability, we can produce sets of connected components. We defined such a set as *highway*. For a given world, multiple highways can exist. Thus, the navigator can travel from one landmark to another only if these two landmarks are on the same highway. Computing *highways* can be done in $O(M+N+K)$, where M and N are the row, column sizes of the grid world and K is the number of landmarks populated. In $O(M+N)$ we bucket sort the cells, once as (i,j) and once as (j,i). Then we have only to find the edges between adjacent points in rows and columns. Since we have K landmarks, the result is a graph with K vertices and $O(K)$ edges; in $O(K)$ time we find its connected components. This implies that it takes $O(M+N+K)$ to find out whether the goal is a place that "you can't get to".

There are two main data structures used by the map-maker module, the transition matrix and the cost matrix. The entry (i,j) in the transition matrix is filled with a direction - *north, east, south* and *west* - and a descriptive feature value of the landmark l_j if it is reachable from landmark l_i. For example, if l_j is north of l_i and its south edge has value 7, i.e., three dots in a row, then the transition matrix entry (i,j) would be $(north, 7)$. The cost matrix contains the cost of such a transition. In our implementation, we computed three different optimality criteria: sensing cost (s), distance traveled (d), and the map length (m). Distance traveled is defined as the number of cells the navigator travels over. Sensing cost is the number of landmarks encountered on the way. Map length is the number of instructions given to the navigator.

The map-maker computes the optimal route, based on the optimality criteria, by Dijkstra's shortest path algorithm. The time complexity for Dijkstra's algorithm is $O(n^2)$ if no fancy data structure is used [3], but if a Fibonacci heap is used as a priority queue, Dijkstra's algorithm takes $O(e + n \log n)$, where e is the number of edges and n is the number of vertices.

Another way of finding the optimal path is to use a heuristic search algorithm such as branch-and-bound or the A^* algorithm [61]. These algorithms are more costly than Dijkstra's algorithm in the worst case, but if the heuristic function is "good", they can be very effective. We found that in the case of sensing cost (s) estimation or map length (m) estimation, the underestimating heuristic that assigns a cost of 1 to objects in the same row/column, and 2 elsewhere occasionally gives dramatic improvement. Also in the case of distance cost (d) estimation, using the Manhattan distance metric between current position and the goal as underestimating heuristic gave us good results.

Note that optimal paths can be computed only if the start and goal landmarks are on the same highway. Inter-highway jumping methods are possible, but costly and possibly inaccurate. We have defined several *wandering* strategies, which are basically a space-filling zig-zag movement of the navigator towards a certain direction. The space that is filled can be rectangular with a specified width (a width of 1 gives standard navigational seeking), or triangular, which eventually fills a specified quarter of the plane, or semi-triangular, which fills an eighth.

Although it is difficult to analytically determine the expected number of highways given a specified object density, a series of experiments suggested that the number of highways is at a maximum when object density is about .03. Below .03, objects tend to be completely isolated; above .03, highways start to link up into one world-wide highway. An analytic proof would have some of the flavor of the proof of the birthday paradox: to start a new highway, each additional object would have to avoid all existing highways, which becomes increasingly difficult.

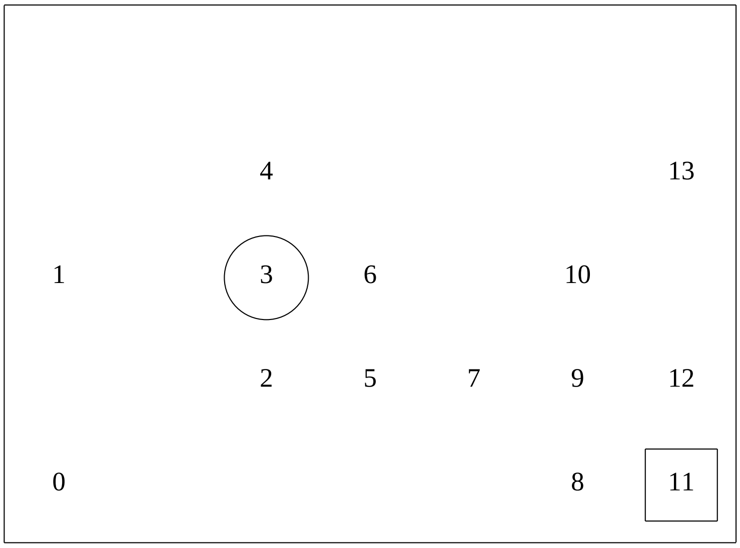


Figure 2.3: Labeling of landmarks in the sample dots world

2.2.6 Experimental results

For our experiments, we have allowed one primary cost optimality criteria and one secondary criteria. Six different paths were possible: sensor-distance, sensor-maplength, distance-sensor, distance-maplength, maplength-sensor, and maplength-distance. The difficult part was coming up with a set of landmark placements so that the paths generated by the map-maker are as distinct as possible. There seemed to be no apparent heuristic to do so, although it is clear that $m \leq s \leq d$. Using the landmark layout in figure 2.3, we have the following paths:

```
d-s : 11  8  9  10  6  3    (d=6, s=5, m=5)
d-m : 11 12  2   3          (d=6, s=6, m=3)
s-d : 11 12 13   4  3       (s=4, d=8, m=4)
s-m : 11  0  1   3          (s=4, d=10, m=3)
m-d : 11 12  2   3          (m=3, d=6, s=6)
m-s : 11  0  1   3          (m=3, d=10, s=4)
```

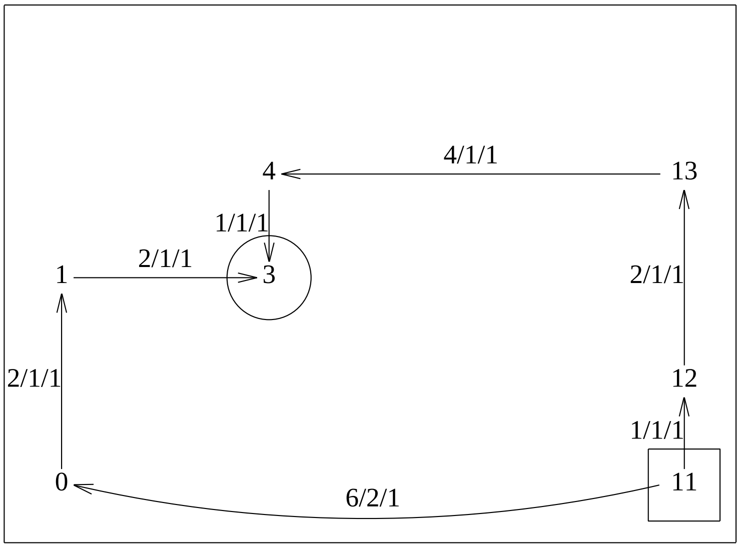


Figure 2.4: Paths generated by the map-maker

Figure 2.4 shows two of the optimal paths (s-m and s-d) generated by the map-maker. The numbers on the arcs indicate costs $d/s/m$. The algorithm to derive optimal paths that satisfies the primary and the secondary criteria had to be carefully designed in order to prune the exponentially large number of paths created. There are several ways to solve this problem. The first is a n-stage search method, where n is the number of optimality criteria being used. Using the primary criteria, the algorithm creates a tree of primary optimal paths, then in the next round of search, considers only the edges that appeared on the primary paths. If we have n optimality criteria, we apply this n times. The second method uses n cost matrices, but computes the optimal paths in a single pass. The search is done using the primary cost matrix and when there is a tie, the secondary cost matrices are considered. This second method increases matrix computation time and space to save search time. A third method uses a single compressed cost matrix and a single-pass. Each entry in the cost matrix is an algebraic combination of primary and secondary costs, encoded in odometric fashion. A single pass of search would suffice. However, one drawback of this method is that if the number of optimality criteria is very large, we would have to use a very large number of bytes to hold each cost matrix entry; further, the computation involving these numbers would also be costly.

2.3 Summary

In this chapter, we presented our previous work in the areas of topological navigation in one and two dimensional spaces. In section 1, we described the initial work of Kender and Leff [31] that provided a theoretical framework for "environmental navigation" in one-dimensional space. One of the most important contributions of this work was in the formal proof that visual landmark selection and direction giving are in fact a *very* hard problem - even in a simplified error-free one-dimensional space. In section 2, we described the "level flying helicopter" navigator, which was a formalization of two-dimensional grid world navigation. A model of topological visual navigator was implemented using a camera mounted robot arm. By using several important criteria for creating custom maps, such as sensing cost, distance of travel, and communication cost, we were able to show the effectiveness of direction giving based on topology of landmarks.

Chapter 3

Definitions

In this chapter, we define basic terms to be used throughout the rest of this dissertation. We describe our models of the “map-maker” and the “navigator”, and define their capabilities and limits. To model a large unstructured environment, we propose the use of randomly populated “dots” world and explain the rationale behind this simplification.

This chapter is organized as follows. In section 1, we discuss our basic assumptions, and explain the objectives of the map-maker and the navigator. Section 2 describes the abstract representations of the world in accordance to the three different views of the world: the real world itself, the map-maker's view of the world, and the navigator's view of the world. In section 3, we give some details in the experimental setting for the implementation of the navigating agent.

3.1 Assumptions

There are two major modules in this project, namely, the map-maker and the navigator. In this section, we present the definitions of these two modules and describe their objectives and capabilities.

The ultimate objective of the map-maker is to generate a "custom map", consisting of directional instructions, so that the navigator will be able to find its way to the destination by executing each of the instructions in a sequence. The map-maker is assumed to be omniscient and error free. It sees the whole environment and knows the exact position of each objects that exists. The map-maker also knows the ability and the limits of the navigator. Therefore, the custom map contains directional instructions that the navigator can handle. The communication between the map-maker and the navigator is done off-line (one-way, one-time). This means that the custom map is given to the navigator in the beginning of the journey and the navigator's only source of directional information is the custom map.

The navigator's capabilities are much more limited. Its view window size is very small relative to the environment. It has limited metric measurement capabilities only within its current view window. However, the accuracy of the its metric measurements is assumed to be low. For example, the navigator can move the window position so that a visible reference object is positioned near one of the four corners of the view window, but it does not have the accuracy to pin point the coordinates of a visible object. Therefore, it is not possible for the map-maker to give, to the navigator, the (x, y) coordinate of a landmark as part of the directional instructions. The navigator is not intelligent enough to decide on its own what it should do, thus totally dependent on the custom map that it is given. It does not have any memory. Therefore, the custom map can be considered as the navigator's intelligence. Table 3.1 summarizes the assumptions that we make on the map-maker and the navigator.

	Map-maker	Navigator
Objective	Generate custom maps	Navigate using custom maps
Visibility	Infinite	Limited to current view window
Metric ability	Yes	Only within current view window
Intelligence	Omniscient	Limited by the custom map
Memory	Large	None
Computing power	Fast	Slow
DOF	0	2 - (x, y)
Communication	Off-line	Off-line

Table 3.1: Capabilities and limits of the map-maker and the navigator

3.2 The World, the map-maker, and the navigator

We distinguish three similar but subtly different perceptions of "the world".

The real world The three-dimensional world as it exists and which is experienced by both the map-maker and the navigator. This is the real world and it is mathematically rich: it is continuous, it has a distance measure, and objects "embedded" in it can have finite extent. Much of this appears to be extraneous to the navigator: many such worlds consist largely of the essentially empty spaces between objects. Therefore, we need to abstract this world into something more simple.

The world model of the map-maker The first level of abstraction is done by the map-maker who is omniscient and error-free, into a graph. That is, the world is conceived as a collection of nodes which represent landmarks, connected by arcs which represent navigable regions without intervening objects. Empty space, distance, and object extent are ignored. Actually, these can be and are considered in the cost matrix by the map-maker. The data structures used in this level of abstraction are parkways and trajectories. The formal definitions of parkway and trajectory will be given in Chapter 4.

The custom map The next level of abstraction occurs when the map-maker communicates (one-way, one-time) with the navigator in the form of a sequence: this is, by some criterion, the best custom map of visual landmarks, ordered by sequence of traversal. We postulate that this custom map is the optimal amount of information that the navigator needs in order to navigate. We assume that there is only one navigator, and that the map-maker's omniscience extends to a perfect model of the navigator's sensory endowment. The navigator perceives the world in the most limited way. Sensory range is limited, to a relatively small area of the world (view window), and there is virtually no memory. However, this is an actual advantage; usually, the less that needs to be sensed or remembered the better, since the cognitive load of the navigator is minimized [67]. Although more intelligent navigators can be modeled, this one chooses to ignore most of the world and its features for the sake of efficient traversal. The custom map [1] can is represented in the form: $(D, M)^*$ D is a qualitative description to distinguish a landmark(s) from its neighboring objects. M is the movement of the robot in the local coordinate using the chosen landmark(s) as the reference point. For example, the *goal* object in figure 3.1, which is the dot located at the top, can be achieved by a custom map as follows: ((*, SW), (Isol, SW), (Obvious, SE), (New, SE), (Isol-pair, SW), (Isol, -)). Here, the symbols $*$ and $-$ are used to indicate "any" and "null", respectively. The initial position of the robot is indicated by the box at the bottom of the diagram. The first instruction of the custom map says "choose any object visible and move the view window so that it becomes positioned at the *south-west* corner of the view window". The result of the robot's movement puts the robot in the position indicated by the next box (second from the bottom). Here, the robot sees many objects, but since the next command is (Isol, SW), the

[1] In Chapter 7, we present the formal grammar of the custom map that includes the trajectory and the error recovery instructions.

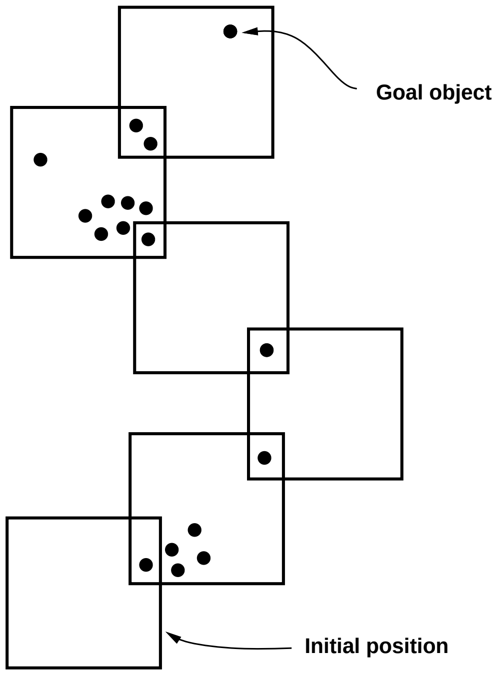


Figure 3.1: An example of navigation using a custom map

robot chooses the isolated object and moves to the next position. The rest of the navigational path instructed by the custom map is depicted likewise.

Designing the "description language", such as isolated, obvious, new, etc., to be used in the custom map is a difficult problem. We describe these issues in Chapter 5. The description language is ultimately used as commands in a custom map.

3.3 Experimental setting

3.3.1 The navigable world

The navigational environment that we are interested in is a three dimensional world, although the current implementation of the navigator (due to its restriction of degrees of freedom) makes the effective environment two dimensional. The navigational terrain itself is a flat surface that is visually uniform. Objects that will be scattered over this flat surface are spherical objects, such as marbles, uniform in size. There is no restrictions on where the objects are placed, except that no objects are allowed to be placed on top of another. The two assumptions of the objects - that they are uniform in size, and that they are placed randomly - emphasize the spatial and topological problems of doing vision and navigation "in the large". That is, firstly, an object can no longer be described by its intrinsic attributes, such as shape, size, or color. Therefore, in order to describe an object, the geometrical relationship of the target object to its neighboring objects must be considered. Secondly, the directions of the movements of the navigator have no external reference to rely on. Our goal is to be able to throw a number of marbles on a table and have our robot successfully navigate through this random world.

In a more complicated world that has a more variety of objects, we can (and often humans do) still apply the same abstraction. For example, in the left diagram of figure 3.2, we see a distribution of houses and trees. In the simplest form of abstraction, every house or tree can be regarded as a point-like object, and the only information extracted by the map-maker is its location in the environment. This is shown on the right diagram of figure 3.2. Another way of abstracting the world would be to categorize the objects according to their types (using whatever categorizational strategy is available to the navigator), and produce n (number of object types) separate dots world. This is shown in figure 3.3. So, if there are features that are sensible by the navigator, they are used to filter the perceptions of the map-maker and the navigator, so that the world (at least for that single command (D, M)), consists only of a point-like space of objects of that type.

3.3.2 Robot movement capability

Our navigator robot has 2 degrees of freedom, in x and y. It is able to view small subarea of the environment looking straight down perpendicular to the en-

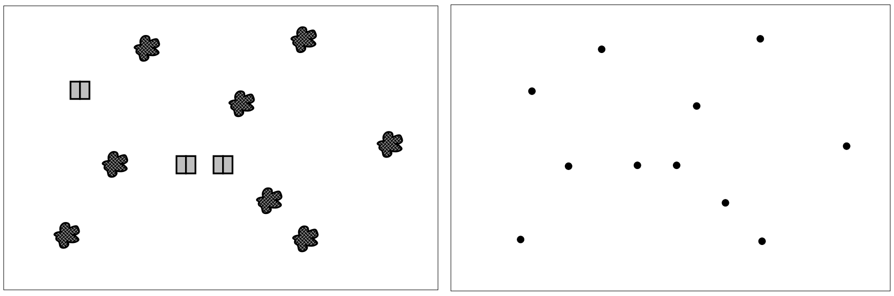


Figure 3.2: A world consisting of trees and houses, and the corresponding dots world representation

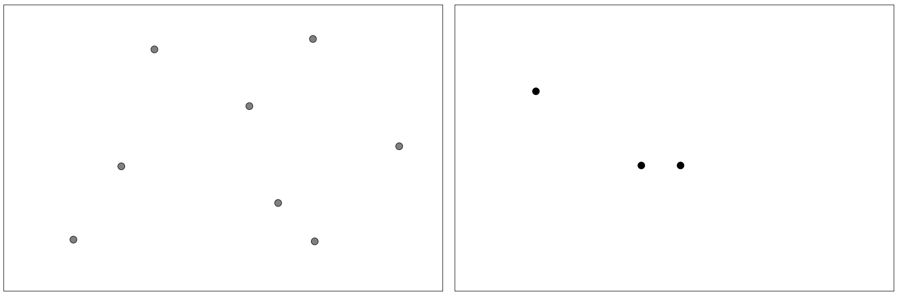


Figure 3.3: A separation of the dots world according the the navigator's sensibility: only trees, or only houses

vironment and z component of the navigator is fixed. Thus we are still in the "level-helicopter" world as in [32], except that we are no longer restricted to a grid structure. For our experiments, we implemented the navigator with an IBM 7575 SCARA robot arm with a CCD camera attached to the robot's end effector. The navigation terrain was the robot arm's workspace. Details are provided in Chapter 8.

Ultimately though, our navigator robot should have 4 degrees of freedom (x, y, z, θ): (x, y) to move to a desired (x, y) position in the random world, z to control the extent of visibility, and θ to look around its current position (pan). For simplicity, the robot navigator will not be given the freedom to control the angle of tilt of the camera. It will therefore be allowed to view the world only in a constant angle (see Chapter 9.)

3.4 Summary

In this chapter, we presented the definitions and assumptions for the map-maker and the navigator models. We explained the needs of data abstraction of filtering out unnecessary information so that the corresponding user (map-maker, navigator) can handle them appropriately. We also suggested the concept of "omniscient map-maker" and "dumb navigator". In this framework, most of time consuming and cognitively constraining tasks are handled by the map-maker model who has all the necessary tools and resources, prior to the navigation. The navigator, whose function is restricted to interpreting and executing commands provided by the map-maker, is assumed to have very little sensory and motor capabilities. This imbalance between the map-maker and the navigator subsystems is justified by the efficiency in traversal and the flexibility in the navigator model. This also enables us to find the lower bounds of the navigator's limits without sacrificing the performance. Finally, we described the experimental setting that was used for our experiments in the domain of navigation in large unstructured environment.

Chapter 4

Parkways and Trajectories

In this chapter, we describe two fundamental data structures, created and used by the map-maker for representing viable paths, called parkways and trajectories. They are graph data structures that contain mostly topological information of the environment. Parkways provide the backbones of viable paths, enabling the navigator to travel by following visible landmarks. Trajectories provide "junctions" for the navigator to jump between disjoint parkways, much the same way highway interchanges let drivers transfer highways. The first section discusses various issues related to parkways, including its formation, functions, and some analysis. Likewise, the second section describes trajectories.

4.1 Parkways

4.1.1 Parkway definition

When the map-maker captures the real world with its global sensor (in our case, a camera), it records all the positions of the populated objects. As stated earlier, we assume that objects are identically sized and shaped. Therefore, the

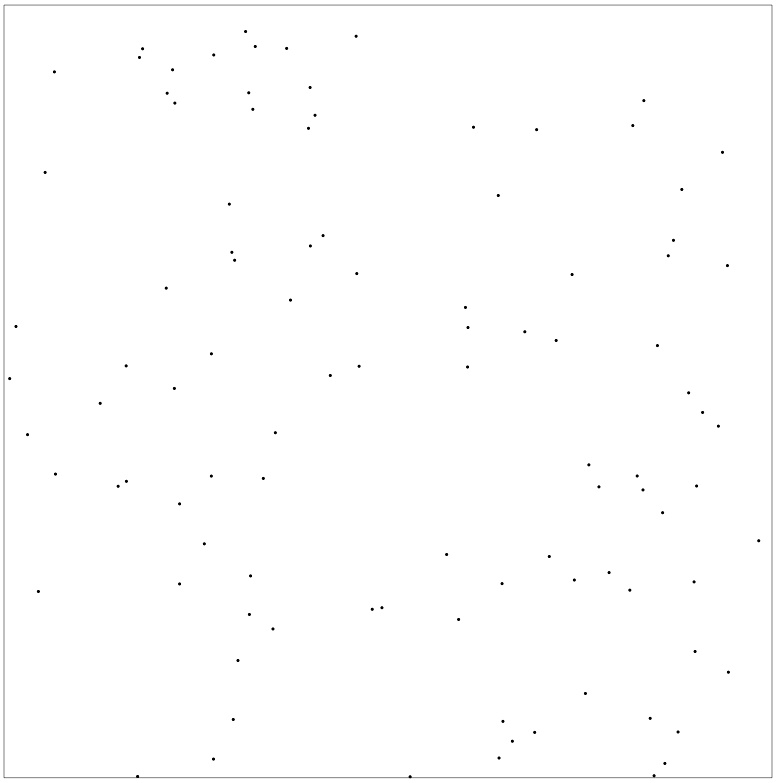


Figure 4.1: A randomly populated environment

intrinsic qualities of each object is not useful for navigation. Figure 4.1 shows an example of the map-maker's view of a randomly populated world with 100 objects. Only the positional information will be used for further processing. The map-maker further abstracts the world into a *graph* data structure, with vertices and edges. Each vertex represents an object and each edge indicates a viable path. There are many ways to decide whether or not two vertices (objects) in this graph are connected by an edge. In the most general case, we define two objects to be "connected" if the two objects can be viewed in the same view window of the navigator. This is the central idea of the parkway concept. Then, by applying the connected component algorithm [3] using the above definition of "connectedness",

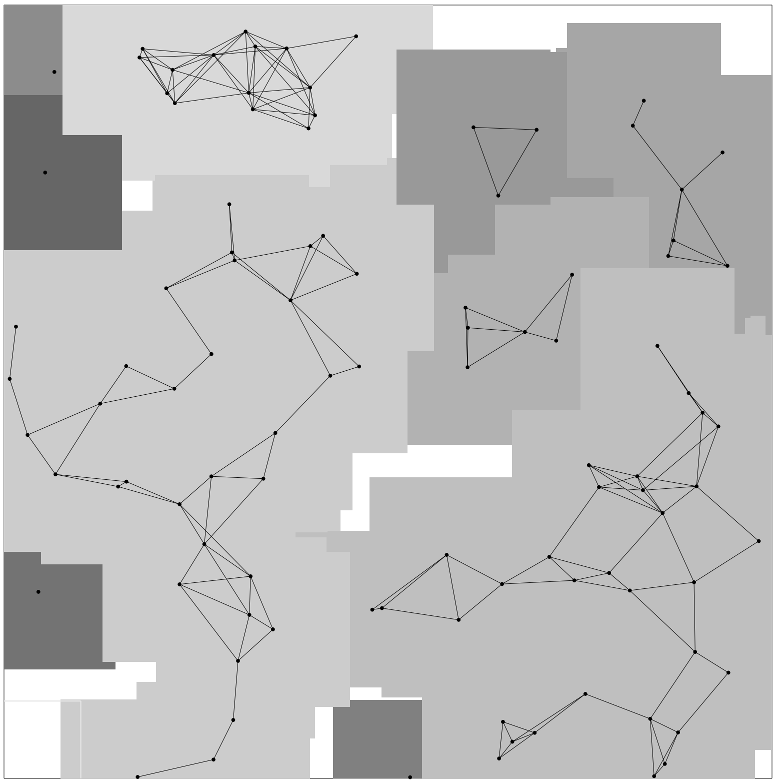


Figure 4.2: Parkways

we can generate discrete sets of connected components. We define a parkway[1] to be such a connected component. Figure 4.2 shows parkways formed on a world of 100 objects using a window size 10×10. The arcs between points indicate that the objects are connected. The regions with various shades indicate the extent of each parkway. These regions are defined as "territories". When a new object is added to the environment, it is absorbed into the parkway corresponding to the territory where the new object is placed. To find the shortest path between two objects in a same parkway, we apply Dijkstra's shortest path algorithm. The candidates of the next node to choose is confined to those objects which are connected to the "reference" object. A reference objects is a previously chosen landmark that the

[1]Parkway will be sometimes referred to as all-parkway when there is a need to distinguish it with other types of parkways (e.g. isolated parkway, that is, a parkway between isolated points).

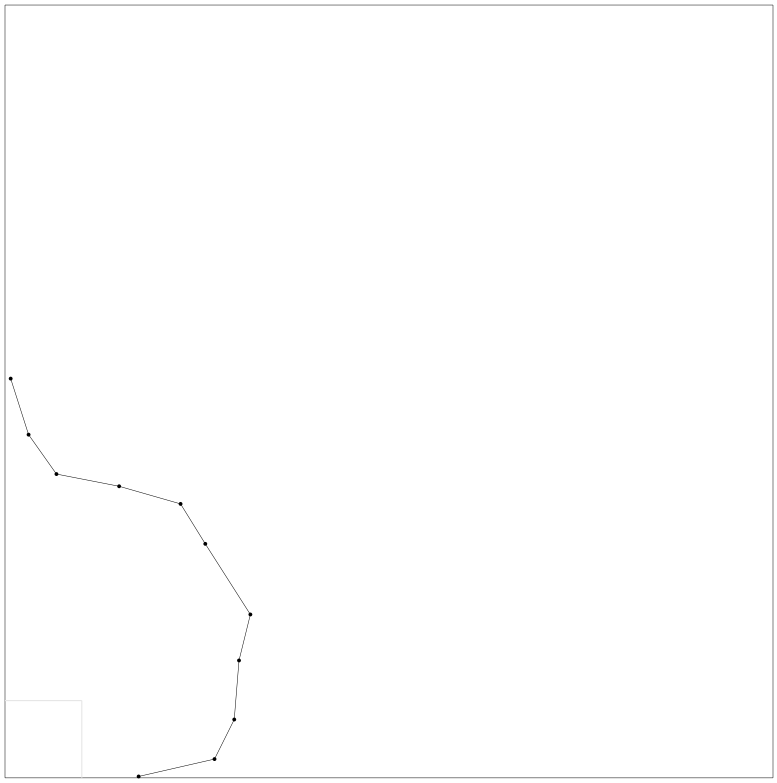


Figure 4.3: Shortest path within a same parkway

navigator uses as an anchor point. This way, we can guarantee that the robot does not move via metric information outside its sensible region, thus eliminating the possibility of a dead reckoned error. Navigation within a parkway is relatively "easy" because the navigator can simply follow visible landmarks. How easy it is to navigate within a parkway will depends on the complexity and reliability of the description language, which we define in Chapter 5. Figure 4.3 shows an example of the shortest path between two objects that are in the same parkway.

4.1.2 Types of parkways

In the previous sections, we defined parkways in the most general sense. Depending on the definition of "connectedness", there can be many different parkway types. In this section, we introduce only "obvious" and "isolated" parkways, but other parkways are also possible by using different descriptors (see Chapter 5).

Obvious parkways

If the number of objects within the view window exceeds 2, we need a qualitative way to describe to the navigator "what you see". This is a hard problem and will be discussed in more detail in Chapter 5. However, it is possible to define and construct a parkway that does not need the "scene descriptors". First, we restrict the position of the reference object to one of four corners in the window coordinate: SE, SW, NE, and NW. An object is an "obvious point" of the current reference object if it is the only other object that is visible. The parkway formed by obvious points is called an "obvious parkway". Figure 4.4 is the obvious parkway formed on our example world of figure 4.1. Note that an obvious parkway is a directed graph, and therefore the paths are not commutative. The all-parkway and the obvious parkway are in fact the upper and lower bounds [2] of the parkway types. In the all-parkway case, every visible object is connected to the reference object, therefore, the number of edges incident to the reference point node is maximum. In the obvious parkway case, only the lone visible objects are connected to their reference points, therefore, the number of incident edges is minimum.

Isolated Parkway

Given a scene with one or more objects, humans can identify "the most isolated" object. This can also be done computationally by using the *mutual neighborhood* concept. Detailed algorithm to derive the isolated point in a scene will be given in Chapter 5.

During the parkway traversal, the navigator has to keep track of the current

[2] Strictly, the lower bound on parkways is the error recoverable obvious parkway.

