Learning and Vision Algorithms for Robot Navigation

by

Margrit Betke

S.M., Massachusetts Institute of Technology (1992)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

Author . . . .
Department of Electrical Engineering and Computer Science
May 18, 1995

Certified by . . . .
Ronald L. Rivest
Professor
Thesis Supervisor

Accepted by . . . .
F. R. Morgenthaler
Chairman, Department Committee on Graduate Students

Learning and Vision Algorithms for Robot Navigation

by

Margrit Betke

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 1995, in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

This thesis studies problems that a mobile robot encounters while it is navigating through its environment. The robot either explores an unknown environment or navigates through a somewhat familiar environment. The thesis addresses the design of algorithms for

1. environment learning,
2. position estimation using landmarks,
3. visual landmark recognition.

In the area of mobile robot environment learning, we introduce the problem of piecemeal learning of an unknown environment: the robot must return to its starting point after each piece of exploration. We give linear time algorithms for exploring environments modeled as grid-graphs with rectangular obstacles. Our best algorithm for piecemeal learning of arbitrary undirected graphs runs in almost linear time.

It is crucial for a mobile robot to be able to localize itself in its environment. We describe a linear time algorithm for localizing the mobile robot in an environment with landmarks. The robot can identify these landmarks and measure their bearings. Given such noisy measurements, the algorithm estimates the robot's position and orientation with respect to the map of the environment. The algorithm makes efficient use of our representation of the landmarks by complex numbers.

The thesis also addresses the problem of how landmarks in the robot's surroundings can be recognized visually. We introduce an efficient, model-based recognition algorithm that exploits a fast version of simulated annealing. To avoid false recognition, we propose a method to select model images by measuring the information content of the images. The performance of the algorithm is demonstrated with real-world images of traffic signs.

Thesis Supervisor: Ronald L. Rivest
Title: Professor

Acknowledgments

Thanks go first to Ron Rivest. I am very grateful to Ron for having been an outstanding advisor. It is fun to work with Ron. He has always treated me as a colleague and friend. Ron always had plenty of time and helpful advice for me. Thank you Ron, Mona Singh, Nick Makris, Baruch Awerbuch, and Leonid Gurvits for collaborating with me on work that turned into this thesis.

I am grateful to Rodney Brooks and Tomaso Poggio for their feedback as members of my thesis committee. Thanks also go to Berthold Horn, Charles Leiserson, and Nancy Lynch for giving me a chance to work with them and learn from them.

I am very grateful to Ron Rivest, Tomaso Poggio, Federico Girosi, and Bob Berwick for arranging my financial support from the NSF (grant ASC-9217041) and the Siemens Corporation.

Thanks to Stephen Judd, Tom Hancock, Steve Hanson, and Russ Greiner of Siemens Corporate Research for the great time I had working with them.

I would like to thank Wilfried Betke for taking some of the pictures that I used in Chapter 4. Many thanks to Michelle Hsu, Thien Lee, Ou-Dan Peng, Acee Agoyo, and Marney Smyth for their help.

Thanks to many friends in the theory group and the AI Lab at MIT. I don't know how I would have survived graduate school without Be Hubbard. "Thank you very much, Be!" Special thanks to Mona Singh for being a wonderful friend. Many thanks to Javed Aslam for all his help. Thanks to Abeer Alwan, Mihir Bellare, Ruth Bergman, Scott Blomquist, Avrim Blum, Bobby Blumofe, Andrew Chou, Tom Cormen, Scott Decatur, Bronwyn Eisenberg, Mike Ernst, Igal Galperin, Rainer Gawlick, Rosario Gennaro, Federico Girosi, Sally Goldman, Lalita Jategaonkar, Tina Kapur, Joe Kilian, Shlomo Kipnis, Phil Klein, Mike Klugerman, Dina Kravets, Aparna Lakshmiratan, John Leo, Pam Lipson, Marina Meila, Partha Niyogi, James Park, Anna Pogosyants, Stephen Ponzio, Phil Rogaway, Alex Russell, Rob Schapire, Eric Schwabe, Roberto Segala, Denise Sergent-Leventhal, Ray Sidney, Pawan Sinha, Donna Slonim, Mark Smith, Cliff Stein, David Williamson, Ethan Wolf, Yiqun Yin, and David Zuckerman.

Many thanks to Wendy Stone for having been a source of strength and support. Special thanks to Trevor Jim, Jane Ko, and Mojdeh Mohtashemi for having been great friends and roommates. Thanks to Nick for helping make these years such interesting ones.

I owe the largest part of my gratitude to my family for their love and support. Thanks to my parents Wilfried and Ludmilla Betke, my sister Elisabeth, and my brother Ulrich. "Herzlichen Dank!"

Publication Notes

Most of the research presented in this thesis has been published elsewhere.

Chapter 2 describes my joint work with Ronald L. Rivest and Mona Singh. It is a revised version of a journal paper that appeared in *Machine Learning* in March 1995 [20]. An extended abstract of this paper is published in the Proceedings of the 1993 Conference on Computational Learning Theory and as an MIT AI-Memo 1474, CBCL-Memo 93 [19].

Chapter 2 also describes my joint work with Ronald L. Rivest, Mona Singh, and Baruch Awerbuch. An extended abstract will be published in the Proceedings of the 1995 Conference on Computational Learning Theory [3]. It is also published as an MIT Laboratory for Computer Science technical memo MIT/LCS/TM–516 in January 1995 [4].

Chapter 3 presents my joint work with Leonid Gurvits [16]. The work was accepted as a regular journal paper at the *IEEE Transactions on Robotics and Automation* in February 1995. An extended abstract of this paper is published in the Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, September 1994, and also as a Siemens Corporate Research Technical Report SCR-94-TR-474, December 1993.

The research in Chapter 4 is joint work with Nicholas C. Makris. An extended abstract of this work will be published in the Proceedings of the Fifth International Conference on Computer Vision in June 1995 [18]. It is also published as an MIT AI-Memo 1510 and CBCL-Memo 109 in December 1994 [17].

Contents

Chapter 1

Introduction

This thesis studies problems that a mobile robot encounters while it is navigating through its environment. Two scenarios are considered: the robot either does not know its environment a priori and needs to explore it, or it is familiar with its environment and needs to navigate through it. For these scenarios the thesis addresses the design of algorithms for

- **environment learning,**
- **position estimation using landmarks,**
- **visual landmark recognition.**

The algorithms describe how a robot can learn about and navigate in its environment while consuming its limited resources as sparingly as possible. The resources may be time, energy, or memory space. Another objective of the algorithms is that the robot must solve its tasks reliably. The performance of the algorithms, i.e., the efficiency and reliability of the algorithms, is analyzed theoretically and/or empirically.

Environment learning and landmark navigation of mobile robots have been very active research fields in recent years. Robotics has evolved to a point where mobile robots are commercially used, for example, as meal delivery robots in hospitals [45]. There are other working prototypes, such as vehicles that navigate autonomously on highways [37, 13, 91] and mobile robots that navigate in office buildings [57, 25, 33, 96, 82, 54, 55]. These successes have motivated and excellerated research in various areas. There is a wide range of work: at one end of the spectrum are systems implemented on physical robots; at the other end are algorithms solving fundamental mathematical problems. This thesis presents algorithms analyzed within a sound theoretical framework as well as results of implementations using real-world data. How do these contributions which span such a wide range of approaches relate to another and to previous work? In the following, we define a classification of navigating robots that categorizes our results and links them with the work of others.

We classify the kinds of problems a navigating robot needs to solve according to the uncertainty involved in the tasks. We distinguish navigating robots by the amount of information they have about their environment. The more a priori knowledge they have, the better they can plan their paths in advance.

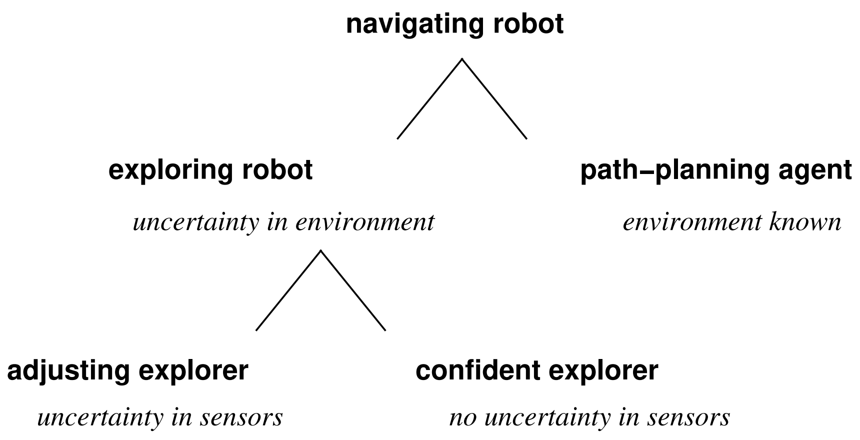

Figure 1-1: Classification of navigating robots.

In the extreme case, the robot's a priori knowledge is assumed to describe and model the environment fully. We call such robots "path-planning agents," because they can plan their navigation routes off-line, prior to any physical movement. The assumption that a robot actually has complete knowledge of its environment can only be justified for certain applications [30, 72, 42].

The thesis addresses more predominant scenarios in which the navigating robot has to cope with uncertainty in its environment. Because of this uncertainty, such robots have a limited ability to plan their navigation route before they start moving. Therefore, we call these robots "exploring robots." Their input is given in an "on-line" fashion: they receive information about the environment piecewise. After each piece of information the robot can make a decision based on the knowledge accumulated so far.

We distinguish between "confident" and "adjusting explorers." The confident explorer is assumed to be able to rely on its sensors undoubtedly. It is confident that the information that it gathers while moving around can be fully trusted. The only uncertainty the confident explorer has to cope with is the uncertainty about the outcome of its actions. For example, it may not know where a path leads to that it is following. A confident explorer may be classified by the type of sensors it is assumed to have. These may be short- or long-range sensors, for example, tactile, sonar, or vision sensors. The thesis describes algorithms for a confident explorer with a short-range sensor such as a tactile sensor.

An adjusting explorer does not trust its sensor inputs completely. It is aware that the information about the environment that it gathers does not describe the real world. Instead, it knows that it builds a map of a self-centered, imaginary world. Sarachik calls this world "solipsistic" [96]. The model of the environment may diverge from the real world as time passes. A confident explorer is assumed not to need a "reality check." In contrast, an adjusting robot must perform this reality check. It can adjust its world model accordingly. The thesis describes an adjusting explorer that uses landmarks to relate its self-centered model and the real world to each other.

The models of the confident and the adjusting explorer have both advantages and disadvantages. Obviously, a real robot is never confident, but always encounters sensor errors. Critics of the confident explorer model argue that the model is simplistic and unrealistic. However, the confident explorer model can be useful for abstracting from low-level engineering details and concentrating on high-level navigation tasks. Therefore, confident explorers often represent their knowledge about their environment using a qualitative, topological map. In contrast, adjusting explorers often represent their knowledge about their environment in quantitative, geometric maps. The thesis follows this way of representing a robot's environment. (An example for an adjusting robot using a topological map is Mataric's system [82].) There is a trade-off: geometric maps are more detailed than topological, but they are more difficult to obtain and use.

1.1 Contributions of the thesis

The thesis combines three independent results in the fields of machine learning, robotics, and computer vision. Each result describes a different approach to address and solve problems that a navigating robot encounters. The level of abstraction decreases with each part of the thesis, i.e.,the first part deals with a theoretical learning problem for robots of the future and the final part deals with the problem of how to utilize sensory data, i.e., real-world images of landmarks, to support robot navigation.

1.1.1 Contributions to machine learning

The research on environment learning presented in this thesis belongs to the area of machine learning. It is about the design and analysis of efficient learning algorithms for confident explorers. Similar formal approaches to environment learning have been studied previously in the context of graph theory, computational geometry, on-line algorithms, and automata theory (e.g., [89, 21, 93]). In this related work, formal models are defined that try to capture the difficulties that lie in the confident explorer's limited knowledge of its environment. In our work, we assume that the unknown environment can be described by a simple discrete model, an undirected graph $G = (V, E)$. We study both arbitrary undirected graphs and the special case of grid graphs with rectangular obstacles. We introduce the problem of *piecemeal learning of an unknown environment:* The robot's goal is to learn a complete map of its environment while satisfying the constraint that exploration must be done a "piece at a time," with a return to the starting point after each piece of exploration.

The efficiency with which the robot learns the map of the graph is measured in terms of the number of edges traversed. The main difficulty in our work lies in designing efficient, but analyzable, robot exploration algorithms.

We show that if the robot is given a bound on the number of edges it is allowed to traverse in each exploration phase, it can explore grid-graphs with rectangular obstacles in a piecemeal manner in linear time. We extend these results and show that the robot can explore any kind of undirected graph piecemeal in almost linear time. Our best algorithm for arbitrary undirected graphs achieves $O(E + V^{1+o(1)})$ running time. We also give a much simpler and elegant but less efficient algorithm for piecemeal learning which runs in $O(E + V^{1.5})$ time. It is an open question whether general graphs can be explored piecemeal in linear time.

1.1.2 Contributions to robotics

Our work on piecemeal learning of unknown environments assumes that the navigating robot is a confident explorer that can rely on its sensors. This avoids a crucial problem addressed extensively in robotics research: a mobile robot often does not know where it is in its environment. Therefore, various strategies have been developed that keep track of the robot's position in its environment (e.g. [22, 41, 38, 2]). For the case of a confident explorer, this is easily solved using dead-reckoning. Since the sensors are not assumed to involve any measurement errors, the trajectory and location of the robot can be computed exactly.

This is not possible for adjusting explorers. Because of inevitable measurement inaccuracies errors in position estimation accumulate over time. Dead-reckoning is not sufficient to localize the robot. Another popular method for localization is the global positioning system (GPS). Unfortunately, it does not work reliably in certain environments.

These problems motivate us to investigate how an adjusting explorer can use landmarks in its surroundings to support position estimation. In this thesis, we present a method to estimate the robot's position reliably and efficiently in the presence of measurement errors. The environment of the robot is modeled as a continuous two-dimensional plane with landmarks at certain points in the plane. Our localization system assumes that the robot is somewhat familliar with its environment, i.e., knows the position of the landmarks on a given map of the environment. It is also assumed that the robot has a camera or other sensors that enable it to both identify landmarks and measure the bearings of one landmarks relative to another landmark. This research was originally motivated by working with "Ratbot," a mobile robot built at Siemens Corporate Research [54].

In this thesis, we show how to estimate the robot's position using a new technique that is based on a complex number representation of the landmarks. The algorithm runs in time linear in the number of landmarks. The performance of the algorithm is illustrated with simulations.

1.1.3 Contributions to vision research

Identifying landmarks in the robot's environment is a difficult problem. A reliable and efficient solution to this problem is essential for the success of the localization method presented in this thesis. Therefore, the thesis also addresses the problem of how landmarks in the robot's surroundings can be recognized visually.

The robot takes images of its surroundings while it is moving through its environment. We address the problem of recognizing a landmark in such an image as the problem of best describing a match between this image and an image of a hypothesized landmark.

In the field of visual object recognition, the idea of matching a model of an object with an image by exhaustive search is standard (e.g. [94]). However, a large number of parameters is necessary to mold the model image for a parameter match. This makes the search space extremely large. We introduce a recognition algorithm that exploits a fast version of simulated annealing to reduce the time needed for recognition by orders of magnitude with respect to an exhaustive search. The normalized correlation coefficient is used as a measure of the match. The robot tries to match the pictures of its surroundings with template images that are automatically created from model images of landmarks. These model images are stored in the robot's database in advance. A fast transformation of the model images into template images is done on-line during the search.

The performance of the algorithm is demonstrated with real-world images of complicated scenes. As an example of landmarks we choose traffic signs. False recognition occurs only for template images with very small information content. To avoid false positive matches, we propose a method to select model images for robust object recognition by measuring the information content of the model images. The algorithm works well in noisy images for model images with high resolution.

1.2 Thesis overview

The thesis is organized into three fully self-contained chapters. Each chapter addresses a different aspect of mobil robot navigation. Chapter 2 considers piecemeal learning of unknown environments. We first motivate the problem of piecemeal exploration. Then we describe related work in machine learning. We present the algorithms for the special case of learning grid graphs with rectangular obstacles and finally the algorithms for arbitrary undirected graphs.

Chapter 3 provides a method for mobile robot localization. First we present related work on landmark navigation and position estimation for mobile robots. Then we give our algorithm and its analysis. Finally, the reliablity of the algorithm is shown with simulations.

Chapter 4 describes how landmarks, for example, traffic signs can be recognized in noisy images. We first describe our method to transform model images into template images. Then we give the algorithm and show its performance on a set of real-world images. Finally, we address problem of false recognition and propose a method to select model images by measuring the information content of the images.

Chapter 5 presents our conclusions and ideas for future research.

Chapter 2

Piecemeal Learning of Unknown Environments

2.1 Introduction

In the area of mobile robot environment learning, we introduce a new problem: an unknown environment that must be learned "piecemeal" – a piece at a time. The robot's goal is to learn a complete map of its environment while satisfying the "piecemeal constraint" that exploration must be done a little at a time, with a return to the starting point after each piece of exploration.

Why might mobile robot exploration be done piecemeal? Robots are useful for exploring environments that are too risky or costly for humans to explore, like the inside of a volcano (as was attempted by CMU's Dante II robot) or the surface of Mars. The robot's hardware may be too expensive or fragile to stay for long in dangerous conditions, such as in the interior of a volcano. Thus, instead of one long exploration phase, it may be best to organize the explorations into different phases, allowing the robot to return to its start position before its hardware breaks down or its batteries run out. At the start position, the robot can cool off, recharge its batteries, or drop off samples collected.

There are situations where humans show piecemeal learning behaviors. For example, a tourist visiting Rome can explore a new section of the city each day before returning to her hotel in the evening. A rookie taxi driver might learn a city bit by bit while returning to base between trips.

The piecemeal constraint means that each of the robot's exploration phases must be of limited duration. We assume that each exploration phase starts and ends at a fixed start position s. This special location might be a refueling station or a base camp. Between exploration phases the robot might perform other unspecified tasks. We are deliberately vague about these tasks and the ultimate purpose of the learned map. We focus exclusively on the problem of building up a model of the environment under a piecemeal constraint, with the expectation that such a model might be used in a variety of applications (such as finding a shortest path to some goal location).

Piecemeal exploration enables "learning on the job", since the phases of piecemeal exploration can help the robot or "learner" improve its performance on the other tasks it fulfills. This is the "exploration/exploitation tradeoff": spending some time exploring and some time exploiting what has been learned. In our work, the only task that the robot must perform in between exploration phases is the task of *efficiently* interrupting and resuming exploration. More precisely, the robot must get back from the point where it interrupted the exploration to the start position s as quickly as possible. It also must find a fast way to move from s to a location where it can resume exploration. The robot exploits what it has learned – the (incomplete) map of the environment – to perform this task.

The efficiency with which the learner explores the environment is measured in terms of the number of edges traversed. The piecemeal constraint can make efficient exploration surprisingly difficult. The main challenge in designing piecemeal learning algorithms lies in finding efficient and analyzable strategies to organize the phases of exploration.

This chapter presents our results on piecemeal learning of arbitrary undirected graphs and of grid graphs with rectangular obstacles. Our best algorithm for piecemeal learning arbitrary undirected graphs takes $O(E + V^{1+o(1)})$ time – this is almost linear. We also describe a simpler but less efficient algorithm which runs in $O(E+V^{1.5})$ time. We give two linear-time algorithms for the piecemeal search of grid graphs with rectangular obstacles. The first algorithm, the "wavefront" algorithm, can be viewed as an optimization of breadth-first search for our problem. The second algorithm, the "ray" algorithm, can be viewed as a variation on depth-first search.

We now give a brief summary of the chapter. Section 2.2 describes related research on environment learning and shortest path exploration. Section 2.3 gives the formal model of the piecemeal learning problem. Section 2.4 describes initial approaches to piecemeal exploration using DFS and BFS. Section 2.5 compares the on- and off-line piecemeal exploration problems and gives an approximate solution to the off-line version of this problem. Section 2.6 proves a lower bound to the general piecemeal learning problem. Section 2.7 introduces the notion of "city-block graphs" and discusses shortest paths in such graphs. Section 2.8 introduces the notion of a wavefront, gives the wavefront algorithm for piecemeal search of city-block graphs, proves it correct, and derives its running time. Section 2.9 introduces the ray algorithm as another way to do piecemeal search of city-block graphs. Section 2.10 describes three algorithms for piecemeal exploration for arbitrary graphs. The final algorithm runs in $O(E + V^{1+o(1)})$ time.

2.2 Previous work

This section describes previous work that is related to the piecemeal learning problem. Our problem formulation bears some similarity to a variety of search problems. But our goal is somewhat different: instead of searching for a particular state or location in a given environment, the robot has the goal of learning a model of the entire environment. However, searching for and finding shortest or almost shortest paths in an unknown environment is a strategy that is crucial for the efficiency of our algorithms. Therefore, we summarize related work on both environment learning and on finding shortest paths in unknown environments. Figure 2-1 shows how the different tasks of a navigating robot that we consider in this section fit into the taxonomy defined in Chapter 1. We also refer to Rao, Kareti, Shi, and Iyengar's [90] well-illustrated survey on mostly theoretical work on "robot navigation in unknown terrains."

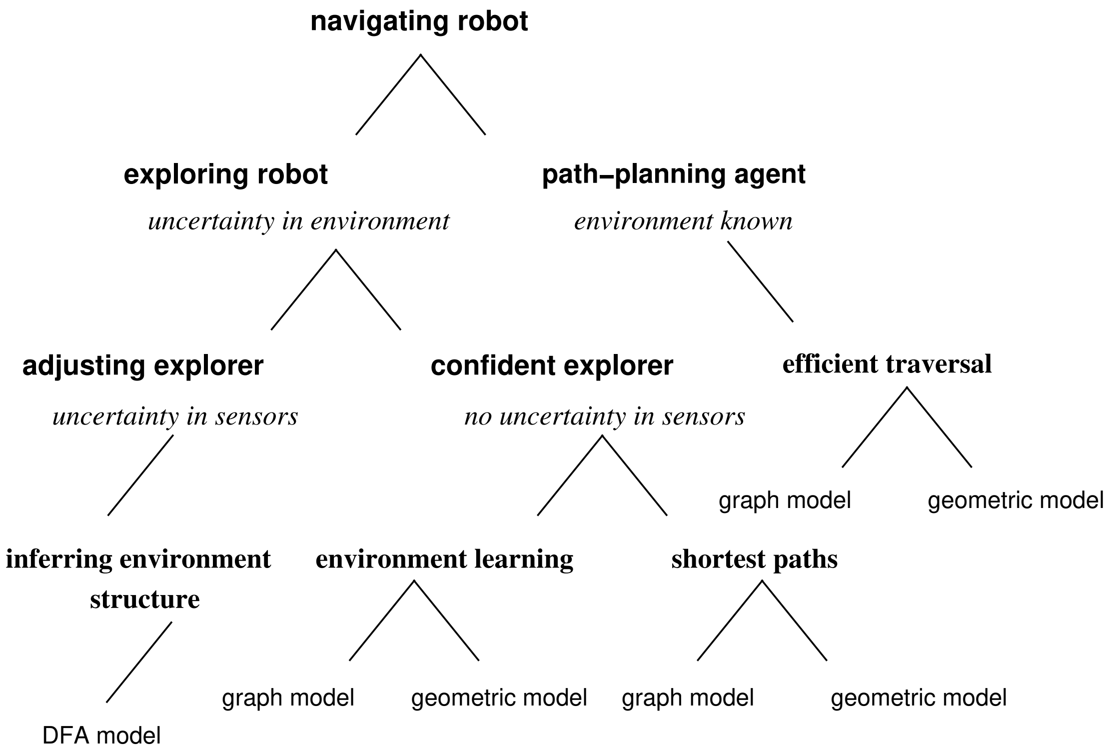

Figure 2-1: Classification of navigating robots, their tasks, and the models of the environments.

2.2.1 Environment learning

One of the first theoretical models for exploring unknown environments was developed by Rivest and Schapire [92, 93]. The work belongs to an area of machine learning called computational learning theory. The model of the environment is a deterministic finite automaton (DFA). Rivest and Schapire's algorithms infer the structure of DFAs correctly with high probability. Related work on inferring the structure of DFAs is done by Basye, Dean, and Kaelbling [12].

Deng and Papadimitriou [40] and Betke [15] address the problem of exploring unknown directed graphs that are characterized by their deficiency, i.e., the "distance" from being Eulerian graphs. Directed graphs may model cities with one-way streets.

The difficulty in learning directed graphs lies in the fact that the robot's moves are irreversible. Bender and Slonim [14] show that two robots can cooperate to learn a directed graph efficiently. They consider a graph model that has indistinguishable vertices. They give an algorithm that runs in expected time polynomial in the number of vertices of the graph.

Deng, Kameda, and Papdimitriou [39] consider the exploration of an environment that can be modeled as the interior or exterior of a rectilinear polygon. They give a competitive algorithm with competitive ratio of 2 (in the L_1-norm). Kleinberg improves this result by giving an algorithm with competitive ratio of 5/4 [66].

Deng, Kameda, and Papdimitriou also show that the interior of a rectilinear polygon with at most k obstacles can be explored using an $O(k)$ competitive algorithm. For polygons of arbitrary shape and with a bounded number of obstacles they show that there exists a competitive algorithm. However, the only ratio that they can prove is huge.

Model	Properties	Results	Authors
directed graphs	deficiency measures distance from being Eulerian graphs, distinguishable nodes	competitive algorithm of ratio 4	[40] [15]
	2 cooperating learners, indistinguishable nodes	algorithm with expected time polynomial in number of nodes	[14]
undirected graphs	piecemeal learning grid graphs with rectangular obstacles	linear algorithms	[19]
	arbitrary graphs	almost linear algorithms	[4]
geometric model	interior & exterior of rectilinear polygons	competitive algorithm	[39]
		of ratio 2	[39]
		of ratio 5/4	[66]
	interior of rectilinear polygon with $\leq k$ obstacles	$O(k)$ competitive algorithm	[39]
	polygons of arbitrary shape, bounded number of obstacles	competitive algorithm exists (huge ratio)	[39]
DFA	learing by experimentation	high probability learning algorithms	[92] [93] [12]

2.2.2 Shortest paths in unknown environments

In the research summarized in this section, it is assumed that the robot is at a start position s and searches the unknown environment for a shortest path to reach the goal t. The environment is either modeled by a graph or by a two-dimensional scene.

The robot is a confident explorer, it does not know the environment in advance, but can trust its sensors.

The "cow-path problem" is defined for a graph-theoretic model. There are w paths leading off start position s and the goal (a pasture) is somewhere on one path. The explorer (a cow) does not known on which path the goal is and tries to find it as quickly as possible. Baeza-Yates, Culberson and Rawlings [8] give an optimal deterministic algorithm which was improved by Kao, Reif, and Tate [62] using randomization. In the case that there are two paths ($w = 2$) the graph is a chain-graph. For this graph, Baeza-Yates, Culberson, and Rawlings use an optimal "linear spiral search" and obtain an algorithm with competitive ratio of 9. Kao, Reif, and Tate give an optimal randomized algorithm with competitive ratio of ≈ 4.5911.

The cow-path problem for the case of two paths is similar to the problem of exploring *layered graphs* of width 2. Layered graphs of width 2, however, allow "shortcuts". For searching such a graph, Papadimitriou and Yannakakis [89] give an optimal algorithm with competitive ratio of 9.

Baeza-Yates, Culberson, and Rawlings [8] describe an algorithm for searching a goal t at distance n from s in a grid graph. If the distance n is known in advance, the algorithm takes $9n-2$ steps. Otherwise it takes $2n^2+5n+2$ steps. The authors also give a lower bound of $2n^2+4n-1$ for the case that the distance n is unknown.

Model	Properties	Results	Authors
graph	*cow-path problem:* w paths from origin s, goal t is on one path	optimal deterministic algorithm	[8]
		randomized algorithm	[62]
	if $w = 2$ (chain graph)	optimal *spiral search* competitive ratio: 9	[8]
		optimal randomized algorithm competitive ratio: ≈ 4.6	[62]
	layered graph, width 2 (same as cow-path but allowing short cuts)	optimal algorithm with competitive ratio: 9	[89]
	grid-graph, distance $d(s,t) = n$ unknown distance n	$9n-2$ steps $2n^2+5n+2$ steps lower bound $2n^2+4n-1$	[8]

For searching an environment that consists of unit-square obstacles in a two-dimensional scene, Papadimitriou and Yannakakis [89] show that a "nearest corner strategy" is optimal. If the obstacles are arbitrarily sized, axis-aligned squares, their algorithm has a competitive ratio of $1/2\sqrt{26} \approx 1.7$. They also provide a lower bound of $\Omega(\sqrt{n})$ for this problem where n is the shortest distance between s and t. This bound is matched by Blum, Raghavan, and Schieber [21]. They describe an $O(\sqrt{n})$ competitive algorithm for the case that the goal is on an "infinite wall." They also give an $O(n2^{\sqrt{\log n}})$ algorithm for the case that the environment can be modeled as a two-dimensional room where s is on the wall of the room and t in the center. This result is improved by Bar-Eli et al. [11] with an $O(n \log n)$ algorithm.