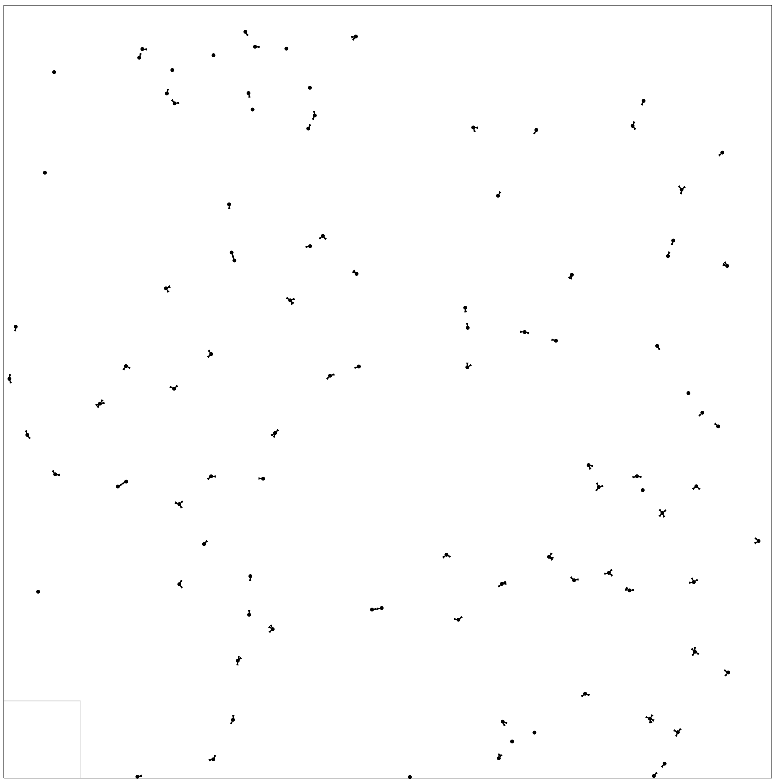


Figure 4.4: Obvious parkways

reference object, which is the landmark that has been previously obtained. When a new landmark is selected, it becomes the new reference point. An object i is defined to be the isolated point to another object r at quadrant $Q \in$ {SE, SW, NE, NW} if (1) Both r and i fit in the same view window with r positioned at the corner Q of the window. (2) Object i is the most isolated object in this view (excluding r). An Isolated parkway has an arc connecting two objects r and i and is constructed as follows.

1. For each object (call this reference point r in the subsequent steps) in the environment, place the window so that the object is positioned in the SE corner.

2. Derive the most isolated point i in the window.

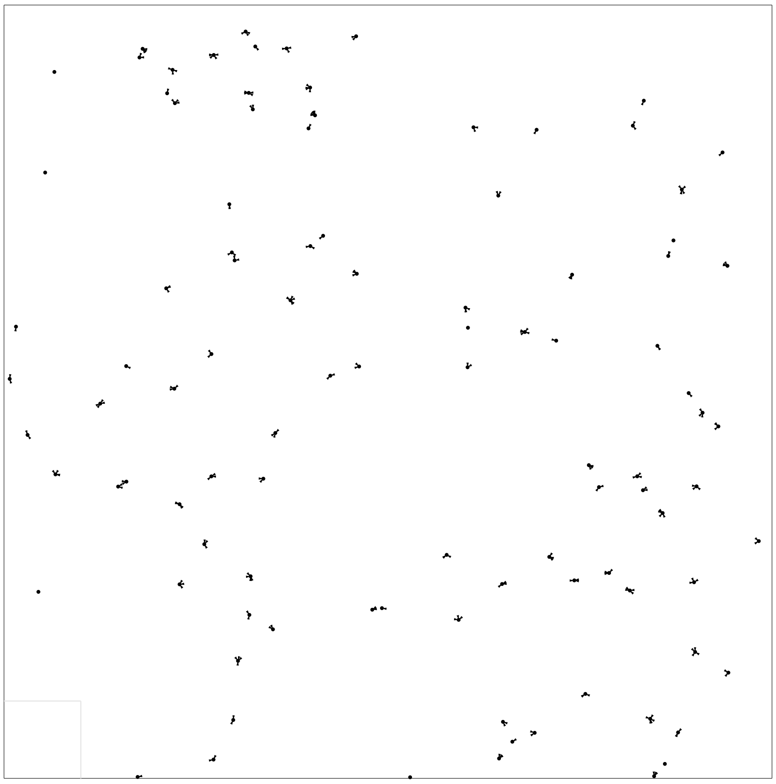


Figure 4.5: Isolated Parkways

3. If the reference point is the winner, disregard it and identify the runner up as the most isolate point i.

4. Update the transition matrix, $T(r, i)$, to 1. (Transition matrix will be describe in Chapter 7.)

5. Repeat with SW, NE, NW corners.

The resulting strongly connected components defined by the transition matrix is the isolated parkway. Figure 4.5 shows the isolated parkways formed in our world of 4.1. Note that an isolated parkway is also a directed graph, and that it is a superset of the obvious parkway.

Reliable parkways

Each parkway path is associated with a reliability measure, as will be explained in detail in Chapter 6. Then the parkway paths can be categorized by parameterizing a threshold value at a certain reliability value. For example, in order to generate a very reliable path, the map-maker may wish to use only parkway paths that have reliability measures greater than or equal to, lets say, 0.8.

Error detectable and recoverable parkways

During navigation, errors and mistakes are sometimes inevitable. In some errorful situations, however, it is possible for the navigator to either identify the error or verify that it is on the correct course. Furthermore, in some cases, the navigator can recover from an error by executing a simple recovery sequence. This process of identifying and recovering from an error is of a great importance [68], and thus will be explored in detail in Chapter 6.

A subgraph of a parkway that is entirely composed of error detectable directed edges (transitions) is called an error detectable parkway. Likewise, if the edges represent error recoverable paths, then the parkway is called an error recoverable parkway. For example, a subgraph of an isolated parkway that is error detectable is an error detectable isolated parkway.

4.1.3 Statistical analysis of parkways

We have carried out some experiments to see the relationships between the object density, the window size, and the number of parkways.

Let the world size be $R(= r \times r)$, the window size be $A(= a \times a)$, and the number of objects populated at time t be t. Then, the density of objects in the world is

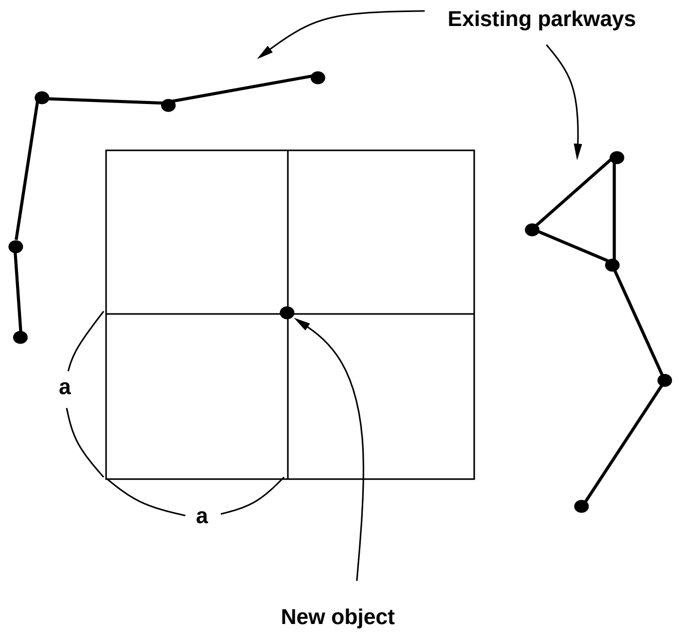


Figure 4.6: Formation of a new parkway

$\frac{t}{R}$, and the average population in a window is $\frac{A}{R}t$. Since we are assuming random placement of the objects, the number of objects in a window can be approximated by a Poisson random variable with parameter $\lambda = \frac{A}{R}t$. The probability of forming a new parkway at time t can now be calculated. We place the t-th object into the world and observe its surrounding area. If there are no other objects within one window sized area surrounding the new object (you can imagine placing 4 windows in a 4-connected way and so that the new object is at the center - see figure 4.6), then a new parkway is formed. Therefore, the probability of forming a new parkway at time t is $e^{-\frac{4A}{R}t}$ $(= e^{-\lambda}\frac{\lambda^0}{0!}$ with $\lambda = \frac{A}{R}t$ in four connected windows). In figure 4.7 the number of parkways formed versus the population of objects is plotted. The five line graphs from top to bottom are associated with window sizes 2×2, 3×3, 4×4 and 10×10, respectively. As stated before, all-parkway and obvious parkway are the upper and lower bounds of the parkway types. Thus in general, the size[3] of an average all-parkway in an environment is bigger than that of an

[3]Size of a parkway can be defined in many ways, but in general, the number of nodes in a parkway is a good measure.

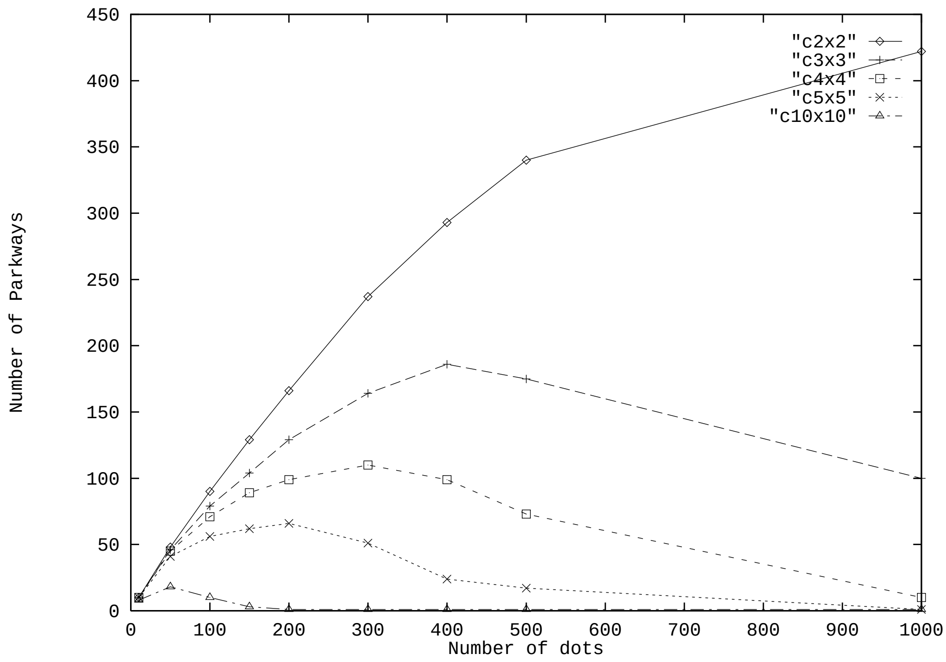


Figure 4.7: Number of parkways versus the population

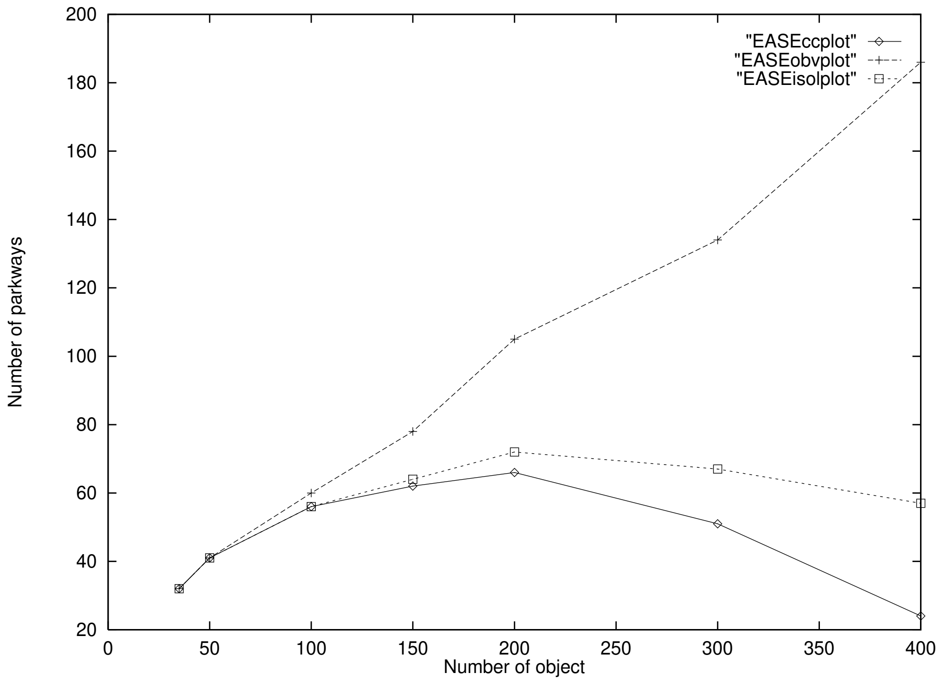


Figure 4.8: Number of all,obvious,isolated parkways versus the population

average isolated parkway or obvious parkway. Conversely, the number of disjoint all-parkways formed in an environment is less than that of an isolated parkway or obvious parkway. Figure 4.8 shows the comparison of the three parkway types.

4.2 Trajectories

Sometimes, start and goal positions can be in mutually disjoint parkways. In this case, following a parkway path will not be sufficient for the navigator to accomplish its task of getting to the goal. Therefore, we need a method to transfer between two parkways. At some point of the traversal, the navigator has to leave a parkway and get to the other parkway without getting lost. Recall, from our original assumptions, that the navigator does not have any metric capabilities outside it view window. Thus, we need a method that utilizes the geometric shape formed by currently visible objects. In order for the method to be robust, it needs to be insensitive (to a certain extent) to rotational, translational, and scale errors.

4.2.1 Trajectory definition

We now describe the proposed method. For each pair of objects in the same view window, we compute the feasibility of reaching a landmark in another parkway by sliding the window in the direction formed by the two objects. This is depicted in figure 4.9. The two objects in the view window form a directional vector. The third object on the upper right corner can be obtained by sliding the view window in this direction. The trajectory of the window movement is represented by a straight line. The inter-parkway paths generated by this method are defined as "trajectories". It is interesting to note that trajectories are not commutative and the graph formed by trajectories is in fact a directed graph. Measurement of the connectedness of the parkways is derived by applying strongly connected

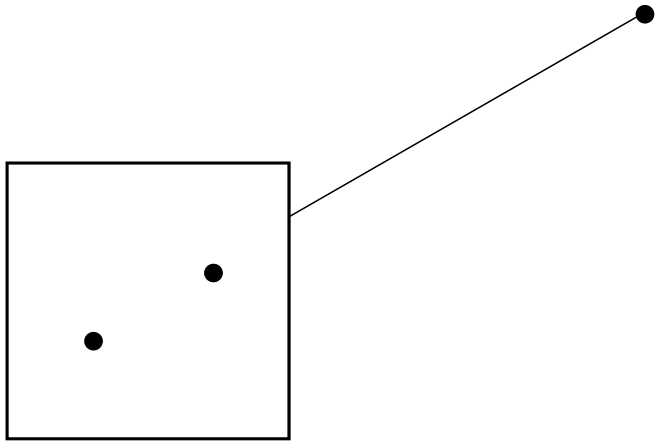


Figure 4.9: An example showing how a trajectory is formed

component algorithm on this directed graph [3]. Figure 4.10 shows an example of trajectories computed on our random world. The straight lines are the trajectories of the window in the direction of the sliding movement. One end of a line is attached to the window at the position to which the new object is expected to appear. The other end of the line is the object that this sliding window is seeking.

4.2.2 Implementation of trajectory traversal

In this section, we describe in detail the implementation of trajectory method to traverse between two disjoint parkways. The basic idea of the trajectory method is to "slide" the view window along the direction formed by two objects within the view window and to see which object gets entered in the window first Note that we are not pin-pointing any coordinates for the navigators movement, but the navigator's movement is solely based on the direction formed by two visible objects. In figure 4.11, we see a view window centered around two objects, $P_1(x_1, y_1)$, and $P_2(x_2, y_2)$. The direction formed by these two points, P_1 and P_2, is defined by the slope (m) of the line that passes through these two points. The trajectory target objects, are then the closest objects to this window along the directions

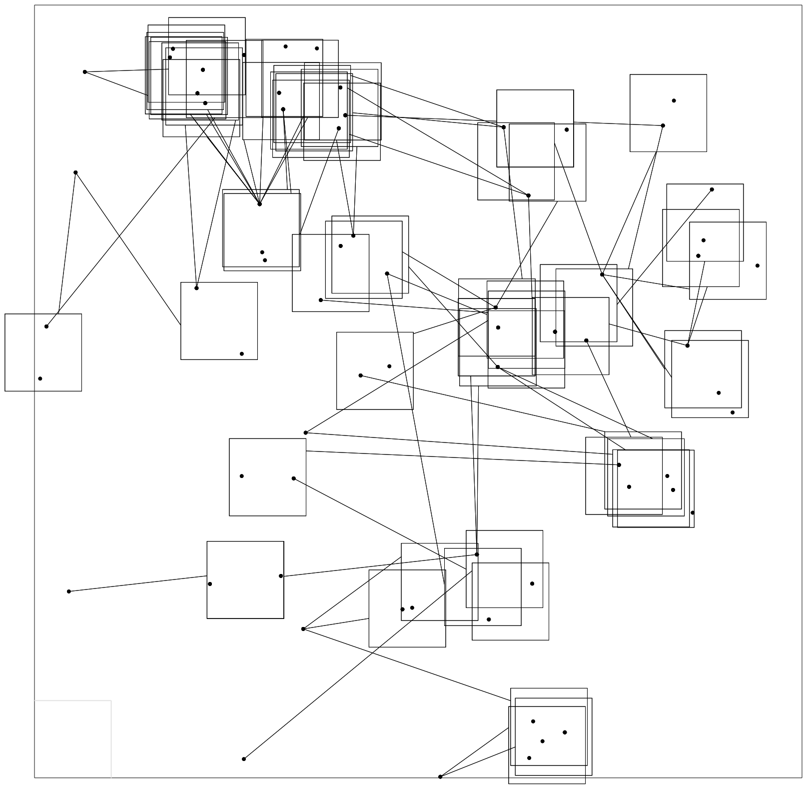


Figure 4.10: Trajectories

(left and right) defined by m, bounded by the two lines, Line1 and Line2. Let us first examine the case when $m \geq 0$, as in figure 4.11, where $m = \frac{y_1 - y_2}{x_1 - x_2}$. Then, we have the following four line equations, corresponding to Line1, Line2, LineA, and LineB, respectively:

$$y = mx - m\frac{x_1 + x_2 - w_x}{2} + \frac{y_1 + y_2 + w_x}{2}$$
$$y = mx - m\frac{x_1 + x_2 + w_x}{2} + \frac{y_1 + y_2 - w_x}{2}$$
$$y = mx - m\frac{x_1 + x_2 + w_x}{2} + \frac{y_1 + y_2 + w_x}{2}$$
$$y = mx - m\frac{x_1 + x_2 - w_x}{2} + \frac{y_1 + y_2 - w_x}{2}$$

The navigator can move in either direction (left, right) based on m. Therefore, for each pair of landmarks, there can be up to two different trajectories. When the

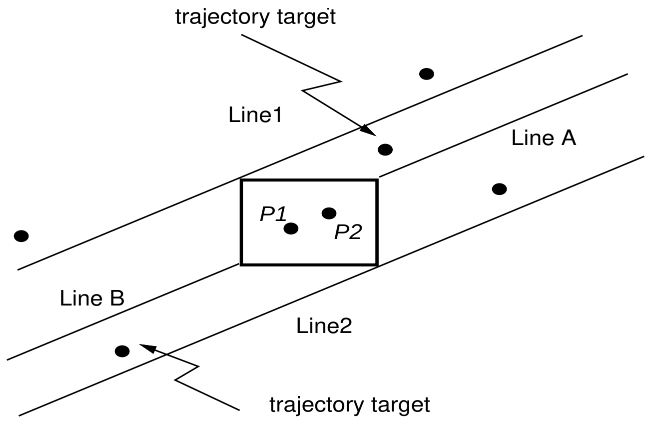


Figure 4.11: Formulation of a trajectory

navigator is moving right, two classes of objects are considered. The first class is the objects that are enclosed by the open polygon bounded by Line1, LineA, and the north part of the view window. The second class is the objects that are enclosed by the open polygon bounded by LineA, Line2, and the east part of the view window. From the objects that are in these two classes, the one which is closest to the view window frame is the trajectory goal object. When the navigator is moving left, also two classes of objects are considered. The first class is the objects that are enclosed by the open polygon bounded by Line2, LineB, and the south part of the view window. The second class is the objects that are enclose by the open polygon bounded by Line1, LineB, and the west part of the view window.

4.2.3 Types of trajectories

In order to form a trajectory, the navigator uses two visible landmarks. Since the navigator has a reference object at all times, it needs to identify a single landmark to make a pair. Depending on the description language, a different object can be selected as the a landmark. For example, using the isolated landmark descriptor, a trajectory is defined by the reference point and the isolated point. A trajectory formed by using isolated point descriptor is called an isolated trajectory.

Likewise, a trajectory formed by using obvious point descriptor is called an obvious trajectory.

As in the parkway case, each trajectory path is associated with a reliability measure. How the reliability measure is derived is explained in detail in Chapter 6. Trajectory paths can then be categorized by parameterizing a threshold value at a certain reliability value. For example, to generate a very reliable path, the map-maker can use trajectory paths that have reliability measures greater than, say, 0.8. In Chapter 6, we will introduce methods to detect and to recover from errors during trajectory movements. Associated with these methods are the error detectable and the error recoverable trajectories.

4.2.4 Statistical analysis of trajectories

As a result of forming trajectory paths, the number of (strongly) connected components decreases. This means the world is more connected as a whole. If there are no trajectory paths, the number of strongly connected components would be equal to the number of existing parkways. The upper curve of figure 4.12 represents the number of such strongly connected components redefined as a result of merging parkways that are connected via trajectories. The lower curve indicates the number of existing parkways. Since the trajectories are "one-way", the measure of connectedness can, at best, be expressed by strongly connectedness. Therefore, we feel that the graph indicates the upper bound (this means in reality, you can usually do better) on the measure of whether or not you can get from one place to another. Note that even at a relatively high population, for example at 450, the number of strongly connected components is not 1. This means that even by using both parkway and trajectories, there are still places that "you can't get to".

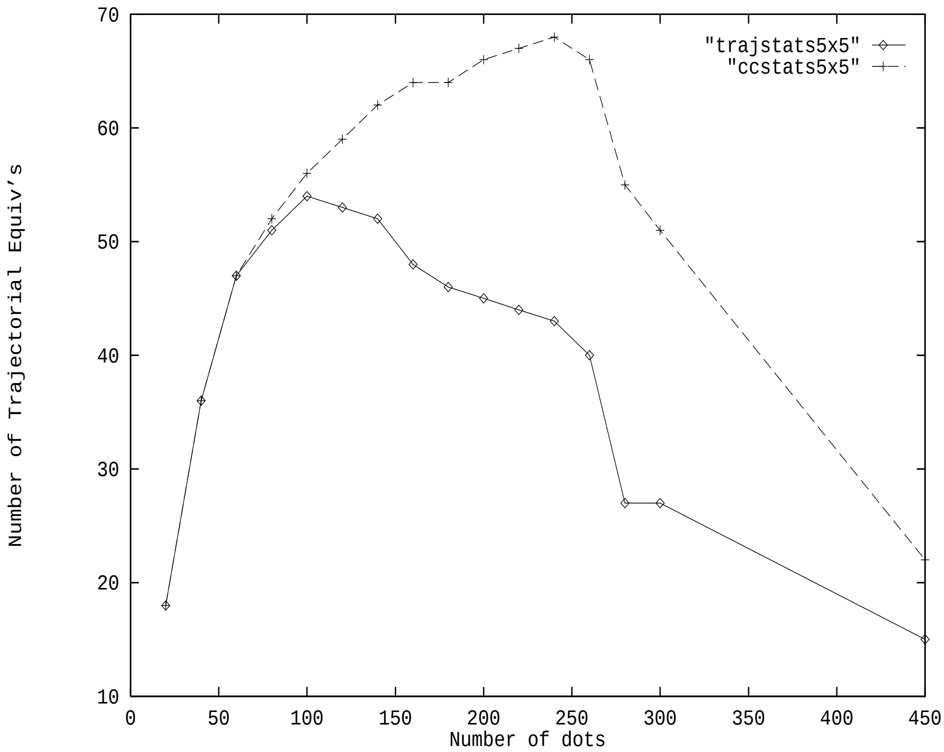


Figure 4.12: The number of strongly connected components as a result of applying the trajectory method

4.2.5 Overall shortest path

In general, the overall shortest path is a combination of parkway path segments and trajectory path segments. An example of a such path is shown in figure 4.13. The starting position is at the top right corner, and the goal position is at the bottom left corner. The parkway path segment is represented by single line segment, and the trajectory path segment is represented by a box accompanied by a single line segment.

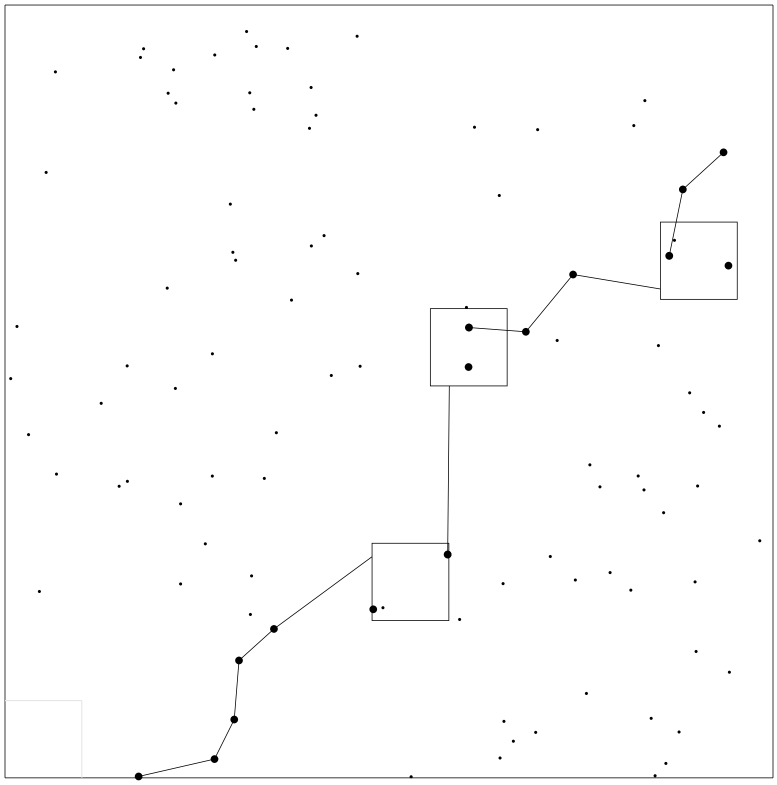


Figure 4.13: Combined shortest path

4.3 Summary

In this chapter, we described data structures called parkways and trajectories, which are topological representations of the world created and used by the map-maker. Parkways define navigable paths based on visible landmarks. Trajectories provide connections between disjoint parkways by allowing the navigator to move based on direction formed by visible landmarks. Any navigational route can be represented as a combination of parkway and trajectory path segments. We studied various types of parkways, including the all-parkway and the obvious parkway, which are the upper and lower bounds of parkway types, respectively. We also described how trajectories are computed and discussed the way trajectories connect existing parkways to provide more viable paths.

Chapter 5

Description Language

In the previous chapter, we examined the navigational environment as perceived through the eyes the map-maker, in which the world is abstracted into topological data structures called parkways and trajectories. This abstraction (representation) was necessary in order for the map-maker to plan paths topologically. When the path planning is complete, the map-maker needs to produce a corresponding custom map that "describes" to the navigator how to follow this path. In this chapter, we discuss various issues in the design and implementation of the description language, the vocabulary of which constitute the custom map. The navigator, whose metric capabilities are assumed to be low, needs directional instructions that are resilient to metric-related errors, such as scale, rotational, and translational errors. It is the map-maker's responsibility to come up with a custom map that will provide the navigator with just the right amount of information in the optimal form. Note that humans prefer this simplified custom map over more complete and redundant form of directional information which may overburdening [67]. This chapter is organized as follows. In Section 1, we discuss the issues in designing a language that can describe a given scene qualitatively. Section 2 describes the basic vocabulary that we have developed, the main function of which is to identify a single object in the scene. In Section 3, we present details in the implementations of some of our environmental descriptors in the basic vocabulary.

5.1 Problem statement

At any given instant, the navigator will have only a small portion of the world at its view, which may contain several objects. The map-maker has to be able to "describe" what the navigator sees in order for the navigator to be able to distinguish a particular object to use as the next reference point. This is a hard problem because of our assumption that the navigable environment consists of randomly placed point-like objects. If we assume that the robot has infinitesimal accuracy, infinite memory and an extremely fast processor, it could keep the bitmap image of each possible view, or at least the coordinates of each landmark within each possible window. However, this is neither practical nor feasible. Realistically, we need some invariants, such as colors or shapes, to describe the immediate environment. Since our world is a "monochrome and point-like" world, we need a qualitative language that can describe the geometrical relations of the visible objects. In other words, we need qualitative and concise descriptions to represent "what you see". The level of detail in the description process would depend on the intelligence and the mobile ability of the navigator.

The simplest non-metric method of discriminating an object would be to enumerate all the objects left to right, top to bottom (*linear ordering*). But this method is very sensitive to the orientation of the camera. If the navigator's camera is even slightly misaligned to the original direction, this method would fail. A slightly more robust method would be to enumerate the objects in a circular order. This would involve a "wrapping around" algorithm, similar to the "gift wrapping" method to derive convex hulls [59]. This method, let us call it *spiral inward* method, is not susceptible to the errors in the rotational angle of the camera that can be critical to the *linear order* method. A *spiral inward* method starts at the current reference object and selects the next object by rotating an imaginary line clockwise until it hits a new object while pivoting one end on the reference object. This process is repeated from each of the selected objects until every object is touched. The starting angle of the line in the beginning can also be encoded. This could eliminate some ambiguities of which object gets selected first. For example, if the current reference object is in one of the four corners, SE, SW, NE, NW, then a possible starting angle would be 180, 90, 270 or 0 degrees. Time complexity of this method is $O(k^2)$, where k is the number of visible objects. Unfortunately, this method too, is very sensitive to small errors, such as calibration error.

A more general method of scene analysis would involve a two step (large scale, small scale) process. In a large scale, computationally determining the clustering of objects would help to narrow down the "candidates" and is a research area by itself. Once the objects have been divided into clusters, the next stage can describe the individual clusters. Known methods for describing individual clusters, or *dot patterns* or *dot figures*, include the *external shape* description method in [14], *internal shape* description method in [60] [72], and a fuzzy logic classification in [2].

Describing a set of points by humans seems to involve dividing and organizing the points into "clusters" or "gestalts". Many cluster detecting algorithms have been developed. These methods are generally divided into hierarchical and non-hierarchical methods. The details can be seen in [23]. Computational methods of analyzing a cluster include algorithms that build data structures such as nearest neighbor graph (NNG), minimum spanning tree (MST), relative neighborhood graph (RNG), Gabriel graph (GG), Delaunay triangle (DT) [60], and sphere of influence graph (SIG) [72]. These methods are considered to describe the *internal* structure of the point sets, whereas the convex-hull or α-hull methods describe the external shapes [14]. A NNG is formed by simply connecting the vertices to produce all nearest neighbor pairs. MST connects every node to some other nodes in such a way that the total length of the edges is minimized and all members of the point set are connected. Zahn [76] uses MST in a graph-theoretical way to describe gestalt clusters. RNG is formed by connecting the nodes x and y if :

$$d(x,y) \leq min(max(d(x,z), d(y,z))|z \in S)$$

where S is the point set. GG is defined by the following relationship:

$$d(x,y) \leq min((d^2(x,z) + d^2(y,z))^{\frac{1}{2}}|z \in S)$$

DT, which is the supergroup of all the preceding groups (and note that every group introduced above is a supergroup of the former group), is the dual of the well known Voronoi diagram. Definition of the SIG from [72]:

> Let $S = x_1, x_2, ..., x_n$ be a finite set of points in a plane. For each point $x \in S$, let r_x be the closest distance to any other point in the set, and let C_x be the circle of radius r_x centered at x. The sphere of influence graph is a graph on S with an edge between points x and y if and only if the circles C_x and C_y intersect in at least two places.

The SIG was originally introduced as a low level primary sketch of a *dot patterned* scene. But it has several properties that other higher level descriptive methods offer. First of all, it delivers either "external" or "internal" shape description in the form of "shape-hulls" or "skeletons". Secondly, the output is not restricted to one piece like most of the other "external" descriptors. Last but not least, it does not require a magic number to compute[1]. For comparison, although α-hull is a very elegant method to describe the external shape of a dot pattern, the parameter, α has to be manually set.

Using these clustering algorithms is not enough to solve our problem of discriminating an object from the rest. It only reduces our problem into sizes more manageable. We still need an discriminating method, such as the one introduced in an earlier section, to distinguish a single point within the cluster.

[1]Later in this chapter, we experiment with using magic numbers in SIG.

5.2 Basic vocabulary

In this section, we introduce some of the basic vocabulary that we have so far developed as part of our description language. An object is defined to be "obvious", if it is the only other object that is visible except for the reference landmark. Two or more objects are defined to be "confusable" if it does not matter which object is to be chosen as the next reference point. The result of the next move of the navigator using any one of the "confusable" points should get the navigator to the same place. The term "new" objects refers to the objects that newly came into the view window as a result of robot's movement. The term "isolated point" refers to a single isolated object as a result of applying a clustering algorithm. The term "isolated pair" refers to a single pair of isolated objects as a result of a clustering algorithm. Figure 5.1 illustrates these methods, and figure 5.2 shows an example of a custom map entries consisting of the basic vocabulary and corresponding actions of the navigator.

5.3 Some details

5.3.1 Identifying an obvious point

Identifying obvious point in a dot scene is very simple. By definition, an obvious point is the only visible object in the dot scene other than the current reference object. Therefore, the precondition for identifying an obvious point is that exactly two objects are visible. Implementation is trivial.

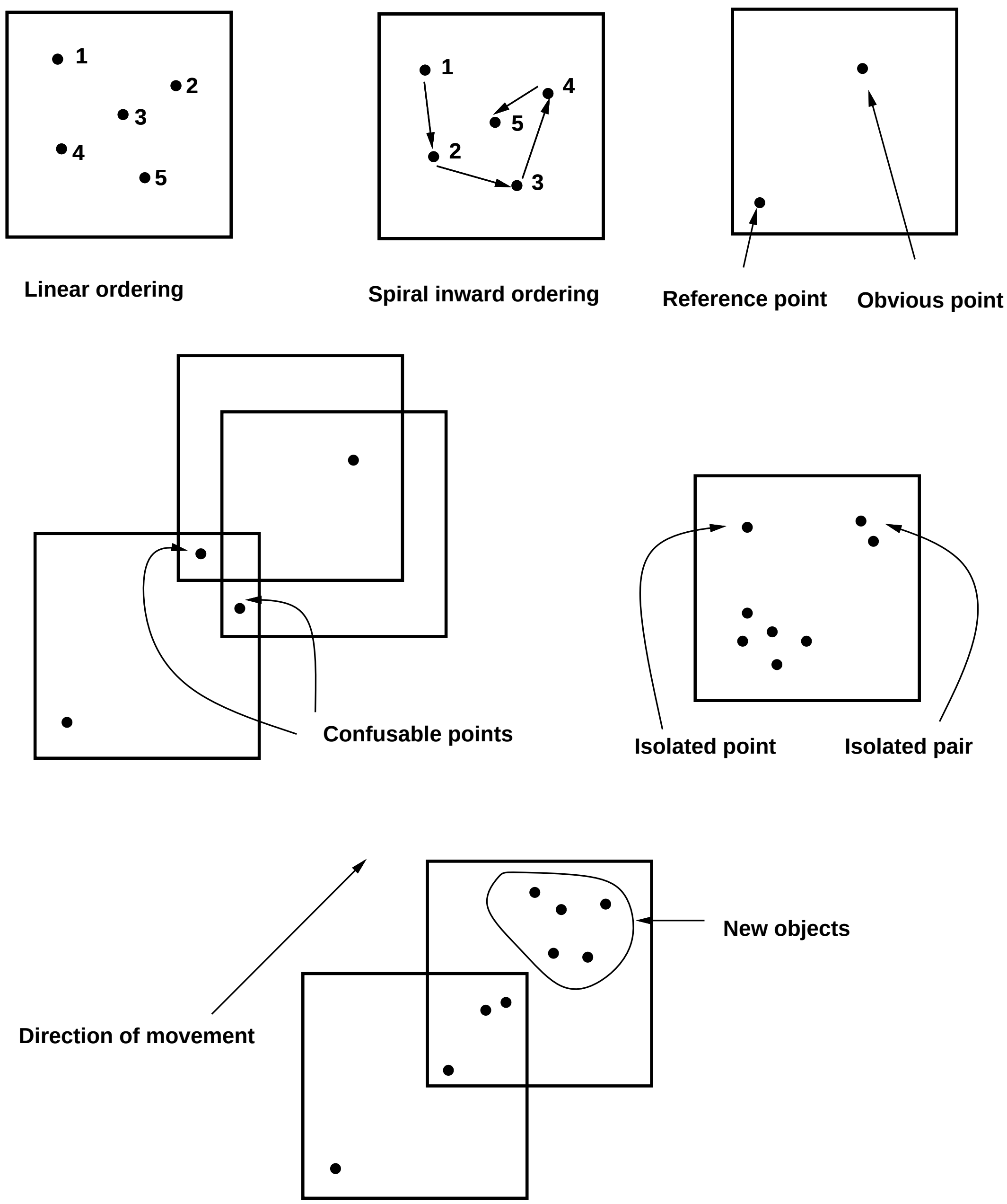


Figure 5.1: The basic description language vocabulary

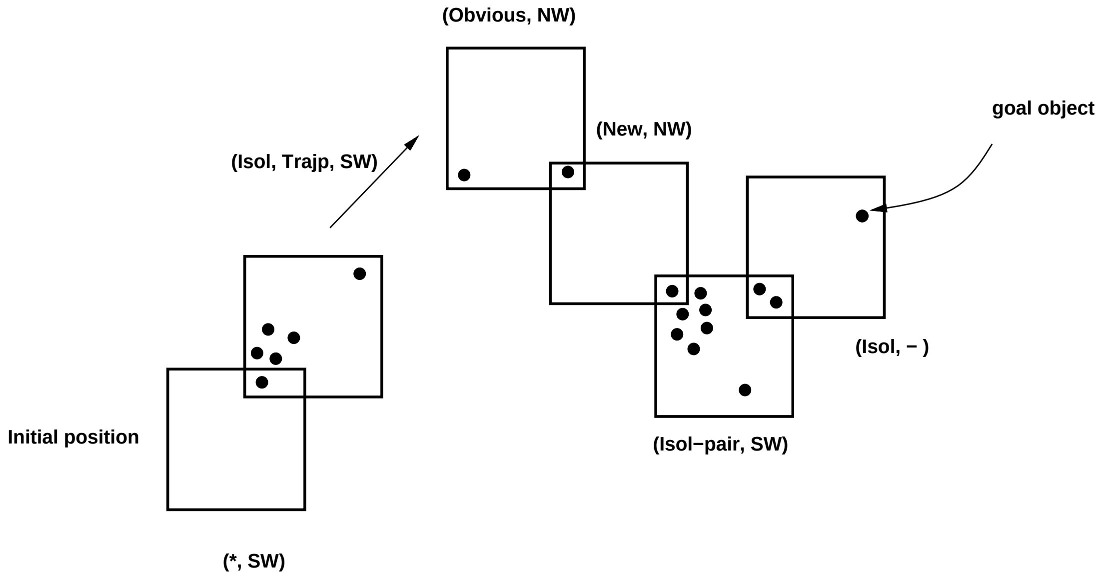


Figure 5.2: An example of a custom map consisting of the basic vocabulary

5.3.2 Identifying an isolated point

Our algorithm to compute the most isolated point in a scene consisting of n point-like objects uses the concept of *mutual neighborhoods*[23]. (Several other definitions of "isolated" led to unstable performance or costly computations.) The algorithm is as follows:

1. For each pair of points in the scene compute the *mnv* (mutual neighborhood value). The *mnv* of two points A and B is the sum of two numbers, representing the order of how close B is to A and the order of how close A is to B relative to all the other objects. For example, if B is 2nd closest point to A, and A is 3rd closest point to B, then the *mnv* is 5. The result is stored in a $n \times n$ matrix where n is the number of visible objects. The objects in figure 5.3 have the *mnv* matrix of table 5.1.

2. For each column of the *mnv* matrix, find the smallest value greater than 0. This value is the *mnv* value between this particular object (that correspond to this particular column) and its "closest" neighbor. Call this value the

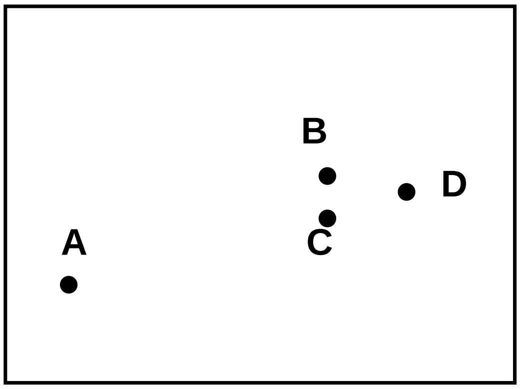


Figure 5.3: An isolated point example

	a	b	c	d
a	0	5	4	6
b	5	0	2	3
c	4	2	0	4
d	6	3	4	0

Table 5.1: *mnv* matrix

"c-value" of this particular object. So in our example, the c-values for A, B, C, and D are 4, 2, 2, and 3 respectively.

3. The object (column) that has largest c-value is the most isolated point in the scene. In our example, A has the largest c-value, of 4, and therefore it is the most isolated point. Note that c-value is a small integer of value at most n.

Example of this algorithm for identifying the most isolated object in dot figure of various population are shown in figures 5.4, 5.5, and 5.6. The large filled circle in each figure is the identified isolated point.

5.3.3 Chronological method to describe a scene

In this section, we explore a heuristic to describe a point as a function of sequence of probabilities. In section 5.1, we described a data structure called

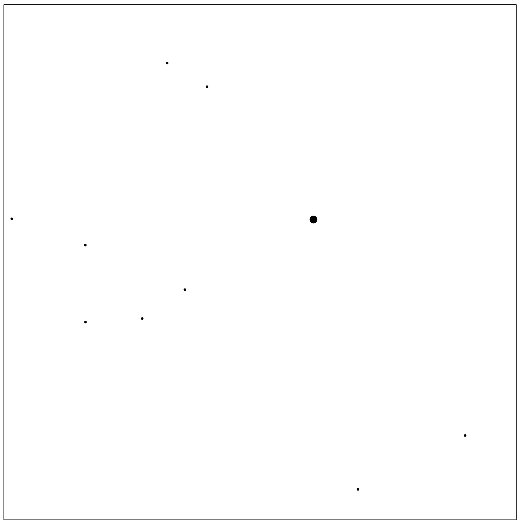


Figure 5.4: Isolated point in a scene with 10 objects

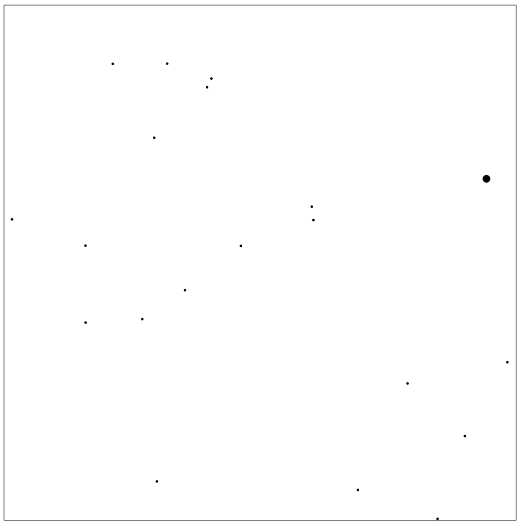


Figure 5.5: Isolated point in a scene with 20 objects

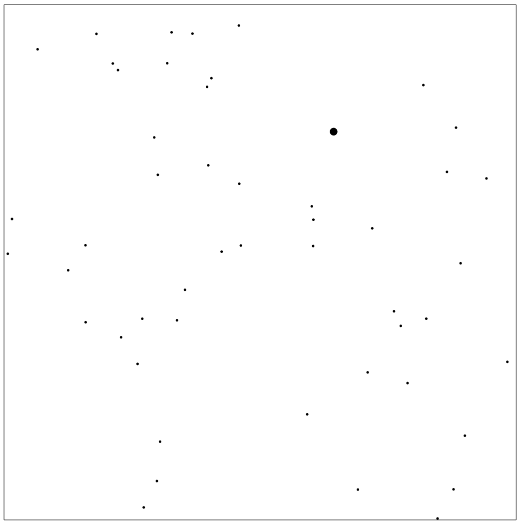


Figure 5.6: Isolated point in a scene with 50 objects

"Sphere of Influence Graph (SIG)" suggested by Toussaint [72], the definition of which has been presented in section 1 of this chapter. We use a modified version of the SIG by using a fraction(P) of the radius r_x to define the connectivity. For example, if P is 0.8, the radius of the circle centered at x would be $0.8 \times r_x$. The clusters that are defined by the resulting graphs differ depending on P. The range of the fraction P is from 0.5 where the graph becomes totally disconnected into individual points, and $+\infty$ where the graph becomes totally connected. Then, any point may be describe by a series of fractions that define SIG's that divide the scene into many clusters. For example, the series of fractions:

$$0.9 \rightarrow 0.89 \rightarrow 0.85 \rightarrow \cdots \rightarrow 0.6$$

disconnects the original complete SIG in a certain sequence. If we use a fixed set of rules, such as "pick the largest cluster", for reducing the problem domain for each of the fraction P in the sequence, certain objects may be identified. Examples of clustering in varying P are shown in figures 5.7, 5.8, and 5.9. Observations of the clusters formed by the SIG in varying fractions revealed that isolated pairs, that are broken out from a cluster early in the stage of the chronological clustering

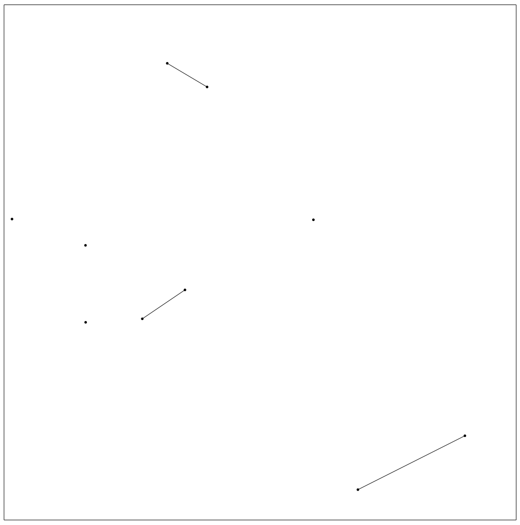


Figure 5.7: SIG in a scene with 10 objects and P = 0.501

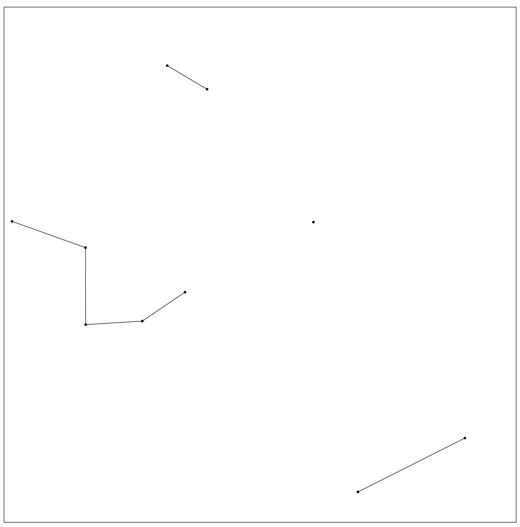


Figure 5.8: SIG in a scene with 10 objects and P = 0.60

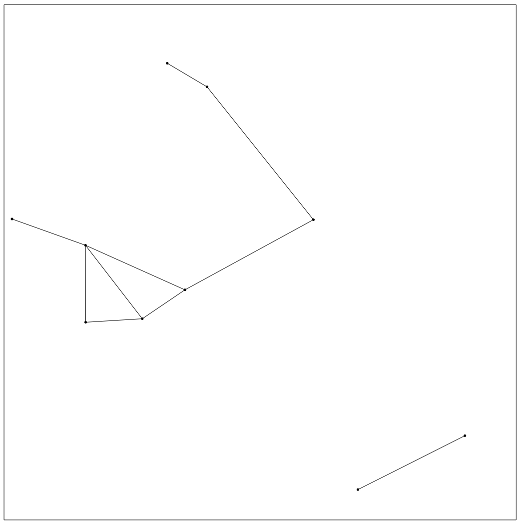


Figure 5.9: SIG in a scene with 10 objects and P = 0.90

method described above, are relatively stable and stay bound together till the very end. Therefore, we feel that the "isolated pair" descriptor, introduced earlier, can be implemented by using the SIG. However, the details have not been worked out in this thesis.

5.4 Summary

In this chapter, we studied the issues in the design and the implementation of the description language, the purpose of which is to identify objects in a scene. This is a hard problem, particularly in a dots world in which objects are identical in size and shape. There are many techiniques readily available for describing dot patterns or dot figures, including clustering algorithms, internal/external shape description algorithms, and fuzzy logic for shape discrimination. But these methods in general do not provide tools for identifying a single object in the scene. Indentifying a single object in the scene is useful in navigation when the landmark can be used as a reference point (or an anchor point.) We developed a basic vocabulary for identifying a single object (or two) that are resilient to metric errors. This includes descriptors such as obvious point, isolated point, isolated pair, confusable points, and new objects. Some of these have been implemented and analyzed.

Chapter 6

Error Handling

In this chapter, we focus on various problems that relate to errors that occur before and during the navigation. In chapter 3, we assumed that the map-maker is error free and that only the navigator is prone to errors. However, due to inherent limitations of sensing elements (in our case, the camera and the lens), and the dynamics of the environment (temperature, lighting, slight displacement of objects in environment, etc.), this assumption is not realistic. In this and following chapters, we relax the error-free assumption of our map-maker. In the beginning of map-making session, the map-maker captures the entire environment with its global camera, and computes the real world (x, y) coordinates of the populated objects. The calculated coordinates contain error. Error is further accumulated by the navigator during the navigation due to its sensory and motor limitations. When the overall error grows large enough, the navigator is prone to make mistakes. Sometimes this will result in the navigator failing to accomplish its task of reaching the destination.

There are two different ways to deal with errors. The first method is "error prevention" and is performed by the map-maker. Prior to the navigational task, the map-maker carefully examines the environment and generates a path that is least affected by error, thereby preventing the navigator from making mistakes. The second method is "error recovery", and is accomplished by the navigator during its navigational task. At each "check-point", the navigator examines its immediate surrounding to verify that it is on the correct course. If an error has been detected, it carries out an error recovery routine to get back to the original course[1]. In this chapter, we explore both of these methods in detail.

The outline of this chapter is as follows. In section 1, we analyze the errors accumulated while deriving (in the map-maker) and using (in the navigator) isolated parkways. In section 2, we examine errors accumulated while deriving and using trajectories. We model these errors statistically and analytically. These error models are then used to estimate the reliability of each generated path. Mistakes during navigation can be minimized (or prevented) by following the most reliable path generated this way. section 3 discusses error detection and error recovery methods for both parkway and trajectory traversals. The basic idea is to have the navigator seek for "uniqueness" in its immediate surrounding. We propose a general framework to detect and to recover from mistakes. Then we discuss a detailed version of the recovery algorithm that uses the Quadrant Occupancy Count (QOC), which we implement.

6.1 Parkway errors

6.1.1 Accuracy of positional information

In this section, we measure the accuracy of the positional information of the objects acquired by the map-maker. In our physical setting of the map-maker, the global camera is placed at a position that can capture the whole environment as

[1]Our error recovery method can be classified as "forward recovery", as defined in [68], because it replans to get back on track, instead of retracing its traveled route.

shown in figure 8.1. The global camera is calibrated by a simple calibration technique, as described in chapter 8, and each object's world coordinates are derived according to the calibration matrix.

We can assume that the (x, y) coordinate of each object is perturbed from its true values via some probability distribution. For simplicity, we assume that the probability distribution of error in each positional information, (x, y), is a Gaussian distribution. We further assume that the standard deviation of the errors in each (x, y) is constant for every object in the environment. To estimate the mean and the standard deviation of the probability distribution of the errors, we placed $n = 56$ objects, each to a known position. Then we calculated the X and Y differences, denoted by $E(x_1), E(x_2), \ldots, E(x_n)$, and $E(y_1), E(y_2), \ldots, E(y_n)$, from the estimated position to the real positions. The sample means and the standard deviations of $E(x_i), E(y_i)(i = 0 \ldots n)$ are:

$$(\mu_x, \mu_y) = (0.031696mm, 0.122321mm)$$

$$(\sigma_x, \sigma_y) = (1.498837mm, 1.065667mm)$$

Because we assume that $E(x_i)(i = 0 \ldots n)$ is a random sample from a normal distribution with mean μ and variance σ^2,

$$\frac{1}{\sigma^2}\sum_{i=1}^{n}(E(x_i) - E(\bar{x}))^2 = \frac{(n-1)S^2}{\sigma^2}$$

has a χ^2 distribution with $(n-1)$ degrees of freedom [46]. The same applies to $E(y_i)$. To find the upper bound of the standard deviation of the error, we use test statistic: $\chi^2 = (n-1)S^2/\sigma_0^2$, and we reject H_0 for values of this statistic larger than $\chi^2_{0.5} = 67.5048$ (based on 50 degrees of freedom, at 5% level). The observed value of the test statistic is

$$\frac{(n-1)}{\sigma_0^2} = \frac{(55)(1.915580)}{2} < 67.5048$$

Thus, H_0 is not rejected. Therefore, for our experiments, we use $\sigma^2 = 2$ as an upper bound for positional errors with confidence of 95%.

6.1.2 Reliability of isolated landmarks

Results of our experiments (See chapter 8) using various custom-maps, populations, and start/goal positions indicated that the navigator tends to fail in subareas which are highly cluttered. To explain this phenomenon, we did a statistical experiment, since the non-linear definition of "isolated" defies analytic solution. We started with a randomly populated window with n objects. To simulate the inherent error of the navigator's position estimation of the visible objects, the position information, (x, y), of each of the n objects was perturbated by a 2 dimensional Gaussian probability distribution. The output of such association produces a distorted position information, $(x + \epsilon_x, y + \epsilon_y)$. Then we applied the isolated point algorithm on the view window with the distorted positional information to see if the algorithm still identifies the correct landmark. The reliability of the isolated point in a view window with n objects is measured by the probability of achieving the correct isolated point. This is approximated by the ratio $\frac{C}{N}$, where N is the number of tries and C is the number of times when the algorithm identifies the correct isolated point. We ran this test with $N = 1000$, with $n = 3, 5, 7, 9, 11, 13$. Figure 6.1 shows the results. The data points near the top line indicates the reliability measures of the isolated point when the standard deviations, σ_x, σ_y, of the position estimations are equal to 1% of the window width W (of window size $W \times W$). The data points near middle and bottom line correspond to the reliability measures of isolated point when standard deviation are 2% and 3% of the window width, respectively. As we can see, the reliability of an isolated point decreases as the number of neighboring points increases. Note also that as the positional error increases (standard deviation of error function increases), the reliability decreases. After experimentation, we modeled the reliability of isolated point with the following equation, which states that the reliability is inversely proportional to the number of neighboring objects and directly (but negatively) proportional with σ.

$$R(n, \sigma) = \frac{a\sigma + b}{n} + c\sigma + d \tag{6.1}$$

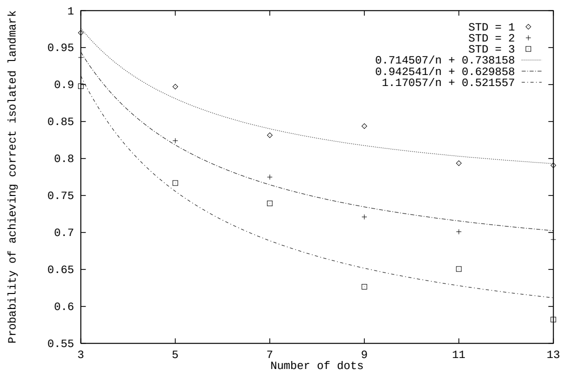


Figure 6.1: The reliability of an isolated landmark decreases as the number of neighboring objects increases

This equation was fitted with the data points using the *Mathematica* package[75]. The result of the curve fitting is also shown in figure 6.1. The resulting equation is:

$$R(n, \sigma) = \frac{0.228034\sigma + 0.486473}{n} - 0.1083\sigma + 0.846459 \tag{6.2}$$

Figure 6.2 is a visualization of this reliability measures in original world of figure 4.1. Each "isolated point" candidate has a box around it. The size of each box is defined as as $(W \times R)^2$, where, W is the width of the navigator's view window, and R is the reliability of the isolated landmark. Basically, the larger the box is, the more reliable the object is as an isolated landmark. This diagram was generated purely for visual purposes of this thesis and is not used in the navigation.

6.1.3 Reliability degrade measures of isolated landmarks

The reliability measure of each isolated point, as describe in the preceding section, is not a sufficient estimation of the actual reliability. This is due to the fact that objects that are not inside the view window but near the boundaries may or may not appear during the actual navigational movement. For example, the

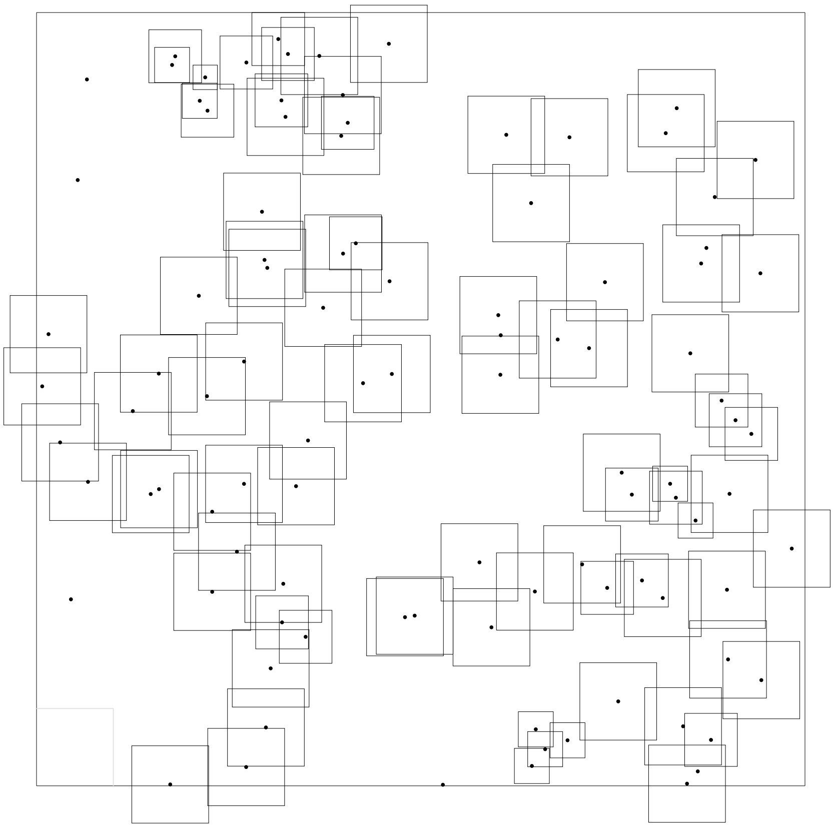


Figure 6.2: The reliability of each isolated landmark, illustrated graphically; larger boxes indicate higher reliability

reliability measure of an isolated point in a view window is accurate only if the surrounding boundary points do not slip into the same view frame.

To estimate how the reliability of an isolated point degrades as a function of number of newly introduced points, we did another statistic experiment. For each given population of $n = 3, 5, 7, 9$, we introduced $k = 2, 4, 6, 8, 10$ new points into the window. Then we compared to see if the original isolated point was stable. For each $\sigma = 1, 2, 3$, we ran the experiment 1000 times. Figures 6.3, 6.4, 6.5 show the result. In each case, we can see that as the number of intruding objects increases, the reliability decreases. The reliability degrade is also a function of σ,

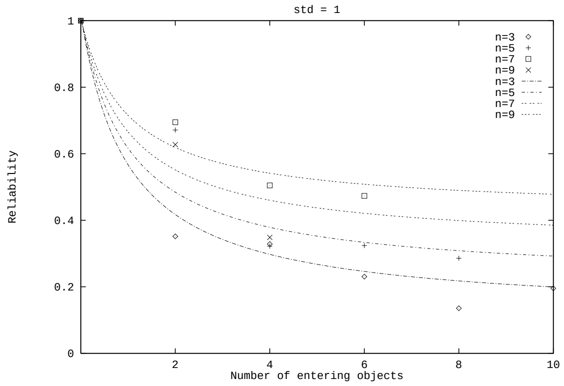


Figure 6.3: The ratio of reliability verses the number of intruding objects, when $\sigma = 1$

with lower reliability in larger σ's. We fitted the data points with equation:

$$D(k, n, \sigma) = \frac{a\sigma + bn + c}{k+1} + d\sigma + en + f \tag{6.3}$$

The fitted graph is also shown in the figures along with the raw data points. The resulting equation is:

$$D(k, n, \sigma) = \frac{0.02315\sigma + 0.05294n + 1.03316}{k+1} - 0.02248\sigma + 0.05125n - 0.01328 \tag{6.4}$$

Interestingly, the reliability degrade is less when the original number objects is greater. This means that an isolated point in a cluttered neighbor is less likely to be influenced by extra points being added in, and is more stable to noise. This is in contrast to the original reliability of each isolated point, where the less cluttered the neighbor, the more reliable is an isolated landmark.