Model	Properties	Results	Authors
geo-metric scene	unit-square obstacles	optimal "nearest corner strategy"	[89]
	axis-aligned squares, arbitrary size distance $d(s,t) = n$	competitive algorithm ratio ≈ 1.7 lower-bound $\Omega(\sqrt{n})$	[89]
	t on "infinite wall"	$O(\sqrt{n})$ competitive algorithm	[21]
	s on wall of room, t in center	$O(n2^{\sqrt{\lg n}})$ algorithm	[21]
		$O(n \lg n)$ algorithm	[11]
	rectilinear polygon with m essential cuts	$O(n)$ competitive algorithm	[66]
	streets = polygons with 2 chains between s and t, each point can see some point on opposite chain	competitive algorithm ratio ≈ 5.71	[65]
		ratio ≈ 2.83	[66]
	arbitrarily shaped obstacles, goal coordinates given	algorithm for path between s and t bounded by 1.5 times the sum of perimeters of all obstacles not farther away from t than s.	[79]
	perfect sensing within disks around landmarks no sensing outside k landmarks	$O(k^4 \log k)$ algorithm	[73]

Kleinberg [66] gives a competitive algorithm for exploring an unknown rectilinear polygon. He also addresses the problem of exploring "streets" which was introduced by Klein [65]. Streets are polygons defined by two chains between s and t where each point on one chain can see some point on the opposite chain. Kleinberg's algorithm has competitive ratio of ≈ 2.83 and improves on Klein's ≈ 5.71 competitive algorithm.

For the case that the coordinates of the goal t are given, Lumelsky and Stepanov [79] show how a path through an obstacle course with arbitrarily shaped obstacles can be found. They do not compare the length of this path to the length of a shortest path. Instead, they show a weaker result: the path is bounded by 1.5 times the sum of the perimeters of all obstacles that are not farther away from t than s.

Lazanas and Latombe [73] bound the path to the goal by the number of landmarks k. They consider the case that the robot's sensing is perfect within disks around these landmarks and null outside. The algorithm runs in time $O(k^4 \log k)$.

2.2.3 Path-planning agents

A related problem on graph traversal is the "Chinese Postman Problem" proposed by Kwan [72] and solved by Edmonds and Johnson [42] (more on this problem in Section 2.5.1). Traversing the inside of a polygon was addressed by Chin and Ntafas [30].

Model	Properties	Results	Authors
graph	undirected/directed graph *Chinese postman problem* traversing each edge once	polynomial algorithm	[72] [42]
geometric	*Shortest watchman route* shortest path inside polygon, each point in polygon is visible at least once during route n = number of sides of polygon	$O(n^4 \lg \lg n)$	[30]

2.3 A formal model of piecemeal learning

Modeling the environment

We model the learner's environment as a finite connected undirected graph $G = (V, E)$ with distinguished start vertex s. Vertices represent accessible locations. Edges represent accessibility: if $\{x, y\} \in E$ then the robot can move from x to y, or back, in a single step. The graph is undirected since we want to model situations in which the robot can go back the same way it came from.

The graph is initially unknown to the exploring robot. Using this model, the problem of piecemeal learning of an unknown environment is defined as the problem of exploring the unknown graph:

- *Explore all edges of a graph.*
- *Return to the start vertex when necessary.*
- *Perform both tasks efficiently.*

We illustrate the exploration process with a simple example in Figure 2-2. The exploration begins at start vertex s. At each stage of the exploration process the robot knows its current position. This means that it knows at which vertex it is; it can see the "name" of the vertex. We assume that the learner can always recognize a previously visited vertex; it never confuses distinct locations.

The explorer can also distinguish the outgoing edges at the current vertex. Each edge has a label that distinguishes it from any other edge. Without loss of generality, we can assume that the edges are ordered. At a vertex, the robot knows which edges it has traversed already. The robot has no vision or long-range sensors. Therefore, it can sense only the edges incident to its current location and does not know where unexplored edges lead to. If the robot chooses one such edge, it explores the edge by traversing it. The robot is not allowed to "teleport" from one vertex to some other distant vertex. It can only move from its current location to an adjacent location at each step. The exploration process terminates when the whole graph is explored.

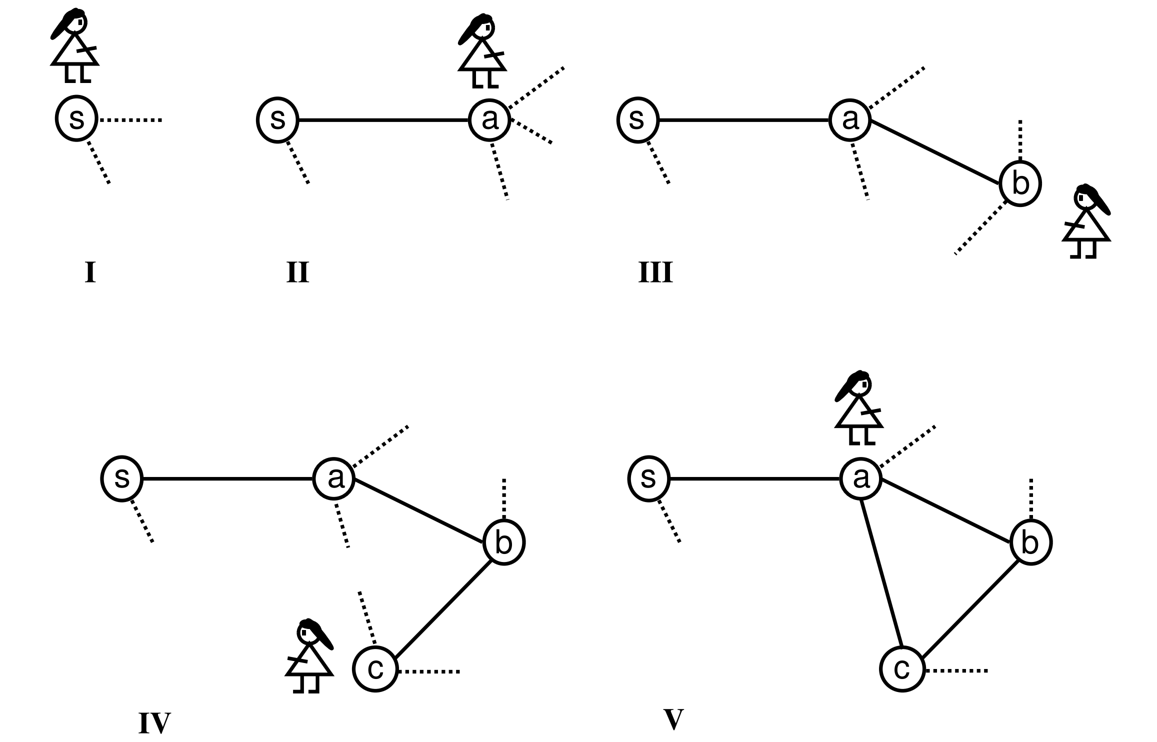

Figure 2-2: The exploration process. Explored edges are shown as solid lines, unexplored edges at explored vertices are shown as dashed lines. Step I: The explorer is at start vertex s and sees two outgoing edges. Step II: The explorer chose edge (s, a) and discovers that it leads to vertex a. Steps III and IV: The explorer discovers vertices b and c. Step V: The explorer discovers edge (c, a) leading back to vertex a. She recognizes vertex a and previously traversed edges (s, a) and (a, b).

Modeling the efficiency of exploration

We are interested in piecemeal algorithms with which the robot can visit every vertex and traverse every edge of the graph efficiently. The efficiency of the algorithm is measured in terms of the number of edges traversed. The time the robot spends "thinking" does not matter. Traversing the mental model is cost-free; traversing edges in the graph is what costs. We assume a uniform cost for an edge traversal. The robot's goal is to minimize the total number of edges traversed.

Modeling the "piecemeal constraint"

The robot is given an upper bound B on the number of steps it can make (edges it can traverse) in one exploration phase. In order to assure that the robot can reach an arbitrary vertex in the graph, do some exploration, and then get back to the start vertex, we assume B allows for at least one round trip between s and any other single vertex in G, and also allows for some number of additional exploration steps. More precisely, we assume $B = (2 + \alpha)r$, where $\alpha > 0$ is some constant, and r is the *radius* of the graph that is the maximum of all shortest-path distances between s and any vertex in G. Note that our definition of the radius of the graph is relative to the start vertex s.

What does the robot know in advance?

The robot knows how to recognize locations in the environment and how to model them as vertices. It models two neighboring locations as adjacent vertices connected by an edge. The robot trusts its sensors and is confident that the recognition and modeling of the environment is done without any error.

We do not assume that the robot a priori knows the size of the environment, i.e., the number of vertices and edges of the graph. The robot initially knows its starting vertex s and the bound B. Advance knowledge of bound B does not mean that the robot has advance knowledge of r and α. However, B may represent physical constraints of the robot's hardware rather than depend on r and α which represent environment constraints. For example, the environment could be huge and the robot's gas tank small. For this scenario, it is obvious that the robot can only use our algorithms to explore the part of the environment that is reachable.

2.4 Initial approaches using DFS and BFS

In this section, we discuss piecemeal search on arbitrary undirected graphs. In particular, we show why "standard approaches" to this problem do not work.

A simple approach to piecemeal exploration is to use an ordinary search algorithm – breadth-first search (BFS) or depth-first search (DFS) – and just interrupt the search as needed to return to visit s. Once the robot has returned to s, it goes back to the vertex at which the search was interrupted and resumes exploration. (Detailed descriptions of BFS and DFS can be found in algorithms textbooks, for example, Cormen, Leiserson, and Rivest [35].)

2.4.1 Depth-first search

In ordinary depth-first search, the search process starts by exploring an edge out of start vertex s and discovering a new vertex adjacent to s. Then an edge out of

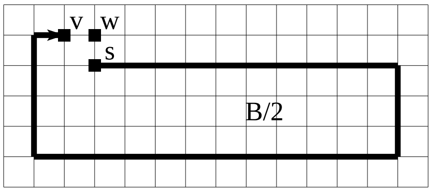

Figure 2-3: The learner reaches vertex v after $B/2$ steps in a depth-first search. Then it must interrupt its search and return to s. It cannot resume exploration at v to get to vertex w, because the known return path is longer than $B/2$, the remaining number of steps allowed in this exploration phase. DFS fails.

this vertex is explored and so forth. In general, edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it. When all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered. This process continues until all edges are explored. In ordinary DFS at most $2|E|$ edges are traversed (each edge is traversed at most once for exploration and once for backtracking).

Piecemeal exploration complicates matters. Since each exploration phase has a limited duration, the search cannot follow an ordinary DFS strategy, but must be interrupted. For example, suppose at step $B/2$ of the first phase of exploration, the robot reaches vertex v of the grid graph illustrated in Figure 2-3. At this point during the depth-first search, the robot must stop exploration and go back to the start location s. The only path it knows from v back to s has length $B/2$. In the second exploration phase, in order for the robot to resume a depth-first search, it must go back to v, the most recently discovered vertex. However, since the robot only knows a path of length $B/2$ to v, it cannot proceed with exploration from v. If it proceeded and did not find a short-cut back to s, it would run out of fuel before it can refuel at s. Therefore, DFS with interruptions due to the piecemeal constraint fails.

Another approach to solve the piecemeal search problem would be to try a *bounded depth-first search* strategy. In bounded DFS, edges are explored out of the most recently discovered vertex v which had depth less than a given bound β. However, a bounded DFS strategy also does not translate into a piecemeal search algorithm for arbitrary undirected graphs.

2.4.2 Breadth-first search

In ordinary breadth-first search, the robot explores all the vertices at the same distance from s before exploring any vertices that are further away from s. Note that in the usual BFS model, the algorithm uses a queue data structure to keep track of which vertex it will search from next. Thus, searching requires extracting a vertex from this queue. However, we model *physical* search: since the robot can only search from its current location, extracting a vertex from this queue corresponds to a change of the robot's position. Our model does not allow the robot to "teleport" to a new position – instead, it must roll or walk there. We say the robot must "relocate" from its current vertex to a previously discovered vertex to resume exploration.

With teleport-free exploration, breadth-first search results in a large number of relocations in graphs in which vertices of the same shortest path distance to s are far away from each other results in a large number of relocations. This makes the search inefficient. For such graphs the cost of relocating between vertices can make the overall cost of BFS quadratic in the number of edges in the graph. More precisely, for a graph (V, E) of radius r we have:

- Exploring all the edges takes time $O(E)$.
- Visiting all vertices with unexplored edges at the same shortest path distance to s can be performed with a depth-first traversal of the BFS tree which takes $O(V)$ time.
- The depth of the BFS tree is r.

Therefore, BFS takes at most $O(E + rV)$ time. Note that a BFS strategy has a crucial advantage over DFS: BFS ensures that the robot always knows a *shortest* path from s to any explored vertex. Whereas a DFS strategy cannot resume exploration at vertices to which it only knows a long path, a BFS strategy can always resume exploration. Knowing shortest paths or approximate shortest paths is a property that is very useful for piecemeal algorithms and an important objective in the design of our algorithms.

2.5 On-line versus off-line

It is beneficial to compare the problem of traversing a familiar environment in a piecemeal manner to the problem of exploring an unknown environment in a piecemeal manner. If a map of the environment is known a priori, traversing this environment in a piecemeal manner can be called an *off-line problem.* If the map is not known a priori and is built during exploration, the piecemeal learning problem can be called an *on-line problem.*

In the following section, we develop a strategy for the off-line problem and give an approximate solution for it. We later adapt this strategy to develop an algorithm for the on-line problem.

2.5.1 Off-line piecemeal search

The off-line problem does not actually have a learning or exploration component. It is rather a search problem. However for simplicity we also use the term "exploration."

In the *off-line* piecemeal search problem, the robot is given a finite connected undirected graph $G = (V, E)$, a start location $s \in V$, and a bound B on the number of edges it may traverse in any exploration phase. The robot's goal is to plan an optimal search of the graph that visits every vertex and traverses every edge, and also satisfies the piecemeal constraint (i.e., each exploration phase traverses at most B edges and starts and ends at the start location).

The *off-line cost* is the number of edges that the robot traverses during piecemeal search. The off-line cost is *optimal* if with the help of the map of the graph, the robot finds the most efficient route that covers every edge of the graph.

The off-line piecemeal search problem is similar to the well-known *Chinese Postman Problem* proposed by Mei-ko Kwan [72]. This is the problem of finding the shortest path that covers every edge of the graph at least once. In our case, the postman must return to the post-office every so often. We call the off-line problem the *"Weak Postman Problem"* since the postman cannot carry all the mail at once, but rather picks up some of it whenever he returns to the starting point. The same problem arises when many postmen must cover the same city with their routes.

The Chinese Postman Problem can be solved by a polynomial time algorithm if the graph is either undirected or directed [42]. The Chinese Postman problem for a mixed graph that has undirected and directed edges was shown to be NP-complete by Papadimitriou [88]. We do not know an optimal off-line algorithm for the Weak Postman Problem; it may be NP-hard. This is an interesting open problem.

We give an efficient off-line piecemeal search algorithm for arbitrary undirected graphs. The algorithm uses an "interrupted depth-first search" and solves the Weak Postman Problem approximately.

Theorem 1 *There exists an approximate solution to the off-line piecemeal search problem for an arbitrary undirected graph $G = (V, E)$ which traverses $O(|E|)$ edges.*

Proof: Assume that the radius of the graph is r and that the number of edges the robot is allowed to traverse in each phase of exploration is $B = (2 + \alpha)r$, for some constant α such that αr is a positive integer. Before the robot starts traversing any edges in the graph, it uses the map of the graph to determine a search route that allows efficient interruptions due to the piecemeal constraint. First, the robot computes a depth-first search tree of the graph. A depth first traversal of this depth-first search tree defines a path P of length $2|E|$ which starts and ends at s and which goes through every vertex and edge in the graph (see Figure 2-4). Then the robot divides this path into segments of length αr. (These segments determine the search per exploration phase.) The robot also computes a shortest path P_a from s to the start of each segment. (These paths allow efficient interruptions.)

The robot then starts the piecemeal exploration of the graph. Each phase of the exploration consists of taking a shortest path P_a from s to the start of a segment, traversing the edges in the segment, and taking a shortest path P_b back to s. For each segment, the robot traverses at most $2r$ edges to get to and from the segment. Since there are $\lceil \frac{2|E|}{\alpha r} \rceil$ segments, there are $\lceil \frac{2|E|}{\alpha r} \rceil - 1$ interruptions, and the number of

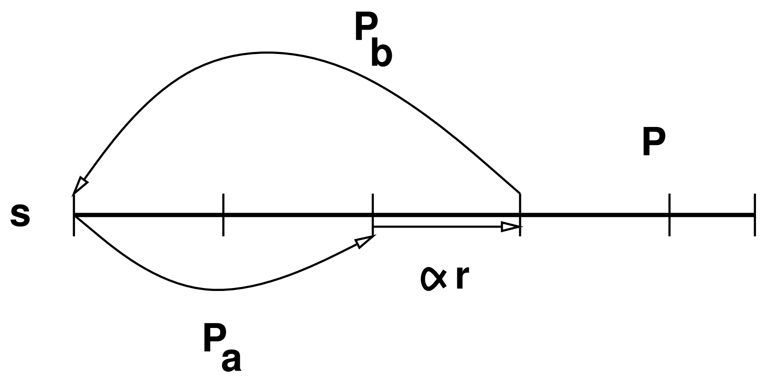

Figure 2-4: Illustration of the proof of Theorem 1.

edge traversals due to interruptions is at most:

$$\begin{aligned}(\left\lceil \frac{2|E|}{\alpha r} \right\rceil - 1)2r &\leq \left(\frac{2|E|}{\alpha r}\right) 2r \\ &\leq \frac{4|E|}{\alpha}.\end{aligned}$$

The number of edge traversals due to exploration and backtracking is at most $2|E|$. Thus the total number of edge traversals is at most $(4/\alpha + 2)|E| = O(E)$. □

2.5.2 On-line piecemeal search

The on-line problem corresponds to what a explorer can really do. It requires that its decisions can only be based on what it has seen so far.

This section shows how we can change the strategy for off-line piecemeal search to obtain a general method for converting certain types of search algorithms into on-line piecemeal exploration algorithms. The "piecemeal theorem" states that an exploration algorithm in which the explorer always knows a path back to the start vertex of length at most r can always be interrupted for refueling without changing the running time of the algorithm by more than a constant factor.

We call an on-line search *optimally interruptible* if the robot always knows a shortest path back to s that can always be composed from the edges that have been explored. We refer to a search as *efficiently interruptible* if it always knows a path back to s via explored edges of length at most the radius of the graph. We say a search algorithm is a *linear time* algorithm if the robot traverses $O(E)$ edges during the search.

Theorem 2 (Piecemeal theorem)*:*

An efficiently interruptible algorithm for exploring an unknown graph $G = (V, E)$ with n vertices and m edges that takes time $T(n, m)$ can be transformed into a piece-

meal learning algorithm that takes time $O(T(n,m))$.

Proof: The proof of this theorem is similar to the proof of Theorem 1. However, there are a few differences. The search is on-line and is being interrrupted during exploration. Instead of using an ordinary search algorithm (like DFS) and interrupting as needed to return to s, we use the given efficiently interruptible search algorithm. Moreover, the cost of the search is not $2|E|$ as in DFS, but at most $cT(n,m)$.

Assume that the radius of the graph is r and that the number of edges the robot is allowed to traverse in each phase of exploration is $B=(2+\alpha)r$, for some constant α such that αr is a positive integer. In each exploration phase, the robot executes αr steps of the original search algorithm, interrupts its search, and returns to the start vertex s (see Figure 2-5). At the beginning of the next phase, the robot returns from s to the appropriate vertex to resume exploration. Then, the robot traverses again αr edges as determined by the original search algorithm and returns to s. Since the search algorithm is efficiently interruptible, the robot knows a path of distance at most r from s to any vertex in the graph. Thus during any exploration phase, the robot traverses at most $2r$ edges for relocation to s and back and αr edges for exploring them. Because $2r+\alpha r=B$, the robot exploits its fuel (bounded by B) efficiently.

Since there are $\lceil \frac{T(n,m)}{\alpha r} \rceil$ segments, there are $\lceil \frac{T(n,m)}{\alpha r} \rceil - 1$ interruptions, and the number of edge traversals due to interruptions is:

$$\begin{aligned}(\left\lceil \frac{T(n,m)}{\alpha r} \right\rceil - 1)2r &\le \frac{T(n,m)}{\alpha r}2r \\ &\le \frac{2T(n,m)}{\alpha}.\end{aligned}$$

Thus, the total number of edge traversals is upper bounded by $(2/\alpha)T(n,m) + cT(n,m) = O(T(n,m))$. □

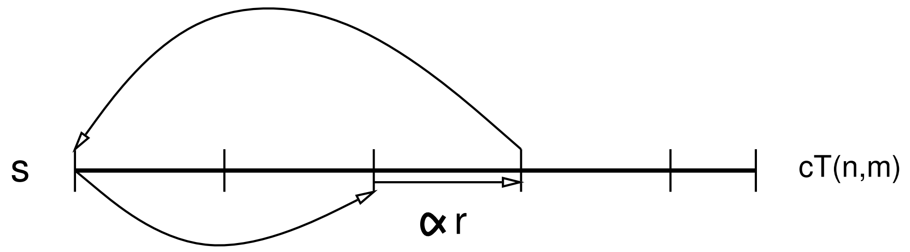

Figure 2-5: Illustration of the proof of Theorem 2.

Corollary 1 *An optimally interruptible, linear-time algorithm for searching an undirected graph can be transformed into a linear-time piecemeal search algorithm.*

Corollary 1 means what we need to do is find optimally interruptible algorithms. Once an optimally interruptible, linear-time search algorithm is found, it can easily be transformed into a linear piecemeal exploration algorithm using the procedure outlined in the proof of Theorem 2 (see also Figure 2-6).

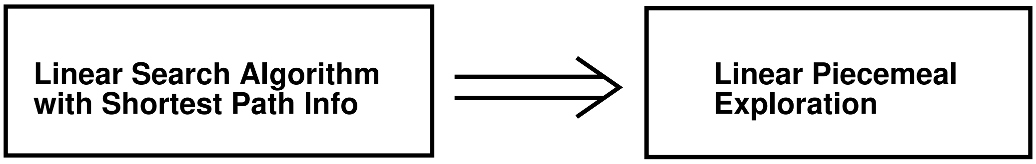

Figure 2-6: Certain linear search algorithms can be transformed into linear piecemeal exploration algorithms.

2.6 A lower bound

Let d_{robot} denote the shortest-path distance from s to the vertex the robot is visiting, and let $d_{unvisited}$ denote the shortest-path distance from s to the vertex nearest to s that is as yet unvisited. (Note that these are dynamic quantities that change as the robot moves around.)

Lemma 1 *A robot which maintains $d_{robot} \leq d_{unvisited}$ (such as BFS) may traverse $\Omega(E^2)$ edges.*

Proof: Consider the graph shown in Figure 2-7 whose vertices are labeled with

$$\{-n, -n+1, \ldots, -1, 0, 1, 2, \ldots, n-1, n\}$$

where s is labeled with "0" and edges connect vertices whose labels are consecutive integers. To achieve $d_{robot} \leq d_{unvisited}$, a teleport-free BFS algorithm would run in quadratic time, traveling back and forth from 1 to -1 to -2 to 2 to 3 .... $\square$

Corollary 2 *Any optimally interruptible search algorithm requires $\Omega(E^2)$ edge traversals in the worst case.*

With teleport-free search, it is generally impossible to maintain $d_{robot} \leq d_{unvisited}$ without a great loss of efficiency: Therefore, in our piecemeal algorithms, we allow the robot to move further away from s than the unvisited vertex nearest to s. For some of our algorithms we maintain $d_{robot} \leq d_{unvisited}$.

2.7 Exploring city-block graphs

We begin our presentation of piecemeal exploration algorithms by focusing on a special class of undirected planar graphs. We call these graphs *city-block graphs*. They

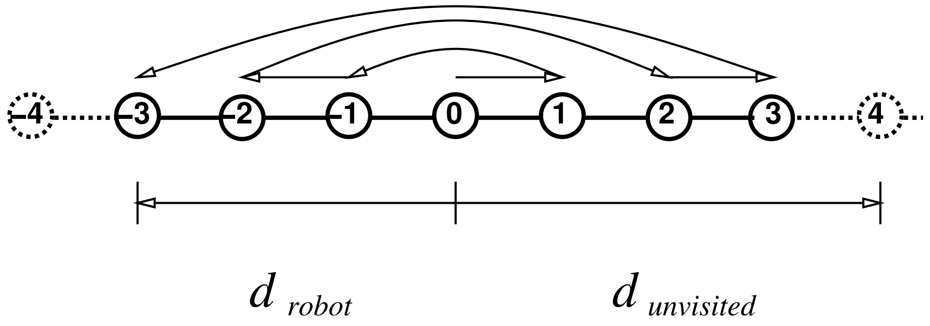

Figure 2-7: A simple graph for which the cost of BFS is quadratic in the number of edges. The explorer travels back and forth from 0 to 1 to -1 to -2 to 2 to 3 to -3. The distance d_{robot} is 3, the distance $d_{unvisited}$ is 4.

model environments such as cities or office buildings in which efficient on-line robot navigation may be needed. City-block graphs are grid graphs: planar graphs with only horizontal and vertical edges. They contain faces that are axis-parallel rectangles. A square face composed of four edges, a 1×1 face, may correspond to a "standard city block." Larger faces may correspond to parks or shopping malls. We call such large faces "obstacles." City-block graphs are also studied by Papadimitriou and Yanakakis [89], Blum, Raghavan, and Schieber [21], and Bar-Eli, Berman, Fiat and Yan [11]. Figure 2-8 gives an example of a city-block graph with fourteen obstacles of different sizes scattered around the start vertex s.

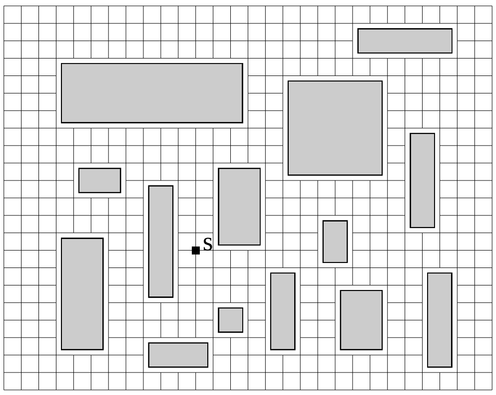

Figure 2-8: A city-block graph with distinguished start vertex s.

An $m \times n$ city-block graph with no obstacles has exactly mn vertices (at points (i, j) for $1 \leq i \leq m$, $1 \leq j \leq n$) and $2mn - (m + n)$ edges (between points at distance 1 from each other). If obstacles are present, the number of vertices is less than mn. We assume that the robot can recognize the directions of incident edges.

For city-block graphs we present two efficient $O(|E|)$ optimally interruptible search algorithms. An optimally interruptible search algorithm is also an efficiently interruptible search algorithm. Using the strategy described in the proof of the piecemeal theorem (Theorem 2), these two search algorithms can be adopted to give efficient piecemeal search algorithms for city-block graphs.

2.7.1 Shortest paths in city-block graphs

An optimally interruptible algorithm maintains at all times knowledge of a shortest path back to s. Since BFS is optimally interruptible, we study BFS in some detail to understand the characteristics of shortest paths in city-block graphs. Also, the algorithm presented next, called "wavefront algorithm," is a modification of BFS. Figure 2-9 illustrates the operation of BFS. Our algorithms depend on the special properties that shortest paths have in city-block graphs.

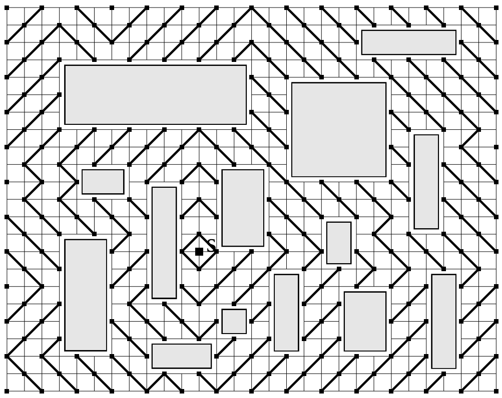

Figure 2-9: Environment explored by breath-first search. The exploration process starting at s is similar to waves expanding from the point where a pebble is dropped in water. Only "wavefronts" at odd distance to s are shown.