6.1.4 Derivation of overall reliability of isolated landmarks

This section starts with an example to illustrate the derivation of the overall reliability of an isolated landmark. Suppose $n = 4$, $k = 2$, $\sigma = 1$ as in figure 6.6. Also assume that the probability of boundary objects, A and B, appearing inside

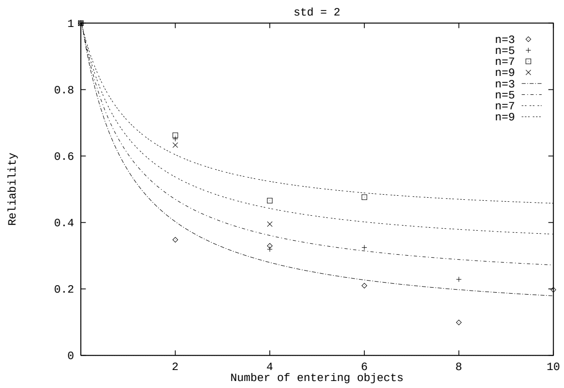


Figure 6.4: The ratio of reliability verses the number of intruding objects, when $\sigma = 2$

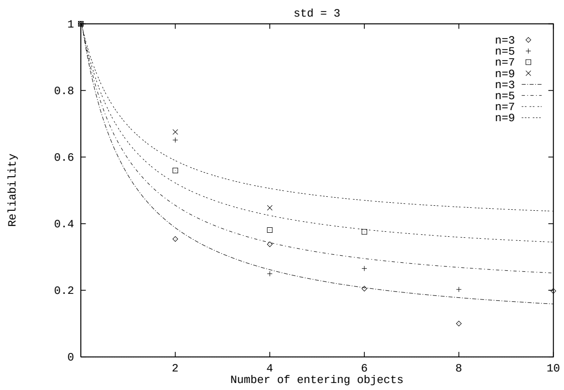


Figure 6.5: The ratio of reliability verses the number of intruding objects, when $\sigma = 3$

the view window are $P(A) = 0.3$ and $P(B) = 0.2$, respectively. Then the overall reliability of the isolated point I is:

$$\begin{aligned} Rel(n, 2, \sigma) &= R(n,\sigma)D(0,n,\sigma)P(\bar{A})P(\bar{B}) + \\ &\quad R(n,\sigma)D(1,n,\sigma)[P(A)P(\bar{B}) + P(\bar{A})P(B)] + \\ &\quad R(n,\sigma)D(2,n,\sigma)P(A)P(B) \end{aligned} \tag{6.5}$$

where $P(\bar{A})$ and $P(\bar{B})$ are the probabilities that objects A and B do not appear in the view window, respectively. $R(n, \sigma)$ is the reliability measure of an isolated point in n-populated window with perturbation parameter σ, as described in

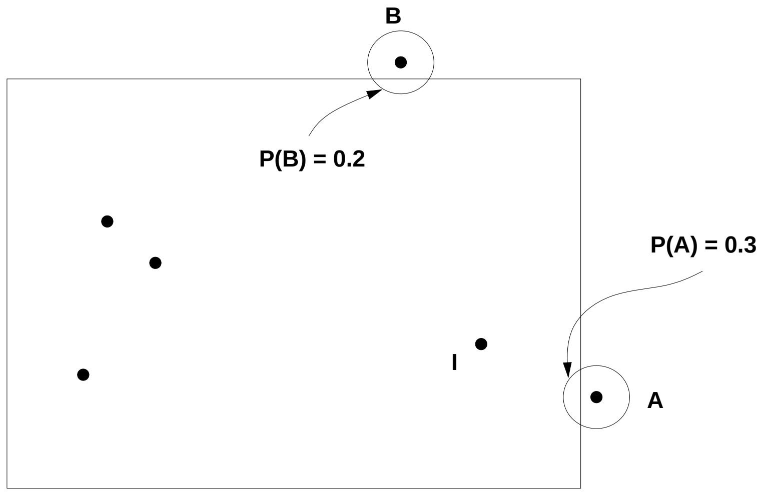


Figure 6.6: Derivation of the overall reliability of the isolated point in a window with 4 objects and 2 possible outliers

equation 6.2. $D(k, n, \sigma)$ is the reliability degrade function for an isolated point in n-populated window with perturbation parameter σ when there are k outliers, as described in equation 6.4 In general, the overall reliability of an isolated point in a window of n objects, with k surrounding objects is:

$$\begin{aligned} Rel(n, k, \sigma) &= R(n, \sigma)D(0, n, \sigma)C(0) + \\ &\quad R(n, \sigma)D(1, n, \sigma)C(1) + \\ &\quad \ldots \\ &\quad R(n, \sigma)D(k, n, \sigma)C(k) \end{aligned} \tag{6.6}$$

where $C(i)$ represent the probability of $i \in \{i | i = 0, 1, \ldots, k\}$ objects intruding the window. Unfortunately, computing $C(i)$ gets combinatorially very expensive as k grows. Therefore, we use an approximation to $Rel(n, k, \sigma)$ by using only first 3 terms of equation 6.6. Explanation of this reasoning is as follows.

Suppose there are k outliers and exactly i of them appear in the view window. Then there are:

$$\begin{pmatrix} k \\ i \end{pmatrix} = \frac{k!}{i!(k-i)!}$$

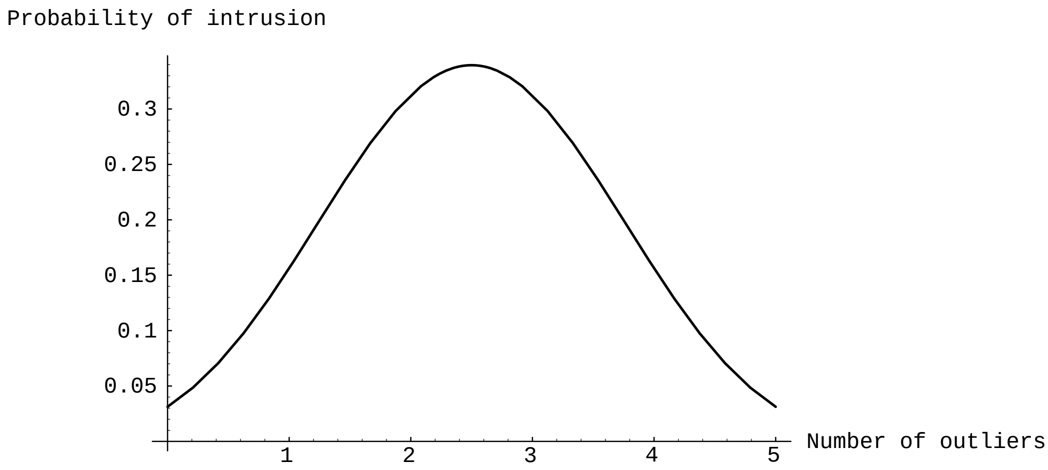


Figure 6.7: Distribution of C(i) when $P(x) = 0.5$ for every object x

possible combinations that i number of objects appeared. We know that $0 \leq P(x) \leq 0.5$, for each object x, of k outsiders. If $P(x) = 0.5$ for all x (when the reliability degrade is maximum), then the distributions of $C(i)$ is as shown in figure 6.7. However, if $P(x) < 0.5$, the peak of the distribution moves towards the left. $D(k, n, \sigma)$ is a monotonically decreasing function, with a sharp decrease rate when k is small (i.e. 0 through 3), as shown in figures 6.3, 6.4, and 6.5. The shape of $Rel(n, k, \sigma)$ depends on the distribution of $C(i) \times D(i, n, \sigma)$. For simplicity, if we assume $P(x) = p$ for all outliers, then $Rel(n, k, \sigma)$ can be written as:

$$Rel(n, k, \sigma) = \frac{k!}{i!(k-i)!} \times p^k (1-p)^{n-k} \times D(k, n, 1) \tag{6.7}$$

Figure 6.8 shows a 3D plot of equation 6.7, when $n = 10$ and $k = 5$. As we can see, the peak of the graph is in the middle when $p = 0.5$ and moves to left as p gets smaller. In the real situation, p of each outlier is much smaller than 0.5. Therefore we can assume that $Rel(n, k, \sigma)$ sharply decreases at small i. As stated earlier, for our experiments, we used $i = 3$ as the cut off point assumed that at most 3 objects outlying at any instant. This approximation is benign because we are *under*-estimating the reliability of an isolated point in crowded regions.

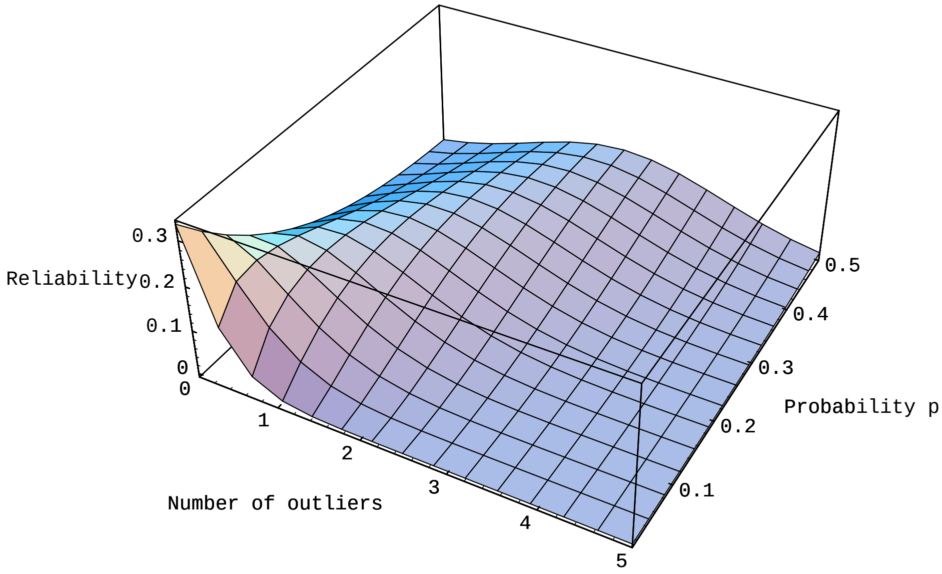


Figure 6.8: Distribution of the overall reliability

6.2 Trajectory errors

Trajectory movement is a form of blind search. The navigator defines a trajectory (direction) based on two landmarks in it current view. Then it simply moves along the formed direction until a new object appears in its view. The actual distance the navigator must travel to reach the trajectory target, denoted by X_T, may be longer than the estimated distance calculated by the map-maker. Likewise, the distance traveled by the navigator before another object is reached, denoted by X_e may be shorter than estimated. If $X_T > X_e$, then the navigator has made an error by selecting a wrong landmark. Note that the error involved in achieving a wrong object during a trajectory movement is strictly one dimensional, as opposed to the two dimensional area involved in the error of "the most isolated-point" method. This is because the trajectory movement is one dimensional. In the isolated point method, the neighboring objects surrounding the actual isolated point contribute to the error. However, the trajectory landmark needs to be distinct in only one direction - the direction of the navigator's trajectory movement.

The errors in trajectory traversal can come from sensor error in determining positional information of objects, or from translation or orientational error of the view window during navigation. The first of these three sources produce a static error, meaning that the error does not change with the traveling distance of the navigator. But the errors due to the second and the third sources are dynamic, meaning that the error grows as the trajectory distance increases.

6.2.1 Static errors

First, let us examine the static error. Let X_1 and X_2 denote the distance from the view window frame to objects A and B, respectively. Assume that X_1 and X_2 follow some probability distributions with density functions $f(x_1)$ and $f(x_2)$ respectively. Since X_1 and X_2 are independent, the joint density function is given by,

$$f(x_1, x_2) = f(x_1)f(x_2)$$

Then, the probability that object B will be reached before object A is given by:

$$P(X_1 > X_2) = \iint_{x_1 > x_2} f(x_1, x_2)dx_1dx_2 \tag{6.8}$$

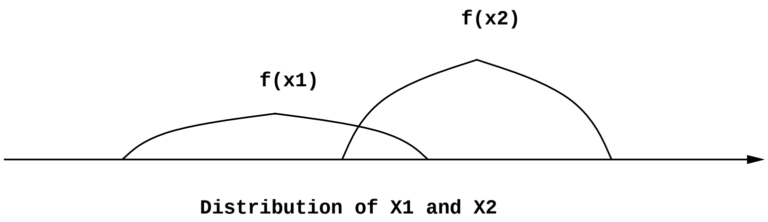
These are shown by figures 6.9 and 6.10. In figure 6.10, $P(X_1 > X_2)$ is simply the area of the blob under the line $x_2 = x_1$.

Uniform distribution of the positional information

In the simplest case, $f(x_1)$ and $f(x_2)$ are uniform distribution density function, as follows (see figure 6.11).

$$X_1 \sim U(a_1, a_2) \quad \text{where } a_2 - a_1 = c$$

$$X_2 \sim U(b_1, b_2) \quad \text{where } b_2 - b_1 = c$$



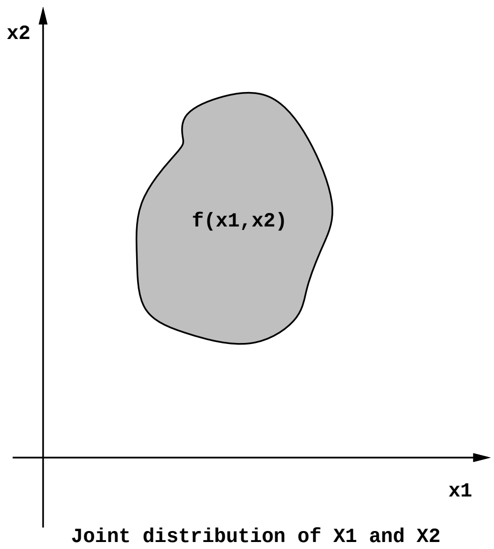


Figure 6.9: Probability distribution of X_1 and X_2

Then the probability that B will be reached before A is given by $P(X_1 > X_2)$, which is the area of the box under the line $x_2 = x_1$ in figure 6.11.

$$\begin{aligned} P(X_1 > X_2) &= \iint_{x_1 > x_2} f(x_1) f(x_2) dx_1 dx_2 \\ &= \frac{1}{2c^2}(a_2 - b_1)^2 \end{aligned} \tag{6.9}$$

Now, the integration is just the area of the box in figure 6.11 under the line $x_2 = x_1$, which is:

$$\frac{1}{2}(a_2 - b_1)^2$$

Therefore,

$$P(X_1 > X_2) = \frac{1}{2c^2}(a_2 - b_1)^2 \tag{6.10}$$

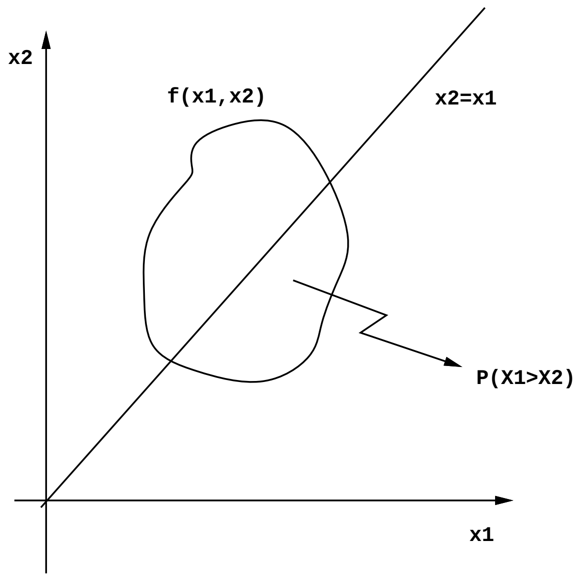


Figure 6.10: Probability that $X_1 > X_2$

Gaussian distribution of the positional information

Let us now assume that $f(x_1)$ and $f(x_2)$ are Gaussian distribution density functions, with parameters $N(a, \sigma^2)$ and $N(b, \sigma^2)$, respectively (shown in figure 6.12).

Then, the probability that B will be achieved before A in a trajectory movement is (see figure 6.12):

$$\begin{aligned} P(X_1 > X_2) &= \iint_{x_1 > x_2} \frac{1}{2\pi\sigma^2} e^{-\frac{(x_1-a)^2+(x_2-b)^2}{2\sigma^2}} dx_1 dx_2 \\ &= \int_{\frac{b-a}{\sqrt{2}}}^{\infty} \int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2} e^{-\frac{(x_1-a)^2+(x_2-b)^2}{2\sigma^2}} dx_1 dx_2 \\ &= \int_{\frac{b-a}{\sqrt{2}}}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_2-b)^2}{2\sigma^2}} dx_2 \\ &= \frac{1}{2}(Erf(\frac{b-a}{2\sigma}) - 1) \end{aligned} \tag{6.11}$$

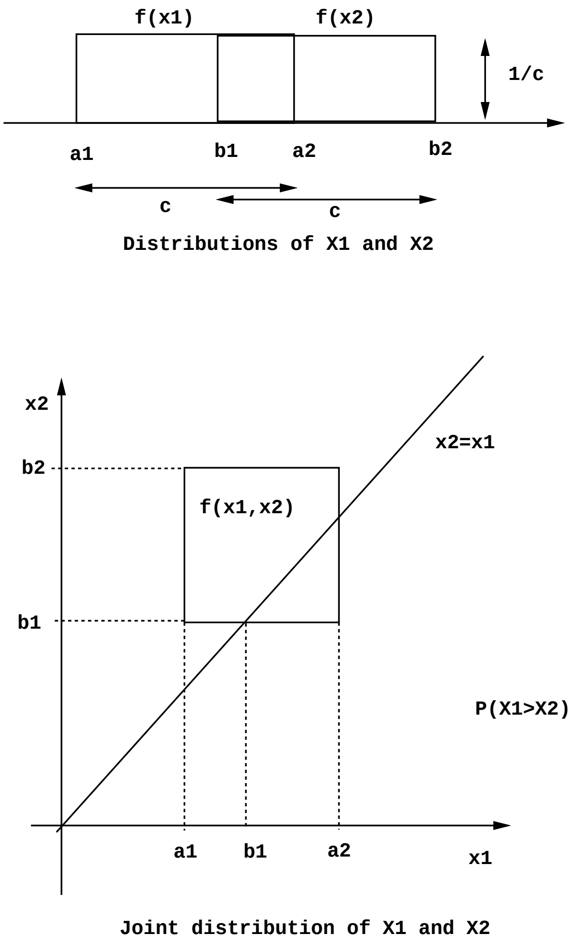


Figure 6.11: Uniform joint probability distribution

6.2.2 Dynamic errors

Dynamic error of the trajectory task depends on (1) how far the robot moves and (2) in which direction the robot moves relative to reference objects. The next two subsections discuss both of these error sources in turn.

Translational Error

Translational error occurs due to the mechanical inaccuracies of the robot. As the robot moves farther, the error grows. Note that whenever a new landmark is identified, the translational error is reset to 0. This is equivalent to the "map

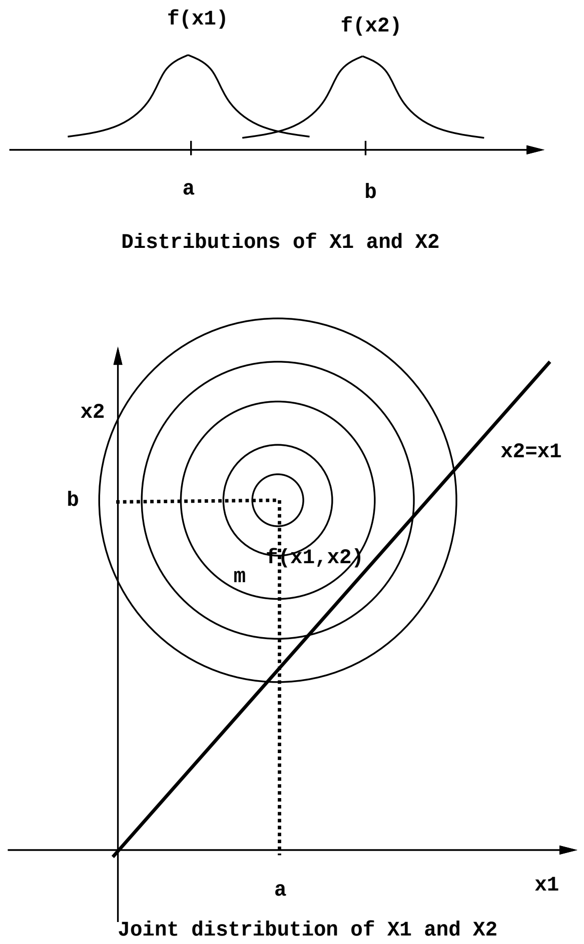


Figure 6.12: Gaussian joint probability distribution

matching" technique used in the vehicle navigation systems [20] [41] [18] [9]. The basic idea is to use a variable size σ that is a function of distance. For example, if object O_2 is 10 times farther away from the window than object O_1, then the distribution of O_2 should be $10\times$ *constant* times more dispersed than that of O_1. This is shown in figure 6.13.

Let d be the distance traveled in the trajectory task. Let Δ be the incremental length of the navigator's movement during the trajectory task. At each incremental movement, leading edges of the navigator's view window are examined to see if an object has newly entered the view. To ensure that the navigator does not overshoot the trajectory target, Δ has to be smaller than the widths of the leading edges. In

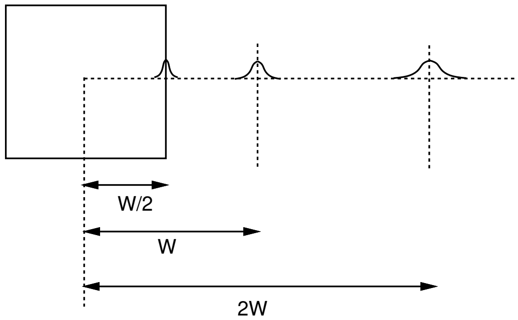


Figure 6.13: Dynamic error function

our setup, we use the diameter of the object (4 mm) as the width of the leading edge.

The accuracy of navigator's movement is found in the 7575 manipulator manual, in which an uncalibrated XY movement is bounded by $\pm 0.20mm$ [27]. Assuming a 2 dimensional Gaussian distribution for the positioning accuracy of the manipulator, and using $\pm 3\sigma$ as an approximation to the given accuracy boundaries, we have:

$$\begin{aligned} E &\sim N(0, (\frac{d}{\Delta}\sigma)^2) \\ &\sim N(0, (\frac{d}{4}00.67)^2) \\ &\sim N(0, (0.01675d)^2) \end{aligned} \tag{6.12}$$

Orientational error

The direction of movement in the trajectory task is defined by two reference objects (that have been previously chosen) in a view window. If the positional information of the two chosen landmarks is not correct, it will affect the accuracy of the trajectory task. For example, let us consider situation in figure 6.14, in which objects A and B are used to define the trajectory. The navigator has already completed step 1 of the trajectory task, which is "centering" the window so that the virtual center of objects A and B is aligned to the center of the view window.

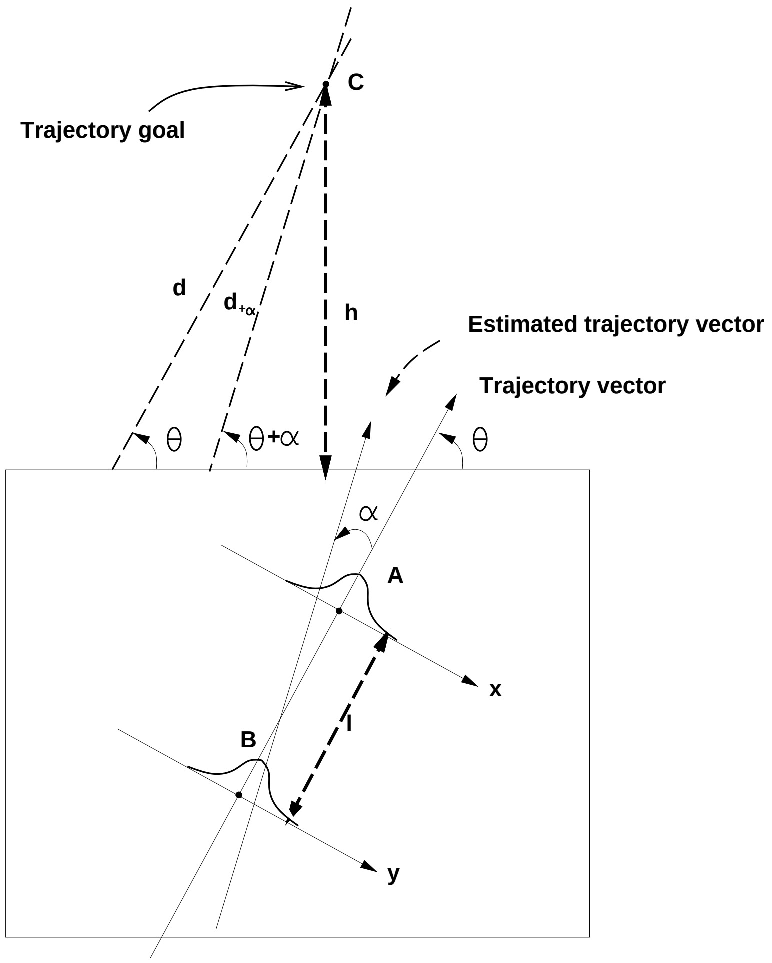


Figure 6.14: Dynamic error during a trajectory movement

It is about to start step 2, which is the sliding movement in the direction formed by objects A and B. Let θ be the incident angle between the trajectory vector and the top leading edge of the window, and let h be the vertical distance between C and the leading edge. Then, the distance d between the leading edge of the view window and the trajectory goal, in the direction of the trajectory vector is:

$$d_\theta = \frac{h}{\sin\theta} \tag{6.13}$$

As previously assumed, the probability distributions of error in the positional estimations of A and B are modeled by two dimensional Gaussian functions. Let X and Y be the deviation of observed positions from the real positions of A and B in the directions perpendicular to the trajectory vector, respectively. Then, the

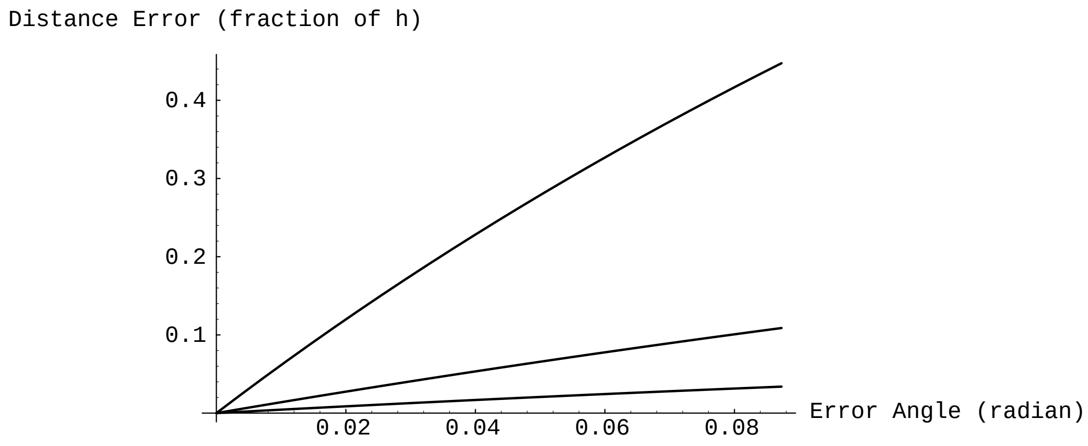


Figure 6.15: Distance error as a function of the trajectory angle

actual distance that the navigator travels to achieve the trajectory goal is given by:

$$d_{\theta+\alpha} = \frac{h}{\sin(\theta+\alpha)} \tag{6.14}$$

The error E is the difference between equation 6.13 and equation 6.14, and is given by:

$$\begin{aligned} E &= d_\theta - d_{\theta+\alpha} \\ &= \frac{h}{\sin\theta} - \frac{h}{\sin(\theta-\alpha)} \\ &= \frac{\sin(\theta+\alpha)-\sin\theta}{\sin\theta\sin(\theta-\alpha)} h \\ &= \frac{\sin\theta\cos\alpha+\cos\theta\sin\alpha-\sin\theta}{\sin^2\theta\cos\alpha+\sin\theta\cos\theta\sin\alpha} h \end{aligned} \tag{6.15}$$

Figure 6.15 shows three instances of E as functions of α , having θ value of $\theta = \frac{1}{8}\pi, \theta = \frac{1}{4}\pi, \theta = \frac{3}{8}\pi$, respectively. Note that the trajectory error decreases as the incident angle θ approaches $\pi/2$. Also note that E is close to a linear function when α is small. The angle between the trajectory vector and the estimate trajectory vector, α is given by:

$$\alpha = \arctan \frac{x-y}{l} \tag{6.16}$$

Since α is small (i.e. $\alpha < 5^o$). we can use the following approximations:

$$\begin{aligned} \sin\alpha &\simeq \alpha & (6.17)\\ \tan\alpha &\simeq \alpha & (6.18)\\ \cos\alpha &\simeq 1 & (6.19)\end{aligned}$$

Therefore,

$$\begin{aligned} \alpha &\simeq \frac{x-y}{l} & (6.20)\\ E &\simeq \frac{\sin\theta + \alpha\cos\theta - sin\theta}{\sin^2\theta + \alpha\sin\theta\cos\theta}h & \\ &\simeq \frac{\alpha\cos\theta}{\sin^2\theta}h & (6.21)\end{aligned}$$

Figure 6.16 shows the result of the approximation. The solid lines represent the approximated distance error function of equation 6.21. Light gray lines represent the actual error as in equation 6.16 when α is negative. Dashed lines represent the actual error when α is positive. The unit of y-axis is in fraction of h. E can be further approximated as a linear function of x and y:

$$\begin{aligned} E &\simeq \frac{\alpha\cos\theta}{\sin^2\theta}h & \\ &= \arctan(\frac{x-y}{l})\frac{\cos\theta}{\sin^2\theta}h & \\ &\simeq \frac{x-y}{l}\frac{\cos\theta}{\sin^2\theta}h & (6.22)\end{aligned}$$

6.2.3 Overall reliability of trajectory movement

As explained in previous sections, sources and distribution assumptions of errors in the trajectory method consist of:

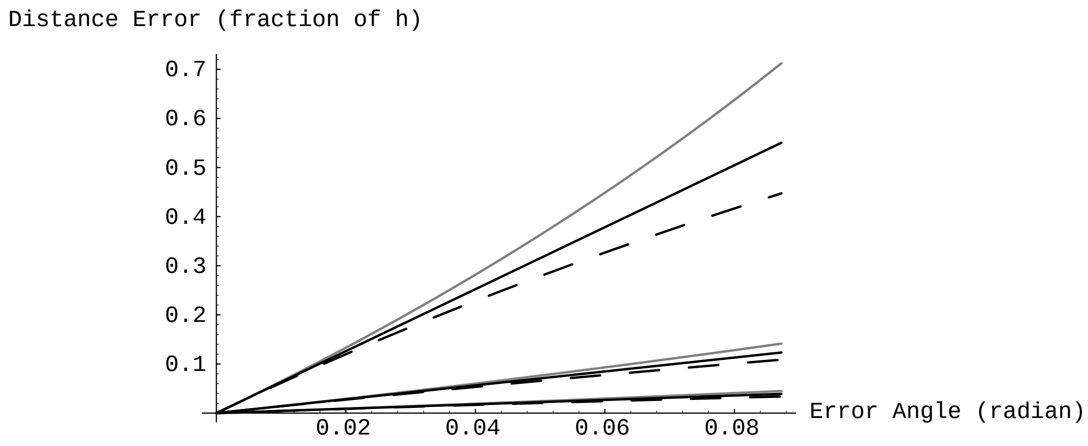


Figure 6.16: Approximation of distance error as a function of the trajectory angle

1. The map-maker's positional error $\sim N(0, \sigma_m^2)$. This depends on the sensory accuracy of the map-maker.

2. The navigator's positional error $\sim N(0, \sigma_n^2)$ This depends on the sensory accuracy of the navigator.

3. The navigator's translational error $\sim N(0, \sigma_t^2)$. This depends on the mechanical accuracy of the navigator and the traveling distance of the navigator for the trajectory task.

4. The navigator's orientational error $\sim N(0, \sigma_o^2)$. This depends on the position estimation errors of the map-maker and the navigator, and the traveling distance of the navigator for the trajectory task.

Note that although the actual distribution of these errors are 2 dimensional, we use 1 dimensional distribution functions because trajectory movement is 1 dimensional. Distribution functions that we use, are in fact 1 dimensional cross sections of the original 2 dimensional distribution functions.

The overall distribution of error for the trajectory movement is a combination of the four error types. From equation 6.22 and the listed items above, we have:

$$\begin{aligned} E &= \frac{x-y}{l}\frac{\cos\theta}{\sin^2\theta}h + z \\ &= \frac{\cos\theta}{l\sin^2\theta}hx - \frac{\cos\theta}{l\sin^2\theta}hy + z \end{aligned} \tag{6.23}$$

Here, z indicates how far off the real trajectory destination is from the estimated distance in the trajectory task. z is the sum of 1, 2, 3 stated above. The distributions of random variables X, Y, and Z, corresponding to x, y, and z, respectively are:

$$X \sim N(0, \sigma_m^2 + \sigma_n^2) \tag{6.24}$$

$$Y \sim N(0, \sigma_m^2 + \sigma_n^2) \tag{6.25}$$

$$Z \sim N(0, \sigma_m^2 + \sigma_n^2 + \sigma_t^2) \tag{6.26}$$

Since E is a linear function of x, y, and z, E is also of Gaussian distribution with parameters:

$$E \sim N(0, (1 + 2(\frac{h\cos\theta}{l\sin\theta^2})^2)(\sigma_m^2 + \sigma_n^2) + \sigma_t^2) \tag{6.27}$$

6.3 Error detection and recovery

6.3.1 Error detection

When a landmark is obtained, the navigator needs to observe its neighborhood in order to verify that it is on the correct course. Usually, the current view of the navigator does not contain enough information for verification. The largest region that the navigator can observe without losing its current reference point is the area covered by the 4 quadrants - SE quadrant, SW quadrant, NE quadrant, and

NW quadrant. When the reference object is at the SE corner of the window, the corresponding area covered by the window is the SE quadrant, and so forth. Mistakes or errors are detected by observing the 4 quadrants relative to the currently achieved landmark.

General idea

A landmark can be characterized by the contents of its 4 quadrants. The contents of the 4 quadrants can be expressed in a *signature* form of:

$$signature = \{\text{Observe(SE)}, \text{Observe(SW)}, \text{Observe(NE)}, \text{Observe(NW)}\}$$

where Observe(Q) is a description of the corresponding quadrant $Q \in$ {SE, SW, NE, NW}. In the isolated landmark selection process, a wrong object may be mistakenly chosen. However, if the signature of the wrongly chosen object is different from the signature of the correct isolated landmark object, then the mistake can be identified by comparing the desired signature to the currently observed one. Therefore, in order for a landmark to be error detectable, it needs to have a unique quadrant signature within its neighborhood.

Implementation of the Observe() function depends on the capabilities of the navigator. For our simple navigator model, we developed several Observe() functions, as listed as follows:

1. Quadrant Occupation Count (QOC) - is defined to be a sum of 4 numbers (0 or 1) indicating the presence or the absence of objects in each quadrant. i.e., QOC $\in \{0, 1, 2, 3, 4\}$. QOC can be represented by a 2-bit number. In figure 6.17, the QOC of object R is 3 because 3 quadrants are occupied.

2. Quadrant Occupation Signature (QOS) - is defined to be a concatenation of 4 numbers (0 or 1) indicating the presence or the absence of objects in each

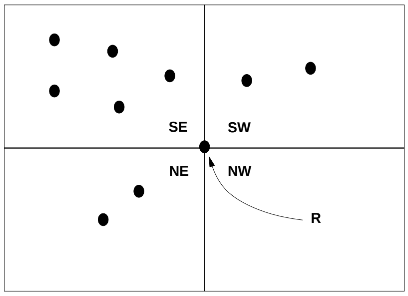


Figure 6.17: An example configuration for the Observe() function

quadrant. i.e., QOS $\in \{0000, 0001, \ldots, 1111\}$. QOS can be represented by a 4-bit number. In figure 6.17, the QOS of object R is 1110.

3. Quadrant Population Count (QPC) - is defined to be a sum of 4 numbers corresponding to the number of visible objects in each quadrant. i.e., QPC $\in \{x|$ integer$\}$. QPC can be represented by an integer. In figure 6.17, the QPC of object R is 9.

4. Quadrant Population Signature (QPS) - is defined to be a concatenation of 4 numbers corresponding to the number of visible objects in each quadrant. i.e., QPS $\in \{x_1x_2x_3x_4|x_i$ is an integer for all $i\}$. QPS can be represented by 4 column valued vectors. In figure 6.17, the QPS of object R is 5220.

For our implementation and experiments, we used the Quadrant Occupancy Count (QOC). Details on how QOC was used for error detection and recovery are explained in following sections.

Error detection using QOC

QOC of a landmark is obtained by moving the view window around the 4 quadrants relative to the landmark and by counting the number of occupied quadrants. Errors can be detected only if the landmark has a unique QOC within the current view window. In figure 6.18, we see there are 5 objects in the view window. R is

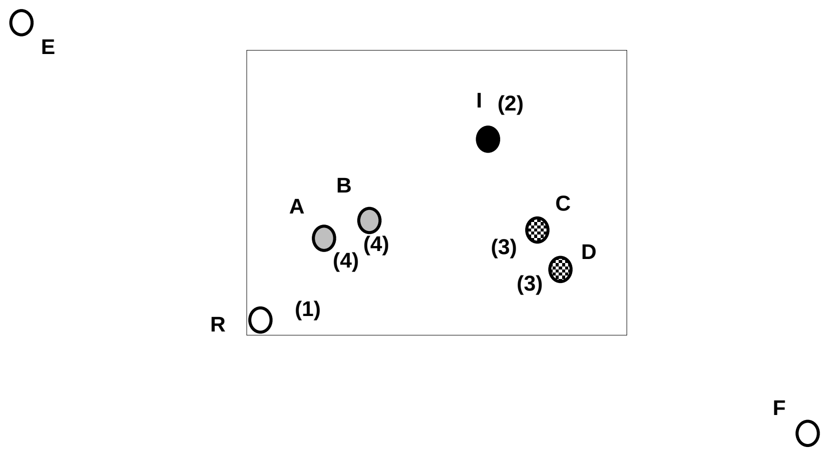


Figure 6.18: An error detectable configuration

A	B	C	D	I	R
4	4	3	3	2	1

Table 6.1: QOC values of objects A, B, C, D, I, and R

the current reference point and I is the desired isolated landmark. QOC of the visible objects are shown in table 6.1. As we can see, QOC of object I is unique within this view window, thus error can be detected by examining the QOC of the obtained landmark.

Optimization of sequencing window positions

To obtain the QOC of a landmark, the navigator needs to examine its 4 quadrants (NE, NW, SE, SW). We know that one of the quadrants is always occupied because of the reference point. Therefore, we need only look at 3 quadrants. If we use QOS to identify a landmark, by sequencing the 3 quadrants in a certain order, we can detect for error (or validate the current move) before all 3 quadrants are examined. For example, let us suppose that NE is occupied by the reference point, and therefore we need to examine only NW, SE, and SW. Let a string BBB indicate the occupancy of the corresponding quadrants, where B=(0,1). Let us

	NW	SE	SW
A	0	1	1
B	1	1	1
I	0	1	0

Table 6.2: Signature strings of objects A, B, and I

further assume that there are 4 objects in the window, A, B, R, and I, where R is the reference point and I is the isolated point. Assume that the signature string for A, B, and I are as in table 6.2. Then, by examining NW only will determine whether or not the navigator is in the correct place, instead of examining all 3 quadrants. Unfortunately, choosing the optimal set of quadrants (either in serial, or in parallel), is NP-Complete because this problem can be reduced to the minimal set covering problem. But, since we have only 3 quadrants to work with, exhaustive search is possible.

6.3.2 Error recovery

In the preceding section, we described methods to detect errors during navigation. Error detection is useful only if there are corresponding error recovery methods available to the navigator. In this section, we describe a such error recovery routine that the navigator can utilize when errorful situations are encountered. In the worst case, when an error is detected, the navigator simply gives up its navigational task and says “I am lost”. But in many cases, by performing a fixed set of “reset” sequences, the navigator is able to get back to its originally intended course.

General algorithm

The basic idea is to assign a fixed action for each QOC (0 to 4). Therefore, the error recovery is a simple "if QOC then ACTION" set. For example, in figure 6.18, objects A and B have QOC of 4. If the navigator chooses either A or B instead of the desired I, it can "correct" the mistake by moving to the SW corner (relative to A or B) and then choosing the isolated point. Likewise, the correction instruction for objects C or D whose QOC is 3, is (SE, Isol).

Formally, an isolated point is error recoverable if:

1. The configuration is error detectable.
2. Self containment: Objects in the same QOC group are positioned so that the originally intended landmark (the true isolated landmark, I) is in the same quadrant related to these objects. For example, in our example of figure 6.18, objects A and B having QOC 4, both have I in quadrant SW.
3. Transitively reachable: From every reachable object (which can be mistaken as the true isolated landmark), there must exist a path to the true isolated landmark. For example, in our figure 6.18, the isolated point of object C in SW quadrant is in fact object B, not I. But I is reachable from B. Therefore, I is transitively reachable from C.

Reachability domain

Before the error-recoverability conditions, as suggested in previous paragraph, can be tested, we need to find all objects that are reachable - the objects that can be wrongly chosen as the true isolated landmark. For example, in figure 6.19, every object in the original view (inside the box with dark solid lines) are reachable. Object E is also reachable from object A and B, but objects F, and G are not.

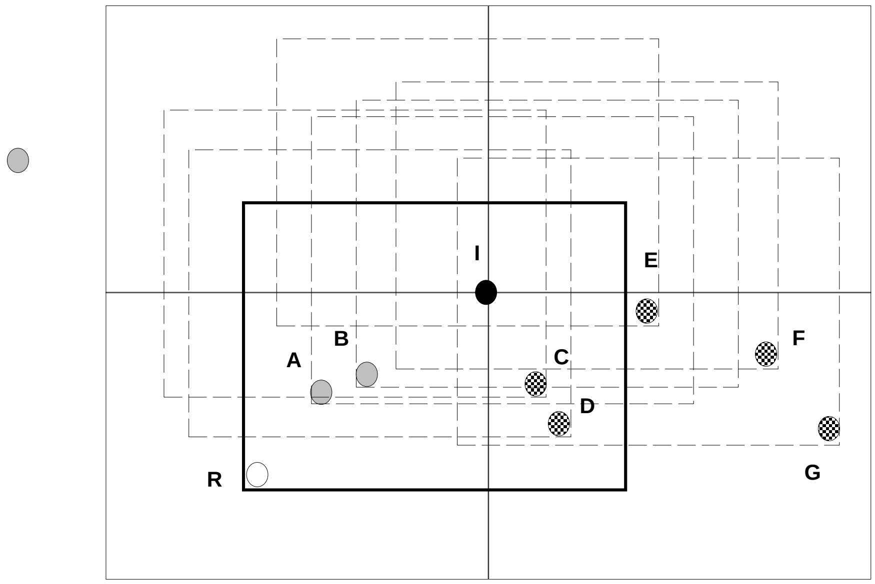


Figure 6.19: An error recovery domain

The following is a simple algorithm to identify all the objects that are reachable during the error recovery procedure.

1. Consider all points within the 4 quadrants spanned by the true isolated point (thin solid lines in figure.) With these k objects, form a $k \times k$ matrix. For example, in the figure, we have 7 objects A, B, C, D, E, F, and G.

2. Initialization of the matrix $reach$[i, j]: If the i-th object is within the original window, update $reach[i, i]$ of the matrix as '1'. Everywhere else, put 0.

3. Update matrix: For each object, find out which other object it covers and update the matrix accordingly. For example, in the figure, object A covers objects B and C, therefore $reach[A, B]$ and $reach[A, C]$ are set to '1'.

4. Reduce the matrix: Examine every column of the matrix to see which column has all-0 entries, and update the corresponding (transpose) row to 0. For example, in the figure, object G has all-0 column. Therefore, every entry in row G of the matrix is set to 0. Repeat step 4 until no further update is possible.

	R	A	B	C	D	E	F	G
R	1	1	1	1	1	0	0	0
A	0	1	1	1	0	1	0	0
B	0	0	1	1	1	0	0	0
C	0	0	1	1	0	0	0	0
D	0	1	1	1	1	0	0	0
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	1	0	0
G	0	0	0	1	1	1	1	1

Table 6.3: *reach*[] matrix before running the domain algorithm

	R	A	B	C	D	E	F	G
R	1	1	1	1	1	0	0	0
A	0	1	1	1	0	1	0	0
B	0	0	1	1	1	0	0	0
C	0	0	1	1	0	0	0	0
D	0	1	1	1	1	0	0	0
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0

Table 6.4: *reach*[] matrix after running the domain algorithm

5. Objects that have non-zero columns are reachable.

Initially, *reach*[] matrix of configuration in figure 6.19, before and after running the algorithm, are shown in tables 6.3 and 6.4. As we can see, F and G have 0-columns, therefore are not reachable. This reachability domain problem can be solved by a version of "topological sorting" algorithm and therefore within time complexity $O(|E| + |V|)$ [3].

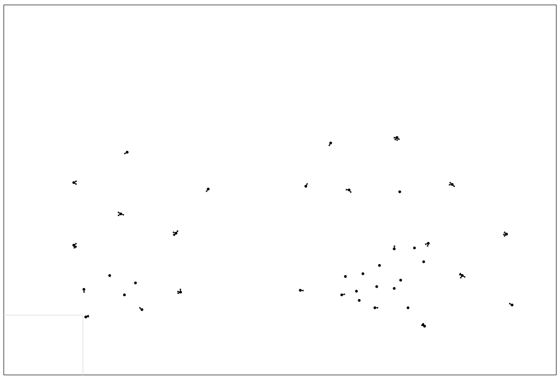


Figure 6.20: Error recoverable parkways - general algorithm

Special case

We can increase the number of error recoverable landmarks by adding a further restriction to the general error recoverable conditions. If we restrict the reachability domain objects such that the reference point, such as R in figures 6.18, 6.19, is not reachable during the error recovery sequence, more landmarks become error recoverable. This is because the inclusion of the reference point as a group candidate further restricts the error recoverable configurations. Figures 6.20 and 6.21 are the error recoverable parkways generated using the general and special algorithms, respectively.

Error detection in trajectories

We can use the QOC method to detect errors in the trajectory movements by appropriately modifying the existing algorithm introduced earlier. In the parkway case, the landmark needed to have a unique QOC within the view window in order for it to be error detectable. In trajectories, the landmark (trajectory target) has to be unique among all the reachable objects. Recall, from previous section, that an object is reachable, during a trajectory movement, if there is a possibility of navigator wrongly obtaining it before the trajectory target.

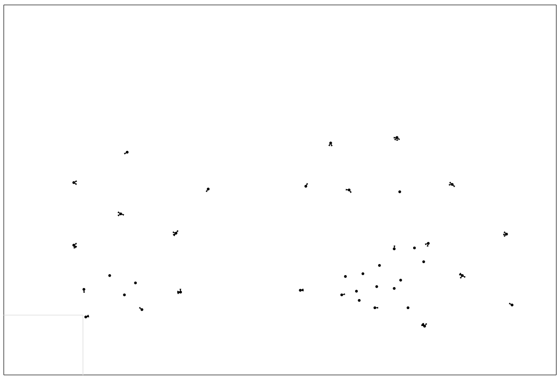


Figure 6.21: Error recoverable parkways - special algorithm

Error recovery in trajectories

To guarantee that a trajectory movement is error recoverable, we need to impose another restriction to the reachability restriction as described in the error detection case. From every reachable object in a trajectory movement, the target landmark must be visible in a same view. Otherwise, even after identifying a mistake, there would be no way to get back to the correct target. If this condition is satisfied, the error recovery algorithm, that was previously introduced for parkways, can be directly used.

Figure 6.22 shows an example of an error correctable path. The initial position of the navigator is at the lower left hand corner. Initial reference object is the dark gray object. The goal object is near the top right hand corner, represented by the light gray object. The numbers next to the objects are the corresponding QOC's. Initially, the navigator sees 5 objects in its view. The first entry in the custom map instructs the navigator to identify isolated landmark and adjust to SW. If it makes a mistake and wrongly chooses an object whose QOC is 4, it is told to adjust to SW and look for an isolated landmark. If it chooses an objects whose QOC is 2, it is told to adjust to SE and look for an isolated landmark. Only when the chosen object's QOC is 3, it is able adjust to SW and continue to the

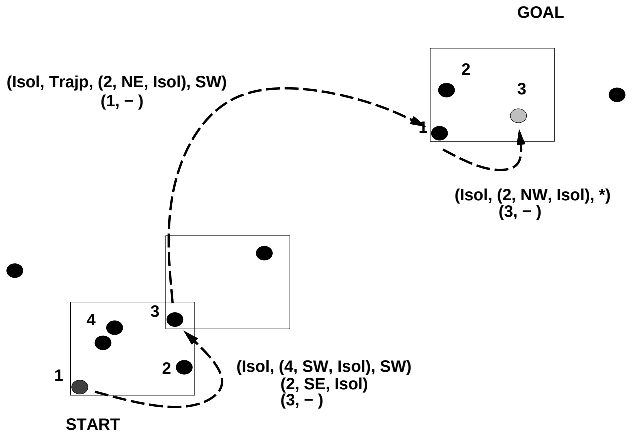


Figure 6.22: A highly error recoverable path

next instruction. The next entry in the custom map instructs the navigator to do a trajectory movement based on directional vector formed by the reference point and the isolated point. If the QOC of the obtained landmark is 2, it is told to adjust to NE and to look for isolated landmark. Otherwise, if it is 1, it can adjust to SW and continue to the next instruction. The last entry in the custom map tells the navigator to identify an isolated landmark as the goal object. If the computed isolated point has QOC of 2, it is told to recover by adjusting to NW, and then by identifying the isolated landmark. Otherwise, if the QOC is 3, the goal has been identified.

6.4 Summary

In this chapter, we examined various types of errors that affect the accuracy of navigation. We determined their sources and modeled them statistically, analytically, and by approximation. As a result of the error models, we are able to incorporate reliability measures into the cost matrix so that an error minimizing path can be generated, as will be shown in Chapter 7. Reliability of an isolated landmark depends on (1) the accuracy of the map-maker's and the navigator's sensors, σ, and (2) the density of the environment n, in which the isolated parkway is located. Thus, when navigating in an isolated parkway, mistakes can be minimized by following paths in sparse areas of the environment rather than cluttered regions. Reliability of a trajectory depends on (1) the accuracy of the map-maker's and the navigator's sensors, σ, (2) the length of the trajectory d, (3) the trajectory angle[2] θ of figure 6.14, and (4) the density of the environment, n, in which the trajectory takes place. Thus, trajectory error can be minimized by taking a short, orthogonal trajectory towards a sparse region.

We proposed a general framework for error detection and error recovery. The basic idea is to have the navigator examine its immediate surrounding at predefined check points (landmarks). Where and how many check points there are, depend on the environment. In general, the check point landmark must have some unique feature that distinguishes it from its neighbors. For our implementation of the error detection and recovery, we used Quadrant Occupancy Count (QOC) to characterize the current surrounding of the navigator, which represents the presence or the absence of objects in the neighborhood of the check point landmark. When the QOC of the obtained landmark does not match the anticipated value, the navigator is told to execute a sequence of error correcting movements to get back on track.

[2]This dependency is an artifact of having a rectangular window.

Chapter 7

Path and Custom Map Generation

The map-maker's main objectives are to compute a navigational path, and to generate a corresponding custom map for the navigator. Only then is the navigator able to start on its journey. This is analogous to a direction giving and direction following situation among humans. For example, inviting a friend to a dinner for the first time usually involves giving directions. First, the direction giver must decide which route is the best for the navigator to take. Usually, the shortest distance route is not the best route to take when the driver is not familiar with the roads. The direction giver must take into consideration various "costs" for the navigation, some of which are listed as follows.

- How short is the travel distance?
- How much time does it take?
- How easy is the road? [8]
- How easy is it to describe? [44]

- How to provide enough information without over burdening the driver? [67]

The first 3 items in the above list are related to the derivation of the route. Depending on the skills and experience of the driver, the priority of these costs may vary. Also, depending on the priority of each of these costs, the resulting route may vary. For example, the shortest distance path may not be the shortest time travel path. Furthermore, the easiest (most reliable) path may be drastically different from the two. If the driver is unskilled or is not familiar with the roads, path derivation should favor easy roads over faster roads. The latter two items in the list are related to the direction giving. Describing how to follow the already derived path is as important as the path generation itself.

The same principle applies to our dots world. Depending on the capabilities of the navigator, and the priorities of the user, different cost optimality criteria can be imposed. For example, the user may want the map-maker program to generate the shortest distance path, the most reliable path, or a compromise of the two. Based on the derived path, the map-maker then needs to come up with a corresponding custom map. The length and the complexity of the custom map may also be of importance in some situations. For example, it may be necessary to limit the size of the custom map for efficient communication from the map-maker to the navigator. The environmental description language defines the command set available to the navigator in the custom map. Each instruction in the custom map needs to be simple [67], and qualitative, for the sake of efficiency and accuracy.

Traditionally, robot navigation theory has been divided into two groups. First is the traditional AI theory applied to navigation where the high level planning is done prior to the navigation. The other is the reactive control theory based on Brook's subsumption architecture [7], in which the navigation is low level in the sense that the robot navigates by reacting to the environment based on its sensory inputs. Our map-maker and navigator models incorporate both of the two approaches. The map-maker's path planning and custom map generation provides the high level preplanned control of the navigator. The navigator's actions are based on the custom map commands and the immediate environmental sensory reading, which can be regarded as reactive control of the robot navigator.

This chapter is organized into two sections. In Section 1, we describe in detail the various aspects of path planning in our dots world, including how the cost measures are incorporated into the algorithm to find optimal paths. In Section 2, we discuss the generation of custom maps corresponding to the generated paths.

7.1 Path generation

7.1.1 Transition and cost matrices

The transition graph or transition matrix, interchangeably, contains information as to whether or not an object can be identified, therefore directly reachable, in reference to another. That is, if object B is directly reachable from object A, the entry of the transition matrix corresponding to (A, B) is set to 1. An example of the transition matrix of the isolated parkway in figure 7.1 is shown in table 7.1. The square box shown to the left of the figure is the window size used to calculate the parkways. The number of disjoint isolated parkways is 4 (although the number of all-parkways is 3) because there are 4 strongly connected components. Objects A, B, C, F, G are in the first isolated parkway. Objects D and E are in the second isolated parkway. This is because transition from object group D, E to A, B, C, F, G is one way. Objects H and I are each on its own parkway. From the observation of the transition matrix, we immediately see that object C is a crucial landmark in this environment because the corresponding column is filled most.

Each entry in the cost matrix, corresponding to the transition matrix entry, is a value indicating the cost of such transition. Depending on the need of the navigator, this value can bear different meanings. For example, this can be spatial

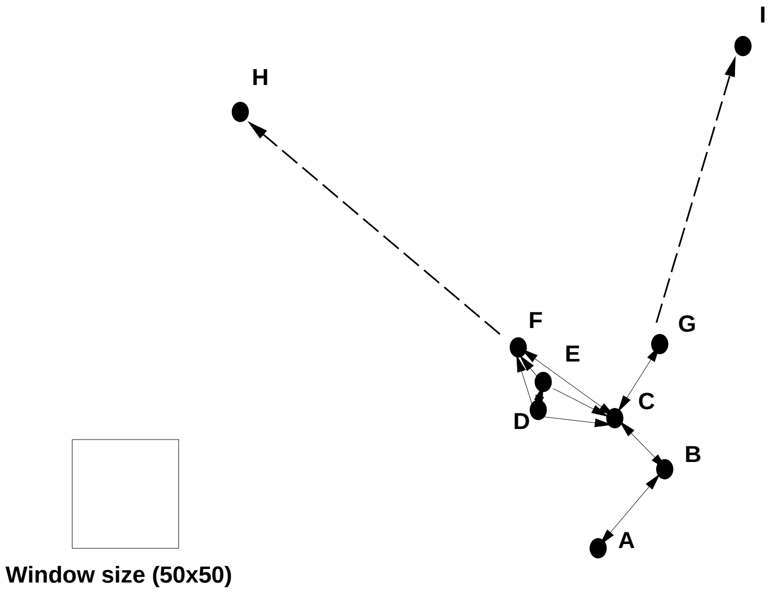


Figure 7.1: An example of isolated parkways

	A	B	C	D	E	F	G	H	I
A		1							
B	1		1						
C		1				1	1		
D			1		1	1			
E			1	1		1			
F			1						
G			1						
H									
I									

Table 7.1: Transition matrix

distance, temporal distance, reliability measure, or any other index of evaluation that can be measured or generated. An example of (distance) cost matrix corresponding to figure 7.1 is shown in table 7.2.

	a	b	c	d	e	f	g	h	i
a		40.31							
b	40.31		38.60						
c		38.60				52.35	59.55		
d			34.13		16.76	35.06			
e			34.67	16.76		20.25			
f			52.35						
g			59.55						
h									
i									

Table 7.2: Cost matrix

7.1.2 Parkway paths

The shortest path in a parkway is generated by Dijkstra's algorithm that searchs single-source shortest path to all other vertices. This requires $O(n^2)$, were n is the number of vertices in the graph. The algorithm is given in the following[3]:

begin
$S \leftarrow \{v_0\}$;
$D[v_0] \leftarrow 0$;
for each v in $V - \{v_0\}$ **do** $D[v] \leftarrow l(v_0, v)$;
while$S \neq V$ **do**
begin
choose a vertex w in $V - S$ such that $D[w]$ is a minimum;
add w to S;
for each v in $V - S$ **do**
$D[v] \leftarrow \mathbf{MIN}(D[v], D[w] + l(w, v))$
end
end

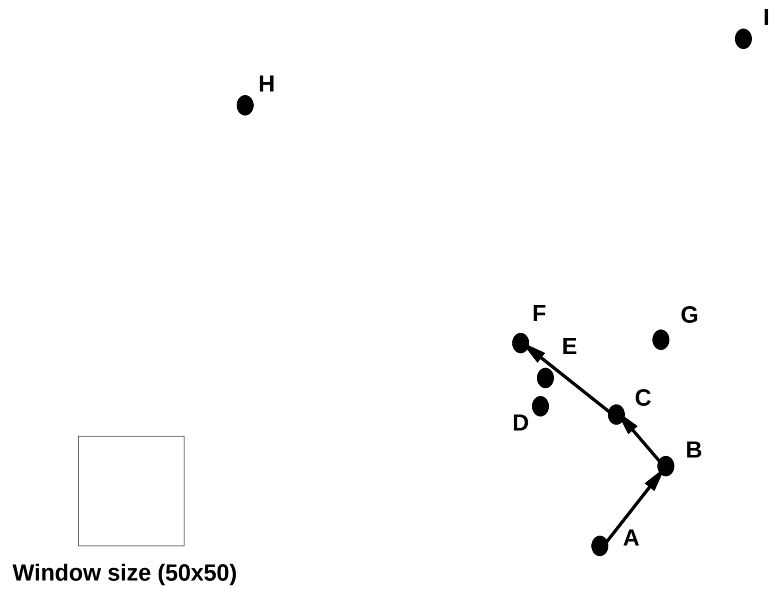


Figure 7.2: An example of the shortest path within a parkway (from A to F)

An example of shortest path in isolated parkway of figure 7.1 is shown in figure 7.2. The starting point is at object A, and the destination is at object F.

7.1.3 Trajectory paths

In our example of figure 7.1, the dotted arrows indicate trajectory paths. The updated transition and cost matrices with trajectory information are shown in tables 7.3 and 7.4. The overall shortest path is a combination of parkway path and trajectory path. An example is shown in figure 7.3.