Let $\delta[v, v']$ denote the length of the shortest path between v and v', and let $d[v]$ denote $\delta[v, s]$, the length of the shortest path from v back to s.

2.7.2 Monotone paths and the four-way decomposition

A city-block graph can be usefully divided into four regions (north, south, east, and west) by four monotone paths: an east-north path, an east-south path, a west-north path, and a west-south path. The east-north path starts from s, proceeds east until it hits an obstacle, then proceeds north until it hits an obstacle, then turns and proceeds east again, and so on. The other paths are similar. An example of a city-block graph divided into four regions by the monotone paths is shown in Figure 2-10. Note that all monotone paths are shortest paths. Furthermore, note that s is included in all four regions, and a path is included in both of its neighboring regions.

In Lemma 2 we show that for any vertex, there is a shortest path to s through only one region. Without loss of generality, we therefore only consider optimally interruptible search algorithms that divide the graph into these four regions, and search these regions separately.

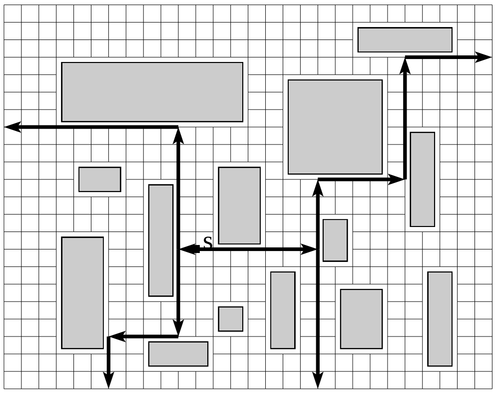

Figure 2-10: The four monotone paths and the four regions.

Lemma 2 *There exists a shortest path from s to any point in a region that only goes through that region.*

Proof: Consider a point v in some region A for which a shortest path P from s to the point v is not entirely contained in region A. Such a scenario is shown in Figure 2-11. We show how to construct another path P' that is entirely contained in region A. (Note that the vertices and edges which make up the monotone paths surrounding a region A are considered to be part of that region).

Since path P starts and ends in region A but is not entirely contained in region A, there must be a point u that is on P and also on one of the monotone paths bordering A. Note that u may be the same as v. Without loss of generality, let u be the last such point, so that the portion of the path from u to v is contained entirely within region A. Then the path P' will consist of the shortest path from s to u along the monotone path that u is on, followed by the portion of P from u to v. This path P' is a shortest path from s to v because P was a shortest path and P' can be no longer than P. □

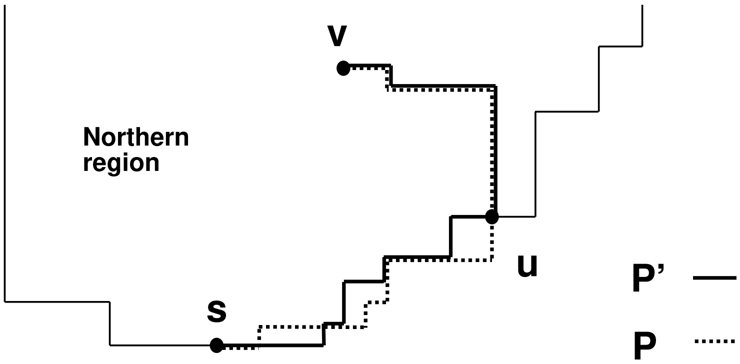

Figure 2-11: Illustrating the proof of Lemma 2 for the northern region.

2.7.3 Canonical shortest paths of city-block graphs

We now make a fundamental observation on the nature of shortest paths from a vertex v back to s. In this section, we consider shortest paths in the northern region; properties of shortest paths in the other regions are similar.

Lemma 3 *For any vertex w in the northern region, there is a canonical shortest path from w to the start vertex s which goes south whenever possible. The canonical shortest path goes east or west only when it is prevented from going south by an obstacle or by the monotone path defining the northern region.*

Proof: We call the length $d[v]$ of the shortest path from v to s the *depth* of vertex v. We show this lemma by induction on the depth of a vertex.

For the base case, it is easy to verify that any vertex v such that $d[v] = 1$ has a canonical shortest path that goes south whenever possible.

For the inductive hypothesis, we assume that the lemma is true for all vertices that have depth $t-1$, and we want to show it is true for all vertices that have depth t. Consider a vertex v at depth t. If there is an obstacle obstructing the vertex that is south of point v or if v is on a horizontal segment of the mononotone path defining the northern region, then it is impossible for the canonical shortest path to go south, and the claim holds.

Thus, assume there is a vertex v_s south of v that can be reached from v in one step. If $d[v_s] = t - 1$ our claim holds: the canonical path goes south from v to v_s and then follows the canonical path of v_s. For contradiction, assume that the depth of v_s is not $t - 1$ and therefore the canonical path from v to s goes east or west of v. In the following we assume it goes west and illustrate this in Figure 2-12. The case that it goes east can be handled similarly.

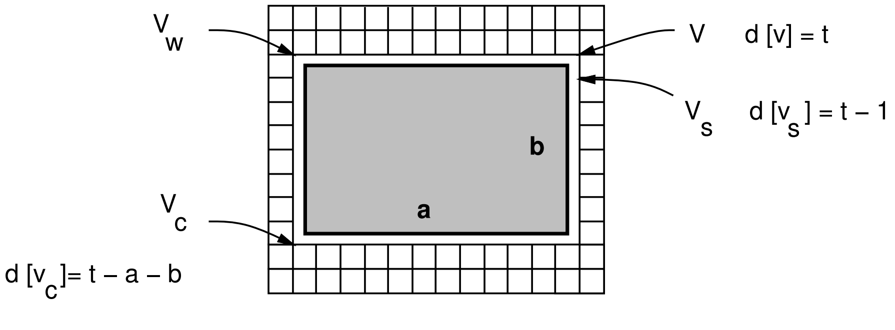

Figure 2-12: Illustration of the proof of Lemma 3

Assume it goes west until it reaches the first vertex which has an edge leading south. Since this vertex v_w has depth $d[v_w] < t$, the induction assumption holds and its canonical path goes south.

The path from v to v_w defines the northern side of some $(a \times b)$ face in the graph. If $a = b = 1$ the face is a normal city block, otherwise it is an obstacle (see Figure 2-12).

The vertex v_c at the southwest corner of the face lies on the canonical path from v to s. Since $d[v] = t$ vertex v_c has depth $d[v_c] = t - a - b$. Furthermore, there is a path from v_c to v_s along the southern and eastern side of the face that has length $a + b - 1$. Since $d[v_c] = t - a - b$, we have $d[v_s] = t - 1$. This finally contradicts the assumption that the depth of v_s was not $t - 1$ and the claim that the canonical path goes south is proven. □

Can two adjacent vertices have the same depth? The following corollary and lemma show that this cannot happen.

Corollary 3 *Consider adjacent vertices v and w in the grid graph where v is north of w. In the northern region, without loss of generality, $d[v] = d[w] + 1$.*

Lemma 4 *Consider adjacent vertices v and w in the grid graph where v is west of w. In the northern region, without loss of generality, $d[v] = d[w] \pm 1$.*

Proof: We prove the lemma by induction on the y-coordinate of the vertices in the northern region. If v and w have the same y-coordinate as s, then we know that $d[v] = d[w] + 1$ if s is east of w and $d[v] = d[w] - 1$ if s is west of v. Assume that the claim is true for vertices v and w with y-coordinate k. In the following we show that it is also true for vertices v and w with y-coordinate $k + 1$.

Consider two cases:

1. If there is no obstacle directly south of v or of w the claim follows by Lemma 3 and the inductive assumption.

2. There is an obstacle directly south of v or of w. We assume without loss of generality that both v and w are on the boundary of the north side of the obstacle. (Note that v or w may, however, be at a corner of the obstacle.)

If our claim did not hold it would mean that $d[v] = d[w]$ for two adjacent vertices v and w (because, in any graph, the d values for adjacent vertices can differ by at most one). This would also mean that all shortest paths from v to s must go through vertex v_w at the north-west corner of the obstacle and all shortest paths from w to s must go through vertex v_e at the north-east corner of the obstacle. However, we next show that there is a point p_m on the boundary of the north side of the obstacle that has shortest paths through both v_e and v_w. The claim of Lemma 4 follows directly.

Given the length k of the north side of the obstacle, the distance x between p_m and v_w can be obtained by solving the following equation: $x + d[v_w] = (k - x) + d[v_e]$. By inductive assumption the following holds: If k is even then $|d[v_e] - d[v_w]|$ is even, and if k is odd then $|d[v_e] - d[v_w]|$ is odd. The distance x is $(k + d[v_e] - d[v_w])/2$. Therefore, distance x is integral and p_m is a vertex in the grid-graph. □

2.8 The wavefront algorithm

The *wavefront algorithm* is based on BFS, but overcomes the BFS's problem of relocation cost. In this section we first develop some preliminary concepts and results based on an analysis of breadth-first search. We then present the wavefront algorithm, prove it to be correct, and show that it runs in linear time.

2.8.1 Properties of wavefronts

In city-block graphs a BFS can be viewed as exploring the graph in waves that expand outward from s, much as waves expand from a pebble thrown into a pond. Figure 2-9 illustrates the "wavefronts" that can arise.

We define wavefronts to contain vertices that all have the same shortest path distance to s and that are somehow "close" to each other. More formally, a *wavefront* w is defined as an ordered list of explored vertices $\langle v_1, v_2, \ldots, v_m \rangle$, $m \geq 1$, such that $d[v_i] = d[v_1]$ for all i, and such that $\delta(v_i, v_{i+1}) \leq 2$ for all i. Lemma 5 proves that the distance between adjacent points in a wavefront is always exactly equal to 2. We call $d[w] = d[v_1]$ the distance of the wavefront.

There is a natural "successor" relationship between BFS wavefronts, as a wavefront at distance t generates a successor at distance $t+1$. We informally consider a *wave* to be a sequence of successive wavefronts. Because of obstacles, however, a wave may

- **split** if it hits an obstacle or
- **merge** with another wave, on the far side of an obstacle.

Two wavefronts are *sibling* wavefronts if they each have exactly one endpoint on the same obstacle and if the waves to which they belong merge on the far side of that obstacle. (Sibling wavefronts do not need to have to same depth.) The point on an obstacle where the waves first meet is called the *meeting point* p_m of the obstacle. In the northern region, meeting points are always on the north side of obstacles, and each obstacle has exactly one meeting point on its northern side. In Figure 2-13 there is a meeting point at the eastern back corner of the obstacle. In Figure 2-14 the meeting point is at the back of the obstacle.

Lemma 5 *A wavefront can only consist of diagonal segments.*

Proof: By definition the distance between adjacent vertices of a wavefront is at most 2. It follows from Corollary 3 and Lemma 4 that neighboring points in the grid cannot be in the same wavefront. Therefore, the distance between adjacent vertices is exactly 2. Thus, the wavefront can only consist of diagonal segments. □

We call the points that connect diagonal segments of different orientation of a wavefront *peak* or *valleys*. A peak has a larger y-coordinate than the y-coordinates of

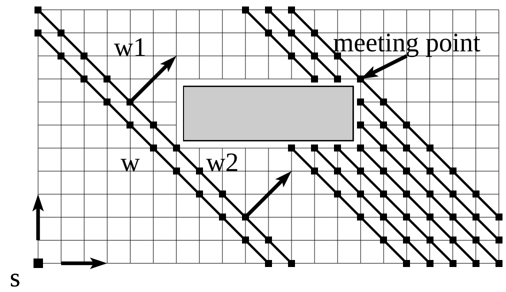

Figure 2-13: Splitting and merging of wavefronts along a corner of an obstacle. Illustration of meeting point and sibling wavefronts.

its adjacent vertices in the wavefront, and a valley has a smaller y-coordinate than the y-coordinates of its adjacent vertices as illustrated in Figure 2-14.

The initial wavefront is just a list containing the start point s. Until a successor of the initial wavefront hits an obstacle, the successor wavefronts always consist of two diagonal segments of equal length connected by a peak. This peak is at the same x-coordinate for these successive wavefronts and at the same x-coordinate as s. Therefore, we say that the *"shape"* of the wavefronts does not change. In the northern region a wavefront can only have descendants that have a different shape if a descendant curls around the northern corners of an obstacle, or when it merges with another wavefront, or splits into other wavefronts. These descendants may have more complicated shapes with several peaks and valleys.

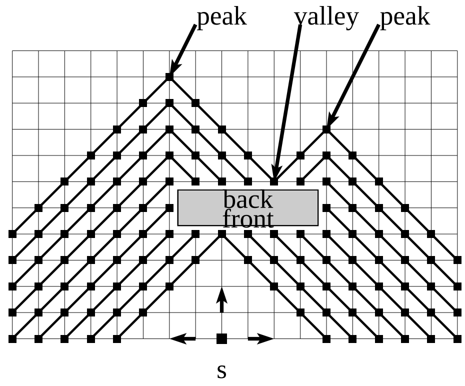

Figure 2-14: Shapes of wavefronts. Illustration of peaks and valleys, and front and back of an obstacle. The meeting point is the lowest point in the valley.

A wavefront w splits whenever its hits an obstacle. More precisely, a vertex v_i in wavefront w is on the boundary of an obstacle and w splits into wavefronts $w_1 = \langle v_1, v_2, \ldots, v_i \rangle$ and $w_2 = \langle v_i, v_{i+1}, \ldots, v_m \rangle$. Then wavefront w_1 propagates around the obstacle in one direction, and wavefront w_2 propagates around in the other direction. Eventually, some descendant wavefront of w_1 and some descendant wavefront of w_2 will have a common point on the boundary of the obstacle – the meeting point. The position of the meeting point is determined by the shape of the wave approaching the obstacle. (In the proof of Lemma 4 vertex p_m is a meeting point and we showed how to calculate its postion once the length k of the north side of the obstacle and the shortest path distances of the vertices v_e and v_w at the north-east and north-west corners of the obstacle are known: The distance from v_w to the meeting point m is $(k + d[v_w] - d[v_e])/2$.)

In the northern region, the *front* of an obstacle is its south side, the *back* of an obstacle is its north side, and the *sides* of an obstacle are its east and west sides. A wave always hits the front of an obstacle first. Consider the shape of a wave before it hits an obstacle and its shape after it passes the obstacle. If a peak of the wavefront hits the obstacle (but not at a corner), this peak will not be part of the shape of the wave after it "passes" the obstacle. Instead, the merged wavefront may have one or two new peaks which have the same x-coordinates as the sides of the obstacle. For example, Figure 2-14 illustrates a merged wavefront of shortest path distance 12 with two peaks. It also has a valley at the meeting point on the boundary of the obstacle.

2.8.2 Description of the wavefront algorithm

The wavefront algorithm, presented in this section, mimics BFS in that it computes exactly the same set of wavefronts as BFS. However, in order to minimize relocation costs, the wavefronts may be computed in a different order than BFS. Rather than computing all the wavefronts at distance t before computing any wavefronts at distance $t + 1$ as BFS does, the wavefront algorithm will continue to follow a particular wave persistently, before it relocates and pushes another wave along.

We define *expanding* a wavefront $w = \langle v_1, v_2, \ldots, v_k \rangle$ as computing a set of zero or more successor wavefronts by looking at the set of all unexplored vertices at distance one from any vertex in w. Every vertex v in a successor wavefront has distance $d[v] = d[w] + 1$. The explorer starts with vertex v_1 and moves to all of its unexplored adjacent vertices. The explorer then moves to the next vertex in the wavefront and explores its adjacent unexplored vertices. It proceeds this way down the vertices of the wavefront. The following lemma shows how to expand a wavefront efficiently. Its proof is illustrated in Figure 2-15.

Lemma 6 *An explorer can expand a wavefront consisting of k vertices by traversing at most $O(k)$ edges.*

Proof: To expand a wavefront $w = \langle v_1, v_2, \ldots, v_k \rangle$ the robot explores all unexplored edges incident to the vertices in w. This can be done efficiently by traversing the path $v_1 \to v_2 \to \ldots \to v_k$ along the pairs of unexplored edges between the vertices in w. This results in at most $2(k-1)$ edge traversals, since neighboring vertices v_i, v_{i+1} for $1 \leq i \leq k$ are at most 2 apart. The successor wavefront might have $k + 2$ vertices. Thus at the beginning of the expansion, at vertex v_1, the robot may have to traverse an additional edge twice and at the end of the expansion, at vertex v_k, an additional edge once. In addition, at any vertex which is a peak, the robot may have to traverse an edge twice. A wavefront has at most $\lceil k/2 \rceil$ peaks. Summing the number of edge

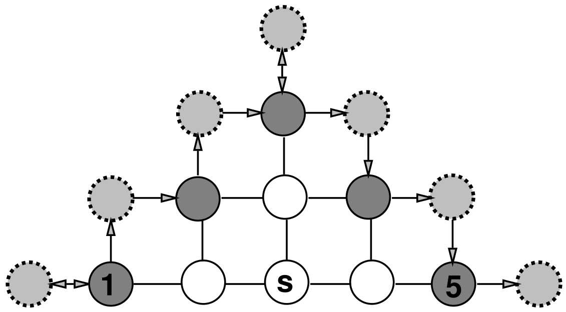

Figure 2-15: Expansion of a wavefront: Wavefront $\langle v_1, \ldots, v_5 \rangle$ consisting of vertices shown with dark shading is expanded into a wavefront with seven vertices shown with light shading. The robot starts the expansion at vertex v_1 and explores the edges by following the arrows.

traversals, we conclude that the robot traverses at most $2(k-1)+3+2\lceil k/2 \rceil = O(k)$ edges. □

Since our algorithm computes exactly the same set of wavefronts as BFS, but persistently pushes one wave along, it is important to make sure the wavefronts are expanded correctly. The one incorrect way to expand a wavefront and get something other than what BFS would obtain is the following: expanding a wavefront that is touching a meeting point before its sibling wavefront has merged with it.

To avoid this incorrect expansion, the wavefront algorithm is *blocked* in the following two situations:

(a) it cannot expand a wavefront from the side around to the back of an obstacle before the meeting point for that obstacle has been set (see Figure 2-16), and

(b) it cannot expand a wavefront that touches a meeting point until its sibling has arrived there as well (see Figure 2-17).

A wavefront w_2 *blocks* a wavefront w_1 if w_2 must be expanded before w_1 can be safely expanded. We also say w_2 and w_1 *interfere*.

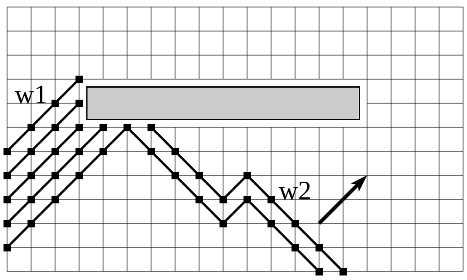

Figure 2-16: Blockage of w_1 by w_2. Wavefront w_1 has finished covering one side of the obstacle and the meeting point is not set yet.

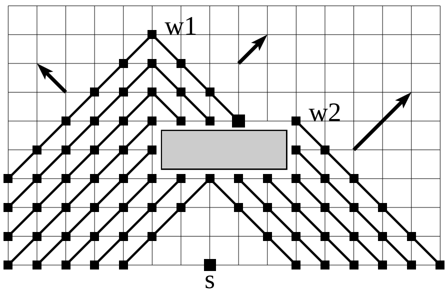

Figure 2-17: Blockage of w_1 by w_2. Wavefront w_1 has reached the meeting point on the obstacle at distance $d[w_1] = d[m] = 10$, but the sibling wavefront w_2 has not; $d[w_2] = 8$.

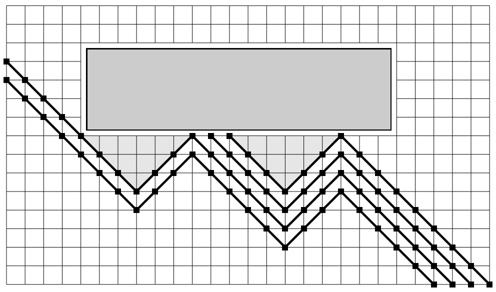

Figure 2-18: Triangular areas (shaded) delineated by two expiring wavefronts.

A wavefront w is an *expiring* wavefront if its descendant wavefronts can never interfere with the expansion of any other wavefronts that now exist or any of their descendants. A wavefront w is an expiring wavefront if its endpoints are both on the front of the same obstacle; w will expand into the region surrounded by the wavefront and the obstacle, and then disappear or "expire." We say that a wavefront expires if it consists of just one vertex with no unexplored neighbors.

Procedure WAVEFRONT-ALGORITHM is search algorithm that can be optimally interrupted to create an efficient piecemeal search algorithm. The algorithm pushes a wave along, i.e., it repeatedly expands one wavefront until the wavefront splits, merges, expires, or is blocked. The WAVEFRONT-ALGORITHM takes as an input a start point s and the boundary coordinates of the environment. It creates four monotone paths and defines the four regions north, south, east, and west as described in Section 2.7.2. Procedure EXPLORE-AREA is called to explore each region separately.

WAVEFRONT-ALGORITHM (s, *boundary*)
create monotone paths
For *region* = north, south, east, and west **Do**
    initialize current wavefront $w := \langle s \rangle$
    EXPLORE-AREA (w, *region*)
    take a shortest path to s

Each iteration of EXPLORE-AREA expands a wavefront. When EXPAND is called on a wavefront w, the explorer starts expanding w from its current location, which is a vertex at one of the end points of wavefront w. It is convenient, however, to think of EXPAND as finding the unexplored neighbors of the vertices in w in parallel.

For each region an ordered list L of all the wavefronts to be expanded is kept. In the northern region, the wavefronts are ordered by the x-coordinate of their west-most point. *Neighboring* wavefronts are wavefronts that are adjacent in the ordered list L of wavefronts. Note that for each pair of neighboring wavefronts there is an obstacle on which both wavefronts have an endpoint.

Depending on what happens during the expansion, the successor wavefront can be split, merged, blocked, or may expire. Note that more than one of these cases may apply.

Initially, each wavefront in the northern region is expanded from its west-most vertex to its east-most vertex. We say wavefronts are expanded in a "clockwise" manner. The direction of expansion changes for the first time in the northern region when a wavefront is blocked by a wavefront to its west (the direction of expansion then becomes "counter-clockwise"). In fact, the direction of expansion changes each time a wavefront is blocked by a wavefront that is in the direction opposite of expansion.

Note that vertices on the monotone paths are considered initially to be unexplored, and that expanding a wavefront returns a successor that is entirely within the same region.

The boundaries of the city-block graph are treated as large obstacles. Each region has been fully explored when its list L of wavefronts is empty.

Procedures MERGE and SPLIT handle the (not necessarily disjoint) cases of merging and splitting wavefronts. Note that we use call-by-reference conventions for the wavefront w and the list L of wavefronts: assignments to these variables within procedures MERGE and SPLIT affect their values in procedure EXPLORE-AREA.

Two wavefronts w and w' are merged when exploration continues around an obstacle. They are removed from the list L of wavefronts and the merged wavefront w'' is included at the appropriate position in L. A wavefront may be merged with two wavefronts, one on each end. Note that a merged wavefront may be blocked.

Procedure SPLIT is called on wavefront w in line 23 of procedure EXPLORE-AREA when the wavefront w hits an obstacle. Wavefront w is split into $w_0, \ldots, w_n$, and the ordered list L of wavefronts and the current wavefront is updated.

Each time procedure RELOCATE(w, dir) is called, the explorer moves from its current location to the appropriate end point of w: in the northern region, if the direction is "clockwise" the explorer moves to the west-most vertex of w, and if the direction is "counter-clockwise," the explorer moves to the east-most vertex of w.

Procedure RELOCATE(w, dir) can be implemented so that when it is called, the explorer simply moves from its current location to the appropriate endpoint of w via a shortest path in the explored area of the graph. However, for analysis purposes, we assume that when RELOCATE(w, dir) is called the explorer moves from its current

```
EXPLORE-AREA (w, region)
initialize list of wavefronts L := ⟨w⟩
initialize direction dir := clockwise
Repeat
    EXPAND current wavefront w to successor wavefront w_s
    RELOCATE (w_s, dir)
    current wavefront w := w_s
    If w is a single vertex with no unexplored neighboring vertices
        Then
            remove w from ordered list L of wavefronts
            If L is not empty
                Then
                    w := neighboring wavefront of w in direction dir
                    RELOCATE (w, dir)
        Else
            replace w by w_s in ordered list L of wavefronts
            If the second back corner of any obstacle(s)
                    has just been explored
                Then set meeting points for those obstacle(s)
            If w can be merged with adjacent wavefront(s)
                Then MERGE (w, L, region, dir)
            If w hits obstacle(s)
                Then SPLIT (w, L, region, dir)
    If L not empty
        Then
            If  w is blocked by neighboring wavefront w' in direction
                    D ∈ { clockwise, counter-clockwise }
                Then
                    dir := reverse D
                    While w is blocked by neighboring wavefront w'
                        Do
                            w := w'
                            RELOCATE (w, dir)
until L is empty
```

```
MERGE (w, L, region, dir)
remove w from list L of wavefronts
While there is a neighboring wavefront w' with which w can merge
Do
    remove w' from list L of wavefronts
    merge w and w' into wavefront w''
    w := w''
    put w in ordered list L of wavefronts
If w is not blocked
    Then RELOCATE (w, dir)
```

SPLIT (w, L, *region*, *dir*)
split w into appropriate wavefronts $w_0, \ldots, w_n$ in standard order
remove w from ordered list L of wavefronts
For $i = 0$ **To** n
    put w_i on ordered list L of wavefronts
    If dir = clockwise
        Then $w := w_0$
        Else $w := w_n$

location to the appropriate end point of w as follows.

- Line 6 of EXPLORE-AREA: the explorer traverses the vertices of the newly expanded wavefront w_s to get back to its appropriate end point.
- Line 14 of EXPLORE-AREA: the explorer traverses edges along the boundary of an obstacle to get to w.
- Line 10 of MERGE: the explorer traverses vertices of the newly merged wavefront w to get to its appropriate end point.
- Line 32 of EXPLORE-AREA: the explorer traverses edges as follows. Suppose the explorer is in the northern region and at the west-most vertex of wavefront w_0, and assume that w is to the east of w_0. Note that both w_0 and w are in the current ordered list of wavefronts L. Thus there is a path between the explorer's current location and wavefront w which "follows the chain" of wavefronts between w_0 and w: Let $w_1, w_2, \ldots, w_k$ be the (possibly empty) list of wavefronts between w_0 and and w, and let $b_0, b_1, \ldots b_{k+1}$ be the obstacles separating wavefronts $w_0, w_1, \ldots, w_k, w$ (i.e., obstacle b_0 is between w_0 and w_1, obstacle b_1 is between w_1 and w_2, and so on). In order to relocate from w_0 to w, the explorer traverses the vertices of wavefront w_0 to get to the east-most vertex of w_0 which is on obstacle b_0. Then the explorer moves along the obstacle b_0 to get to the west-point vertex of w_1. From there the explorer traverses the vertices in wavefront w_1 to get to the east-most vertex of w_1 which is on obstacle b_1. The explorer continues traversing edges in this manner, alternating between traversing wavefronts and traversing obstacles, until it reaches wavefront w.

2.8.3 Correctness of the wavefront algorithm

The following theorems establish the correctness of the wavefront algorithm.

Theorem 3 *The algorithm* EXPLORE-AREA *expands wavefronts so as to maintain optimal interruptability.*

Proof: The key observations are

1. there is a canonical shortest path from any vertex v to s which goes south whenever possible, but east or west around obstacles and

2. a wavefront is never expanded beyond a meeting point.

We show by induction on the number of stages in the algorithm that at any time our algorithm knows the shortest path from s to any explored vertex in the north region. Each stage of the algorithm is an expansion of a wavefront.

The shortest path property is trivially true when the number of stages $k = 1$. There is initially only one wavefront, the start point. Now we assume all wavefronts that exist just after the k-th stage satisfy the shortest path property, and we want to show that all wavefronts that exist just after the $k+1$-st stage also satisfy the shortest path property.

Consider a wavefront w in the k-th stage which the algorithm has expanded in the $k+1$-st stage to w_n. We claim that all vertices in w_n have shortest path length $d[w]+1$. Note that any vertex in w_n which is directly north of a vertex in w in the grid graph has shortest path length $d[w_n] = d[w]+1$. This is because by Lemma 3 there is a shortest path from any vertex in the graph to s which goes south whenever possible.