7.1.4 Random walk

So far, we only talked about traversal methods (parkway and trajectory traversals) that have deterministic cost measures. This means that no matter how many time the robot navigates on a pre-determined route, the cost will be pretty much the same within a given tolerance, because it will be traveling along the same path.

	a	b	c	d	e	f	g	h	i
a		1							
b	1		1						
c		1				1	1	1	1
d			1		1	1			
e			1	1		1			
f			1					1	
g			1						1
h									
i									

Table 7.3: Transition matrix

	a	b	c	d	e	f	g	h	i
a		40.31							
b	40.31		38.60						
c		38.60				52.35	59.55	132.52	159.93
d			34.13		16.76	35.06			
e			34.67	16.76		20.25		132.52	
f			52.35						
g			59.55						159.93
h									
i									

Table 7.4: Cost matrix

Deterministic traversal is possible only if the navigator is able to keep track of its location in respect to visible landmarks. In another words, the navigator has to be able to constantly keep updating its "you are here" pointer as it moves along [35] [31].

However, in some complex areas of the environment, it may be too confusing or even impossible to keep an accurate "you are here" pointer. For example, if there are more than one object that resembles the currently sought landmark, the navigator will lose some degree of confidence as it decides on one. One solution to this problem is to use error verification and recovery methods to reinitialize the navigator's confidence at certain check points, or to recover from its mistakes,

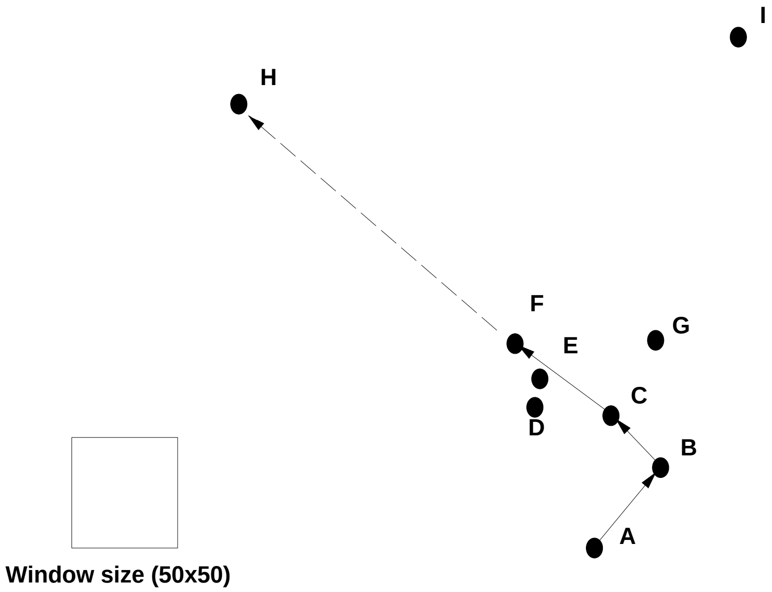


Figure 7.3: An example of the overall shortest path including a trajectory (from A to H)

when errors are detected, as presented in Chapter 6. Another solution is to have the navigator ramble around until it finds a specific landmark that has a unique feature associated with it. Then the navigator is able to reset itself.

Formally, we define random walk in a parkway as follows:

1. Choose a random quadrant relative to the current reference point (if empty, choose another one, etc.)

2. In the chosen quadrant, choose randomly any object and use it as the next reference point.

Using random walk for navigation

A random walk can be used during the navigation only if the currently sought landmark has a unique QOC within its parkway. The navigator simply executes the random walk behavior and at each selected object, it examines the QOC to see if it has reached the goal landmark. This can be a time consuming process if the parkway is large. But in a small parkway with only a few objects (3,4, or 5), random walk can be cost effective.

A parkway with n nodes can have at least n and at most $n(n-1)$ directed edges. In general, the probability of a parkway having a unique QOC object decreases rapidly as the number of nodes increases. This is because there are only 5 QOCs (0 through 4) and as the number of nodes increases, the probability of using up the 5 QOCs increases. Due to the two reasons stated above (time complexity, availability), small parkways are more suitable for random walks than large parkways.

In figures 7.4, 7.5, and 7.6, we see parkways formed in environments with 25, 50, and 100 objects, respectively. As before, the window size (10×10) is shown at the lower left-hand corner of each figure. Objects with unique QOC (within its parkway) are represented by large black circles. As we can see, the number of unique QOC landmarks decreases rapidly as the population increases. In figure 7.4, most objects are by themselves parkways, thus these objects have unique QOC within their own parkways. Figure 7.6 is more interesting. There are 10 parkways altogether. In the top right-hand quadrant of the environment, we can see 3 small parkways, each having one or more unique QOC landmarks. These parkways can serve as good trajectory targets because the navigator can verify its path by randomly walking to the landmarks.

Probability model for cost prediction

We can use Markov process to model the probability of reaching the unique QOC landmark within a parkway. For example, let us assume we have 5 objects in a parkway with the transition probability matrix, P, as shown in table 7.5. Suppose object B is the object we are looking for, that has a unique QOC. Then

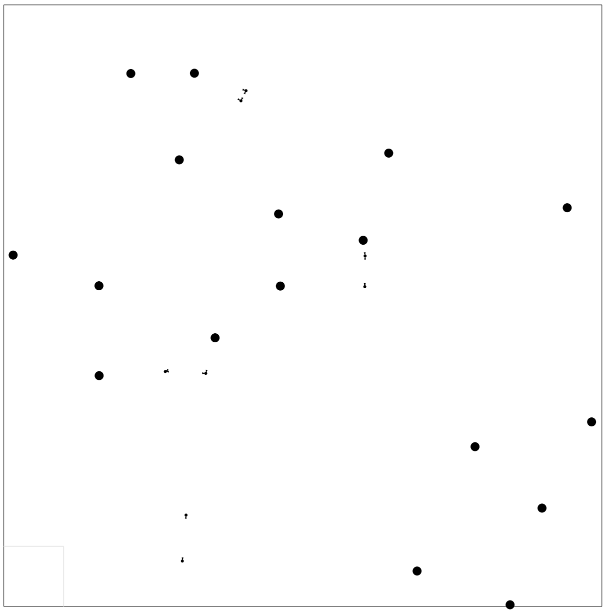


Figure 7.4: Random walk landmarks in an environment with 25 object

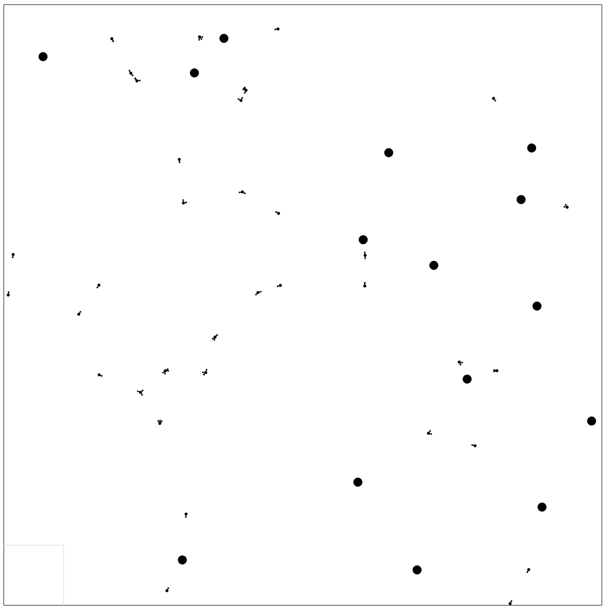


Figure 7.5: Random walk landmarks in an environment with 50 objects

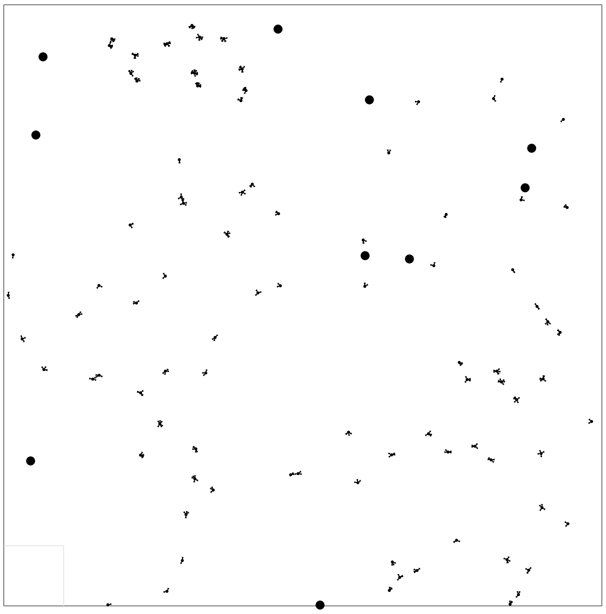


Figure 7.6: Random walk landmarks in an environment with 100 objects

	A	B	C	D	E
A	0	0.5	0.5	0	0
B	1	0	0	0	0
C	0	0	0	1	0
D	0	0.25	0.25	0	0.5
E	0.5	0	0	0.5	0

Table 7.5: Probabilistic transition matrix

we can find in the matrix, the probabilities of reaching B in one step from other objects, from column B. Probability of reaching the landmark in n-steps can be calculated by raising P matrix to n-th power. Then, if the matrix entry for the object with a unique QOC has a probability value greater than a certain threshold value, the object may be used as a random walk landmark.

7.1.5 Tie breaking heuristics

Heuristics in choosing isolated points

Sometimes, particularly in sparse areas, two or more landmarks in the navigator's view window will seem equally isolated to the navigator, i.e., they have the same *mnv c-values* described in Chapter 5. To break the tie, we use a heuristic to choose the isolated landmark that is farthest away from the current reference point. This method is based on the observation that the navigator tends to travel towards the goal (and away from the starting location) at any instance of navigation. Therefore, by selecting the farthest landmark from the current reference point, the navigator usually moves closer to the goal. This is in contrast to a more usual search heuristic, which would select the landmark closest to the goal. The reason we use the former is that the global position of the goal (or even of the two competing landmarks) is unknown to the topologically driven navigator.

Heuristics in trajectory tasks

As described in Chapter 6, the orientational error in trajectory is minimized when the leading edge of the view window and the line of trajectory is at right angle[1], that is θ is $\pi/2$. Therefore, by favoring trajectories whose angle of incidence is close to $\pi/2$, we can get a more reliable path. Also, the error angle α is inversely proportional to the distance between two reference objects. Therefore, the heuristic of choosing the object that is farthest away from the current reference point also applies to the trajectory task.

[1]As stated earlier, this is an artifact of having a rectangular view window.

7.1.6 Definition of optimal path

Depending on the needs, different constraints can be imposed into the computation of the optimal path. Some examples of optimality criteria are:

- Distance - to minimize the travel distance.
- Reliability - to maximize the probability of reaching the destination.
- Travel time - to minimize the travel time.
- Sensing cost - to minimize the cognitive strain of the navigator involved in processing its sensory inputs.
- Communication cost - to minimize the number of entries in the custom map.

Currently, we have implemented cost functions that estimate the distance of navigation path D and the reliability of navigation path R. One of map-maker's responsibility is to generate a path that either minimizes D, or maximizes R. Unfortunately, in some environments, these two cost estimates are in a direct conflict of each other. For example, if the shortest distance path involves in a highly cluttered area, the reliability would be low. Conversely, a reliable path that avoids highly cluttered areas may force the navigator to detour around a shortest path. Here, we suggest a third function C that compromises D and R, defined as: $C = \log D - \log R$. Using C as our cost estimate for travel, we can apply Dijkstra's shortest path algorithm to derive a path that minimizes the D/R ratio. The generated path will tend to favor short paths and sparsely populated areas. Note that R is a function of not only n (population) but also σ (position estimate error) as in equations 6.2 and 6.23. If σ is small, the path will resemble the D-path and if σ is large, the path will resemble the R-path. This means that if the navigator's metric ability within its window is good, the optimal path will be the metrically shortest path. On the other hand, if the navigator's metric ability is

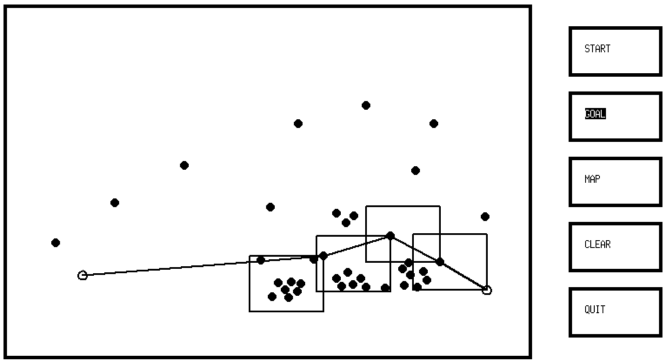


Figure 7.7: The shortest distance path

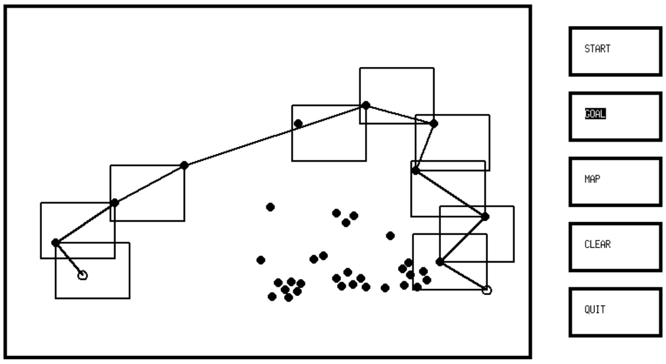


Figure 7.8: The most reliable path

poor, the optimal path will be the one that least confuses the navigator. Examples of shortest distance path, most reliable path, and compromised path are shown in figures 7.7, 7.8, and 7.9, respectively.

7.1.7 Context-based landmarks

The shortest path in a parkway network does not guarantee the shortest travel path for the navigator because the cost (distance traveled) of achieving a landmark depends on the current "context" of the landmark. For example, consider a path generated within a parkway. Each arc of the path represents the distance between a reference point and a subsequent landmark. However, depending on which corner

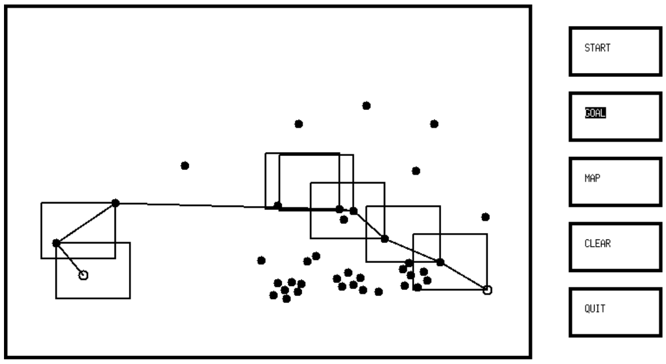


Figure 7.9: A compromised path

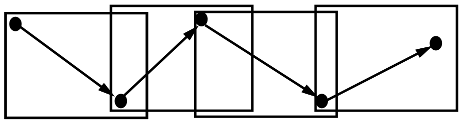


Figure 7.10: Zigzag arrows indicate the physical distances between landmarks, whereas the actual movement of the navigator is almost linear

of the view window (SE, SW, NE, NW) the new landmark is placed, the actual traveled distance of the navigator may or may not be equal to the arc length. In an extreme situation, we can visualize a parkway path that zigzags but the actual traveling movement of the navigator is linear and the traveled distance is much smaller (see figure 7.10.) This is analogous to a driver following a series of landmarks, such as buildings, in which the physical distances between landmarks is not equal to the odometer reading. Using our navigator model, each landmark has 4 different contexts, corresponding to the SE, SW, NE, and NW corners of the navigator's view window. In order to implement the context-based landmark following, data structures to represent parkways, trajectories and the cost matrix, must be modified. Instead of n nodes in the parkway network, we now have $4n$ nodes, namely, the 4 ways each landmark can be "seen" by the navigator. The size of the cost matrix grows by a factor of 4^2, but stays very sparse. Out of $16n^2$ cells, at most $12n$ are used. Therefore, a sparse matrix representation is needed for storage efficiency. Time complexity of the search algorithm is also increases,

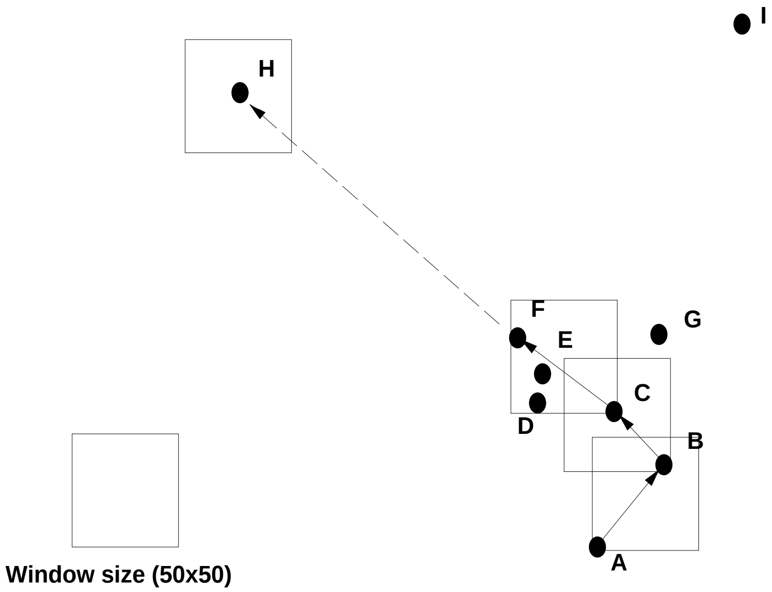


Figure 7.11: A context-based path example

but by a factor of 4^2 since Dijskstra takes $O(n^2)$. The shortest context-based path in our example parkway of figure 7.1 is shown if figure 7.11.

Cost of achieving context-based landmarks

The cost of distance traveled varies depending on which context of a landmark the navigator chooses during the navigation. For example, the distance cost of choosing the SE context of a landmark is different from the distance cost of choosing the NW context of a landmark.

Parkway traversal cost The distance cost of obtaining an isolated point and moving to the next position depends on which corner of the view window the newly achieved isolated point is to be positioned, i.e., SE, SW, NE, NW. The cost for each of the context is simply the distance between two points, C and L. Here, C is the corner point of current view frame corresponding to the corner context of the new landmark, and L is the position of the new landmark. This is shown in

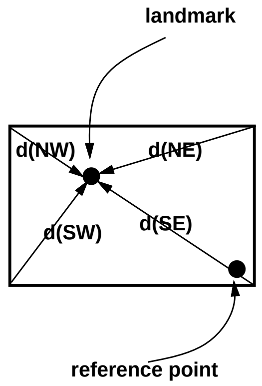


Figure 7.12: The cost of parkway context

figure 7.12, where $d(\mathrm{NW})$ is the cost of achieving the NW context of the landmark, etc.

Trajectory traversal cost The distance cost of a trajectory movement consists of three components:

1. The distance cost to center the window to the virtual midpoint of the current reference point and the new landmark, indicated by arrow 1 in figure 7.13.

2. The distance from the window edge to the trajectory goal point in the direction of the trajectory, indicated by arrow 2 in figure 7.13.

3. The cost to adjust to the appropriate context (SE, SW, NE, NW), indicated by arrow 3 in figure 7.13.

7.2 Custom map generation

For the actual navigational task, the navigator solely depends on the custom map created by the navigator. The custom map contains the navigational instructions that correspond to the parkway or trajectory path segments. Each entry in

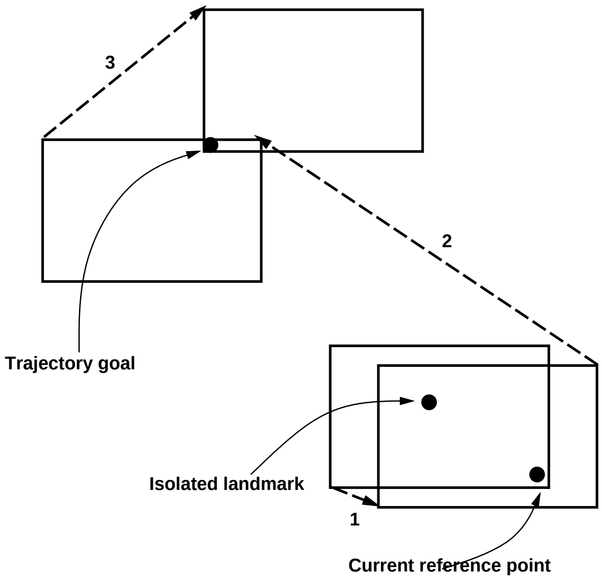


Figure 7.13: The cost of trajectory context

the custom map instructs the navigator (1) what to look for (landmark seek phase), (2) how to verify (error detection), (3) how to correct itself (error recovery), (4) and where to go with respect to the landmark(adjust phase). In Chapter 4, we discussed the environmental description language for qualitatively identifying a landmark. The vocabulary in this description language defines the command set available to the navigator.

7.2.1 Grammar

Each entry in the custom map is of the form $(D, [T], [E], M)$. D is a description language vocabulary that identifies a landmark in current view. $[T]$ is an optional term that indicates trajectory, where $T \in \{\text{Trajp, Trajn}\}$. Trajp indicates trajectory in the positive direction (from the reference point to the isolated point), and vice versa. $[E]$ is another optional term which consists of a set of instructions for error recovery sequences as described in Chapter 6. Finally, M is the corner designator to indicate which of the 4 corners (SE, SW, NE, NW) the chosen landmark is to be positioned. Symbols $*$ and $-$ are used to indicate “any” and “null”,

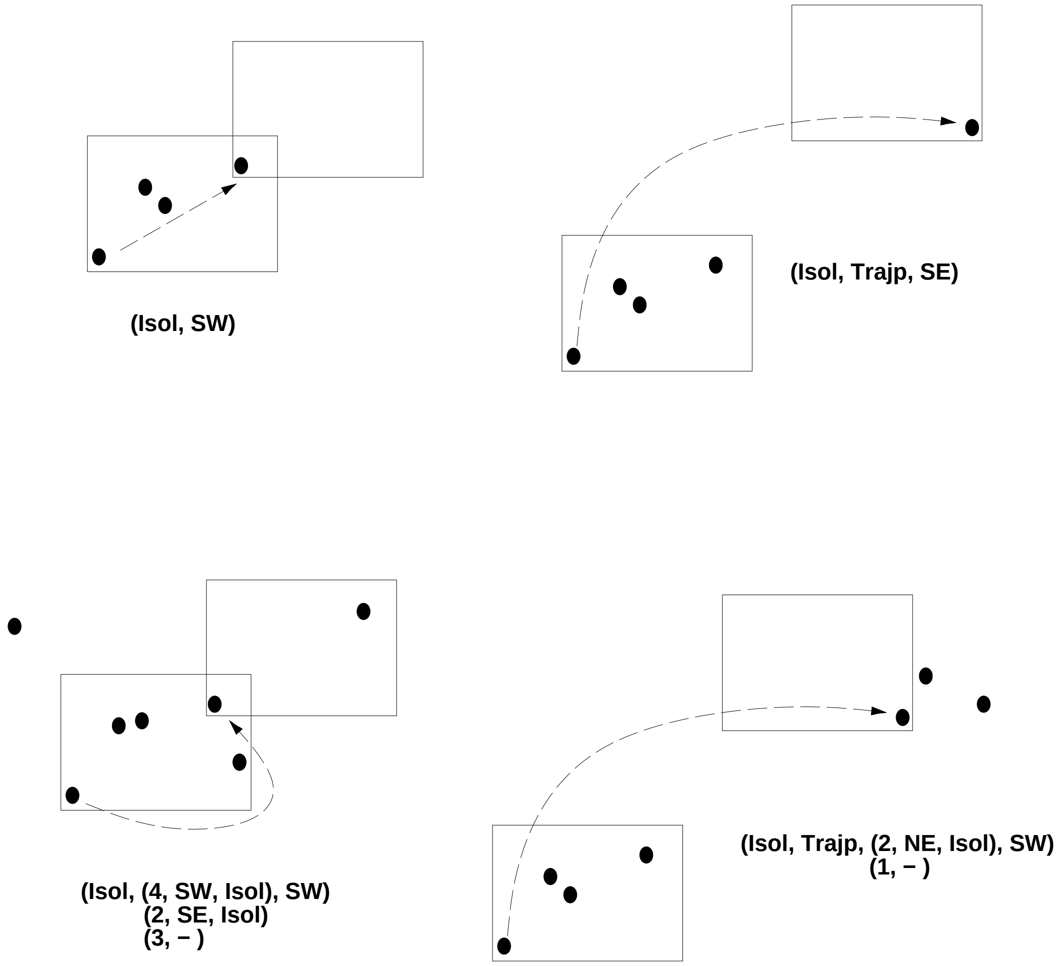


Figure 7.14: Examples of custom map entries

respectively. For example, instruction (*, SW) means to choose "any" object in the view and adjust to SW corner. (Obvious, -) means that the navigator can select the obvious point and do nothing (usually it means that it has reached the goal).

Some examples are shown in figure 7.14. In the simplest case, we have the custom map entry, (Isol, SW) corresponding to the top left hand corner diagram of figure 7.14. (Isol, Trajp, SE) is shown on the top right hand corner. The custom map entry becomes more complex when error recovery instructions are included, as in the bottom diagrams. To demonstrate the compactness of the custom map, we have provided some examples of real life directional instructions (invitation to a

Custom map command	Real world example
(Isol, SW)	Go to the right side of the modern looking building
(Isol, Trajp, SE)	Move towards the modern looking building until you see a church. Go to the right side of the church.
(Isol, (2, NE, Isol), SW) (3, -)	Go to the church. If the entrance is looking at a 2 way intersection, you are at a wrong place. In that case, take the road that goes to the left and it will bring you to the correct church. The entrance of the church should see a 3 way intersection. Go to the right side of the church for the reception.
(Isol, Trajp, (2, NE, Isol), SW) (3, -)	Move towards the modern looking building until you see a church. Go to entrance of the church to see if it sees a 3 way intersection. If you see a 2 way street, you are at a wrong church. In that case, take the road that goes to the left and it will bring you to the correct church. Go to the right side of the church for reception.

Figure 7.15: Custom map entries versus real life directional instructions

church wedding) in figure 7.15 that roughly correspond to our custom map entries.

7.2.2 Examples

Let us go back to figures 7.7, 7.8, and 7.9, and see what the corresponding custom maps are. First, the custom map for the shortest distance path in figure

7.7 is given by:

((Isol, SE), (Isol, NE), (Isol, NE), (Isol, Trajp, (2, NW, Isol), (1, -), -))

There are 4 entries in the custom map. First three take the navigator from the starting position (shown on the lower right hand corner in the figure) to the end of the first parkway. There are no error recovery sequences embedded in the first three entries. In the first and the third entry commands, neither error recovery nor error detection is possible because the landmarks in the corresponding windows do not have unique QOC's. In the second entry, the navigator sees only one object (obvious point) in its view. Therefore error verification is implicit and error recovery is not necessary. In the last entry of the custom map, the navigator is told to obtain the isolated point and move in the positive direction of the formed trajectory. If the newly appearing object has QOC of 2, then the navigator is told to move to NW and to pursue the isolated landmark to recover from the error. Otherwise, if the QOC is 1, it has reached the destination.

The custom map for the most reliable path in figure 7.8 is given by:

((Isol, SW), (Isol, SE), (Isol, SW), (Isol, SE), (Isol, NE), (Isol, Trajp, NE), (Isol, NE), (Isol, NW), (Isol, -))

Here the navigator travels in a detoured route to avoid cluttered regions. Apart from the initial position, where there are 4 objects (excluding the reference point), every other view window has only 1 object to choose the isolated point from. Therefore error recovery routines are unnecessary in the corresponding entries. Note that reliability tends to imply "implicitly error recoverable", although the two concepts are not the same. In the first entry, error detection and recovery are not possible because the isolated landmark does not have a unique QOC.

The custom map for the compromised path in figure 7.9 is given by:

((Isol, SE), (Isol, SE), (Isol, SE), (Trajp, NE), (Isol, NW), (Isol, -))

Here we see that the navigator tries to avoid cluttered region avoiding detouring too much from the shortest distance path. None of the custom map entries has error recovery sequences embedded, mostly because the navigator follows obvious points. From these examples, we note a weakness of our vocabulary. A path segment with no error recovery necessary is treated the same way as a path segment with no error recovery possible. Currently, both lead to no explicit directional instruction in the custom map. This is the issue of "knowing to be lost" we are still puzzling over.

7.3 Summary

Path planning and custom map generation are the two main objectives of the map-maker. In this chapter, we studied in detail how the data structures, such as parkways, trajectories, transition and cost matrices, are used for the computation of the optimal path for the navigator. We demonstrated that various cost optimality can be used to produce different paths. For example, we saw differences between shortest distance path, most reliable path, shortest travel (context) path, etc, each of which are optimal in its own sense.

We defined the grammar for the custom map entries. The basic structure of a custom map command is of the form "seek-verify-adjust", according to which the navigator "seeks" for a landmark, "verifies" the correctness using the error verification (correction) sequence, and "adjusts" to get ready for the next move.

Chapter 8

Implementation and Experiments

In this chapter, we describe the implementation of the map-maker and the navigator. We also discuss the results of the navigational experiments performed. Hardware components of the map-maker and the navigator consists of CCD cameras, an IBM robot arm, PIPE, and a Sun workstation. Software components are comprised of an X-windows based user interface, high-level reasoning system for map-maker and the navigator written in C language on the Unix platform, programs written in AML for the robot control, and programs written in ASPIPE for basic image processing of PIPE. These are presented in section 1. In section 2, we present the results of our simulated and actual experimentations, and analyze the success and the failure cases.

8.1 Implementation

We have implemented our map-maker and navigator using an IBM 7575 SCARA robot arm, two Sony XC-77 CCD cameras, PIPE, which is a high speed real-time image processor, and a Sun SPARC workstation for high level control of the navigator and as the map-maker. Figure 8.1 and 8.2 show the configuration of the

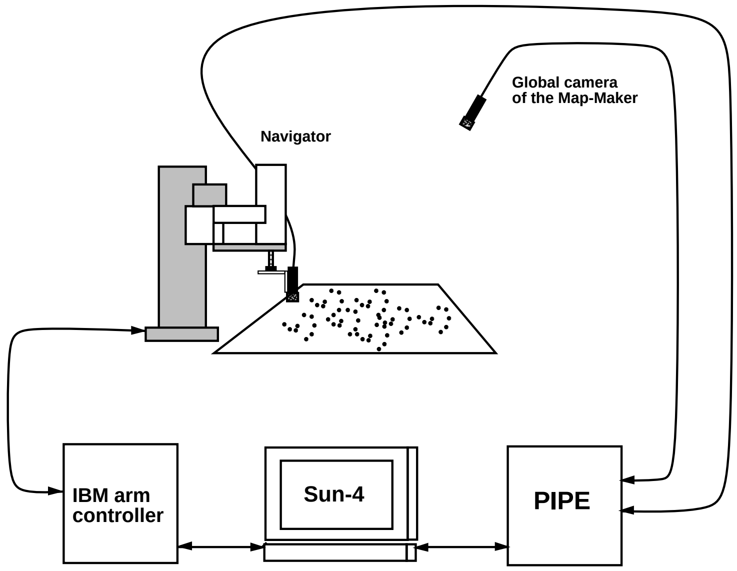


Figure 8.1: The experimental setup configuration

map-maker and the navigator.

8.1.1 Map-maker

The map-maker is comprised of one CCD camera located at a global position that can capture the whole workspace of the navigator. This camera is attached to PIPE which grabs the image and sends it to the Sun workstation which runs the "map-making" program based on the centroid information of the scattered objects. The map-maker generates a file called "custommap" which contains the list of directional instructions for the navigator, as described in chapter 7.

Calibrating the map-maker

The (almost) omnisciency assumption of the map-maker requires that the image captured by the global camera (see Figure 8.1) correctly be used to generate the position of each object. To account for image distortions due to translation, rotation, and perspective, we use a simple geometric calibration matrix A that

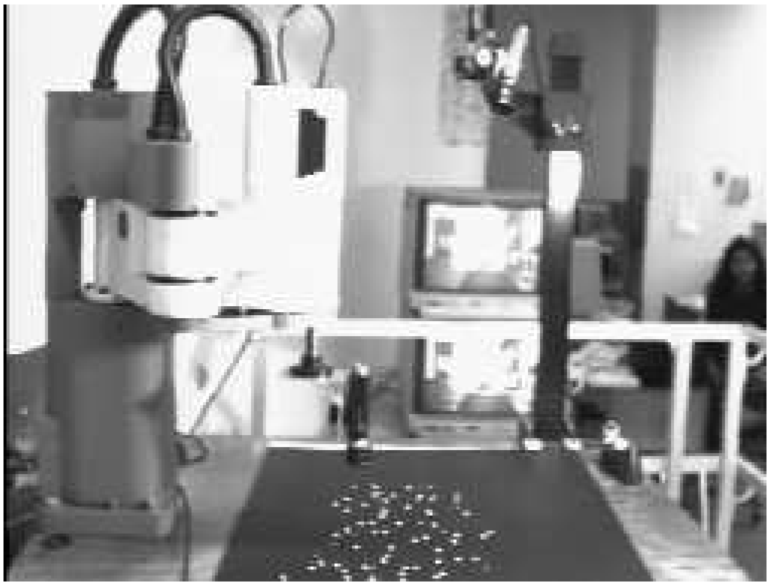


Figure 8.2: The map-maker and the navigator

transforms the homogeneous world coordinates $(X, Y, Z, 1)$ into the homogeneous camera coordinates $(U, V, 1)$ [21]. The transformation is represented by the following equation:

$$\vec{c_h} = A\vec{w_h} \tag{8.1}$$

where $\vec{c_h}$ is the homogeneous vector representation of the camera coordinates, A is the transformation matrix, and $\vec{w_h}$ is the homogeneous vector representation of the world coordinates. More precisely:

$$\begin{bmatrix} c_{h1} \\ c_{h2} \\ c_{h3} \\ c_{h4} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{8.2}$$

The camera coordinates, U and V are then:

$$U = \frac{c_{h1}}{c_{h4}} \tag{8.3}$$

$$V = \frac{c_{h2}}{c_{h4}} \tag{8.4}$$

Substituting equation 8.3 and equation 8.4 in equation 8.2 and expanding the matrix products yields:

$$Uc_{h4} = a_{11}X + a_{12}Y + a_{13}Z + a_{14} \tag{8.5}$$

$$Vc_{h4} = a_{21}X + a_{22}Y + a_{23}Z + a_{24} \tag{8.6}$$

$$c_{h4} = a_{41}X + a_{42}Y + a_{43}Z + a_{44} \tag{8.7}$$

We do not expand the matrix product for ch_4 because it represents the z-coordinate of the camera. Substituting equation 8.7 into equations 8.5 and 8.6 yields:

$$a_{11}X + a_{12}Y + a_{13}Z - a_{41}UX - a_{42}UY - a_{43}UZ - a_{44}U + a_{14} = 0 \tag{8.8}$$

$$a_{21}X + a_{22}Y + a_{23}Z - a_{41}VX - a_{42}VY - a_{43}VZ - a_{44}V + a_{24} = 0 \tag{8.9}$$

Since there are 12 unknowns, and two equations, we need at least 6 data points to solve equations 8.8 and 8.9. We used 24 dots uniformly distributed on the workspace of the robot. The camera parameters (the elements of matrix A) were then estimated by solving equations 8.8 and 8.9 by least squares. Figure 8.3 shows an image of the calibration dots captured by the global camera. The centroid information of these dots in the camera coordinates, together with the real world coordinates of the dots were inputed into a program that computes the Singular Value Decomposition (SVD) for the input matrix and does the least square fit estimation.

8.1.2 Navigator

The navigator is comprised of a second camera attached to the IBM robot arm. This camera is also connected to PIPE for image processing of each scene as the

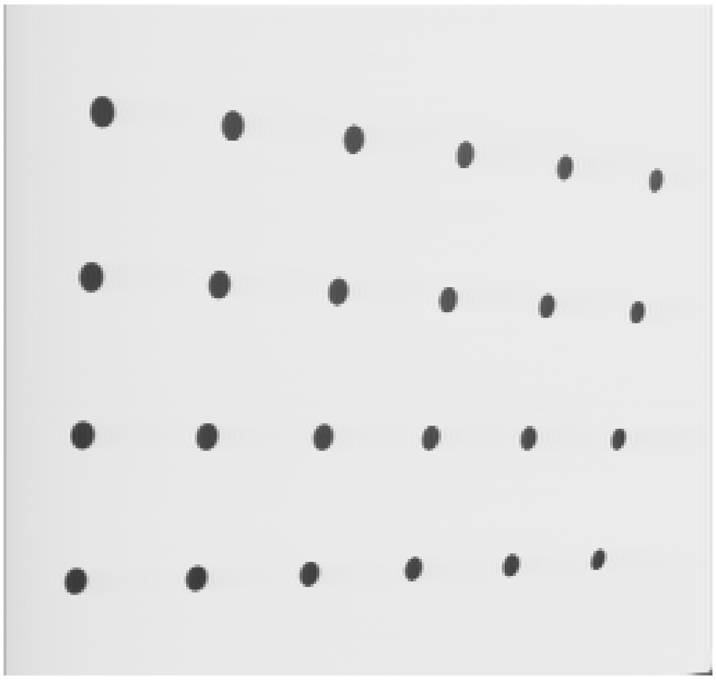


Figure 8.3: The calibration dots viewed from the global camera

navigator moves along. For each directional instruction in the custom map, the derivation of the most isolated point and the amount of "adjust" of the robot for the corresponding instruction is computed by the Sun workstation. It then sends out low level instructions to the IBM arm controller for the actual movement.

8.1.3 User interface program

The user interface program of the map-maker for the purpose of interaction with human is implemented on the Sun workstation running X-Windows systems. When the map-maker program is started, computed position of each scattered object in the environment is collected and displayed on the interface windows. Using available command buttons, such as START, GOAL, MAP, CLEAR, and QUIT, the user is able to choose the initial position and the destination, and to ask the map-maker to compute optimal path and to generate custom map. Figure 8.4 shows the user interface program.

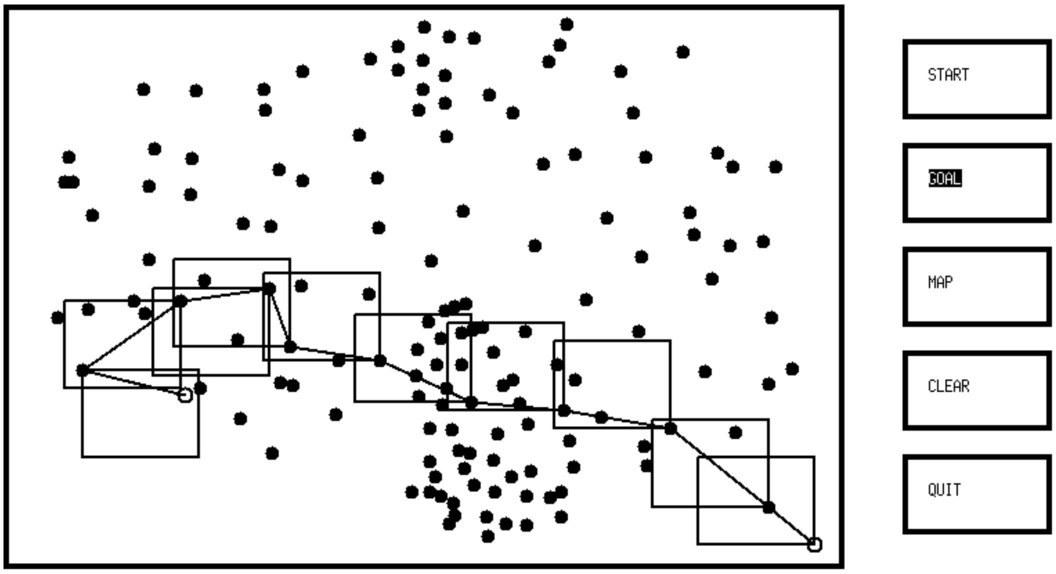


Figure 8.4: An example showing a path that passes through a highly cluttered unreliable region

8.2 Experiments

8.2.1 Sparsely versus densely populated environment

As described in chapter 6, our statistical experiments indicated that the isolated point descriptor is less reliable in a scene with dense population. We also noted that the reliability is a function of σ, which measures the accuracy of the sensor. For details of the error models, refer to chapter 6. Our test runs with the robot arm navigator agreed with these statistical results. The robot tends to fail in (1) highly populated environments, and in (2) areas of environments where there are large clusters of objects. Figure 8.4 shows a path created by the map maker that passes through a wall of highly cluttered region of the environment. Figure 8.5 shows the same environment with an adjusted path that avoids the unreliable cluttered region.

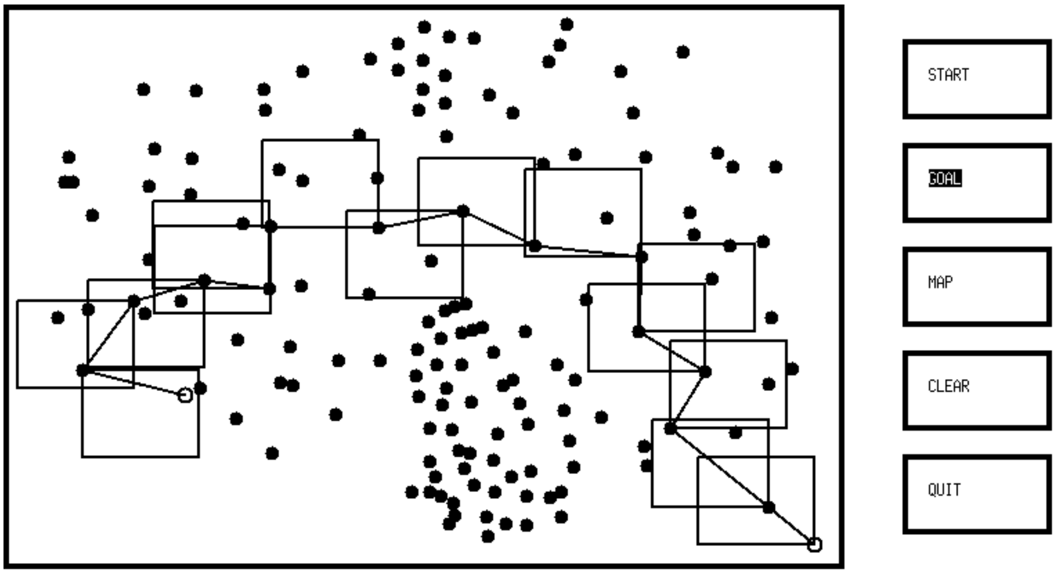


Figure 8.5: A result of using the "reliability" criteria to avoid the cluttered regions

8.2.2 Cost optimality criteria

Minimum travel distance

Minimizing the travel distance is done in terms of the context-based distance as described in chapter 7. The reliability of the shortest distance path is sometimes very low when there are cluttered regions along the way. We see a such path in figure 7.7. The corresponding costs for this path are:

```
Distance cost       : 444.930
Unreliability cost  :   0.539
```

The unreliability cost is measured by $-\log(R)$, where R is the reliability. We use unreliability measure instead of R for the purpose of deriving path using Dijkstra's shortest path algorithm, which tries to minimize cost.

Maximum reliability

Maximizing the reliability (or minimizing the unreliability) of overall path usually yields longer distance path as described in chapter 7. We see a such reliable but detouring path in figure 7.8. The corresponding costs for this path are:

```
Distance cost       : 731.506
Unreliability cost  :   0.127
```

Hybrid method

Use of a compromised cost function (to minimize D/R ratio), as described in chapter 7, yields a path that is a neither too long nor too unreliable. We see a such path in figure 7.9. The corresponding costs for this path are:

```
Distance cost       : 457.931
Unreliability cost  :   0.214
```

As we can see, the shortest distance path has the lowest reliability (highest unreliability measure), whereas the most reliable path has the highest cost in terms of distance travel. The compromised path has an intermediate cost values in both measures.

8.2.3 Effects of sensor accuracies

Accuracy of the map-maker's and the navigator's sensors are modeled in the positional information error estimator, as presented in chapter 6. Depending on the standard deviation of the positional error, which is directly related to the sensor accuracy, the reliability of each path segment may differ. In this section we compare paths generated at different sensor accuracies (σ).

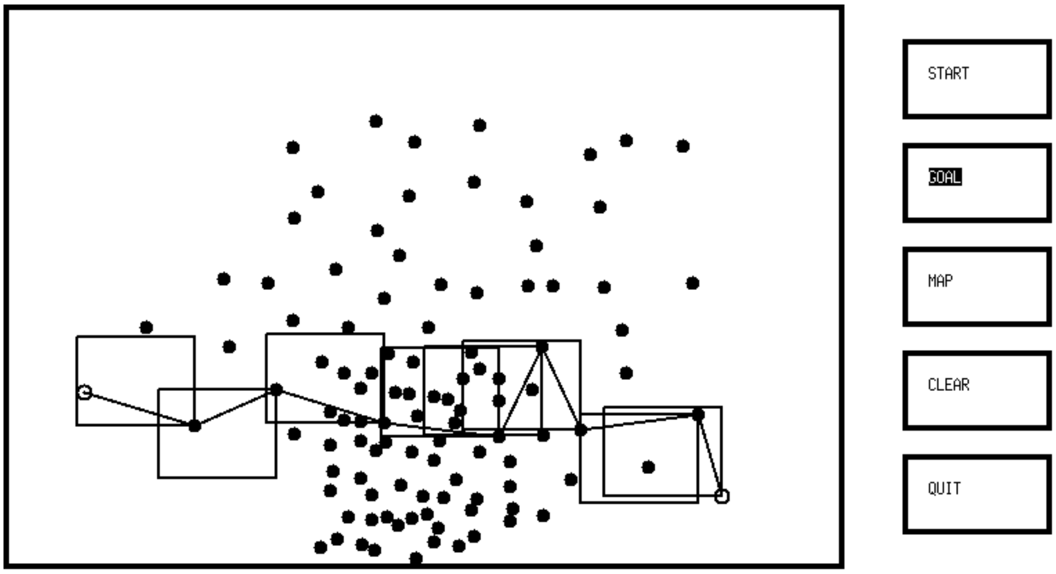


Figure 8.6: D/R path when σ is small (1 mm)

Small σ - accurate sensor

As σ decreases, the reliability of an isolated point increases. Figure 8.6 shows a path generated by the map-maker using the compromised cost function D/R when σ is equal to 1 mm. Because R is large (high reliability), the computed path is close to the shortest distance path. Note that the navigator passes through the cluttered region. The corresponding costs for this path are:

```
Distance cost       : 420.57
Unreliability cost  :   0.91
```

Large σ - inaccurate sensor

As σ gets large, reliability of an isolated point decreases, as shown in figure 6.1. Figure 8.7 shows a D/R path generated by the map-maker when σ is equal to 5 mm. Because R is large (low reliability), the computed path resembles more to

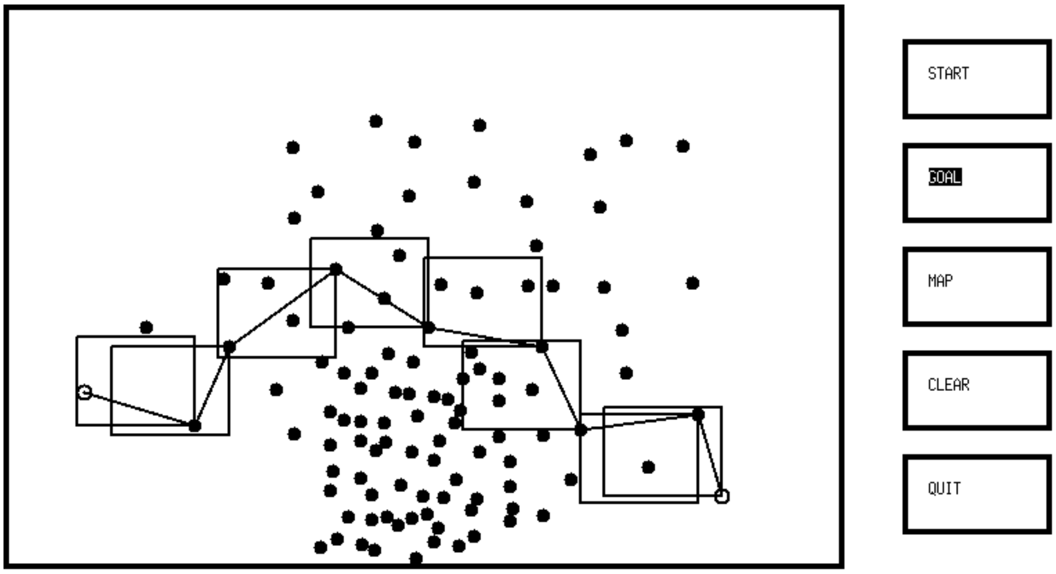


Figure 8.7: D/R path when σ is large (5 mm)

the most reliable path. Note that the navigator tries to avoid the cluttered region. The corresponding costs for this path are:

```
Distance cost       : 460.43
Unreliability cost  :   2.27
```

8.2.4 Context-based navigation

As previously stated in chapter 7, the distance between landmarks is not necessarily equal to the actual travel distance (context-based.) In an extreme case, as in figure 7.10 we saw that the context-based shortest path can be a zigzag type landmark path. In figure 8.8, we see a shortest path generated in terms of minimizing the distances between the landmarks. In figure 8.9, we see the shortest context-based path for the same start-goal configuration. The landmark distance cost and the context based travel distance cost for the path in figure 8.8 are:

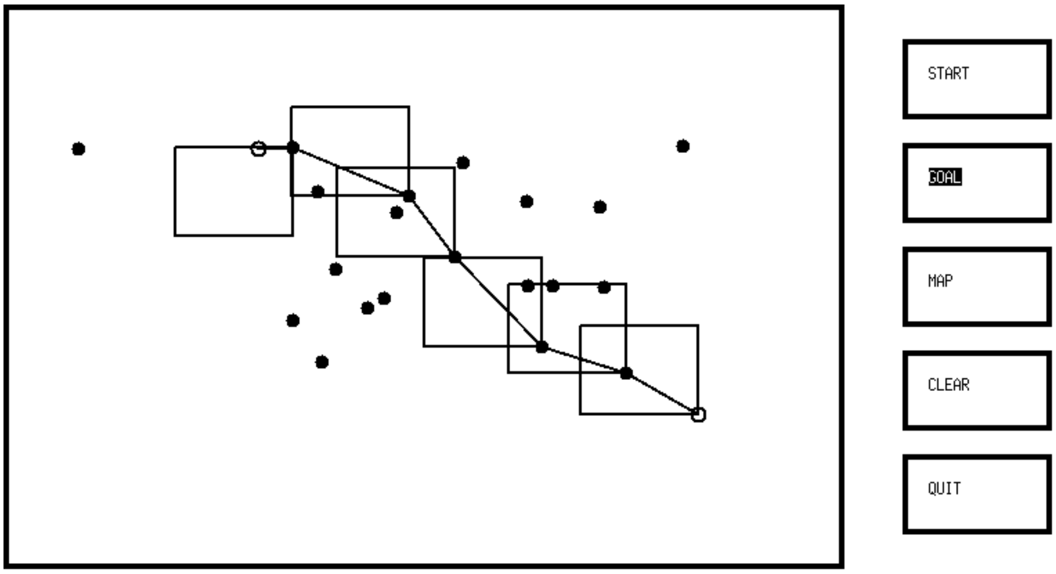


Figure 8.8: The shortest landmark distance path

```
Landmark based distance cost: 385.465
Context  based distance cost: 358.639
```

Landmark distance cost and the context based travel distance cost for the path in figure 8.9 are:

```
Landmark based distance cost: 411.413
Context  based distance cost: 347.039
```

8.2.5 Error detection and recovery

We have implemented the QOC method as described in chapter 6. In figure 8.10, we see a path generated in a simple environment. The starting position is at the lower right-hand corner. The corresponding custom map for this path is:

> ((Isol, SE), (Isol, (4, NE, Isol), (3, -), SE), (Isol, SE), (Isol, (4, NE, Isol), (2, -), -))

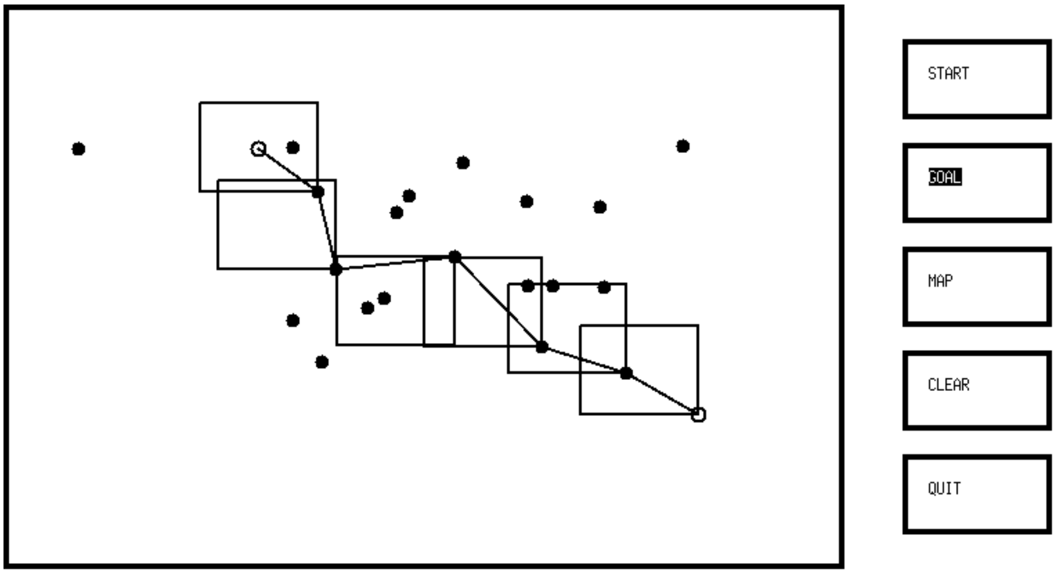


Figure 8.9: The shortest context-based distance path

In our test run with the robot, the navigator failed to achieve the correct landmark during the execution of the second entry of this custom map (corresponding to the second window from the right in the path). The correct landmark was marginally out of the navigator's visual window. Therefore, the navigator incorrectly identified the top-most object in the window as the isolated landmark. When the error verification sequence was executed, QOC was 4, which indicated the robot to adjust to NE and to recapture the correct landmark. In this case, the error recovery routine handled the error caused by the object which was positioned near the boundary of the view window. After the error recovery routine, the navigator was able to get back on track, and eventually get to the destination successfully. Note that this journey would have been a failure if not for the error recovery routine. Effectively, the error recovery routine has increased the reliability of this path.

Figure 8.11 shows a path generated by the map maker that is highly error correctable. The navigator starts near lower left-hand corner of the environment and navigates to the destination at the lower right-hand corner. This path con-

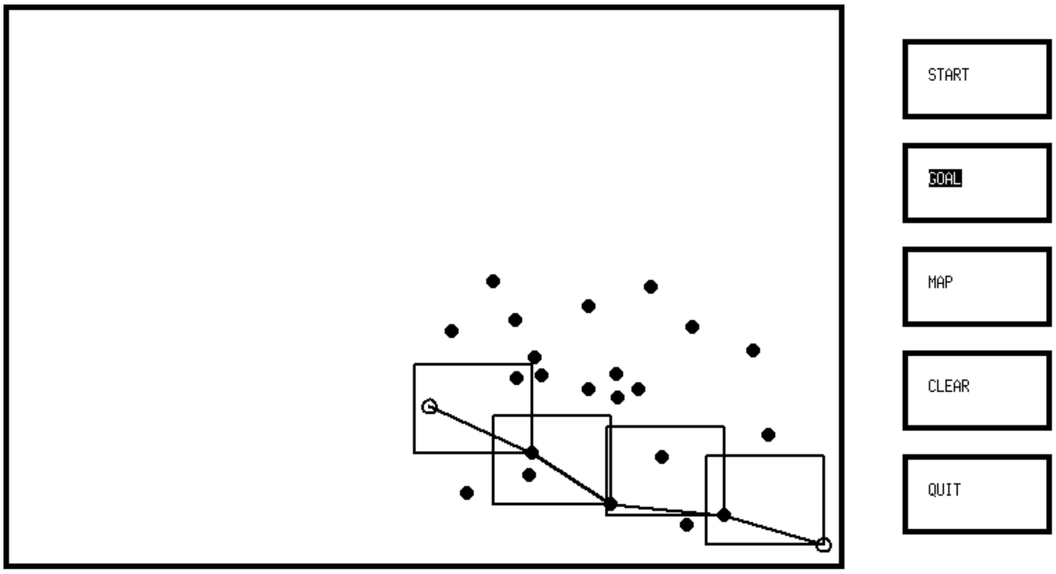


Figure 8.10: An example of a simple error recoverable path

sists of both parkway path segments and trajectory path segments that are error recoverable. Corresponding commands in the custom map (as generated on the user interface display during the map-making session), including error detection and correction statements are shown in figure 8.12.

8.2.6 Failures

In general, the system was most robust in sparsely populated environments and tended to fail where the objects were heavily cluttered. Long trajectory path traversals sometimes failed when the traveling direction was at an acute angle (rather than near an orthogonal angle). These were expected results according to our error analysis discussed in Chapter 6. In this section, we present some specific failure cases.

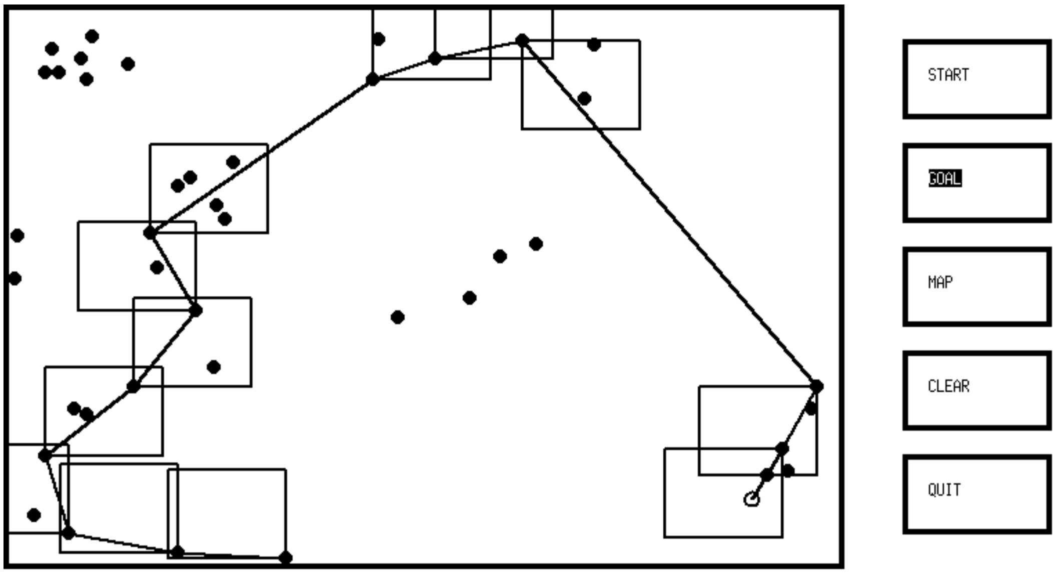


Figure 8.11: A highly error correctable path

Parkway traversal failures

Unreliability due to clutters According to equation 6.2 in Chapter 6, the isolated landmark identification tends to be unreliable in dense areas. In figure 8.13, we see a navigational path generated by the map-maker, with the start position located in the lower right corner and the goal position located in the upper left corner of the environment. The navigator failed during the execution of the 6th movement (corresponding to the second rectangle from the left). In the figure, we see the correct choice of the isolated landmark which is indicated by the a line segment connecting it from the reference point. However, during the actual test run, our navigator selected the object in the view located in the top left corner of the window.