The only time any vertex v_n in w_n is not directly north of a vertex in w is when w is expanded around the back of an obstacle. This can only occur for a vertes that is either the west-most or east-most vertex of a wavefront. Without loss of generality we assume that v is the west-most point on w and is expanded to its western adjacent vertex v_n. If the meeting point is west of v_n or at v_n, the shortest path distance of v_n is $d[w_n] = d[w]+1$ by Lemma 4. The algorithm maintains a correct shortest path that goes from v_n through v and then follows v's shortest path. (Note that the algorithm does not expand eastwards beyond the meeting point.) Thus, the wavefront that includes vertex v_n is expanded correctly so as to maintain shortest path information. □

Theorem 4 *There is always a wavefront that is not blocked.*

Proof: We consider exploration in the northern region. The key observations are that

(1) neighboring wavefronts cannot simultaneously block each other and

(2) the east-most wavefront in the north region cannot be blocked by anything to its east, and the west-most wavefront in the north region cannot be blocked by anything to its west.

Thus the robot can always "follow a chain" of wavefronts to either its east or west to find an unblocked wavefront. More precisely, the robot must traverse the edges that connect the vertices in each wavefront in the list of wavefronts L between its current wavefront w the unblocked wavefront $\hat{w}$. It must also traverse the edges on the boundaries of the obstacles between these wavefronts. Note that neighboring wavefronts in list L each have at least one endpoint that lies on the boundary of the same obstacle.

A neighboring wavefront is either a sibling wavefront or an expiring wavefront. An expiring wavefront can never block neighboring wavefronts. Therefore, it suffices to show that sibling wavefronts cannot block each other.

Before we show that sibling wavefronts cannot block each other we need the following. The first time an obstacle is discovered by some wavefront, we call the point that the wavefront hits the obstacle the *discovery* point. (Note that there may be more than one such point. We arbitrarily choose one of these points.) In the north region, we split up the wavefronts adjacent to each obstacle into an *east wave* and a *west wave*. We call the set of all these wavefronts which are between the discovery point and the meeting point of the obstacle in a clockwise manner the west wave. We define the east wave of an obstacle respectively.

The discovery point of an obstacle b is always at the front of b. The wavefront that hits at b is split into two wavefronts, one of which is in the east wave and one of which is in the west wave of the obstacle. Such a scenario is shown in Figures 2-16 and 2-17. We claim that a descendent wavefront w_1 in the west wave and a descendant wavefront w_2 in the east wave cannot simultaneously block each other. Assume that the algorithm is trying to expand w_1 but that wavefront w_2 blocks w_1. Wavefront w_2 can only block w_1 if one of the following two cases applies. In both cases, we show that w_1 cannot also block w_2.

In the first case, w_1 is about to expand to the back of obstacle b, but both of the back corners of obstacle b have not been explored, and thus the meeting point has not been determined (e.g. 2-16). Wavefront w_2 can only be blocked by w_1 if w_2 is either already at the meeting point of the obstacle or about to expand to the back of the obstacle. Since none of the back corners of obstacle b have been explored, neither of these two possibilities holds. Thus, wavefront w_1 does not block w_2. (In Figure 2-16 w_2 is expanded next.)

In the second case, w_1 has reached the meeting point at the back of b (e.g. see Figure 2-17). Therefore, both back corners of the obstacle have been explored and w_1 is not blocking w_2.

We have just shown that if w_2 blocks w_1 then w_1 cannot also block w_2. Thus, the algorithm tries to pick w_2 as the nearest unblocked wavefront to w_1. However, w_2 may be blocked by its sibling wavefront w_3 on a different obstacle b'. For this case, we have to show that this sibling wavefront w_3 is not blocked, or that its sibling wavefront w_4 on yet another obstacle b'' is not blocked and so forth. Without loss of generality, we assume that the wavefronts are blocked by wavefronts towards the east. Proceeding towards the east along the chain of wavefronts will eventually lead to a wavefront which is not blocked: the east-most wavefront in the northern region. The east-most wavefront is adjacent to the initial monotone east-north path. Therefore, it cannot be blocked by a wavefront towards the east. □

Theorem 5 *The wavefront algorithm is an optimally interruptible piecemeal search algorithm for city-block graphs.*

Proof: To show the correctness of a piecemeal algorithm that uses our wavefront algorithm for exploration with interruption, we show that the wavefront algorithm maintains the shortest path property and explores the entire environment.

Theorem 3 shows by induction on shortest path length that the wavefront algorithm mimics breadth-first search. Thus it is optimally interruptible.

Theorem 4 shows that the algorithm does not terminate until all vertices have been explored. Completeness follows. □

2.8.4 Efficiency of the wavefront algorithm

In this section we show the number of edges traversed the wavefront algorithm is linear in the number of edges in the city-block graph. By the piecemeal theorem, we obtain a linear piecemeal exploration algorithm for city-block graphs.

The only times the robot traverses edges are when procedures EXPAND, and RELOCATE are called. In addition, it traverses edges to create the monotone paths and to get back to s between calls to EXPLORE-AREA. Thus, we count the number of edges traversed for each of these cases. In Lemmas 7 to 10, we analyze the number of edges traversed by the robot due to calls of RELOCATE. Theorem 6 uses these lemmas and calculates the total number of edges traversed by the wavefront algorithm.

Lemma 7 *An edge is traversed at most once due to relocations after a wavefront has expired (line 14 of* EXPLORE-AREA*).*

Proof: Assume that the robot is in the northern region and expanding wavefronts in a clockwise direction. Suppose wavefront w has just expired onto obstacle b (i.e., it is a single vertex with all of its adjacent edges explored). The robot now must relocate along obstacle b to its neighboring wavefront w' to the east. Since w' is also adjacent to obstacle b, the robot is only traversing edges on the obstacle b.

Note that at this point of exploration, there is no wavefront west of w which will expire onto obstacle b. This is because expiring wavefronts are never blocked, and thus the direction of expansion cannot be changed due to an expiring wavefront. So, when a wavefront is split, the robot always chooses the west-most wavefront to expand first. Thus, the wavefronts which expire onto obstacle b are explored in a west to east manner. For relocations after wavefronts have expired on obstacle b the robot always moves east along the boundary of this obstacle and only traversed each edge on the boundary once. □

Lemma 8 *An edge is traversed at most once due to relocations after wavefronts have merged (line 10 of* MERGE*).*

Proof: Assume that the robot is in the northern region and expanding wavefronts in a clockwise direction. Before a call to procedure MERGE, the robot is at the west-most end vertex of wavefront w. Note that wavefront w can be merged with at most two wavefronts, one at each end. If the resulting wavefront w_m is unblocked, procedure RELOCATE is called and the robot must traverse w_m to its west-most vertex. However, since wavefront w_m is unblocked, w_m can immediately be expanded and is not traversed again. □

Lemma 9 *At most one wavefront from the east wave of an obstacle is blocked by one or more wavefronts in the west wave. At most one wavefront from the west wave is blocked by one or more wavefronts in the east wave.*

Proof: Consider the west wave of an obstacle. By the definition of blocking, there are only two possible wavefronts in the west wave that can be blocked. One wavefront is adjacent to the back corner of the obstacle. Call this wavefront w_1. The other wavefront is adjacent to the meeting point of the obstacle. Call this wavefront w_2.

We first show that if w_1 is blocked then w_2 will not be blocked also. Then we also know that if w_2 is blocked then w_1 must not have been blocked. Thus at most one wavefront in the west wave is blocked.

If w_1 is blocked by one or more wavefronts in the east wave then these wavefronts can be expanded to the meeting point of the obstacle without interference from w_1. That is, wavefront w_1 cannot block any wavefront in the east wave, and thus there will be no traversals around the boundary of the obstacle until the east wave has reached the meeting point. At this point, the west wave can be expanded to the meeting point without any wavefronts in the east wave blocking any wavefronts in the west wave.

Similarly, we know that at most one wavefront from the west wave is blocked by one or more wavefronts in the east wave. □

Lemma 10 *An edge is traversed at most three times due to relocation after blockage (line 33 of* EXPLORE-AREA*).*

Proof: Without loss of generality, we assume that the wavefronts are blocked by wavefronts towards the east. Proceeding towards the east along the chain of wavefronts will eventually lead to a wavefront which is not blocked, since the east-most wavefront is adjacent to the initial monotone east-north path.

First we show that any wavefront is traversed at most once due to blockage. Then we show that the boundary of any obstacle is traversed at most twice due to blockage. Note that pairs of edges connecting vertices in a wavefront may also be edges which are on the boundaries of obstacles. Thus any edge is traversed at most three times due to relocation after blockage.

We know from Theorem 4 that there is always a wavefront that is not blocked. Assume that the robot is at a wavefront w which is blocked by a wavefront to its east. Following the chain of wavefronts to the east leads to an unblocked wavefront w'. This results in one traversal of the wavefronts. Now this wavefront w' is expanded until it is blocked by some wavefront w''. Note that wavefront w'' cannot be to the west of w', since we know that the wavefront west of w' is blocked by w'. (We have shown in the proof of Theorem 4 that if w_1 blocks w_2 then w_2 does not block w_1.) The robot will not move to any wavefronts west of wavefront w' until a descendant of w' no longer blocks the wavefront immediately to its west. Once this is the case, then the west wavefront can immediately be expanded. Similarly, we go back through the chain of wavefronts, since - as the robot proceeds west - it expands each wavefront in the chain. Thus the robot never traverses any wavefront more than once due to blockage.

Now we consider the number of traversals, due to blockage, of edges on the boundary of obstacles. As wavefronts expand, their descendant wavefronts may still be adjacent to the same obstacles. Thus, we need to make sure that the edges on the boundaries of obstacles are not traversed too often due to relocation because of blockage. That is, the robot does not move back and forth between wavefronts on different sides of an obstacle. Lemma 9 shown above implies that each edge on the boundary of the obstacle is traversed at most twice due to blockage.

Thus, since the edges on the boundary of an obstacle may be part of the pairs of edges connecting vertices in a wavefront, the total number of times any edge can be traversed due to blockage is at most three. □

Theorem 6 *The wavefront algorithm runs in time linear in the number of edges in the city-block graph.*

Proof: We show that the total number of edge traversals is no more than $15|E|$. Note that when the procedures CREATE-MONOTONE-PATHS, EXPAND, and RELOCATE are called, the robot traverses edges in the environment. In addition, the robot traverses edges in the environment to get back to s after exploration of each of the four regions. These are the only times the robot actually traverses edges in the environment. Thus, to calculate the total number of edge traversals, we count the edge traversals for each of these cases.

The robot traverses the edges on the monotone paths *once* when it explores them, and *once* to get back to the start point. This is clearly at most $2|E|$ edge traversals. The robot walks back to s four times after exploring each of the four regions. Thus the number of edges traversed here is at most $4|E|$. Lemma 6 implies that the total number of edge traversals caused by procedure EXPAND is at most $2|E|$. We now only need to consider the edge traversals due to calls to procedure RELOCATE.

Procedure RELOCATE is called four times within EXPLORE-AREA and MERGE.

The four calls are due to expansion (line 6 of Explore-area), expiring (line 14 of Explore-area), merging (line 10 of Merge) and blocking (line 33 of Explore-area). Relocations after expanding a wavefront results in a total of $|E|$ edge traversals. Lemma 7 shows that edges are traversed at most twice due to expiring wavefronts. Lemma 8 shows that edges are traversed at most once due to relocations after merges. Finally, Lemma 10 shows that edges are traversed at most three times due to relocations after blockage. Thus the total number of edge traversals due to calls of procedure Relocate is at most $7|E|$.

Thus the total number edges traversed by the wavefront algorithm is at most $15|E|$. A more careful analysis of the wavefront algorithm can improve the constant factor. □

Theorem 7 *A piecemeal algorithm based on the wavefront algorithm runs in time linear in the number of edges in the city-block graph.*

Proof: This follows immediately from Theorem 5 and Theorem 6. □

2.9 The ray algorithm

We now give another efficient optimally interruptible search algorithm, called the *ray algorithm*. The ray algorithm is a variant of DFS that always knows a shortest path back to s. This thus yields another efficient piecemeal algorithm for searching a city-block graph.

The ray algorithm also starts by finding the four monotone paths, and splitting the graph into four regions to be searched separately. The algorithm explores in a manner similar to depth-first search, with the following exceptions. Assume that it is exploring the northern region. The basic operation is to explore a northern-going "ray" as far as possible, and then to return to the start point of the ray. Along the way, side-excursions of one-step are made to ensure the traversal of east-west edges that touch the ray. Optimal interruptability is always maintained: the ray algorithm will not traverse a ray until it knows a shortest path to s from the base of the ray (and thus a shortest path to s from any point on the ray, by Lemma 2).

The ray algorithm is illustrated in Figure 2-19. From each point on the (horizontal segments of the) monotone paths bordering the northern region, a north-going ray is explored. On each such ray, exploration proceeds north until blocked by an obstacle or the boundary of the city-block graph. Then the robot backtracks to the beginning of the ray and starts exploring a neighboring ray. As described so far, each obstacle creates a "shadow region" of unexplored vertices to its north. These shadow regions are explored as follows. Once the two back corners of an obstacle are explored, the shortest paths to the vertices at the back of an obstacle are then known; the "meeting point" is then determined. Once the meeting point for an obstacle is known, the

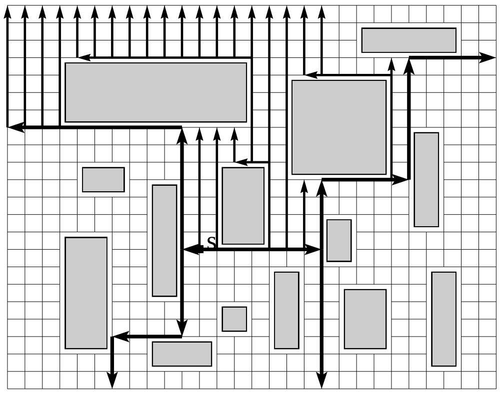

Figure 2-19: Operation of the ray algorithm.

shortest path from s to each vertex on the back border of the obstacle is known. The robot can then explore north-going rays starting at each vertex at the back border of the obstacle. There may be further obstacles that were all or partially in the shadow regions; their shadow regions are handled in the same manner.

We note that not all paths to s in the "search tree" defined by the ray algorithm are shortest paths; the tree path may go one way around an obstacle. Since the algorithm knows that the shortest path goes the other way around, it is nonetheless an optimally interruptible search algorithm.

Theorem 8 *The ray algorithm is a linear-time optimally interruptible search algorithm that can be transformed into a linear-time piecemeal search of a city-block graph.*

Proof: This follows from the properties of city-block graphs proved in Section 4, and the above discussion. In the ray algorithm each edge is traversed at most a constant number of times. The linearity of the corresponding piecemeal search algorithm then follows from Theorem 2. □

2.10 Piecemeal learning of arbitrary graphs

The piecemeal theorem (Theorem 2) states that an exploration algorithm in which the explorer always knows a short path back to the start vertex of length at most the radius of the graph can be interrupted for refueling without changing the running time of the algorithm by more than a constant factor (see Figure 2-20).

All the algorithms we present in this section are efficiently interruptible, and thus give piecemeal learning algorithms for undirected graphs. The main theorem we show for piecemeal learning is:

Theorem 9 *Piecemeal learning of a general undirected graph $G = (V, E)$ can be done in time $O(E + V^{1+o(1)})$.*

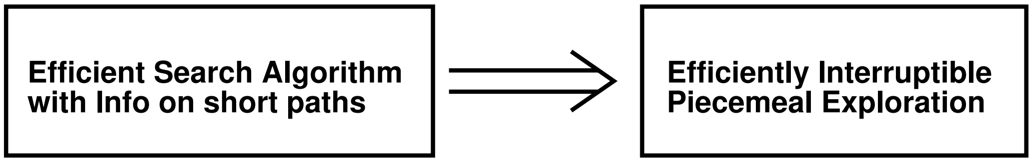

Figure 2-20: Illustration of the piecemeal theorem.

Proof: In the Recursive Strip algorithm, given in Section 2.10.4, the robot always knows a path from its current location back to the start vertex of length at most the radius of the graph. We show in Section 2.10.4 that the running time of this algorithm is $O(E + V2^{O(\sqrt{\log V \log\log V})})$. By Theorem 2, this algorithm can be interrupted efficiently to give a piecemeal learning algorithm. $\Box$

We need the following definitions in this section: A *layer* in a BFS tree consists of vertices that have the same shortest path distance to the start vertex. A *source vertex* is a root of a BFS tree. A *global BFS tree* is a BFS tree in which the root is the start vertex of the exploration. A *BFS forest* is a set of BFS trees. Two BFS trees in a BFS forest may be linked by a *connection edge*. A *frontier vertex* is a vertex that is incident to unexplored edges. A frontier vertex is *expanded* when the robot has traversed all the unexplored edges incident to it.

2.10.1 Bounded BFS

The traditional BFS algorithm expands frontier vertices layer by layer. As shown in Section 2.4.2 this algorithm runs in time $O(E + rV)$ in the teleport-free model. The procedure Bounded-BFS describes a version of the traditional BFS procedure that has been modified for our teleport-free BFS model in two respects. First, the robot does not relocate to frontier vertices that have no unexplored edges (see Figure 2-21). Second, it only explores vertices within a given distance-bound L of the given source vertex $\hat{s}$. (The first modification, while seemingly straightforward, is essential for the efficiency of our more complex algorithms that use Bounded-BFS as a subroutine at various source vertices). A procedure call of the form Bounded-BFS(s, r) with start vertex s and the radius of the graph r as input causes the robot to explore the entire graph.

Bounded-BFS can be used "locally" to explore a subtree of the graph's global BFS tree. This subtree has as a root source vertex $\hat{s}$ and extends to some depth L, a parameter to be chosen later. Bounded-BFS expands vertices layer by layer. At a given layer, it first determines all vertices at the same shortest path distance to the source vertex. Then for each such vertex it moves to it if the vertex has an incident edge that is unexplored.

In the following section we describe how Bounded-BFS can be used for efficient graph exploration.

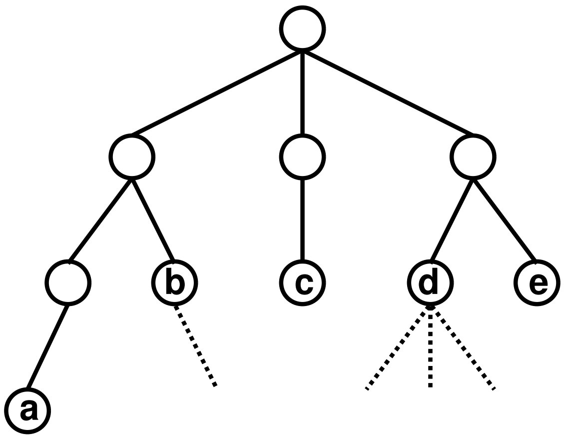

Figure 2-21: BOUNDED-BFS: Explored edges are shown as solid lines, unexplored edges as dashed lines. After the explorer has reached vertex a following BOUNDED-BFS, it moves to vertices b and d to expand them, but knows that it need not revisit vertices c and e.

BOUNDED-BFS($\hat{s}$, L)
for $i = 0$ **to** $L - 1$ **do**
    $V_i \leftarrow$ all vertices at distance i from $\hat{s}$
    for each $u \in V_i$ **do**
        if u has any incident unexplored edges
            then
                relocate to u
                traverse each unexplored edge incident to u
relocate to $\hat{s}$

2.10.2 An exploration algorithm: STRIP-EXPLORATION

This section describes an efficiently interruptible algorithm for undirected graphs with running time $O(E + V^{1.5})$. It is based on breadth-first search.

Awerbuch and Gallager [5, 6] give a distributed BFS algorithm which partitions the network in *strips*, where each strip is a group of L consecutive layers. (Here L is a parameter to be chosen.) All vertices in strip $i - 1$ are expanded before any vertices in strip i are expanded. Our algorithm, STRIP-EXPLORATION, uses the idea of search in strips in a new way.

Our goal is to find an efficient strategy for the robot to obtain shortest path information on-line during the exploration by building a global BFS tree without actually following BFS (since BFS is inefficient). We first describe a "naive strategy" on which our more efficient STRIP-EXPLORATION algorithm is based. This strategy is illustrated in Figure 2-22.

The robot explores the graph from s by building a global BFS tree with root s and depth r. It explores this tree in strips of length L. While the robot is exploring strip i, the global BFS tree only extends to strip $i - 1$: the robot knows shortest

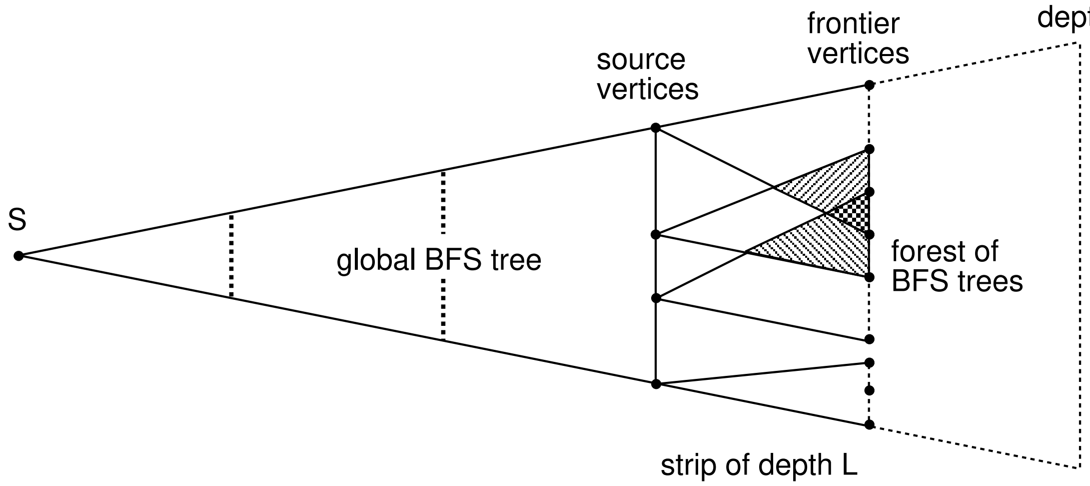

Figure 2-22: Strip-Exploration: The graph shown is explored by building a global BFS tree of depth r out of five strips. At the point shown during the exploration the global BFS tree extends to the end of the third strip, and the fourth strip has just been explored. The fourth strip consists of a forest of four BFS trees. In the naive algorithm, the shaded areas of the BFS forest are retraversed completely. In Strip-Exploration, the shaded areas are passed through more than once only if necessary to get to frontier vertices.

paths to s from any vertex in strips $1, \ldots, i-1$, but only "short paths" to s from the vertices in strip i. We show later how to determine parameter L so that these paths are indeed short enough to allow efficient interruptions due to the piecemeal constraint.

In the following we describe how the robot explores a strip. Assume there are k frontier vertices $v_1, \ldots, v_k$ of depth d_{i-1} when the exploration of strip $i-1$ is concluded. Each such vertex is included in the set of source vertices for strip i. The robot explores strip i by building a BFS tree of depth L from each source vertex $v_j, 1 \leq j \leq k$. At this point during the exploration, since the robot has a map of the global BFS tree up to depth d_{i-1} and a map of the BFS forest of depth L, it can combine both maps and create a map of the global BFS tree extending to depth $d_i = d_{i-1} + L$. Using this map the robot continues exploring the graph, strip by strip, until it has traversed every vertex and edge in the graph.

The naive algorithm described above is inefficient, due to the overlap between the trees in the BFS forest, causing portions of each strip to be repeatedly re-explored. (A forest of overlapping trees is illustrated in Figures 2-22 and 2-23.)

The procedure Strip-Exploration presented below solves this problem by using the Bounded-BFS procedure as the basic subroutine, instead of using a naive BFS. Following procedure Strip-Exploration, the robot searches in a breadth-first manner, but ignores previously explored territory. The only time the robot traverses edges which have been previously explored is when moving to a frontier vertex it is about to expand. This results in retraversal of some edges in previously explored

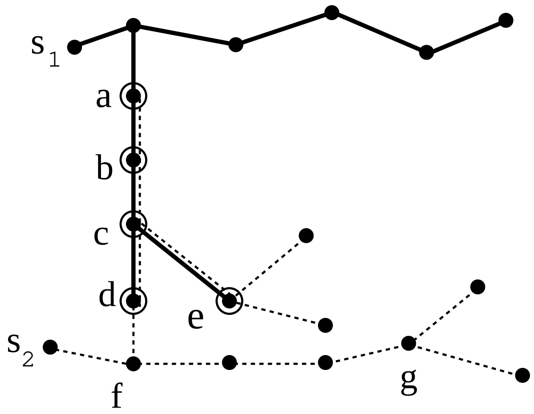

Figure 2-23: Contrasting BFS and BOUNDED-BFS: Consider doing a BFS of depth $L = 5$ from source vertex s_1 (solid line), followed by a BFS of depth $L = 5$ from s_2 (dashed line). After connection edge (f, d) is found during BFS$(s_2, 5)$, the only vertex in the BFS tree with root s_1 that is a frontier vertex and needs to be expanded is vertex e. However, BFS from s_2 revisits vertices a, b, c, d and e. On the other hand, BOUNDED-BFS from s_2 only revisits vertices d, c and e to expand e.

1. STRIP-EXPLORATION(s, r, L)
2. *number-of-strips* $\leftarrow \lceil r/L \rceil$
3. *sources* $\leftarrow \{s\}$
4. **for** $i = 1$ **to** *number-of-strips* **do**
5.   **for** each $u \in$ *set-of-sources* **do**
6.   Relocate to u
7.   BOUNDED-BFS(u, L)
8.   *set-of-sources* $\leftarrow$ all frontier vertices

territory, but not as many as in the naive algorithm.

STRIP-EXPLORATION takes as an input start vertex s, radius r, and strip length L. The number of strips to be explored is $\lceil r/L \rceil$. Initially, the only frontier vertex is the start vertex (line 3 of STRIP-EXPLORATION). It is the only source vertex for exploring the first strip. After the exploration of each strip the new source vertices are determined (line 8 of STRIP-EXPLORATION). The robot moves to each source (line 6) and follows a BOUNDED-BFS of depth L from there (line 7). Since all frontier vertices, i.e., vertices with unexplored edges, are expanded eventually during STRIP-EXPLORATION the graph is explored entirely.

If we choose L to be $\sqrt{r}$ [1], we have the following theorem:

Theorem 10 STRIP-EXPLORATION *is a piecemeal exploration algorithm that runs in time* $O(E + V^{1.5})$.

Proof:

[1]Without loss of generality, we assume here and in the following sections that $\sqrt{r}$ is an integer.

First we show that STRIP-EXPLORATION(s, r, L) runs in time $O(E + V^{1.5})$ for $L = \sqrt{r}$. The cost of STRIP-EXPLORATION is determined by how many edges the robot traverses. The robot moves when line 6 in STRIP-EXPLORATION and lines 7-9 of BOUNDED-BFS are called. First we count edge traversals due to line 6 of STRIP-EXPLORATION.

Assume the robot relocates between source vertices for a given strip i. For these relocations, the robot can mentally construct a tree connecting these vertices. This tree is a subtree of the global BFS tree explored so far. The robot can move between the source vertices by doing a depth-first traversal of this tree. Since this tree has at most $2V$ vertices, the number of edge traversals due to relocations between source vertices for this strip is at most $2V$. Since there are $\lceil r/L \rceil$ strips, the total number of edge traversals due to relocations between source vertices is at most $2rV/L + 2r$.

Now we count edge traversals for repeatedly executing the BOUNDED-BFS algorithm. First for the robot to expand all vertices and explore all edges it traverses $2E$ edges (line 8 of BOUNDED-BFS). Next, since the depth of a strip is L, each time line 2.10.1 of procedure BOUNDED-BFS is called, at most L edges are traversed. To account for relocations in line 7 of procedure BOUNDED-BFS, we use the following scheme for "charging" edge traversals. Say the robot is within a call of the BOUNDED-BFS algorithm. It has just expanded a vertex u and will now relocate to a vertex v to expand it. Vertex v is charged for the edges traversed to relocate from u to v. (We are only considering relocations within the same call of the BOUNDED-BFS algorithm; relocations between calls of the BOUNDED-BFS algorithm were considered above.) Source vertices are not charged anything. Moreover, the robot can always relocate from u to v by going from u to the source vertex of the current local BFS, and then to v, traversing at most $2L$ edges. Thus, each vertex is charged at most $2L$ when it is expanded. BOUNDED-BFS never relocates to a vertex v unless it can expand vertex v (i.e., unless v is adjacent to unexplored edges). Thus, all relocations are charged to the expansion of some vertex, and the total number of edge traversals due to relocation is at most $2LV$.

Thus the total number of edge traversals of STRIP-EXPLORATION is at most $2rV/L + 2r + 3LV + 2E$, which is $O(rV/L + r + LV)$. When L is chosen to be $\sqrt{r}$, this gives $O(E + V^{1.5})$ edge traversals.

When we use strip length $L = \sqrt{r}$, the robot always knows a path to s that is of length at most r. Therefore, it follows from Theorem 2 that STRIP-EXPLORATION can be interrupted efficiently and yields a piecemeal learning algorithm.

□

2.10.3 The iterative strip algorithm

In this section, we describe ITERATIVE-STRIP, an algorithm similar to the STRIP-EXPLORATION algorithm. It is an efficiently interruptible algorithm for undirected graphs inspired by Awerbuch and Gallager's [5] distributed iterative BFS algorithm.

Although its running time of $O((V^{1.5} + E) \log V)$ is worse than the running time of STRIP-EXPLORATION, its recursive version (described in the next section) is more efficient than STRIP-EXPLORATION. (It is not clear how to recursively implement STRIP-EXPLORATION as efficiently, because the search trees in a strip are not disjoint.)

Following the ITERATIVE-STRIP, the robot grows a global BFS tree with root s strip by strip, similar to STRIP-EXPLORATION. The length of each strip is $\sqrt{r}$. Unlike STRIP-EXPLORATION, each strip is processed several times before it has correctly deepened the global BFS tree by $\sqrt{r}$. We next explain the algorithm's behavior on a typical strip by describing how a strip is processed for the first time, and then for the remaining iterations.

In the first iteration, a strip is explored much as in STRIP-EXPLORATION. The robot explores a tree of depth $\sqrt{r}$ from each source vertex, by exploring in a breadth-first manner from each source vertex, but without retraversing previous trees. Whenever the robot finds a *connection* edge linking the current tree to another tree in the same strip, it does not penetrate into the other tree. Unlike STRIP-EXPLORATION, the robot does not traverse explored edges to get to the frontier vertices on other trees. Therefore, after the first iteration, the trees explored are *approximate BFS trees*, which may have frontier vertices with depth less than $\sqrt{r}$ from some source vertex. We called these vertices *active frontier vertices*. Frontier vertices are inactive if their depth is at least $\sqrt{r}$ from all source vertices in the strip. A scenario for the first iteration of a strip is illustrated in Figure 2-24.

After the first iteration, the current strip may not yet extend the global BFS tree by depth $\sqrt{r}$, so more iterations are needed until all frontier vertices are inactive and the global BFS tree is extended by depth $\sqrt{r}$.

To explore the rest of the strip efficiently, the robot uses the important property that two trees linked by a connection edge form a *connected component* within the strip (see Figure 2-25). The graph to be explored is connected, and thus forms one connected component; but we refer to connected components of the explored portion of the graph contained within the strip. Using the connections within the strip, the robot does not have to traverse any edges outside of the current strip to relocate between active frontier vertices in the same connected component.

In the second and later iterations (see Figure 2-26), the robot works on one connected component at a time. The robot explores active frontier vertices in a connected component as follows. It computes (mentally) a spanning tree of the vertices in the current strip. This spanning tree lies within the strip. The robot visits the vertices in the strip in an order determined by a DFS of the spanning tree. Let d be the least depth of any active frontier vertex in the component from a source vertex. As the robot visits active frontier vertices of depth d, it expands them. It then recomputes the spanning tree (since the component now has new vertices) and again traverses this tree, expanding vertices of the appropriate next depth d'. This process continues until the active frontier vertices become inactive, i.e., no active frontier vertex in the connected component has distance less than $\sqrt{r}$ from some source vertex in the

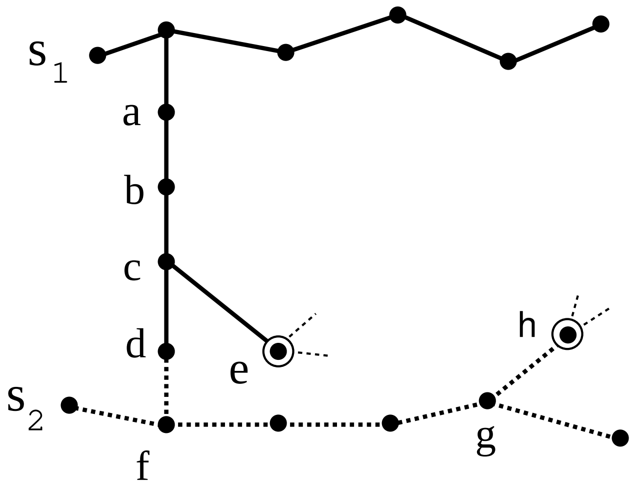

Figure 2-24: Illustrating the first iteration of the exploration of a strip in ITERATIVE-STRIP: First a BFS tree T_1 of length $L = 5$ from source s_1 is built (solid line). At this point, vertices d and e are active frontier vertices. While a tree T_2 of length $L = 5$ from s_2 is explored (dashed line) a connection edge (f, d) is found. At this point, edges in tree T_1 are *not* retraversed. At the end of the iteration, tree T_2 is an approximate BFS tree and the strip contains frontier vertices e and h. Frontier vertex e is active and will be expanded in the next iteration. Frontier vertex h is inactive and will be expanded when the next strip is explored (unless a shorter path to h is found during further exploration of the strip).

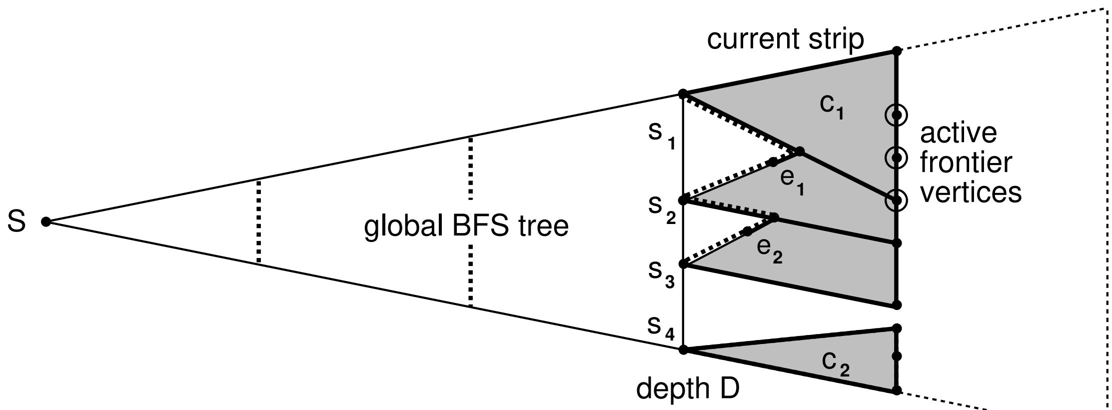

Figure 2-25: The ITERATIVE-STRIP algorithm after the first iteration on the fourth strip. Two connected components c_1, c_2 have been explored. The connection edges e_1 and e_2 link the first three approximate BFS trees. The dashed line shows how source vertices s_1, s_2, s_3 connect within the strip. There are three active frontier vertices with depth less than $D + \sqrt{r}$.

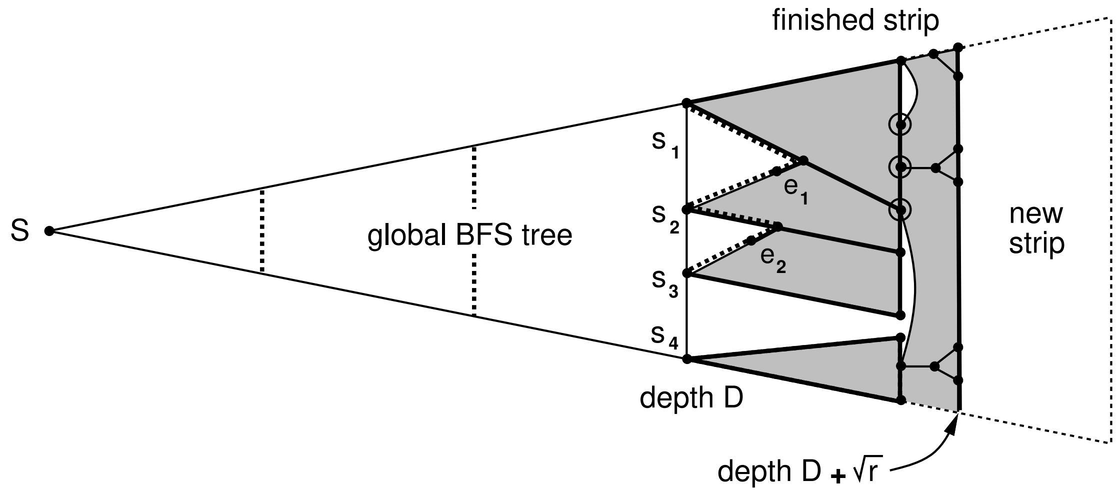

Figure 2-26: The ITERATIVE-STRIP algorithm after the second iteration. Now the circled vertices which were active frontier vertices at the beginning of the iteration are expanded. One of the expansions resulted in a connection edge. Now the strip consists of only one connected component (shaded area). There are six frontier vertices which become source vertices of the next strip. All frontier vertices have depth $D + \sqrt{r}$.

component.

In the next iteration the robot combines the components now connected by new connection edges and explores the new active frontier vertices in these components as described above. Lemma 11 states that at most $\log V$ iterations cause all frontier vertices in a strip to become inactive; then the strip is fully explored and these frontier vertices become source vertices of the next strip.

Procedure ITERATIVE-STRIP takes as an input the start vertex s and the radius of the graph. The first for-loop (line 2 of ITERATIVE-STRIP) splits up the search into $\sqrt{r}$ strips of length $\sqrt{r}$. In each strip a list of source vertices is kept (line 3). The robot moves to a source vertex (line 4) and explores the strip from there to depth $\sqrt{r}$ in a breath-first manner but without ever retraversing already explored edges (line 5). In the second and subsequent iteration of a strip (lines 6-13) a list of *active connected components* in the strip is maintained; a connected component is active if it contains active frontier vertices. If there are not any active component in a strip, the strip is fully explored (line 6). Otherwise, in each active component c the active frontier vertices are ordered by their depths in c (lines 9-10). The robot moves to each active frontier vertex following a spanning tree in c and expands it (line 11).

Lemma 11 *At most* $\log V$ *iterations per strip are needed to explore a strip and extend the global BFS tree by depth* $\sqrt{r}$.

Proof sketch: If there are initially l source vertices, then after the first iteration there are at most l connected components. If a connected component is not linked with another connected component during some iteration, then it does not contain

```
1.  ITERATIVE-STRIP(s, r)
2.  for i = 1 to √r do
3.      for each source vertex u in strip i do
4.          relocate to u
5.          BFS from u to depth √r, but do not enter previously explored territory
6.      while there are any active connected components do
7.          for each active connected component c do
8.              repeat
9.                  let v1, v2, v3, ... be active frontier vertices exclusively in c
10.                     with smallest depth among active frontier vertices in c
11.                 relocate to each of v1, v2, v3, ..., and expand
12.             until no more active frontier vertices exclusively in c
13.         determine new and active connected components
```

active frontier vertices to be expanded in the next iteration. If two components become connected during an iteration, they compose a new connected component and the number of connected components decreased. Thus, each iteration halves the number of components with active frontier vertices. After at most $\log V$ iterations there is no connected component with active frontier vertices left. The robot then has a complete map of the current strip and of the global BFS tree built in previous strips, so it can combine this information and extend the global BFS tree by depth $\sqrt{r}$. □

Theorem 11 ITERATIVE-STRIP *runs in time* $O((E + V^{1.5}) \log V)$.

Proof sketch: We first count the number of edge traversals per strip. Let V_i and E_i be the number of vertices and edges explored in strip i. For each active connected component, vertices of distance t from some source vertex are expanded by computing a spanning tree of the component, doing a DFS of the spanning tree, and expanding all vertices of distance t from some source vertex (lines $7 - 14$). At each iteration (line 6), active connected components are disjoint, so relocating to all vertices in the strip of distance exactly t takes at most $O(V_i)$ edge traversals. Thus, in one iteration, relocating to all vertices in the strip within distance $\sqrt{r}$ takes at most $O(\sqrt{r}V_i)$ edge traversals. Moreover, note that in order for the robot to expand each vertex, it traverses at most $O(E_i)$ edges. Thus, the total number of edge traversals for strip i is $O(E_i + \sqrt{r}V_i)$. Combining this with Lemma 11, and noting that $r \leq V$, proves the theorem. □

2.10.4 The recursive strip algorithm

In this section, we describe an efficiently interruptible algorithm called RECURSIVE-STRIP, which gives a piecemeal learning algorithm which traverses at most $O(E + V^{1+o(1)})$ edges. RECURSIVE-STRIP is the recursive version of ITERATIVE-STRIP; it provides a recursive structure which coordinates the exploration of strips, of approximate BFS trees, and of connected components in a different manner than ITERATIVE-STRIP. The robot still, however, builds a global BFS tree from start vertex s strip by strip. The robot expands vertices at the bottom level of recursion.

In the RECURSIVE-STRIP algorithm, the depth of each strip depends on the level of recursion (see Figure 2-27). If there are k levels of recursion, then the algorithm starts at the top level by splitting the search of G into V/d_{k-1} strips of depth d_{k-1}. Each of these is split into d_{k-1}/d_{k-2} searches of strips of depth d_{k-2}, etc. We have $V = d_k > d_{k-1} > \ldots > d_1 > d_0 = 1$.

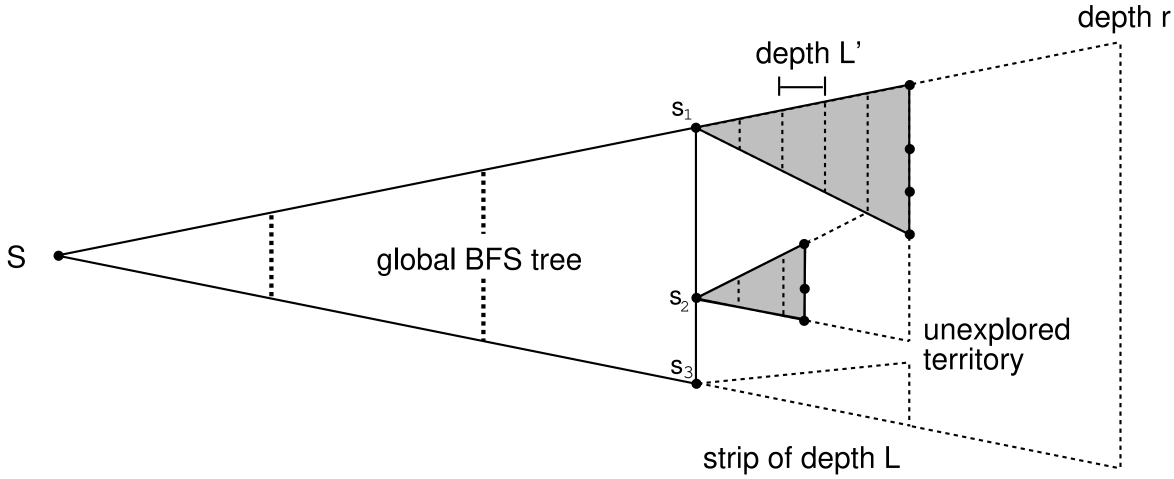

Figure 2-27: The RECURSIVE-STRIP algorithm processing an approximate BFS tree from source vertex s_2 to depth $d_{k-1} = L$. Recursive calls within the tree are of depth $d_{k-2} = L'$.

Each recursive call of the algorithm is passed a set of source vertices called *sources*, the *depth* to which it must search, and a set T of all vertices in the strip already known to be less than distance *depth* from one of the sources. The algorithm traverses all edges and visits all vertices within distance *depth* of the sources that have not yet been processed by other recursive calls at this level. RECURSIVE-STRIP($\{s\}, r, \{s\}$) is called to explore the entire graph.

At recursion level i, the algorithm divides the search into strips and processes each strip in turn, as follows. Suppose the strip has l source vertices $v_1, \ldots, v_l$. The strip is processed in at most $\log l = O(\log V)$ iterations. In each iteration, the algorithm partitions T into maximal sets $T_1, T_2, \ldots, T_k$ such that each set is known to be connected within the strip. Let S_c denote the source vertices in T_c. A DFS of the spanning tree of the vertices in T gives an order for the source vertices in $S_1, S_2, \ldots, S_k$; this spanning tree is used for efficient relocations between these source vertices. Note that all source vertices are known to be connected through the spanning tree of the vertices in T, but they might not be connected within the substrips. Since relocations between the vertices in S_c in the next level of recursion use a spanning tree of T_c, for efficiency the vertices of T_c must be connected within the strip. After partitioning the vertices into connected components within the strip, for each connected component T_c, the

```
1.  RECURSIVE-STRIP(sources, depth, T)
2.  if depth = 1
3.      then let v_1, v_2, ..., v_k be the depth-first ordering of sources in spanning tree
4.      for i = 1 to k do
5.          relocate to v_i
6.          if v_i has adjacent unexplored edges then traverse v_i's incident edges
7.      T = T ∪ {newly discovered vertices}
8.      return
9.  else  determine next depth
10.     number-of-strips ← depth/next-depth
11.     for i = 1 to number-of-strips do
12.         determine set of source vertices
13.         for j = 1 to number-of-iterations do
14.             partition vertices in T into maximal sets T_1, T_2, ..., T_k such that
15.                 vertices in each T_c are known to be connected within strip i
16.             for each T_c in suitable order do
17.                 let S_c be the source vertices in T_c
18.                 relocate to some source s ∈ S_c
19.                 RECURSIVE-STRIP(S_c, next-depth, T_c)
20.                 T = T ∪ T_c
21. relocate to some s ∈ sources
22. return
```

algorithm relocates (along a spanning tree) to some arbitrary source vertex in S_c. It then calls the algorithm recursively with S_c, the depth of the strip, and the vertices T_c which are connected to the sources S_c within the strip.

The remaining iterations in the strip combine the connected components until the strip is finished (line 13). Then the algorithm continues with the next strip in the same level of recursion (line 11), or, if it finished the last strip, it relocates to its starting position and returns to the next higher level of recursion (line 8).

Theorem 12 RECURSIVE-STRIP *runs in time* $O(E + V^{1+o(1)})$.

Proof sketch: First we observe that each vertex is expanded at most once, so there are at most $O(E+V)$ edge traversals due to exploration at line 6 in the bottom level of recursion.

We now count the edge traversals for relocations. For a particular level-i call of RECURSIVE-STRIP, let C_i denote the number of edge traversals due to relocations, and let E_i denote the number of distinct edges that are traversed due to relocation. Let V_i denote the number of vertices incident to these edges and whose incident edges are all known at the end of this call. We normalize the cost C_i at level i by the number of vertices V_i and use an upper bound ρ_i on C_i/V_i. Thus, if the depth of recursion is m then the total number of edge traversals is bounded by $O(V\rho_m)$.

Consider a level-i call. First we count the number of edge traversals for relocation between source vertices. Since all the source vertices in the call are connected by a tree of size $O(V_i)$, relocating to all source vertices at the start of one strip takes $O(V_i)$ edge traversals. With d_i/d_{i-1} strips and $\log V$ iterations per strip, there are

$$V_i \log V \frac{d_i}{d_{i-1}}$$

edge traversals for relocations between source vertices.

Second we count traversals for recursive calls within the call. Note that our algorithm avoids re-exploring previously explored edges. Thus, for a level-i call, when working on a particular strip l, for each iteration within this strip, the sets of vertices whose edges are explored in each recursive call are disjoint. Suppose that, in this strip, in one iteration the procedure makes m recursive calls, each at level $i-1$. Then let $C_{i-1}^{(j)}$, $1 \leq j \leq m$, denote the number of edge traversals due to relocations resulting from the j-th recursive call, and let $V_{i-1}^{(j)}$ denote the number of vertices adjacent to these edges. Furthermore, let $V_{l,i}$ denote the number of vertices which are in strip l of this procedure call at recursion level i. Then we would like first to calculate

$$\sum_{j=1}^{m} C_{i-1}^{(j)}$$

which is the number of edge traversals due to relocation in recursive calls in one iteration within this strip. This is at most

$$\sum_{j=1}^{m} \rho_{i-1} V_{i-1}^{(j)} = \rho_{i-1} \sum_{j=1}^{m} V_{i-1}^{(j)}.$$

Since the recursive calls are disjoint, $\sum_{j=1}^{m} V_{i-1}^{(j)} = V_{l,i}$. Thus the number of edge traversals due to relocations in recursive calls in *one* iteration within this strip is at most $\rho_{i-1} V_{l,i}$. Finally, since there are $\log V$ iterations in each strip, and all strips are disjoint from each other, the number of edge traversals due to recursive calls is at most

$$\rho_{i-1} V_i \log V.$$

Third, note that we relocate once at the end of each procedure call of Recursive-Strip (see line 21). This results in at most V_i edge traversals.

The costs of the three cases described above are additive and thus, the number of edge traversals due to relocation is described by the recurrence

$$C_i \leq V_i \log V \frac{d_i}{d_{i-1}} + \rho_{i-1} V_i \log V + V_i.$$

Normalizing the recurrence by the set of vertices V_i, we get the following recurrence:

$$\rho_i \leq \left(\frac{d_i}{d_{i-1}} + \rho_{i-1} \right) \log V + O(1).$$

Solving the recurrence for ρ_m gives:

$$\begin{aligned}
\rho_m &\leq \log V\left(\frac{d_m}{d_{m-1}}\right) + \log^2 V\left(\frac{d_{m-1}}{d_{m-2}}\right) + \ldots + \log^m V\left(\frac{d_1}{d_0}\right) + \log^m V c_0 + \sum_{i=0}^{m-1} \log^i V \\
&\leq \log V\left(\frac{d_m}{d_{m-1}}\right) + \log^2 V\left(\frac{d_{m-1}}{d_{m-2}}\right) + \ldots + \log^m V\left(\frac{d_1}{d_0}\right) + O(\log^m V) \\
&\leq \sum_{i=1}^{m} \log^i V\left(\frac{d_{m+1-i}}{d_{m-i}}\right) + O(\log^m V)
\end{aligned}$$

We note that $\rho_0 = O(1)$, since at the bottom level, if there are V' vertices expanded, then the number of edge traversals due to relocation is $O(V')$. Furthermore, notice that the product of the m terms in the recurrence is

$$\prod_{i=1}^{m} \log^i V\left(\frac{d_{m+1-i}}{d_{m-i}}\right) = d_m (\log V)^{\frac{(m+1)m}{2}} = r(\log V)^{\frac{(m+1)m}{2}}$$

(in the worst case), and that the sum of these terms is minimized by setting each of theses terms to the m-th root of the product. (Note that this also specifies how to calculate depth d_{i-1} from depth d_i.) Minimizing, we get:

$$\begin{aligned}
\rho_m &\leq m\, r^{1/m} (\log V)^{(m+1)/2} + O(\log^m V) \\
&\leq 2^{\log m} \cdot 2^{\frac{\log r}{m}} \cdot 2^{\frac{m+1}{2}\log\log V} + O(2^{m \log\log V})
\end{aligned}$$

Choosing $m = \sqrt{\frac{\log V}{\log\log V}}$ gives us

$$\begin{aligned}
\rho_m &\leq 2^{\log\sqrt{\log V} - \log\sqrt{\log\log V} + \frac{\sqrt{\log\log V}}{\sqrt{\log V}}\log V + (\frac{\sqrt{\log V}}{\sqrt{\log\log V}}+1)\frac{1}{2}\log\log V} + O(2^{\frac{\sqrt{\log V}}{\sqrt{\log\log V}}\log\log V}) \\
&\leq 2^{\log\log V - \frac{1}{2}\log\log\log V + \frac{3}{2}\sqrt{\log V \log\log V}} + O(2^{\sqrt{\log V \log\log V}}) \\
&= 2^{O(\sqrt{\log V \log\log V})}
\end{aligned}$$

and thus

$$C_m = V 2^{O(\sqrt{\log V \log\log V})}$$

which is $V^{1+o(1)}$. Adding the edge traversals for relocation to the edge traversals for exploration gives us $O(E + V^{1+o(1)})$ edge traversals total. □

2.10.5 Open problems

We have presented an efficient $O(E + V^{1+o(1)})$ algorithm for piecemeal learning of arbitrary, undirected graphs. The only lower bound known for this problem is the trivial linear bound $\Omega(E+V)$. It is open whether there is a linear-time algorithm for piecemeal learning of general graphs.

Chapter 3

Mobile Robot Position Estimation using Landmarks

3.1 Introduction

This chapter describes an efficient algorithm for localizing a mobile robot in an environment with landmarks. The robot has sensors that both identify landmarks and measure their bearings relative to each other. Such sensor information is generally uncertain and contains noise. Given the positions of possibly misidentified landmarks on a 2-D map of the environment and noisy measurements of their bearings relative to each other, the algorithm estimates the robot's position with respect to the map of the environment. The algorithm makes efficient use of the geometry of the problem; specifically, the representation of the landmarks by complex numbers. The algorithm runs in time linear in the number of landmarks. Results of simulations are presented that explore the strength of the algorithm.

Why is mobile robot localization important? A robot cannot accurately execute its commands. As a mobile robot moves through its environment, its actual position and orientation always differs from the position and orientation that it is commanded to hold. Wheel slippage is a major source of error. The errors accumulate and the location uncertainty increases over time. Dead-reckoning is not sufficient to locate the robot. Therefore, sensory feedback is needed to locate the robot in its environment.

Consider an autonomous agent, which could be a mobile robot or a human traveler, who uses a map to navigate through an environment that contains landmarks. The landmarks are marked on the agent's map. The autonomous agent also has a tool which can measure angles. The agent may use the following algorithm to identify its location in the environment:

1. Identify surrounding landmarks in the environment.

2. Find the corresponding landmarks on the map.

3. Measure the bearings of the landmarks relative to each other.

4. Compute your position efficiently.

If the first three steps can be executed without errors three landmarks are sufficient to compute the position of the agent, unless the agent's position and these three landmarks either form a circle, or lie on one line. Then the localization problem does not have a unique solution.

However, in real life an agent makes two kinds of mistakes: (1) some angles are measured with small errors and (2) some landmarks are misidentified. Another source of mistakes could be errors in the map itself. Suppose that the majority of mistakes are of the first type. In this situation we address the following problems:

- Estimating the position and orientation of the agent efficiently.
- Finding misidentified landmarks and large angle measurement errors.

This chapter presents an algorithm that estimates the agent's position in $O(n)$ operations where n is the number of landmarks on the 2-D map. Large errors due to misidentified landmarks and erroneous angle measurements can be found, discarded and the algorithm can be repeated without them with improved results.

Our work is motivated by a mobile robot called Ratbot that was built at the Learning Systems Department at Siemens Corporate Research. Ratbot is used as a test bed for various machine learning approaches to robot navigation. Ratbot navigates through the corridors of the building at Siemens Corporate Research. It is equipped with a camera that points upwards onto a reflective ball which acts like a mirror of the surroundings (see Figure 3-1). Straight lines of objects like pictures, doors, and walls look like arcs in the images taken by Ratbot's camera. The camera setup is due to Judd [61]. Only a circular, one-dimensional strip of the brightness of the image is analyzed. Landmarks like doors, pictures, and fire extinguishers appear as dark bands on a strip. The strips provide information on the bearing of one landmark relative to another landmark, but not on the distance of the landmarks to the camera (i.e., depth information).

To find the corresponding features of an image that is taken during navigation and an image that is taken in advance and stored in a database, only the strips of the two images need to be compared. Therefore, the correspondence problem, i.e., the problem of identifying features in two images that are the projections of the same object in the environment, is much easier for pictures taken by Ratbot. Only one-dimensional strips of pixel values of two pictures need to be compared. Hancock and Judd provide an algorithm that finds matching features in two image strips [54].

Once corresponding landmarks are identified, a description of the matching landmarks within the global map of the robot's environment is given. The map of the environment is a two-dimensional description of the office building through which the robot navigates. The position of the robot with respect to this map is unknown and needs to be determined.

Our position estimation algorithm has been used by Greiner and Isukapalli [50] to determine which of the landmarks that are visible to Ratbot are useful for its

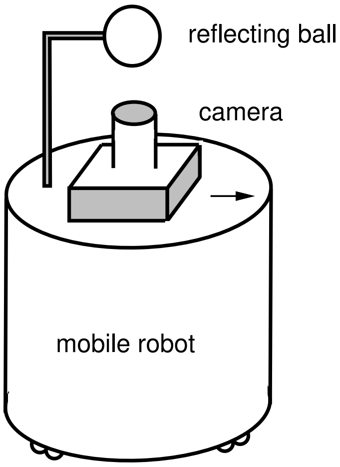

Figure 3-1: Illustration of a robot with a camera set-up that provides "fish-eyed" circular images of the surroundings.

position estimate. However, our algorithm is not restricted to robots that have the same geometry as Ratbot's camera system. Our techniques are general and can be used for other camera setups. Our algorithm could be used not only for a robot moving in an office building, but also for a robot moving in an assembly plant, or for an autonomous vehicle in a city.