Marginal landmarks In figure 8.14, we see a navigational path generated by the map-maker, with the start position located in the lower right corner and the goal position located in the upper left corner of the environment. The navigator

```
 FROM  TO CMD     DISTANCE   LOGPROB   COST

  90   42 ISOL     78.102    -0.000   78.102
          VERIFY QOC -> 2 (TRIVIAL CASE)

  42   30 ISOL     80.231    -0.000   80.231
          VERIFY QOC -> 2 (TRIVIAL CASE)

  30   27 ISOL     88.865     0.024   88.865
          VERIFY QOC -> 3 (CORRECTABLE)
              If QOC -> 2 (SW, ISOL)

  27   55 ISOL     81.216     0.087   81.216
          VERIFY QOC -> 2 (CORRECTABLE)
              If QOC -> 3 (SW, ISOL)

  55   58 ISOL     68.600     0.024   68.600
          VERIFY QOC -> 3 (CORRECTABLE)
              If QOC -> 2 (SE, ISOL)

  58    3 ISOL     77.104     0.024   77.104
          VERIFY QOC -> 2 (CORRECTABLE)
              If QOC -> 3, (SE, ISOL)

   3   87 TRAJP  226.500     0.350  226.500
          VERIFY QOC -> 1 (CORRECTABLE)
              If QOC -> 2, (NE, ISOL)

  87   79 ISOL     47.434     0.024   47.434
          VERIFY QOC -> 3 (CORRECTABLE)
              If QOC -> 2, (NW, ISOL)

  79   81 ISOL     81.688    -0.000   81.688
          VERIFY QOC -> 2 (TRIVIAL CASE)

  81  100 TRAJP  402.040     0.268  402.040
          VERIFY QOC -> 1 (CORRECTABLE)
              If QOC -> 2, (SW, ISOL)

 100  164 ISOL     51.478     0.127   51.478
          UNVERIFIABLE (Not Correctable)

 164  108 ISOL      0.000     0.024    0.000
          VERIFY QOC -> 1 (CORRECTABLE)
              If QOC -> 2, (NE, ISOL)
Overall distance is 1283.258911
Overall -Log reliability is 0.951362
rum>
```

Figure 8.12: Custom map commands as shown to the user during a map-making session

failed during the execution of the 7th movement (corresponding to the second box from the top). In the figure, we see that the correct choice of the isolated landmark is located very near the boundary of the navigator's view window. In the test run, however, this object did not appear in the view window, and therefore a wrong choice of isolated landmark was selected.

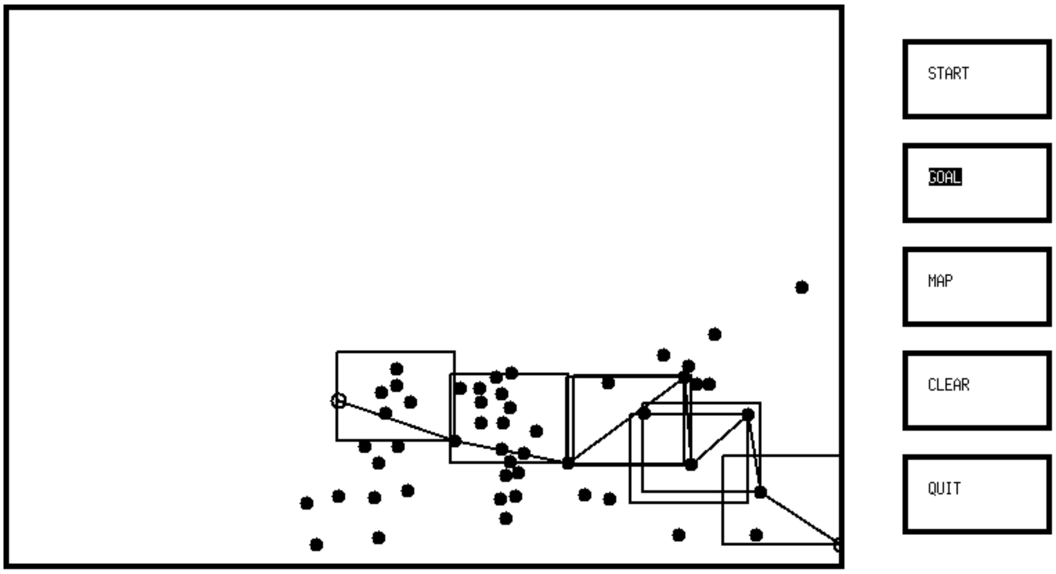


Figure 8.13: A failed path due to passing through a cluttered region

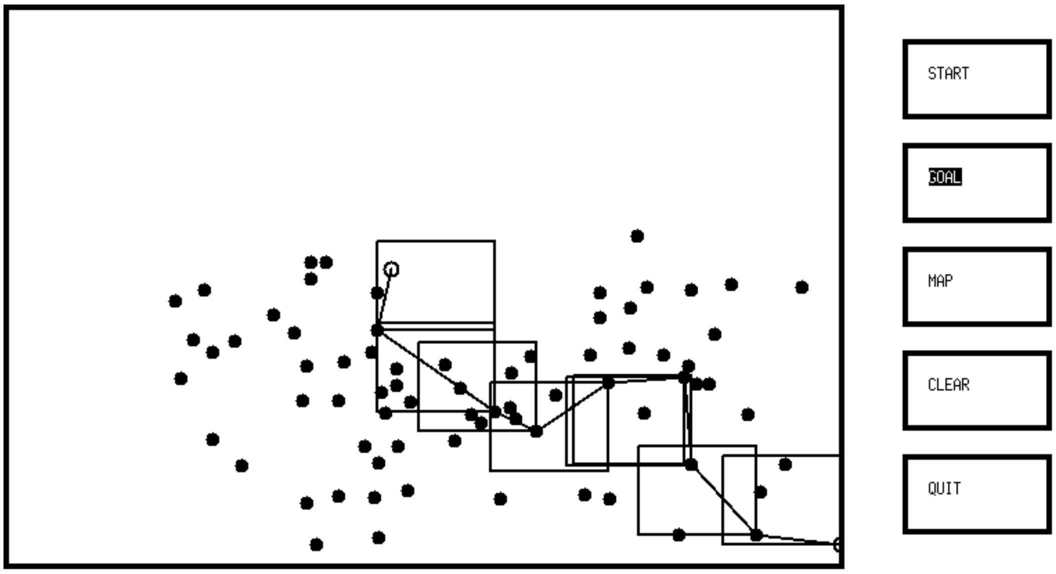


Figure 8.14: A failed path due to a marginally out landmark

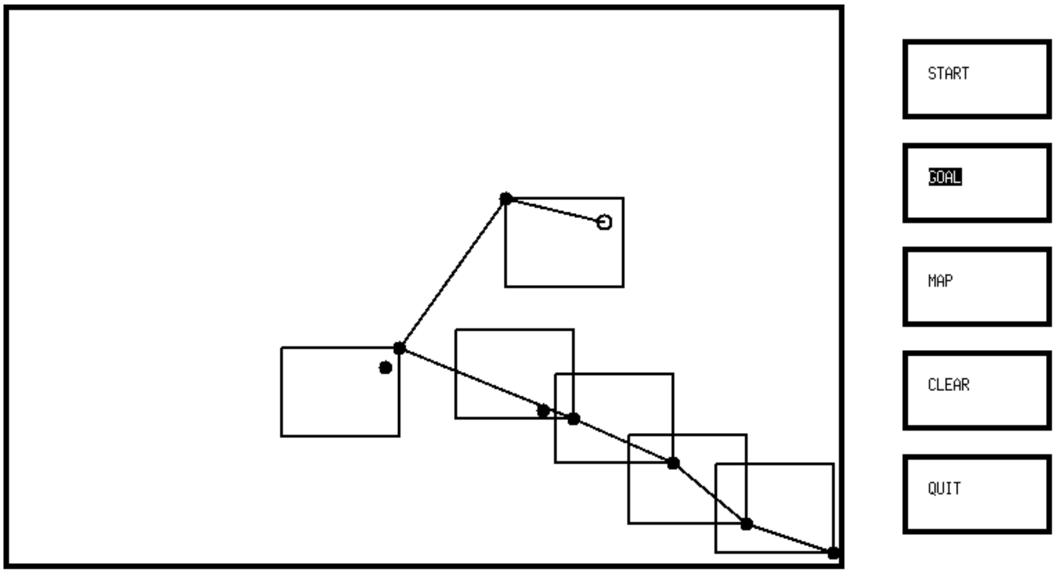


Figure 8.15: A failed trajectory path

Trajectory traversal failures

In figure 8.15, we see a navigational path generated by the map-maker, with the start position located in the lower right corner and the goal position located at the top of the screen. The navigation failed during the 4th movement (correnponding to the 4th line segment counting from the starting landmark) in which the navigator obtained a wrong trajectory target. The reason for this failure is due to the sensory error of the map-maker and the orientational error of the navigator. More specifically, (1) the trajectory angle is at an acute angle and (2) the objects that formed the trajectory vector are very close to each other. These factors contribute to producing small l and large $\cos\theta$ in 6.22, resulting in large E.

8.2.7 Categorization of success and failure cases

The navigation can be either a success, in which case the navigator correctly reaches the destination, or a failure, in which case the navigator ends up in a wrong place. We have categorized these as the following.

Success

1. The navigator follows the correct path and reaches the goal directly. The navigator may or may not be equipped with error handling routines. If the navigator is provided with error handling routines, all verify correctly during the navigation without having the need to execute recovery sequences. The navigator is aware of the success.

2. The navigator follows the correct path and reaches the goal using error recovery routines. The navigator is aware of the error corrections, and of the success of the journey.

3. The navigator follows wrong path, but ends up correctly. In this case, the navigator makes incorrect moves and/or incorrect validations and/or recoveries, but somehow ends up in the right place without knowing its mistakes. The navigator is aware of the success, but is unaware of its mistakes.

Failure

1. The navigator stops due to inability to proceed, case 1:. In this case, the navigator looks for a landmark, but is unable to locate it. For example, the custom map instructs the navigator to identify an obvious point, but there are more than one objects visible (no obvious point). The navigator is totally "lost" and the navigation has failed. The navigator is aware of the failure.

2. The navigator stops due to inability to proceed, case 2: Consider a situation where a landmark is error detectable, but not error recoverable. If correct landmark is verified, the navigator is able to proceed with its confidence reset. But if the error verification fails, the navigator is unable to go on because an error recovery sequence is unavailable. In this case, the navigator

is not completely "lost", but cannot proceed. The navigation is aware of the failure.

3. The navigator follows an incorrect path to the end. In this case, the navigator is unaware of its mistakes (incorrect moves and/or validations), and still thinks it has reached the correct goal. The navigation has failed. The navigator is unaware of the mistakes and is also unaware of the failure.

8.3 Summary

In this chapter, we described the implementation of the map-maker and the navigator. The map-maker is comprised of a global camera, PIPE, a Sun workstation with a X-Windows based user interface. The navigator is implemented using another camera, IBM robot arm, PIPE, and a Sun workstation. Also presented was various experimental results and analysis, including (1) Effects of navigation in sparse versus dense populated environments; (2) Path generation using various cost optimality criteria; (3) Comparison of the shortest landmark distance path and the shortest context-based distance path; (4) Effect of sensor accuracies in path generation; (5) Tests of error detection and recovery algorithms; (6) Examples of failures; and (7) Analysis of success and failure cases in the navigation test runs.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

We have discussed a new method for topological navigation in a large unstructured environment, where metric information is unreliable and expensive [39] [32] [69]. Most of the existing qualitative robot navigators emphasize the importance of landmarks, and their navigations depend heavily on the usage of visual landmarks. However, they also assume that landmarks can be readily identified. The base of our argument is that without a detailed criteria for selecting good landmarks, topological based navigation is not possible.

In this dissertation, we started out by asking several important questions concerning landmarks, such as, (1) what is a good landmark? (2) what features of a landmark are important? (3) what sensors are to be used to recognize a landmark efficiently? (4) how to describe a landmark to the navigator? (5) how to detect errors and (6) how to recover from them. During our efforts to answer these questions, further interesting and important issues were brought up, such as:

- What should be the roles of the direction giver and the direction follower.

- What is the lower bounds of the navigator's intelligence and ability for efficient and accurate navigation?
- What kind of environments are good or bad for navigation? and how to quantify this ?
- Qualitative methods to characterize landmarks so that a navigator with low metric ability can correctly identify them.
- Can we formally define what a landmark is? and assign "goodness" measure to it?
- What can we do to prevent, detect, and recover from errors?

We have addressed these issues carefully in this dissertation.

In chapter 2, we presented the definitions and the assumptions for our topological navigational system. For the purpose of efficiency and of conceptual modularity, we suggested the dichotomy of roles in the navigation system. The first module is the map-maker (direction giver), who has an accurate knowledge of the environment and of the navigator. It abstracts the world into data structures that are mostly topological. The map-maker computes viable path based the configurations of visible landmarks, and generates customized directional instructions for the navigator in a qualitative and efficient way (i.e. verbal), all prior to the actual navigation. The second module is the navigator (direction follower), who is assumed to have very limited sensor range and almost no metric capabilities. Its objective is to get to the destination by executing each of the provided directional instructions in a sequence. The navigator's view of the world is, therefore, an even more abstracted one. We used a very simple navigator model as the direction follower for the sake of: (1) Efficiency during traversal (less cognitive strain, the faster). Most of difficult and time-consuming reasoning is done by the map-maker in the pre-processing stage. During the navigation, because a robot must

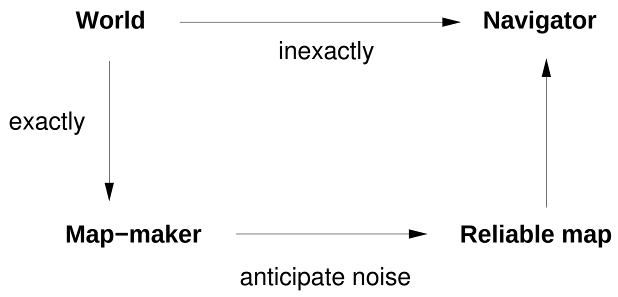


Figure 9.1: The abstraction of the world for the map-maker and the navigator

act in real time relative to a task environment, it has limited time to assimilate its observations to perform appropriate actions. At the same time, its computing capacity and working memory can be shared with other pressing tasks. (2) Simple navigator model because it costs less to construct and operate a simple navigation. (3) Determining the lower bound of the navigator's intelligence and memory without sacrificing the performance. A graphical representation of the relationships between the world, the map-maker, and the navigator is shown in figure 9.1.

In Chapter 4, we introduced abstract data structures called parkways and trajectories which are abstractions of the real world into topologically oriented path representations. Using parkways and trajectories enabled us to define cost measures of the travel route, and to quantitatively evaluate its goodness. Chapter 5 presented a lower level of abstraction of the world, as the navigator experiences it. We suggested the development of description language to identify landmarks qualitatively for robustness and efficiency. We provided a basic description language for our experimental dots world, and gave details in the implementation of the isolated point descriptor. In Chapter 6, we analyzed and modeled various sources of errors, thereby assigning the "goodness" of each landmark based on the computed reliability. We studied error detectability and recoverability in both parkway and trajectory paths and proposed general schemes. As a special case, we analyzed in detail, and implemented error detection and error recovery methods that use of Quadrant Occupancy Count (QOC). In Chapter 7, we discussed path planning

	Dots world	Real world
World abstraction	World → Graph → Custom map	World → Road maps → Directions
Navigational methods	Parkways and trajectories	By landmarks and their topology
Direction giving	Custom maps	Verbal directions
Preferred path	Depends on the accuracy of the navigator and the reliability of the path	Depends on the proficiency of the navigator and the easiness of the route
Error detection	QOC verification	"You'll meet a 4–way intersection"
Error recovery	QOC based recovery	"If you are at a 3–way intersection you've come too far. Take the middle road and it will bring you to the church"

Figure 9.2: Comparison of the map-maker and the navigator to the real world direction giver and the follower

and custom map generation. We observed the relationship between viable paths (parkway and trajectory paths) and the corresponding transition and cost graphs. The transition matrix provided us with a computational definition of landmark (columns mostly filled), and the cost matrix defined the "goodness". We have implemented a demonstration system using a camera mounted robot arm that navigates over a two-dimensional planar world with point-like objects randomly populated on it. The use of point-like objects forced us to define a landmark with respect to its neighboring objects, rather than by its intrisnic qualities. The freedom to place the objects randomly eliminated the usage of external structural cues (e.g. compass, roads) for navigation. The implementation and the experimental results of this demonstrative system was reported in Chapter 8. Figure 9.2 summarizes the correspondence of our navigational system to real world direction giving and following tasks.

As a whole, we feel that we have provided enough evidence for our belief that large scale navigation should be more focused on high-level qualitative reasonings. In our experiments, we showed that high-level topological information of the environment was sufficiently efficient and accurate for navigation in our simplified world. This does not imply that qualitative information is the only useful tool in navigation in general. Whenever metrically accurate information is available, the navigator should use it, but only if the cost can be justified. There is much to be done for this framework to be blended into a real environment. But we claim that the multi-level abstraction of the world offers an effective guidelines for the acquisition and organization of knowledge for more robust and efficient navigation "in the large."

9.2 Future work

9.2.1 Further implementation of current theory

The following is a list of topics that were not fully explored in this dissertation. Nevertheless, we believe that they are important issues that deserve further investigation.

Random walk

We would like to further investigate the "expected" travel time and reachability associated with the random walk. As suggested in Chapter 7, this will involve using a Markov process models.

Confusable points

We have not fully worked out the details in the use of confusable points, as suggested in the basic vocabulary of Chapter 5. Computation of confusable points would require a dynamic programming component for the path generation algorithm. In the simplest case, a confusable path will involve confusability in just one frame of navigator's view. In the other end of the spectrum, confusable paths will have only the start and goal landmarks in common. The rest of the route can consist of different landmarks and path segments.

Custom map grammar

As is the case with any natural or computer programming languages, our custom map language grammar has evolved from a simple and elegant representation to a more complex and not so elegant one. We may need to modify the structure of the grammar for the future enhancement and expansion of the directional commands.

9.2.2 Future research

Related to the current topological navigation system that we implemented, and in order to make it into a more sophisticated and realistic model for the real world navigation, we suggest for future research the following.

Varying z-axis

Allowing a degree of freedom in the z-axis will have many effects on the navigation problem. Changing the height of the navigator is equivalent to changing the size of the navigator's view window. If one goes further up, the view window size of the navigator will get bigger. This implies that the overall number of parkways will grow smaller, but the number of objects in a given view window will grow bigger. At a certain height, the number of parkways will converge to 1. The main tradeoff in varying the window size seems to be between giving the direction versus describing what the navigator sees. On the one extreme, the navigator will be so close to the surface that at most one object will be seen at a given time. In this case, the direction has to be given using a fixed set of directional information (North, East, South, West), and some metric distance measures. On the other extreme is when the navigator is at a height that sees everything. Here, the goal object (landmark) has to be "described" in terms of what it sees. Therefore, if the navigator has the ability to deadreckon accurately, the height of the navigator can be set to *low*, and if the map-maker has a good description language, the height can be set to *high*.

Allowing tilt of the camera angle

The simplest form of our problem is to have our navigator view the world directly from above. This would be a two dimensional navigation problem. However, as previously stated, we would like to have a more realistic model of the navigator, in which it has an oblique view of the world. One big problem that emerges from this is that the objects now can be partly or fully occluded by other objects. Therefore, an efficient and accurate method of dealing with occluded objects must be devised. This is already addressed, in a primitive way, by the notion of "context".

Object classes

The objects that will be used as our landmarks are spheres. Usage of a sphere reduces the burden of the vision system for recognizing objects. The main reason is that a sphere appears as a circle no matter where the camera is located with respect to the object. The centroid of a circle then can be easily calculated. These properties of sphere enabled us to use real images and to treat the objects as point-like. It would be interesting for us to use differently shaped objects. On the other extreme of object shapes is the pole. A pole has an interesting property that when it is viewed from directly above, it appears as a circle. When it is viewed from the horizon, it appears as a rectangle. Anywhere in between it appears as a combination of two elliptical disks and a rectangle.

Guided paths

Currently, path generation is performed entirely by the map-maker. The user is given only the freedom to choose start and goal locations. It would be interesting to allow the user to have some control in the path planning. In terms of the user interface program, the user will be able to suggest a rough estimation of desired route. This can be done by (1) selection of desired intermediate goals, or by (2) use of an input device, such as a mouse, for the rough suggestion of the desired route. The former method would involve in a similar search algorithm, but with multiple start and multiple goal positions. The latter may require a little more sophisticated technique, such as the use of an artificial potential field, to represent a bias.

Bibliography

[1] E. Abbott. *Flatland: A Romance of Many Dimensions.* Dover Publications, 1952.

[2] A. Abella. Extracting geometric shapes from a set of points. In *Proceedings of the Image Understanding Workshop*, 1991.

[3] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[4] Aspex Inc. *Introduction to PIPE Model 1 Systems*, December 1987.

[5] Aspex Inc., New York, NY. *CONSOFT Reference Manual*, 1991.

[6] R. A. Brooks. Solving the find-path problem by good representation of free space. In *Proc. AAAI-82*, pages 381–387, 1982.

[7] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[8] W. G. Chase. Spatial representation of taxi drivers. In D. R. Rogers and J. A. Sloboda, editors, *The acquisition of symbolic skills.* Plenum, 1982.

[9] W. C. Collier. In-vehicle route guidance systems using map matched dead reckoning. In *IEEE Position Location and Navigation Symposium*, 1990.

[10] J. L. Crowley. Navigation for an intelligent mobile robot. *IEEE Journal of Robotics and Automation*, RA-1(1):31–41, 1985.

[11] D. Dai and D. T. Lawton. Range-free qualitative navigation. In *Proceedings of the IEEE Robotics and Automation*, 1993.

[12] R. A. Dork. Satellite navigation systems for land vehicles. In *IEEE Position Location and Navigation Symposium*, 1986.

[13] G. Dudek, P. Freedman, and S. Hadjres. Using uncertain sensing data to create reliable maps: an algorithm for exploring/mapping unknown graph-like world. In *SPIE Cambridge Symposium on Optical and Optoelectronic Engineering, Intelligent Robotics and Computer Vision*, 1992.

[14] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4):pp:551–559, July 1983.

[15] R. J. Elliott and M. E. Lesk. Route finding in street maps by computers and people. In *Proceedings, AAAI-82 Conference*, 1982.

[16] R. L. French. Automobile navigation: where is it going? In *IEEE Position Location and Navigation Symposium*, 1986.

[17] R. L. French. Historical overview of automobile navigation technology. *IEEE Transactions on Vehicular Technology*, 1986.

[18] R. L. French. Automobile navigation in the past, present, and future. In *Proceedings Auto-Carto 8*, pages 542–551, 1987.

[19] R. L. French. Road transport informatics: The next 20 years. Technical Report 881175, Society of Automotive Engineers, 1988.

[20] R. L. French and G. M. Lang. Automatic route control system. *IEEE Transactions on Vehicular Technology*, vt-22(2):36–41, 1973.

[21] K. Fu, R. Gonzalez, and L. C.S.G. *Robotics: Control, Sensing, Vision, and Intelligence*. McGraw Hill, 1987.

[22] G. Giralt, R. Sobek, and R. Chatila. A multi-level planning and navigation system for a mobile robot. In *IJCAI-79*, pages pp:335–337, 1979.

[23] K. C. Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighborhood. *Pattern Recognition*, 10:pp105–112, 1978.

[24] P. Haeussermann. On-board computer system for navigation orientation and route optimization. Technical Report 840485, Society of Automotive Engineers, 1985.

[25] C. B. Harris, L. A. Klesh, E. J. Krakiwsky, K. H. A., and N. S. T. Lee. Digital map dependent functions of automatic vehicle location systems. In *IEEE Position Location and Navigation Symposium*, 1988.

[26] S. K. Honey, M. S. White, and W. B. Zavoli. Extending low cost land navigation into systems information distribution and control. In *IEEE Position Location and Navigation Symposium*, 1988.

[27] IBM. *IBM 7575 and 7576 Manufacturing Systems Hardware Library Site Preparation, Installation, and Specifications, Part Number 70X8867*, 1st edition, August 1986.

[28] IBM. *IBM 7575 and 7576 Manufacturing Systems Software Library AML/2 Manufacturing Control System User's Guide, Part Number 67X1370*, 1st edition, August 1986.

[29] H. A. Karimi and E. J. Krakiwsky. Design concepts for knowledge-based route guidance system. In *IEEE Position Location and Navigation Symposium*, 1988.

[30] H. A. Karimi, E. J. Krakiwsky, C. Harris, G. Craig, and R. Goss. A relational database model for an avl system and an expert system for optimal route selection. In *Proceedings Auto-Carto 8*, 1987.

[31] J. R. Kender and A. Leff. Why direction-giving is hard: The complexity of linear navigation by landmarks. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1656–1658, 1989.

[32] J. R. Kender, I.-P. Park, and D. Yang. A formalization and implementation of topological visual navigation in two dimensions. In *SPIE International Symposia*, 1990.

[33] E. J. Krakiwsky, H. A. Karimi, C. Harris, and J. George. Research into electronic maps and automatic vehicle location. In *Proceedings Auto-Carto 8*, 1987.

[34] D. J. Kriegman, T. O. Binford, and T. Sumanaweera. Generic models for robot navigation. In *Proceedings of the Image Understanding Workshop*, 1988.

[35] B. Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.

[36] B. Kuipers. On representing commonsense knowledge. In *Associative Networks*, pages 393–408. Academic Press, 1979.

[37] B. Kuipers. The cognitive map: could it have been any other way? In H. J. Pick and L. Acredolo, editors, *Spatial Orientation: Theory, Research, and Application*, pages 345–359. Plenum Press, 1983.

[38] B. Kuipers. Modeling human knowledge of routes: partial knowledge and individual variation. In *Proceedings, AAAI-83 Conference*, 1983.

[39] B. J. Kuipers and Y.-T. Byun. A robust, qualitative approach to a spatial learning mobile robot. In *SPIE Cambridge Symposium on Optical and Optoelectronic Engineering, Advances in Intelligent Robotics Systems*, 1988.

[40] T. Levitt and L. D. T. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44(3):305–360, 1990.

[41] T. W. Lezniak, R. W. Lewis, and M. R. A. A dead reckoning/map correlation system for automatic vehicle tracking. *IEEE Transactions on Vehicular Technology*, vt-26(1):47–60, 1977.

[42] T. Lozano-Perez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems Mans and Cybernetics*, SMC-11(10):681–698, October 1981.

[43] D. Mark and M. McGranaghan. Effective provision of navigation assistance to drivers: a cognitive science approach. In *Proceedings Auto-Carto*, 1986.

[44] D. M. Mark. Finding simple routes: 'ease of description' as an objective function in automated route selection. In *Proceedings, Second conference on Artificial Intelligence Applications (IEEE)*, pages 577–581, 1985.

[45] D. M. Mark. On giving and receiving directions: cartographic and cognitive issues. In *Proceedings Auto-Carto 8*, 1987.

[46] W. Mendenhall, L. R. Scheaffer, and D. D. Wackerly. *Mathematical Statistics with Applications*. Duxbury Press, 3rd edition, 1990.

[47] H.-G. Metzler. Computer vision applied to vehicle operation. Technical Report 881167, Society of Automotive Engineers, 1988.

[48] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation*, pages 116–121, 1985.

[49] K. Onoguchi, M. Watannabe, and Y. Okamoto. A visual navigation system using a multi-information local map. In *IEEE International Conference on Robotics and Automation*, pages 767–774, 1990.

[50] N. E. P. and Zechnall. Digital map data bases for autonomous vehicle navigation systems. In *IEEE Position Location and Navigation Symposium*, 1986.

[51] I.-P. Park. Towards automatic vehicle navigation. *Columbia University, Department of Computer Science, Technical Report*, 1991.

[52] I.-P. Park and J. R. Kender. Navigation using isolated landmarks. In *SPIE Cambridge Symposium on Optical and Optoelectronic Engineering, Intelligent Robotics and Computer Vision*, 1992.

[53] I.-P. Park and J. R. Kender. Qualitative environmental navigation. In *Proceedings of the Image Understanding Workshop*, 1993.

[54] I.-P. Park and J. R. Kender. Using isolated landmarks and trajectories in robot navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society Press, 1993.

[55] H. W. Perry. Some remarkable mechanical road guides. *Scientific American*, 104(2), January 1911.

[56] T. K. Peuker. Cartographic data structures. *The American Cartographer*, 2:55–69, 1975.

[57] O. Pilsak. Eva - an electronic traffic pilot for motorists. Technical report, Society of Automotive Engineers, 1986.

[58] B. Pinette. Qualitative homing: Ii. handling incorrectly-identified landmarks. In *SPIE Cambridge Symposium on Optical and Optoelectronic Engineering, Intelligent Robotics and Computer Vision*, 1992.

[59] F. P. Preparata and M. I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1985.

[60] J. D. Radke. On the shape of a set of points. In G. T. Toussaint, editor, *Computational Morphology*, pages 105–136. Elsevier Science Publishers B.V., 1988.

[61] E. Rich. *Artificial Intelligence.* McGraw-Hill, 1983.

[62] C. K. Riesbeck. You can't miss it: Judging the clarity of directions. *Cognitive Science*, 4:285–303, 1980.

[63] B. A. Roberts and L. Vallot. Vision aided inertial navigation. In *IEEE Position Location and Navigation Symposium*, 1990.

[64] A. Silverman. Concepts and technology for navigation equipment. In *IEEE Position Location and Navigation Symposium*, 1988.

[65] M. H. Soldo. Reactive and preplanned control in a mobile robot. In *Proceedings: Image Understanding Workshop*, 1990.

[66] L. A. Streeter and D. Vitello. A profile of drivers' map-reading abilities. *Human Factors*, 28(2):223–239, 1986.

[67] L. A. Streeter, D. Vitello, and S. A. Wonsiewica. How to tell people wherer to go: Comparing navigational aids. *International Journal of Man-Machine Studies*, 22:549–562, 1985.

[68] E. Stuck. *Detecting and Diagnosing Mistakes in Inexact Vision-based Navigation*. PhD thesis, University of Minnesota, 1992.

[69] K. T. Sutherland and W. B. Thompson. Inexact navigation. In *IEEE International Conference on Robotics and Automation*, 1993.

[70] P. Thorndyke and B. Hayes-Roth. Differences in spatial knowledge acquired from maps and navigation. *Cognitive Psychology*, 14:560–589, 1982.

[71] C. Thorpe and J. Gowdy. Annotated maps for autonomous land vehicles. In *Proceedings: Image Understanding Workshop*, 1990.

[72] G. T. Toussaint. A graph-theoretical primal sketch. In G. T. Toussaint, editor, *Computational Morphology*, pages 229–260. Elsevier Science Publishers B.V., 1988.

[73] B. Ward and G. McCalla. Error detection and recovery in a dynamic planning environment. In *Proceedings, AAAI-82 Conference*, pages 172–175, 1982.

[74] M. White. Digital map requirements of vehicle navigation. In *Proceedings Auto-Carto 8*, pages pp.552–561, 1987.

[75] S. Wolfram. *Mathematica*. Addison-Wesley, 1988.

[76] C. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(1):68–86, January 1971.

[77] W. B. Zavoli and S. K. Honey. Map matching augmented dead reckoning. In *IEEE 1986 Vehicular Technology Conference*, 1986.

[78] J. Y. Zheng and S. Tsuji. Panoramic representation for route recognition by a mobile robot. *International Journal of Computer Vision*, 9(1):55–76, October 1992.