The chapter is organized as follows. Section 3.2 describes work related to our problem. Section 3.3 gives a formal description of our problem. Section 3.4 discusses how the robot's position can be estimated using a triangulation approach. Section 3.5 describes our position estimation algorithm which uses a different approach that represents landmarks with complex numbers. Section 3.6 proposes how to incorporate our algorithm in an on-line navigation algorithm that estimates the robot's position while the robot is moving.

3.2 Related work

In recent years several authors have used triangulation approaches for mobile robot localization. Triangulation is a well-known technique for finding a position by means of bearings from fixed points in the environment.

Sugihara [99] addresses a version of the localization problem where the points in the environment look identical (to the contrary, we use landmarks that are distinguishable). The robot observes k such points in an environment that contains n points. It measures the direction of k rays each of which emanates from the robots position and "pierces" through an observed point.

Sugihara gives an $O(n^3 \lg n)$ algorithm for finding the robot's position and orientation such that each ray pierces at least one of the n points in the environment. His algorithm is an improved version of a naive approach that works as follows:

Repeat until the position found:

1. Choose four rays and four points in the environment.
2. Compute a unique candidate position using triangulation with three rays and three points.
3. Check the consistency of the candidate position using the fourth ray.

Since there are $n(n-1)(n-2)(n-3)$ permutations of a set of four points, the naive algorithm takes $O(n^4)$ time.

Avis and Imai [2] address the same problem as Sugihara and describe an $O(kn^3)$ algorithm. They also give an $O(k^2n^3)$ version of their algorithm for the case that angle measurements have errors (they assume worst-case error bounds). The work of both Sugihara and Avis/Imai is of a theoretical nature.

Krotkov [68] uses Sugihara's $O(n^4)$ algorithm for experiments with angle measurement errors. He runs simulations for which the angle measurements are uniformly distributed on an interval of width 2 degrees. He reports "approximately correct" solutions for robot position and orientation.

Sutherland and Thompson [100] address the localization problem using *distinguishable* landmarks in the environment (as we do). They also assume worst-case error bounds on angle measurements. For a given error range, they compute the "area of uncertainty" that is the area in which the robot may self-locate. For any given error in angle measurement, the size of the localization error varies depending on the configuration of landmarks. Sutherland and Thompson give an algorithm for choosing landmarks that reduces the area of uncertainty. The algorithm averages three position estimates, each is obtained by triangulation using three different landmarks. They find that configurations that are close to a configuration where landmarks and robot are located on a single circle should be avoided. They describe results of simulations on real topographic map data.

None of the work described above uses trajectory integration or dead-reckoning as means to support position estimation. We take the same approach, since the method we develop does not depend on the robot's history of commands. The remainder of this section describes work in which the history of commands is used.

There is a wide variety of approaches to handle uncertainties associated with a mobile robot. Some approaches differ from our approach in the way the robot's environment is represented. We use a two-dimensional metric map to describe the environment of the robot.

Brooks [23] proposes a method to model and reduce uncertainties in the position of a moving robot that uses both topological and metric representations of the environment. Relationships between metrically represented regions of the environment are represented in a topological map. He calls such regions "uncertainty manifolds." They bound the robot's location estimate in space.

Mataric [82] describes a physical robot for environment-learning and navigation that uses a landmark-based topological representation of the environment. Elfes [43] describes a navigation system that consists of several map-building levels: it builds symbolic maps, geometric maps, and sonar sensor maps.

A "multi-level navigation system" designed at Advanced Decision Systems uses both metrical and topological maps [76, 75, 69]. The system incorporates a triangulation method for landmark navigation and has been tested in simulations.

Chatila and Laumond [28] and Smith and Cheeseman [97] propose to use Gaussian distributions to represent and estimate positional uncertainty. Several authors follow such a statistical approach for robot navigation, for example, Kosaka and Kak [67], Crowley [38], Leonard and Durrant-Whyte [74], Watanabe and Yuta [102], Burlina, DeMenthon and Davis [27] and Matthies and Shafer [83]. We show empirically that the distribution of the position estimate that our method computes is normal if certain assumptions on the error in the input data hold.

Approaches to handle uncertainties in a robot's position can also be distinguished by the kind of data available to the robot. Our method uses visual input. For example, Drumheller [41] proposes a method to determine the position and orientation of a mobile robot inside a room using sonar data. Chenavier and Crowley [29] estimate the robot's location by combining measurements of landmark bearings with odometry data. Cooper and Durrant-Whyte [34] demonstrate the limitations of GPS and show how to incorporate GPS position measurements into an outdoor navigation system. Kleeman [64] uses ultrasonic beacons for mobile robot localization.

We address mobile robot localization as a problem of relating robot-centered information with global map information. Relating a camera-centered coordinate system to an external coordinate system is called *exterior orientation* in the field of photogrammetry (see for example [56]). Photogrammetry deals with the problem of how objects in the environment are related to their images in camera systems with different orientations and positions. For robot manipulators, the problem of relating a camera-centered coordinate system to a world-centered coordinate system is called the *hand-eye transformation*. The camera information (eye) is used to guide a robot arm or a mobile robot (hand) [36]. Lozano-Pérez, Mason, and Taylor [78] develop a method to eliminate position uncertainty (without representing it). The method is based on the geometry of the robot's task.

3.3 The position estimation problem

In this section we define the problem of estimating a robot's position and orientation in its environment given a global map of the environment and bearings of landmarks measured relative to each other at the robot's position.

We call the coordinate system of the map of the environment the *external* coordinate system. It is spanned by axes $x^{(e)}$ and $y^{(e)}$. We distinguish it from the *robot-centered* coordinate system spanned by axes $x^{(r)}$ and $y^{(r)}$. A landmark z can be described in both coordinate systems. Vector $z^{(e)}$ describes a landmark in the external coordinate system (see Figure 3-2, left). Vector $z^{(r)}$ describes a landmark in the robot-centered coordinate system (see Figure 3-2, right).

The robot's *position* is described by vector $p = (p_x, p_y)$ in the external coordinate system. Vector p links the origins of both coordinate systems. The robot is oriented in direction of axis $x^{(r)}$. The robot's *orientation* is described by angle θ between the external coordinate axis $x^{(e)}$ and the robot-centered coordinate axis $x^{(r)}$. Figure 3-3 illustrates how an external and a robot-centered coordinate system are linked given position vector p and orientation angle θ.

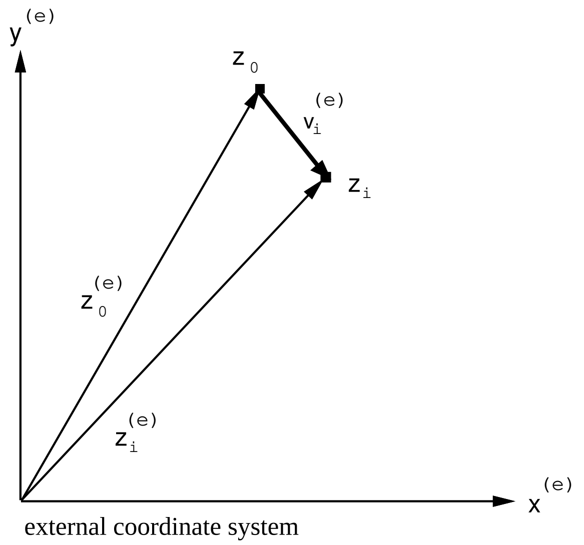

Figure 3-2: On the left: Environment with external coordinate system $(x^{(e)}, y^{(e)})$. Two landmarks, z_0 and z_i, are shown. On the right: Environment with robot coordinate system $(x^{(r)}, y^{(r)})$. The robot's position is illustrated by a black circled disk. The robot is orientated in the direction of the coordinate axis $x^{(r)}$.

Having a map of the robot's environment means that the vectors $z_0^{(e)}, \ldots z_n^{(e)}$ in the external coordinate system are given. The robot measures the bearing of each landmark relative to the direction in which the robot is oriented: Angle τ_i is the angle subtended by landmark vector $z_i^{(r)}$ and axis $x^{(r)}$ for $i = 0, \ldots, n$:

$$\tau_i = \angle(z_i^{(r)}, x^{(r)}).$$

We call the angle obtained from measuring the bearing of one landmark relative to another landmark the "visual angle" between the two landmarks [100].

The robot does not have a way to measure depth, i.e., it does not know its distance $|z_i^{(r)}|$ to the i-th landmark z_i for all i.

Note that if the orientation angle θ is zero then we can express the vector between two landmarks z_0 and z_i as vector $v_i^{(r)} = z_0^{(r)} - z_i^{(r)} = v_i^{(e)} = z_0^{(e)} - z_i^{(e)}$ (see Figure 3-3).

Now we can formulate our problem: Given the external positions $z_0^{(e)}, \ldots, z_n^{(e)}$ of $n+1$ landmarks and corresponding angle measurements $\tau_0, \ldots \tau_n$, estimate the position p and the orientation θ of the robot in the environment.

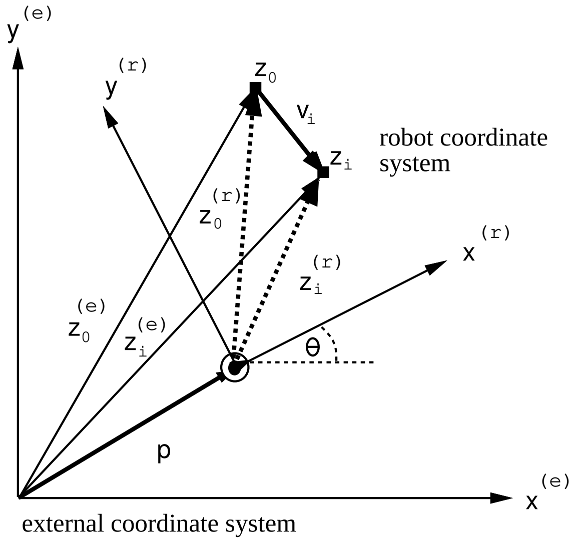

Figure 3-3: Environment with external and robot coordinate system: Both vector $z_i^{(e)}$ in the external coordinate system and vector $z_i^{(r)}$ in the robot coordinate system point to landmark z_i. The robot's location is defined by position vector p and orientation angle θ.

3.4 A triangulation approach

This section shows how to apply well-known triangulation techniques to the robot localization problem. First we consider the localization problem for perfect data, then for noisy data.

3.4.1 Triangulation with perfect data

If the visual angle φ between two landmarks z_0 and z_1 measured at an unknown position p and the distance between the landmarks is known, then position p lies on an arc of a circle spanned by landmarks z_0 and z_1 [47]. (See Figure 3-4 (a).)

A third landmark z_2 is needed to determine position p uniquely: it is the intersection of the circle through landmarks z_0 and z_1 and the circle through landmarks z_1 and z_2 (see Figure 3-4 (b)). Note that p cannot be determined uniquely if landmarks z_0, z_1 and z_2 lie on a circle.

One approach to compute position p is to describe the two circles analytically and determine their intersection. The law of cosine can be used to compute the radius of each circle given the visual angles and distances between the landmarks. The coordinates of the center of the circles can then be computed from the coordinates of landmarks z_0, z_1 and z_2. Once vector p is known, the orientation of the robot, i.e., the direction of $x^{(r)}$ is known.

Another way of triangulating position p is to determine the distance between the robot and the three landmarks. We follow this approach in the next section.

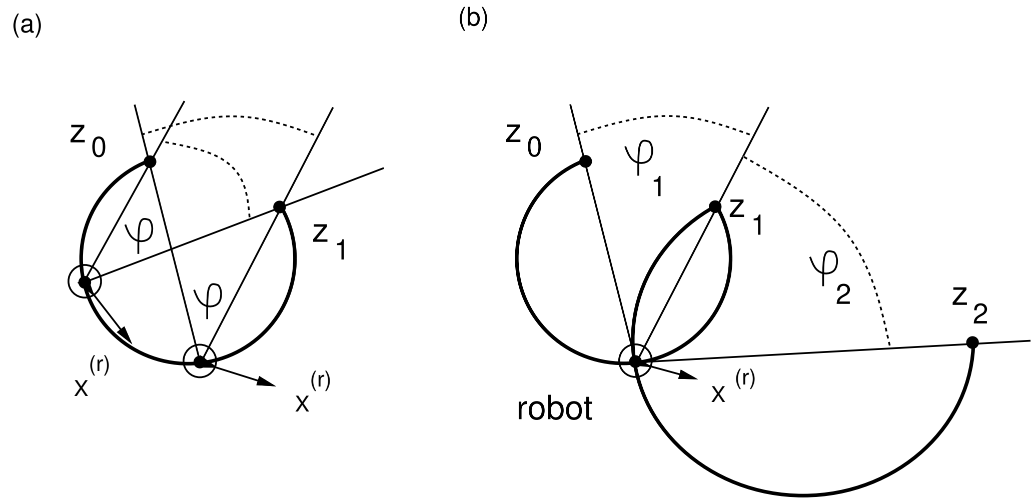

Figure 3-4: Triangulation with perfect data: (a) Scenario with two landmarks: The robot's position is somewhere on the arc shown. The robot is oriented in the direction of $x^{(r)}$. (b) Scenario with three landmarks: The robot's position is at the point of intersection of the two arcs shown. The robot's orientation can be uniquely determined from $x^{(r)}$.

3.4.2 Triangulation with noisy data

The triangulation method described above does not compute the exact position if the input data is noisy. Using several landmarks yields a more robust position estimate. Given n landmarks, there are $\binom{n}{3}$ combinations of three landmarks that can be used to compute $\binom{n}{3}$ position estimates by the triangulation method described above. One way of combining these estimates to obtain a final position estimate is to compute their average. (Other procedures for combining these estimates exist.)

In the following we outline a different triangulation method that uses n landmarks and is based on estimating the distance between the robot and the landmarks. (Note that this is not the method that we propose in this chapter).

Applying the law of cosine the relationship between two landmarks z_i and z_j can be written as

$$|v_{ij}|^2 = |z_i^{(r)}|^2 + |z_j^{(r)}|^2 - 2|z_i^{(r)}||z_j^{(r)}| \cos\varphi_{ij}$$

where $|v_{ij}|$ is the (known) distance between z_i and z_j, φ_{ij} is the visual angle between z_i and z_j at the robot's position, and $|z_i^{(r)}|$ and $|z_j^{(r)}|$ are the unknown distances to the landmarks (see Figure 3-5). Angle φ_{ij} is computed from measured angles τ_i and τ_j: $\varphi_{ij} = \tau_i - \tau_j$.

We can apply the law of cosine to all possible sets of two landmarks and get a system of equations which is overdetermined and can be solved using a least squares approach. Once the lengths $|z_i^{(r)}|, \ldots, |z_n^{(r)}|$ are found, another overdetermined set of equations needs to be solved to find the position $p = (p_x, p_y)$ of the robot: $|z_i^{(r)}|^2 =$

$(x_i - p_x)^2 + (y_i - p_y)^2$ for $i = 0, \ldots, n$ where x_i and y_i are the coordinates of $z_i^{(r)}$. Once vector p is estimated, we can get an estimate for the orientation angle θ by computing vector $z_i^{(r)} = z_i^{(e)} - p$ for some i and then $\theta = \angle(z_i^{(r)}, x^{(e)}) - \tau_i$.

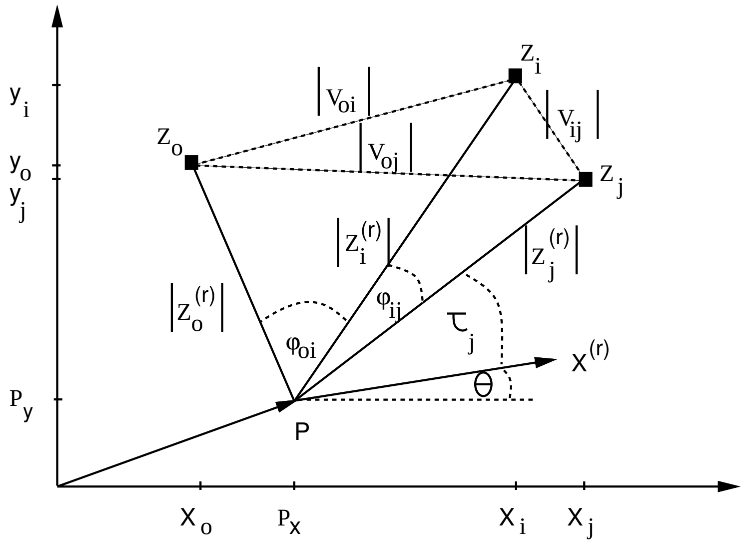

Figure 3-5: Position estimation using the law of cosine.

In the following we illustrate the method with three landmarks z_0, z_i and z_j (see also Figure 3-5). The law of cosine gives us

$$
\begin{aligned}
|v_{0i}|^2 &= |z_0^{(r)}|^2 + |z_i^{(r)}|^2 - 2|z_0^{(r)}||z_i^{(r)}| \cos\varphi_{i0} \\
|v_{0j}|^2 &= |z_0^{(r)}|^2 + |z_j^{(r)}|^2 - 2|z_0^{(r)}||z_j^{(r)}| \cos\varphi_{j0} \\
|v_{ij}|^2 &= |z_i^{(r)}|^2 + |z_j^{(r)}|^2 - 2|z_i^{(r)}||z_j^{(r)}| \cos\varphi_{ij}
\end{aligned}
$$

First we could solve this system of nonlinear equations using a least squares method. (A review of the least squares method for solving linear equations is given in Section 3.5.1.) Then we could express the now known lengths $|z_0^{(r)}|, |z_i^{(r)}|$ and $|z_j^{(r)}|$ by the (unknown) coordinates of the position $p = (p_x, p_y)$ of the robot and the coordinates (x, y) of the landmarks:

$$
\begin{aligned}
|z_0^{(r)}|^2 &= (x_0 - p_x)^2 + (y_0 - p_y)^2 \\
|z_i^{(r)}|^2 &= (x_i - p_x)^2 + (y_i - p_y)^2 \\
|z_j^{(r)}|^2 &= (x_j - p_x)^2 + (y_j - p_y)^2
\end{aligned}
$$

Subtracting the second and third equation from the first we get two equations linear in the unknowns p_x and p_y.

$$
\begin{aligned}
|z_0^{(r)}|^2 - |z_i^{(r)}|^2 &= x_0^2 - x_i^2 + 2p_x(x_i - x_0) + y_0^2 - y_i^2 + 2p_y(y_i - y_0) \\
|z_0^{(r)}|^2 - |z_j^{(r)}|^2 &= x_0^2 - x_j^2 + 2p_x(x_j - x_0) + y_0^2 - y_j^2 + 2p_y(y_j - y_0)
\end{aligned}
$$

Finally, we can solve for p_x and p_y and obtain an estimate for the position $p = (p_x, p_y)$.

Triangulation with noisy data as described above is based on solving nonlinear equations with complicated closed-form solutions. However, standard algorithms that provide least-squares solutions for large numbers of nonlinear equations take too long for real-time robot navigation. In the following section, we introduce a new *linear-time* approach to localizing a mobile robot which is based on a least-squares solution of a linear set of equations.

3.5 Linear position estimation

This section describes an efficient localization algorithm that runs in time linear in the number of landmarks. Instead of using the triangulation method described above, we use a method based on the complex number representation of the landmarks. This representation is the key idea to get a set of linear equations whose solution is a set of position estimates. (Triangulation methods may also provide a set of position estimates but as a solution to nonlinear equations). We describe this set of m linear equations as a vector equation. To solve this particular vector equation, we do not need an $O(m^3)$ matrix inversion algorithm. Instead we introduce an $O(m)$ algorithm which takes advantage of the special structure of the matrix and the right side of the vector equation. In the following, we describe how to set up and solve this vector equation.

A landmark z can be written as a complex number in the robot-centered coordinate system. For each landmark $z_i^{(r)}$ we have an expression

$$z_i^{(r)} = l_i \, e^{\,j\tau_i} \qquad\qquad \text{for } i = 1, \ldots, n$$

where length l_i is the unknown distance of the robot to landmark $z_i^{(r)}$, angle τ_i is the measured angle between $z_i^{(r)}$ and the coordinate axis $x^{(r)}$, and $j = \sqrt{-1}$.

We pick a reference landmark $z_0^{(r)}$ and compute the angles $\varphi_i = \tau_i - \tau_0$ for $i = 1, \ldots, n$. (Landmark $z_0^{(r)}$ may be chosen to be the most reliable landmark if such reliability information is available.) Dividing the complex number representation of landmarks $z_1^{(r)}, \ldots, z_n^{(r)}$ by $z_0^{(r)}$ for $i = 1, \ldots, n$ yields a set of equations that includes the visual angles φ_i between landmark z_i and landmark z_0:

$$\frac{z_i^{(r)}}{z_0^{(r)}} = \frac{l_i}{l_0} \, e^{\,j(\tau_i - \tau_0)} = \frac{l_i}{l_0} \, e^{\,j\varphi_i}. \tag{3.1}$$

Given that $z_i^{(r)} = z_0^{(r)} + v_i^{(r)}$, Equation (3.1) becomes

$$\frac{z_0^{(r)} + v_i^{(r)}}{z_0^{(r)}} = \frac{l_i}{l_0} \, e^{\,j\varphi_i} \qquad\qquad \text{for } i = 1, \ldots, n.$$

After some algebra we obtain a set of equations whose unknowns are vectors $z_0^{(r)}$ and $v_i^{(r)}$ and ratios l_i/l_0, for $i = 1, \ldots, n$:

$$\frac{1}{z_0^{(r)}} = \frac{l_i}{l_0} \frac{1}{v_i^{(r)}} \, e^{\, j\varphi_i} - \frac{1}{v_i^{(r)}} \qquad \text{for } i = 1, \ldots, n. \tag{3.2}$$

To remove the dependence on $z_0^{(r)}$, we substitute the left-hand side of Equation (3.2) with the expression on the right-hand side for a different index k:

$$\frac{l_k}{l_0} \frac{1}{v_k^{(r)}} e^{j\varphi_k} - \frac{1}{v_k^{(r)}} = \frac{l_i}{l_0} \frac{1}{v_i^{(r)}} e^{j\varphi_i} - \frac{1}{v_i^{(r)}} \qquad \text{for } i, k = 1, \ldots, n, \quad \text{and } k \neq i. \tag{3.3}$$

The only unknowns in Equation (3.3) are vectors $v_i^{(r)}$ and ratios l_i/l_0. Since the angles $\varphi_1, \ldots, \varphi_n$ in Equation (3.3) are independent of the orientation of the robot-centered coordinate system, we can rewrite Equation (3.3) for a robot-centered coordinate system with a different orientation:

$$r_k \frac{1}{\hat{v}_k^{(r)}} e^{j\varphi_k} - \frac{1}{\hat{v}_k^{(r)}} = r_i \frac{1}{\hat{v}_i^{(r)}} e^{j\varphi_i} - \frac{1}{\hat{v}_i^{(r)}} \qquad \text{for } i, k = 1, \ldots, n, \quad \text{and } k \neq i \tag{3.4}$$

where landmark z_i is described by vector $\hat{z}_i^{(r)}$, $r_i = |\hat{z}_i^{(r)}|/|\hat{z}_0^{(r)}|$ and $\hat{v}_i^{(r)} = \hat{z}_i^{(r)} - \hat{z}_0^{(r)}$. We choose the robot-centered coordinate system that is oriented the same way as the external coordinate system. Thus, its axes $\hat{x}^{(r)}$ and $\hat{y}^{(r)}$ are parallel to axes $x^{(e)}$ and $y^{(e)}$, respectively (see Figure 3-6). We pick this particular coordinate system, because we can then write

$$\hat{v}_i^{(r)} = v_i^{(e)} = z_i^{(e)} - z_0^{(e)}$$

for $i = 1, \ldots, n$ which yields equations

$$r_k \frac{1}{v_k^{(e)}} e^{j\varphi_k} - \frac{1}{v_k^{(e)}} = r_i \frac{1}{v_i^{(e)}} e^{j\varphi_i} - \frac{1}{v_i^{(e)}} \qquad \text{for } i, k = 1, \ldots, n, \quad \text{and } k \neq i \tag{3.5}$$

whose only unknowns are ratios $r_1, \ldots, r_n$.

Note that using the robot-centered coordinate system that is spanned by axes $\hat{x}^{(r)}$ and $\hat{y}^{(r)}$ does **not** imply that the robot knows its orientation in the environment a priori. It does not know its orientation angle $\theta = \angle(x^{(r)}, x^{(e)}) = \angle(x^{(r)}, \hat{x}^{(r)})$. Instead, once equations (3.5) are solved we can compute vector $\hat{z}_0^{(r)}$ which can then be used to compute the robot's orientation

$$\theta = \angle(\hat{z}_0^{(r)}, \hat{x}^{(r)}) - \tau_0 \tag{3.6}$$

as illustrated in Figure 3-6.

The set of equations (3.5) can be transformed into a matrix equation $A\, r = c$ where r, c, and A can be defined as follows: Vector $r = (r_1, \ldots, r_n)$ is the vector of the unknown ratios of the length of vectors $\hat{z}_1^{(r)}, \ldots, \hat{z}_n^{(r)}$. Vector c is a $n(n-1)$ dimensional

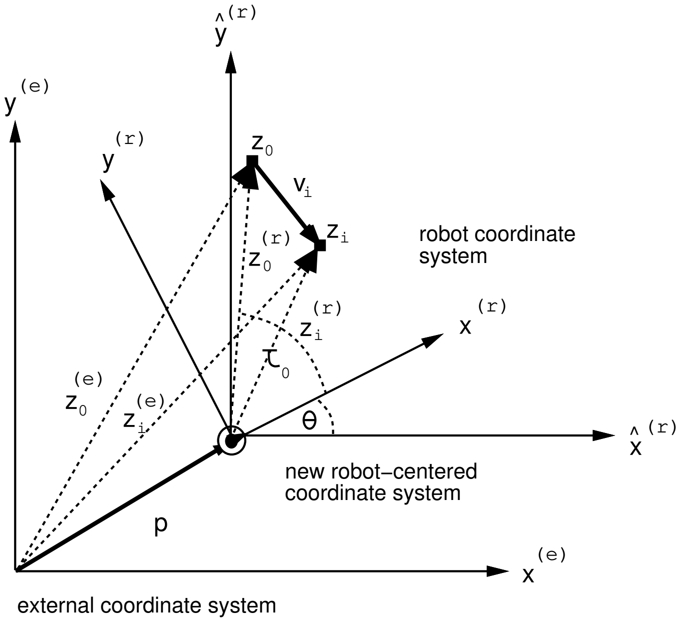

Figure 3-6: External coordinate system spanned by $x^{(e)}$ and $y^{(e)}$, robot-centered coordinate system spanned by $x^{(r)}$ and $y^{(r)}$ and robot-centered coordinate system spanned by $\hat{x}^{(r)}$ and $\hat{y}^{(r)}$. The robot is facing in direction of axis $x^{(r)}$.

vector of differences of complex numbers $c_i = (c_{xi}, c_{yi}) = 1/\hat{v}_i^{(r)}$ and matrix A is a $n(n-1) \times n$ matrix consisting of complex numbers $b_i = (b_{xi}, b_{yi}) = c_i\, e^{\,j\varphi_i}$:

$$A = \begin{pmatrix} b_1 & -b_2 & 0 & 0 & \dots & 0 & 0 \\ b_1 & 0 & -b_3 & 0 & \dots & 0 & 0 \\ b_1 & 0 & 0 & -b_4 & \dots & 0 & 0 \\ \vdots & & & & & & \\ b_1 & 0 & 0 & 0 & \dots & 0 & -b_n \\ -b_1 & b_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & b_2 & -b_3 & 0 & \dots & 0 & 0 \\ 0 & b_2 & 0 & -b_4 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & b_2 & 0 & 0 & \dots & 0 & -b_n \\ \vdots & & & & & & \\ -b_1 & 0 & 0 & 0 & \dots & 0 & b_n \\ 0 & -b_2 & 0 & 0 & \dots & 0 & b_n \\ 0 & 0 & -b_3 & 0 & \dots & 0 & b_n \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & -b_{n-1} & b_n \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} c_1 - c_2 \\ c_1 - c_3 \\ c_1 - c_4 \\ \vdots \\ c_1 - c_n \\ c_2 - c_1 \\ c_2 - c_3 \\ c_2 - c_4 \\ \vdots \\ c_2 - c_n \\ \vdots \\ c_n - c_1 \\ c_n - c_2 \\ c_n - c_3 \\ \vdots \\ c_n - c_{n-1} \end{pmatrix}$$

A solution of equation $A\,r = c$ yields a vector r whose components r_i can be used to solve equation

$$\frac{1}{\hat{z}_0^{(r)}} = r_i \frac{1}{v_i^{(e)}} e^{j\varphi_i} - \frac{1}{v_i^{(e)}} \tag{3.7}$$

for some i to obtain vector $\hat{z}_0^{(r)}$. (This equation corresponds to Equation (3.2) and describes the vectors with respect to the robot-centered coordinate system spanned by axes $\hat{x}^{(r)}$ and $\hat{y}^{(r)}$.) Once $\hat{z}_0^{(r)}$ is known, the robot position p is

$$p = z_0^{(e)} - \hat{z}_0^{(r)}.$$

Note that the system of linear equations $A\,r = c$ is overdetermined. We have n unknowns $r_1, \ldots, r_n$ and $n(n-1)$ equations. These equations may be inconsistent, since we have measurement errors. Therefore, we use a "least squares method" to find a solution for which the average error in the $n(n-1)$ equations is minimized.

3.5.1 Least squares errors and the pseudo inverse

The "least squares method" is a standard method for solving overdetermined systems of equations [98]. In the following we rederive this method for our particular overdetermined system of equations.

To solve the matrix equation $A\,r = c$ we choose ratio vector r so as to minimize the "average" error in the $n(n-1)$ equations. We define the average error by the sum of squares

$$E^2 = ||A\,r - c||^2.$$

If there is an exact solution to $A\,r = c$, the error is $E = 0$. In practice it is unlikely that there is no error, therefore, we solve the minimization problem

$$\min\{r : E^2\} = \min\{r : ||A\,r - c||^2\}.$$

The necessary condition of the squared error E^2 to be minimal is that its derivative with respect to vector r must be zero. The derivative is

$$\begin{aligned}
\frac{d}{dr}E^2 &= \frac{d}{dr}||A\,r - c||^2 \\
&= \frac{d}{dr}(A\,r - c)^T(A\,r - c) \\
&= \frac{d}{dr}(r^T A^T Ar - r^T A^T c - c^T Ar + c^T c) \\
&= A^T Ar + (A^T A)^T r - 2A^T c \\
&= 2\,A^T Ar - 2A^T c
\end{aligned}$$

where A^T is the conjugate transpose of A. Setting the derivative to zero gives us

$$0 \;=\; 2A^T Ar - 2A^T c$$

$$
\begin{aligned}
A^T A r &= A^T c \\
r &= (A^T A)^{-1} A^T c,
\end{aligned}
$$

provided matrix $A^T A$ is nonsingular. In Section 3.5.3 we give the necessary and sufficient conditions for $A^T A$ being nonsingular and argue that they will almost always be fulfilled in practice.

Matrix $A^+ = (A^T A)^{-1} A^T$ is often called the *pseudo inverse* of matrix A (see for example [1, 98]). In the following section we describe an algorithm called POSITION ESTIMATOR which efficiently calculates vector $r = A^+ c = (A^T A)^{-1} A^T c$ such that the average error in the equations is minimized.

3.5.2 The position estimation algorithm

In this section we describe procedure POSITION ESTIMATOR that estimates the position and the orientation of the robot. In the description of the algorithm we use two kinds of multiplications of complex numbers $a = (Re(a), Im(a))$ and $b = (Re(b), Im(b))$: the standard complex multiplication

$$ab = (Re(a)Re(b) - Im(a)Im(b),\ Re(a)Im(b) + Re(b)Im(a))$$

and the inner product

$$\bar{a}^T b = Re(a)Re(b) + Im(a)Im(b)$$

where $\bar{a} = (Re(a), -Im(a))$ is the conjugate of a. For notational convenience we write the inner product of complex numbers a and b as $a^T b$.

Procedure POSITION ESTIMATOR takes as an input the positions of the landmarks $z_0^{(e)}, \ldots, z_n^{(e)}$ (as given in the external coordinate system) and the angles $\varphi_1, \ldots \varphi_n$.

After the initialization of vectors v, c, b and $\sum_{i=1}^n c_i$ in lines 2–6, POSITION ESTIMATOR first computes vector $s = \frac{1}{2} A^T c$ in lines 7 and 8 as follows: For each component of vector $s = \frac{1}{2} A^T c$ the procedure computes

$$
\begin{aligned}
s_i &= (n-1) b_i^T c_i - \sum_{j, j \neq i} b_i^T c_j \\
&= n b_i^T c_i - \sum_{j=1}^n b_i^T c_j \\
&= n b_i^T c_i - b_i^T \sum_{j=1}^n c_j \\
&= n(b_{xi} c_{xi} + b_{xi} c_{xi}) - (b_{xi} \sum_{j=1}^n c_{xj} + b_{yi} \sum_{j=1}^n c_{yj}). \qquad (3.8)
\end{aligned}
$$

Procedure POSITION ESTIMATOR then determines $r = 2(A^T A)^{-1} s$ by calling procedure RATIOS-CALCULATOR in line 9. It next calculates n position estimates $p_1, \ldots, p_n$

$\textsc{Position-estimator}(z_0^{(e)}, \ldots, z_n^{(e)}, \varphi_1, \ldots \varphi_n)$

1. $\textit{sum-of-}c_i = 0$
2. **for** $i = 1$ **to** n (initialization)
3. $\quad v_i = z_0^{(e)} - z_i^{(e)}$
4. $\quad c_i = 1/v_i$
5. $\quad \textit{sum-of-}c_i = \textit{sum-of-}c_i + c_i$ (calculates $\sum_{i=1}^n c_i$)
6. $\quad b_i = c_i e^{j\varphi_i}$
7. **for** $i = 1$ **to** n (calculates $\frac{1}{2}A^T c$)
8. $\quad s_i = n b_i^T c_i - b_i^T \; \textit{sum-of-}c_i$
9. $\textsc{Ratios-calculator}(b_1, \ldots, b_n, s_1, \ldots, s_n)$ (returns r)
10. **for** $i = 1$ **to** n
11. $\quad \hat{z}_{0,i}^{(r)} = 1/(r_i b_i - c_i)$
12. $\quad p_i = z_0^{(e)} - \hat{z}_{0,i}^{(r)}$
13. $p = \frac{1}{n}\sum_{i=1}^n p_i$ (robot position p)
14. $\theta = \angle(\hat{z}_0^{(r)}, \hat{x}^{(r)}) - \tau_0$

(lines 10-12).

In line 11 of procedure Position estimator, a set of n solutions $\hat{z}_{0,1}^{(r)} \ldots, \hat{z}_{0,n}^{(r)}$ for the position of landmark $\hat{z}_0^{(r)}$ in the robot-centered coordinate system is calculated using equation (3.7):

$$\hat{z}_{0,i}^{(r)} = \frac{1}{r_i b_i - c_i} \quad \text{for } i = 1, \ldots, n.$$

In line 13 of procedure Position estimator, a set of estimates $p_1, \ldots, p_n$ for the robot position is calculated using the solutions for the position of landmark z_0.

If there is no noise in the measurements, the vectors $\hat{z}_{0,i}^{(r)}$ are the same for all indices i. In practice there will be noise, so we take the centroid

$$p = \frac{1}{n}\sum_{i=1}^n p_i$$

of the position estimates $p_1, \ldots, p_n$ as an average to obtain an estimate of the position p of the robot (line 13 of procedure Position estimator)[1]. In line 14 the orientation angle is computed using Equation (3.6).

Procedure Ratios-calculator takes as an input the vectors b and s and returns the vector $r = 2(A^T A)^{-1}s$. First Ratios-calculator calculates vectors $b_x = (b_{x1}, \ldots, b_{xn})$ and $b_y = (b_{y1}, \ldots, b_{yn})$ where b_{xi} is the real coordinate of b_i and b_{yi} is

[1] Note that taking the centroid may not be the best way to average the n estimates. We are currently working on determining the best averaging coefficients [53]. They may not be the same for the x- and y-coordinates of p_i.

the imaginary coordinate of b_i. Then it exploits the special form of matrix A^TA:

$$A^TA = 2\begin{pmatrix} (n-1)\, b_1^Tb_1 & -b_1^Tb_2 & -b_1^Tb_3 & \dots & -b_1^Tb_n \\ -b_2^Tb_1 & (n-1)\, b_2^Tb_2 & -b_2^Tb_3 & \dots & -b_2^Tb_n \\ \vdots & & & & \\ -b_n^Tb_1 & -b_n^Tb_2 & -b_n^Tb_3 & \dots & (n-1)\, b_n^Tb_n \end{pmatrix}$$

Instead of inverting the matrix A^TA directly, we write matrix A^TA as a sum of a diagonal matrix and two matrices that are outer products (or Grammian matrices, see Albert [1]). In this special form, the matrix A^TA can be inverted efficiently:

$$A^TA = 2\,(nD - b_xb_x^T - b_yb_y^T)$$

where D is a diagonal matrix whose ith entry is $b_i^Tb_i = b_{xi}^2 + b_{yi}^2$, and the outer product $b_xb_x^T$ is defined as follows

$$b_xb_x^T = \begin{pmatrix} b_{x1}^2 & b_{x1}b_{x2} & \dots & b_{x1}b_{xn} \\ b_{x2}b_{x1} & b_{x2}^2 & \dots & b_{x2}b_{xn} \\ & \vdots & & \\ b_{xn}b_{x1} & b_{xn}b_{x2} & \dots & b_{xn}^2 \end{pmatrix}.$$

Matrix $b_yb_y^T$ can be written analogously.

Then $r = (A^TA)^{-1}A^Tc = (A^TA)^{-1}\,2\,s = 1/2\,(nD - b_xb_x^T - b_yb_y^T)^{-1}\,2\,s = (nD - b_xb_x^T - b_yb_y^T)^{-1}s$. In this form r can be calculated (without inverting $nD - b_xb_x^T - b_yb_y^T$ directly) using the following well-known lemma [1]:

Lemma 12 *If matrix K is nonsingular and $(K^{-1}h)^Th \neq 1$, then*

$$(K - hh^T)^{-1} = K^{-1} + \frac{K^{-1}hh^TK^{-1}}{1 - (K^{-1}h)^Th}$$

Proof: The proof follows directly by verifying equation

$$I = (K - hh^T)(K^{-1} + \frac{K^{-1}hh^TK^{-1}}{1 - (K^{-1}h)^Th})$$

where I is the identity matrix. □

We apply the formula given in Lemma 12 twice to invert matrix A^TA (We show in Section 3.5.3 that the requirements of Lemma 12, i.e., matrix K is nonsingular and $(K^{-1}h)^Th \neq 1$, are indeed met.)

Since $r = (A^TA)^{-1}A^Tc = ((nD - b_xb_x^T) - b_yb_y^T)^{-1}s$, we can apply Lemma 12 to $K = (nD - b_xb_x^T)$ and $h = b_y$ and get

$$r \;=\; ((nD - b_xb_x^T) - b_yb_y^T)^{-1}s$$

$$\begin{aligned} &= K^{-1}s + \frac{K^{-1}b_y b_y^T K^{-1}}{1-(K^{-1}b_y)^T b_y} s \\ &= K^{-1}s + \frac{(K^{-1}b_y)((K^{-1}b_y)^T s)}{1-(K^{-1}b_y)^T b_y} \end{aligned} \tag{3.9}$$

Applying Lemma 12 again, the first term $K^{-1}s$ can be expressed as

$$K^{-1}s = (nD - b_x b_x^T)^{-1}s = (nD)^{-1}s + \frac{(nD)^{-1}b_x((nD)^{-1}b_x)^T s}{1-((nD)^{-1}b_x)^T b_x}, \tag{3.10}$$

Vector $K^{-1}b_y = (nD - b_x b_x^T)^{-1}b_y$ in Equation (3.9) is calculated by applying the formula given in Lemma 12 which yields

$$K^{-1}b_y = (nD)^{-1}b_y + \frac{(nD)^{-1}b_x((nD)^{-1}b_x)^T b_y}{1-((nD)^{-1}b_x)^T b_x}. \tag{3.11}$$

The inverse of a diagonal matrix is a matrix whose entries on the diagonal are inverted. Thus, the ith diagonal entry of $(nD)^{-1}$ is $1/(nb_i^T b_i) = 1/(n(b_{xi}^2 + b_{yi}^2))$.

We can now obtain the full expression for $r = (A^TA)^{-1}A^Tc = (nD - b_x b_x^T - b_y b_y^T)^{-1}s$ by substituting $K^{-1}s$, $K^{-1}b_y$, and $(nD)^{-1}$ into Equation (3.9).

If the i-th landmark is very close to the robot's position, the least squares method described above may yield a negative r_i. In this case ratio r_i should be positive, but almost zero. Thus, we assign it to be zero in procedure RATIOS-CALCULATOR.

RATIOS-CALCULATOR$(b_1, \ldots, b_n, s_1, \ldots, s_n)$

1. compute diagonal matrix $(nD)^{-1}$
2. compute vector $K^{-1}s$ using Equation (3.10)
3. compute vector $K^{-1}b_y$ using Equation (3.11)
4. compute vector $r = (r_i, \ldots, r_n)$ using Equation (3.9)
5. **for** $i = 1$ **to** n
6.   **if** $r_i < 0$ **then** $r_i \leftarrow 0$

3.5.3 Correctness and analysis of POSITION ESTIMATOR

In order to establish the correctness of procedure POSITION ESTIMATOR we first need to prove the following lemmas. We use the notation $D = diag(d_i)$ to denote a diagonal matrix D whose ith diagonal element is d_i. Given some matrix M we write $M > 0$ if M is positive definite and $M \geq 0$ if M is positive semidefinite.

Lemma 13 *Let matrix $D = diag(b_{xi}^2 + b_{yi}^2)$, $b_x^T = (b_{x1}, \ldots, b_{xn}$, and $b_y^T = (b_{y1}, \ldots, b_{yn}$. Matrix $A^TA = 2(nD - b_x b_x^T - b_y b_y^T)$ is nonsingular if and only if the vectors b_x and b_y are linearly independent.*

Proof: Since $A^T A \geq 0$, we know that $A^T A$ is singular if and only if there exists a vector z, $z \neq 0$ such that $z^T A^T A z = 0$ or equivalently

$$\begin{aligned} z^T(nD - b_x b_x^T - b_y b_y^T)z &= 0 \\ z^T n D z &= z^T b_x b_x^T z + z^T b_y b_y^T z \\ n z^T diag(b_{xi}^2 + b_{yi}^2) z &= (z^T b_x)(b_x^T z) + (z^T b_y)(b_y^T z) \\ n \sum_{i=1}^{n} (b_{xi}^2 + b_{yi}^2) z_i^2 &= (\sum_{i=1}^{n} b_{xi} z_i)^2 + (\sum_{i=1}^{n} b_{yi} z_i)^2. \end{aligned}$$

Note that for any numbers $a_1, \ldots, a_n$ we know that $(\sum_{i=1}^n a_i)^2 \leq n \sum_{i=1}^n a_i^2$ and the equality holds if and only if $a_i = a_j$ for all indices i and j. Therefore, $A^T A$ is singular if and only if

$$b_{xi} z_i = b_{xj} z_j \quad \text{and} \quad b_{yi} z_i = b_{yj} z_j$$

for all indices i and j. This means that $b_x = \alpha z$ and $b_y = \beta z$ for some constants α and β. Thus we get that $b_x = (\alpha/\beta)\, b_y$ which means that b_x and b_y are linearly dependent. $\square$

The following lemma gives the geometrical interpretation of Lemma 13.

Lemma 14 *Matrix $A^T A = 2(nD - b_x b_x^T - b_y b_y^T)$ is singular if and only if the landmarks and the robot's position are arranged in a perfect circle or on a line.*

Proof: According to Lemma 13 we have to find a geometric criterion for vectors b_x and b_y to be linearly dependent. The rank of the $n \times 2$ matrix $b_x b_y$ whose two columns consist of vectors b_x and b_y is one if vectors b_x and b_y are collinear. Note that

$$rank(b_x b_y) = rank \begin{pmatrix} b_x^T \\ b_y^T \end{pmatrix}$$

where $\binom{b_x^T}{b_y^T}$ is the matrix whose rows are vectors b_x^T and b_y^T. By definition (see Section 3.5.2) we have

$$\begin{pmatrix} b_x^T \\ b_y^T \end{pmatrix} = \begin{pmatrix} b_{x1} \ldots b_{xn} \\ b_{y1} \ldots b_{yn} \end{pmatrix} = (b_1 \ldots b_n)$$

since $b_k = (b_{xk}, b_{yk})^T$, for $1 \leq k \leq n$. Therefore, it is sufficient for the proof to find the geometrical interpretation of the collinearity of the 2-D vectors b_k, for $1 \leq k \leq n$. That is, we need a geometrical interpretation for the existence of a complex number η such that $b_k = \alpha_k \eta$, where α_k is a real number.

We also have

$$b_k = c_k e^{j\varphi_k} = \frac{e^{j\varphi_k}}{v_k} = \frac{e^{j\varphi_k}}{z_k - z_0}$$

where vector v_k connects landmark z_0 with the landmarks z_k and φ_k is the visual angle between z_0 and z_k from the robot's position for $1 \leq k \leq n$ as illustrated in Figure 3-7. (Note that $b_k = (b_{xk}, b_{yk})^T$ is *not* a vector in the robot-centered coordinate system.)

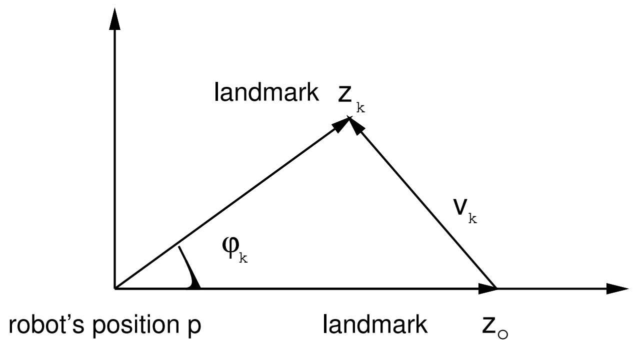

Figure 3-7: Robot coordinate system defined as a z-plane in the proof of Lemma 14.

Without loss of generality, we use a robot-centered coordinate system in which the x-axis is aligned with the vector to landmark z_0 (see Figure 3-7). The landmarks in this coordinate system are defined as $z_k = |z_k|e^{j\varphi_k}$ for $0 < k \leq n$. Thus, b_k can also be expressed as

$$b_k = \frac{z_k}{z_k - z_0}\ \frac{1}{|z_k|} = \frac{z_k}{z_k - z_0}k$$

where k is defined to be $1/|z_k|$. Then vectors b_k, $1 \leq k \leq b_n$, are linearly independent if for all indices k

$$b_k = \frac{z_k}{z_k - z_0}k = \alpha_k \eta$$

for some fixed complex number η and real α_k. We can also write this as

$$\frac{z_k}{z_k - z_0} = \beta_k \eta$$

for $\beta_k = \alpha_k / k$ which is equivalent to $\beta_k \eta z_0 = \beta_k \eta z_k - z_k$ and

$$\frac{\beta_k \eta z_0}{\beta_k \eta - 1} = z_k$$

for $\beta_k \eta - 1 \neq 0$. We can view this as a mapping f of points $w = \beta_k \eta$ in the complex w-plane into the complex z-plane where each landmark z_k is a point in the z-plane:

$$f(w) = \frac{z_0 w}{w - 1} = z.$$

This mapping is called a Möbius transformation (see for example [31]). The Möbius transformation maps the line $w = \beta\eta$ in the w-plane into a circle in the z-plane. Since $f(0) = 0$, the circle passes through the origin of the z-plane. Thus, landmarks $z_1, \ldots, z_n$ and the robot at $(0, 0)$ lie on a circle if the vectors b_k, $1 \leq k \leq b_n$, are linearly independent. Landmark z_0 lies on the same circle since

$$\lim_{\beta \to \infty} \frac{\beta \eta z_0}{\beta \eta - 1} = \lim_{\beta \to \infty} \frac{\eta z_0}{\eta - 1/\beta} = z_0.$$

Note that we assumed $\beta_k \eta - 1 \neq 0$. If $\beta_k \eta = 1$, then we know that $b_k = \frac{e^{j\varphi_k}}{z_k - z_0} = k\beta_k\eta = k$ which means that $\varphi_k = 0$. In this case it follows that all landmarks $z_0, \ldots, z_n$ lie on the real axis of the z-plane. Thus, the landmarks and the robot lie on one line.

The properties of a Möbius transformation also hold for inverse transforms. Therefore, given landmarks arranged on a circle or on a line, it follows that vectors b_k, $1 \leq k \leq b_n$, are linearly independent and matrix $A^T A$ is singular. $\square$

The singularity condition is easily tested through some preprocessing that makes sure that the landmarks are not all arranged in a circle or on a line when procedure Position Estimator is called. In practice it is highly unlikely that the landmarks all lie on a perfect circle or line including the robot's position.

Lemma 15 *If landmarks are correctly identified and there is no noise in the measurements of the visual angles between the landmarks, then procedure* Position Estimator *returns the actual position of the robot.*

Proof: Let $\hat{z}_i^{(r)} = \hat{l}_i e^{j\hat{\tau}}$ be the vector to the i-th landmark in the robot coordinate system that is oriented as the external coordinate system. Note that the actual position of the robot is at $p_{actual} = z_0^{(e)} - \hat{z}_0^{(r)}$. The algorithm calculates $p = z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \hat{z}_{0,i}^{(r)}$ which is equivalent to

$$\begin{aligned}
p &= z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \hat{z}_{0,i}^{(r)} \\
&= z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \frac{1}{r_i b_i - c_i} \\
&= z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\frac{\hat{l}_i}{\hat{l}_0}\frac{1}{\hat{v}_i}e^{j\varphi_i} - \frac{1}{\hat{v}_i}} \\
&= z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \frac{\hat{l}_0 \hat{v}_i}{\hat{l}_i e^{j(\hat{\tau}_i - \hat{\tau}_0)} - \hat{l}_0} \\
&= z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \frac{\hat{l}_0 e^{j\hat{\tau}_0}(\hat{z}_i^{(r)} - \hat{z}_0^{(r)})}{\hat{l}_i e^{j\hat{\tau}_i} - \hat{l}_0 e^{j\hat{\tau}_0}} \\
&= z_0^{(e)} - \frac{1}{n}\sum_{i=1}^{n} \frac{\hat{z}_0^{(r)}(\hat{z}_i^{(r)} - \hat{z}_0^{(r)})}{\hat{z}_i^{(r)} - \hat{z}_0^{(r)}} \\
&= z_0^{(e)} - \frac{1}{n} n \hat{z}_0^{(r)} \\
&= z_0^{(e)} - \hat{z}_0^{(r)}.
\end{aligned}$$

Thus, the algorithm calculates $p = p_{actual}$. $\square$

Theorem 13 *Given a set of noisy measurements of landmarks which are not all arranged in a circle or line algorithm* Position Estimator *is an algorithm that* *determines the position of the robot such that the square of errors in the equations $Ar = c$ is minimized.* □

Proof: The proof follows from the fact that the pseudo inverse A^+ solves the problem $min\{r : ||Ar - c||^2\}$ optimally as shown in Section 3.5.1. Algorithm Position Estimator determines the position of the robot by calculating vector $r = A^+c$. Lemma 13 and Lemma 14 establish that the pseudo inverse A^+ can be computed (since A^TA is nonsingular) if the landmarks are not all arranged on a circle or line. □

Theorem 14 *The algorithm* Position Estimator *runs in time linear in the number of landmarks.*

Proof: Due to the special form of vector $s = \frac{1}{2}A^Tc$ it can be calculated in time linear in n by first computing $\sum_{j=1}^{n} c_{jx}$ and $\sum_{j=1}^{n} c_{jx}$ and then computing each entry s_i of vector s following Equation (3.8).

Algorithm Position Estimator calls procedure Ratios-calculator which calculates vector r by substituting matrix $(nD)^{-1}$ and vectors $K^{-1}s$ and $K^{-1}b_y$ into Equation (3.9). Matrix $(nD)^{-1}$ can be computed in $O(n)$ time if we represent this diagonal matrix by a vector of its diagonal entries. The ith diagonal entry of $(nD)^{-1}$ is $1/(n(b_{xi}^2 + b_{yi}^2))$. Using this vector representation means that multiplying matrix $(nD)^{-1}$ with vectors s and b_x can also be done in linear time. Dot products $((nD)^{-1}b_x)^T s$ and $((nD)^{-1}b_x)^T b_x$ take $O(n)$ time to compute. Given vectors $(nD)^{-1}s$ and $(nD)^{-1}b_x$ and scalars $((nD)^{-1}b_x)^T s$ and $((nD)^{-1}b_x)^T b_x$, the computation of Equation (3.10) can be concluded in $O(1)$ time.

With a similar argument we can show that Equations (3.11) and (3.9) can be computed in linear time. □

3.5.4 Quality of Position Estimator

This section reports results of simulation experiments using procedure Position Estimate. In the experiments, the actual robot position $p_{actual} = (p_{x,actual}, p_{y,actual})$ is compared to the position $p = (p_x, p_y)$ that is computed by procedure Position Estimate. The actual robot orientation θ_{actual} is compared to the orientation θ that is also computed by procedure Position Estimate.

In the experiments, the errors $|p_{x,actual} - p_x|$ and $|p_{y,actual} - p_y|$ in the x- and y-coordinates of the position are computed. The length of the error vector $||p_{actual} - p||$ and the orientation error $|\theta_{actual} - \theta|$ are also computed.

Without loss of generality, the actual robot position p_{actual} is assumed to be at the origin of the map, i.e., $p_{actual} = (0, 0)$ and the robot is assumed to be oriented in the direction of the x-axis, i.e., $\theta_{actual} = 0°$.

First we randomly place eleven landmarks in a $10m \times 10m$ area using a uniform distribution where the robot is at a corner of this area. Note that this scenario uses only a quarter of the robot's surrounding area. It assumes that the robot cannot get omnidirectional input as Ratbot, but instead has a conventional camera setup.

In an attempt to simulate worst-case measurement errors, we first add a constant fraction $\Delta\varphi$ of a degree to *each* angle. Figure 3-8 shows results of experiments in which the added noise $\Delta\varphi$ ranges from 0° to 5°. We averaged our results over 100,000 scenarios with 11 landmarks, distributed uniformly in the 10 by 10 meter area. The length of the error vector $p_{actual} - p$ is 57 cm for 1° noise, 1.1 m for 2° noise, and 2.57 m for 5° for noise in *every* angle. The error in the position angle θ is 2° for 1° angle noise, 4.2° for 2° angle noise, and 9.8° for 5° angle noise.

Next we show results of experiments in which the angle errors are not fixed, but uniformly distributed between $[-\Delta\varphi, \Delta\varphi]$ where $\Delta\varphi$ also ranges from 0° to 5°. Figures 3-9 – 3-12 show the results of the experiments with 40,000 different environments that include 11 landmarks. Figure 3-9 shows the length of the error vector as a function of the visual angle error. The length of the error vector $p_{actual} - p$ is 19 cm for angle noise randomly chosen within $[-1°, 1°]$, 38 cm for angle noise randomly chosen within $[-2°, 2°]$ and 90 cm for angle noise randomly chosen within $[-5°, 5°]$. The error in position angle θ is on average zero.

Figures 3-10, 3-11 and 3-12 show how the errors, $\|p_{actual} - p\|$, $|p_{x,actual} - p_x|$, $|p_{y,actual} - p_y|$, and $|\theta_{actual} - \theta|$ are distributed for $\Delta\varphi = 1°$, $\Delta\varphi = 2°$, and $\Delta\varphi = 5°$. Figure 3-13 shows the computed positions of the robot in the two-dimensional environment for $\Delta\varphi = 1°$, $\Delta\varphi = 2°$, and $\Delta\varphi = 5°$. Note that the distributions for position errors $|p_{x,actual} - p_x|$ and $|p_{y,actual} - p_y|$ are almost normal.

In the next set of experiments, the robot is in the center of a 20 $m \times 20$ m area with eleven landmarks. We expect better results for this scenario in which the robot can take advantage of information provided by all of its surrounding. Indeed, our results show that the error in position has length 8.9 cm for $\Delta\varphi$ uniformly distributed within $[-1, 1]$, length 17.6 cm for $\Delta\varphi$ uniformly distributed within $[-2, 2]$, and length 42.6 cm for $\Delta\varphi$ uniformly distributed within $[-5, 5]$. The results of the experiments are summarized in Figures 3-14 and 3-15.

Note that procedure POSITION ESTIMATOR computes n position estimates $p_1, \ldots,$ p_n and averages them to obtain estimate p. Among $p_1, \ldots, p_n$, there may be some *outliers* that are not very close to the final position estimate p. However, in our experiments we observed that there is a one-to-one correspondence between each landmark z_i and estimate p_i. Each outlier p_i corresponds to either a misidentified landmark z_i or a noisy angle measurement for landmark z_i. Figure 3-16 illustrates this behavior of POSITION ESTIMATOR. Given an input of 21 landmarks $z_0, \ldots, z_{20}$, the procedure computes 20 position estimates $p_1, \ldots, p_{20}$. To model outlier errors resulting either from two noisy angle measurements or from two misidentified landmarks, we add a fixed error of 20° to two arbitrary angles. The other angles have an additive error uniformly distributed within $[-1°, 1°]$.

Our experiments show that the outlier landmarks produce position estimates that

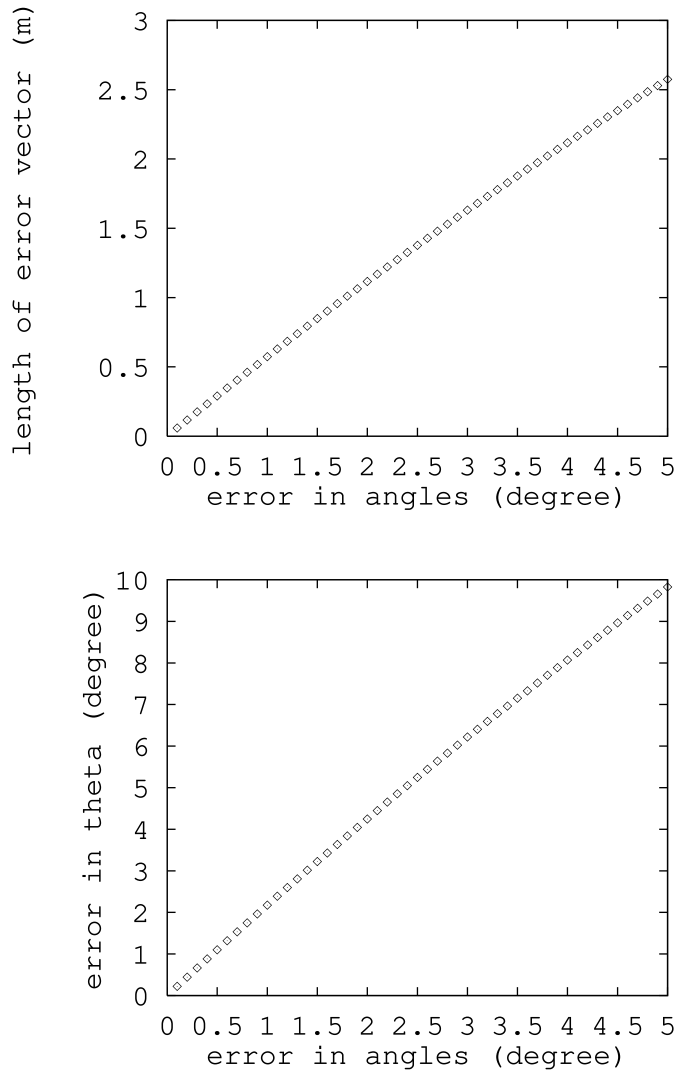

Figure 3-8: Simulation results using algorithm POSITION ESTIMATOR on an input of 11 landmarks and angle measurements with constant error. The error $\|p_{actual} - p\|$ in meters (top) and the error $|\theta_{actual} - \theta|$ in degrees (bottom) are shown as functions of $\Delta\varphi$ in degrees.

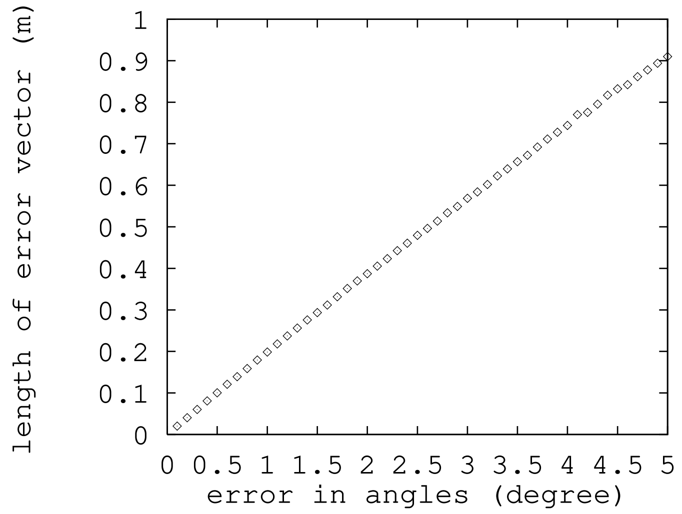

Figure 3-9: Simulation results using algorithm POSITION ESTIMATOR on an input of 11 landmarks and angle measurements with error uniformly distributed over $[-\Delta\varphi, \Delta\varphi]$ where $\Delta\varphi$ ranges from 0 to 5 degrees. The error $\|p_{actual} - p\|$ in meters is shown as a function of $\Delta\varphi$ in degrees.

are far from the actual position of the robot compared to the other estimates. So instead of taking the centroid of all position estimates $p_1, \ldots, p_{20}$ in line 12 of POSITION ESTIMATOR, we can obtain better results if we modify the algorithm such that we discard the outlier position estimates and calculate the centroid of the remaining points.

However, it is even better to call POSITION ESTIMATOR on an input of 19 landmarks (without the outlier landmarks). Figure 3-16 shows the results of our computation on two scenarios in which the length of the error vector is computed. For each of the scenarios, we show position estimates that are computed with and without the outlier landmarks. In the first scenario, the length of the error vector is 24.0 cm when computed with outlier landmarks, and 6.0 cm when computed without the outlier landmarks. In the second scenario, the length of the error vector is 127 cm when computed with outlier landmarks, and 1.6 cm when computed without the outlier landmarks.

3.6 Conclusions

We described an algorithm for estimating the position and orientation of a mobile robot given noisy input data. The key advantages of our POSITION ESTIMATOR over the approach described in Section 3.4 is that (1) the algorithm runs in time linear in the number of landmarks, (2) the algorithm provides a position estimate which is very close to the actual robot position and orientation, and (3) large errors in some measurements (i.e., outliers) can be found.

Our algorithm does not use information about the motion of the mobile robot:

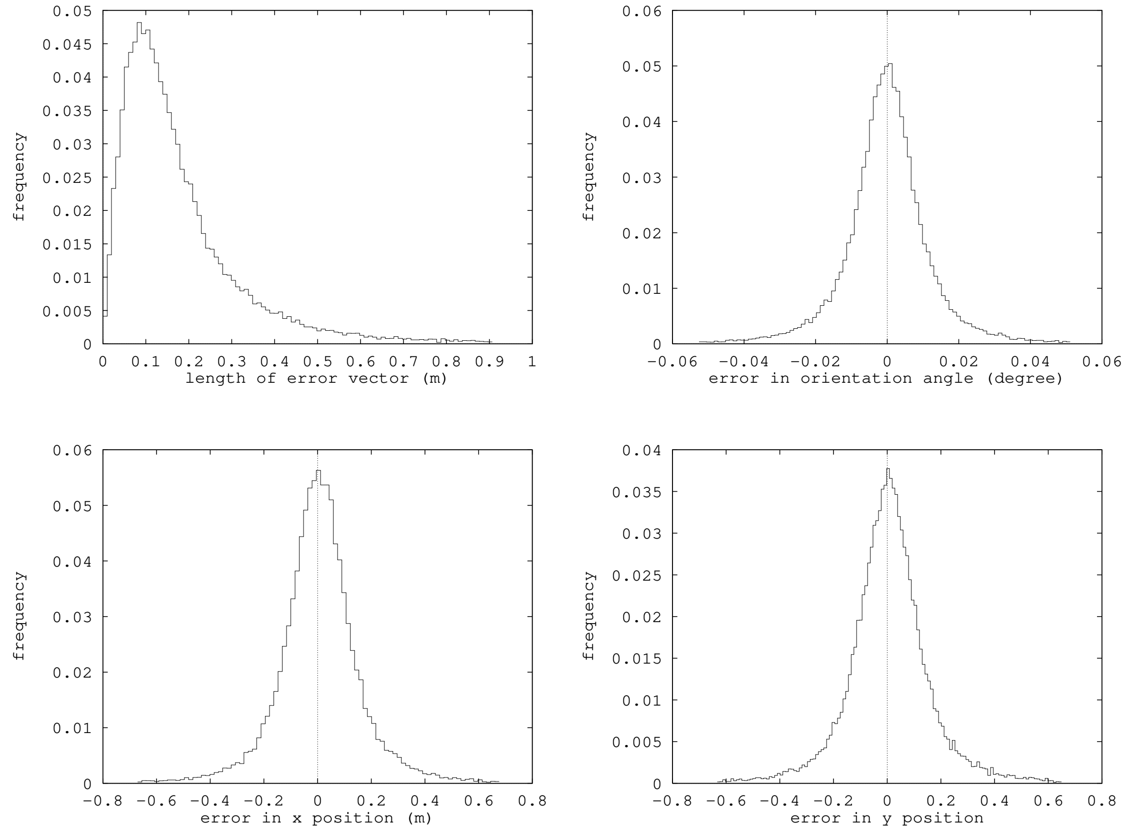

Figure 3-10: Distribution of errors for $\Delta\varphi = 1°$. On the top left, the distribution of the length of error vector $\|p_{actual} - p\|$ with mean 19.7 cm and standard deviation 23.9 cm. On the top right, the distribution of the error in orientation $|\theta_{actual} - \theta|$ with almost zero mean and standard deviation 0.017°. On the bottom left, the distribution of error in x position $|p_{x,actual} - p_x|$ with mean 0.93 cm and standard deviation 0.22 cm. On the bottom right, the distribution of error in y position $|p_{y,actual} - p_y|$ with mean 10.6 cm and standard deviation 21.4 cm.

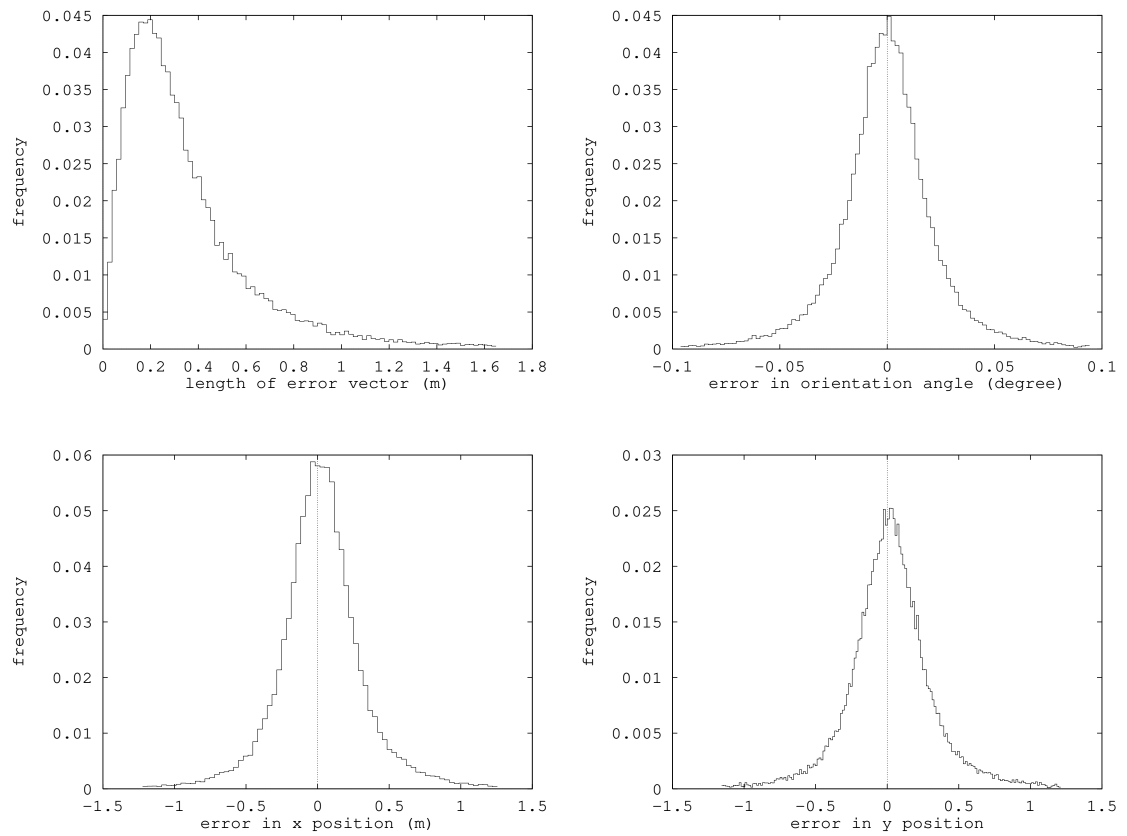

Figure 3-11: Distribution of errors for $\Delta\varphi = 2°$. On the top left, the distribution of the length of error vector $\|p_{actual} - p\|$ with mean 38.5 cm and standard deviation 42.7 cm. On the top right, the distribution of the error in orientation $|\theta_{actual} - \theta|$ with almost zero mean and standard deviation 0.032°. On the bottom left, the distribution of error in x position $|p_{x,actual} - p_x|$ with mean 3.04 cm and standard deviation 41.4 cm. On the bottom right, the distribution of error in y position $|p_{y,actual} - p_y|$ with mean 3.29 cm and standard deviation 39.6 cm.

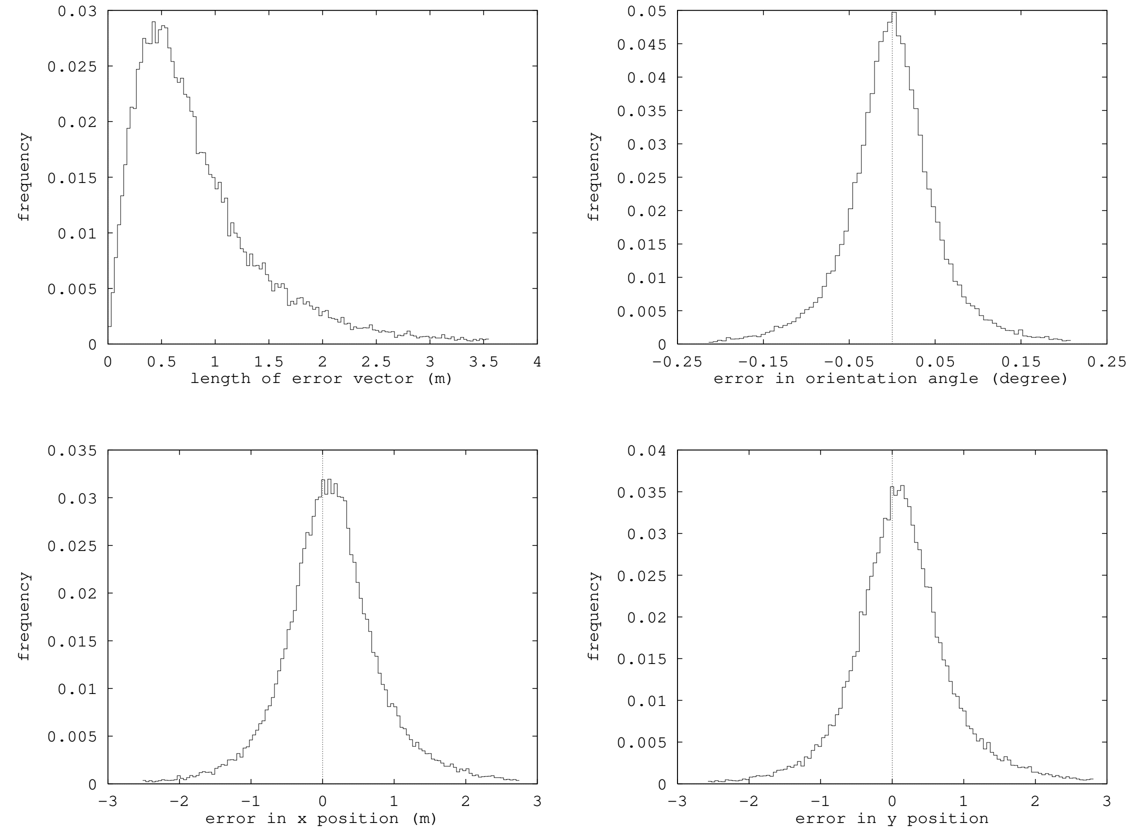

Figure 3-12: Distribution of errors for $\Delta\varphi = 5°$. On the top left, the distribution of the length of error vector $\|p_{actual} - p\|$ with mean 90.8 cm and standard deviation 88.3 cm. On the top right, the distribution of the error in orientation $|\theta_{actual} - \theta|$ with almost zero mean and standard deviation 0.071°. On the bottom left, the distribution of error in x position $|p_{x,actual} - p_x|$ with mean 14.6 cm and standard deviation 87.3 cm. On the bottom right, the distribution of error in y position $|p_{y,actual} - p_y|$ with mean 15.0 cm and standard deviation 89.4 cm.

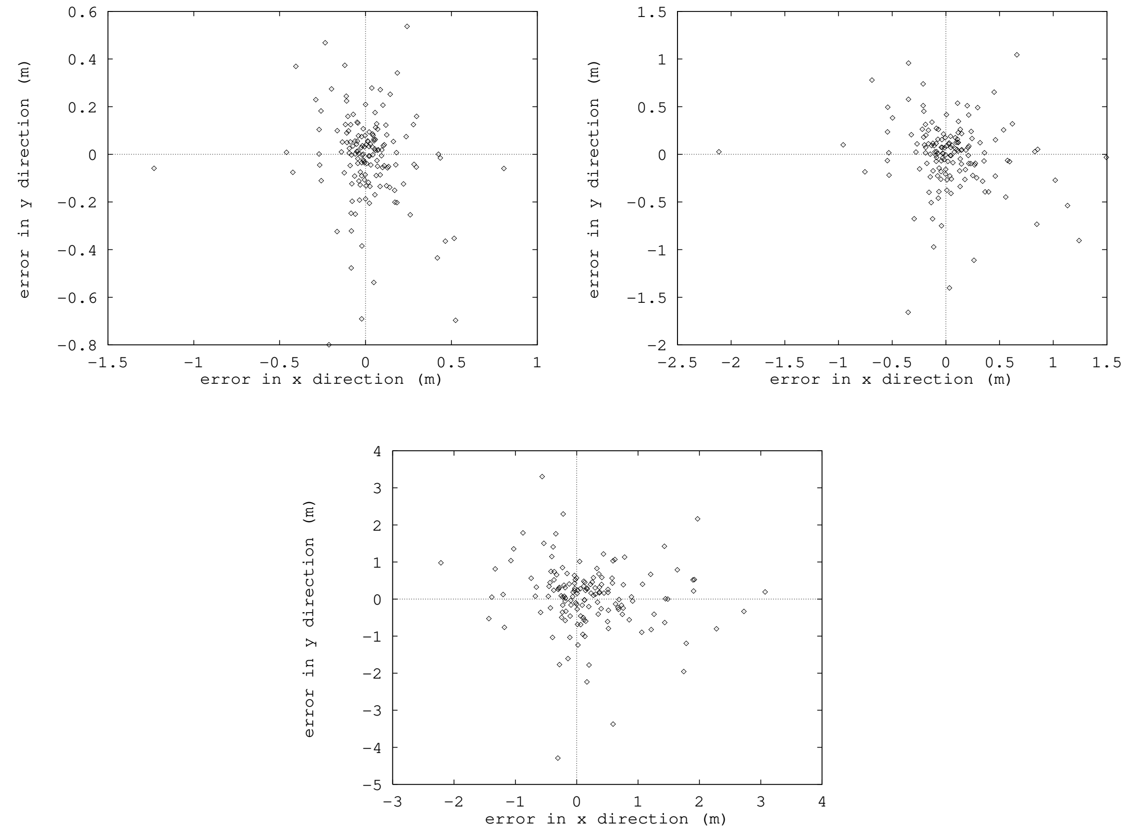

Figure 3-13: Computed positions of the robot in the two-dimensional environment for $\Delta\varphi = 1°$ (top left), $\Delta\varphi = 2°$ (top right), and $\Delta\varphi = 5°$ (bottom).

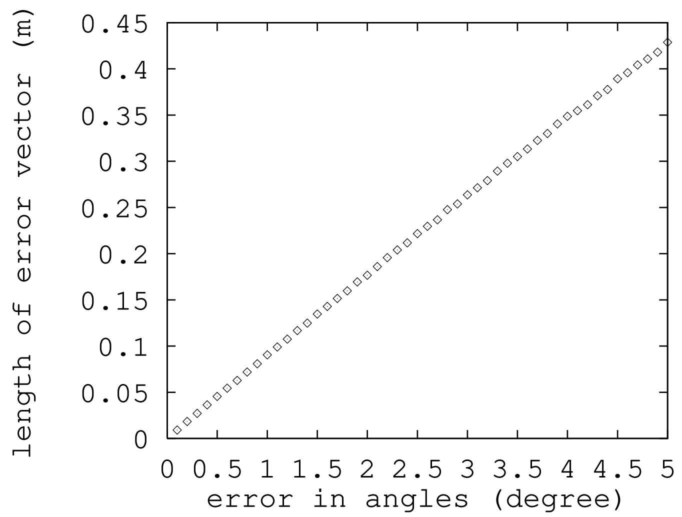

Figure 3-14: Simulation results using algorithm POSITION ESTIMATOR on an input of 11 landmarks surrounding the robot and angle measurements with error uniformly distributed over $[-\Delta\varphi, \Delta\varphi]$ where $\Delta\varphi$ ranges from 0 to 5 degrees. The error $\|p_{actual} - p\|$ in meters is shown as a function of measurement errors in degrees.

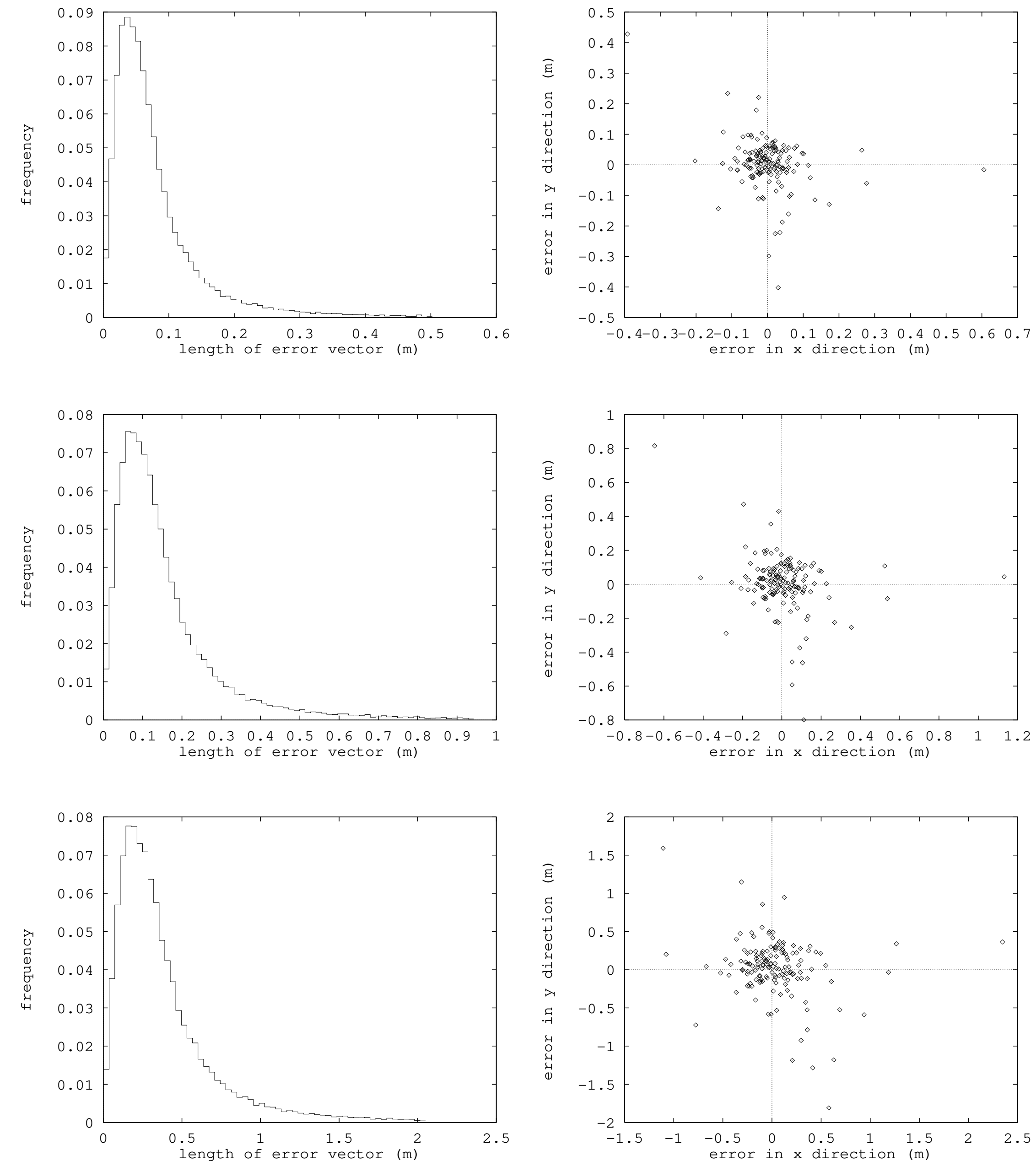

Figure 3-15: Experiments in which the robot is surrounded by 11 landmarks. On the left, the distributions of the length of error vectors $\|p_{actual} - p\|$ for $\Delta\varphi$ uniformly distributed within $[-1, 1]$, $[-2, 2]$, and $[-5, 5]$. On the right, computed positions of the robot in the two-dimensional environment for $\Delta\varphi = 1°, 2°$ and $5°$.

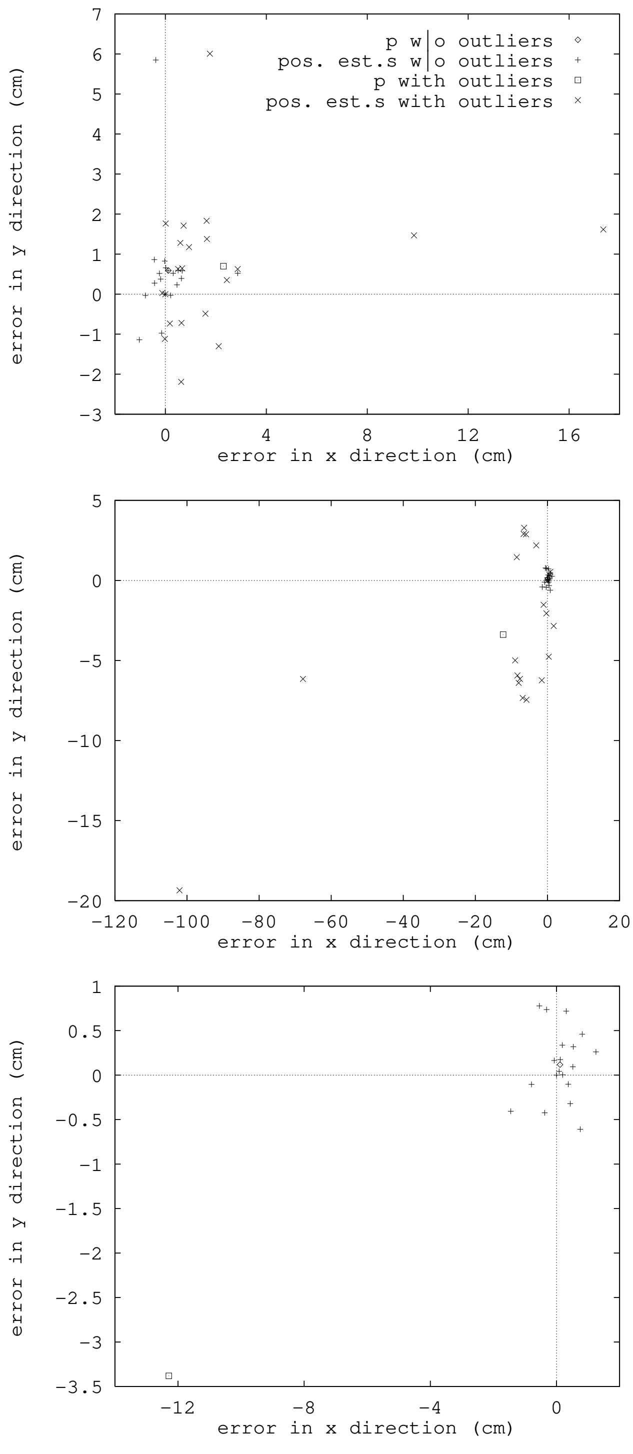

Figure 3-16: Two experiments with 21 landmarks including two outliers. In the first experiment (top), position estimate p computed with outliers is at $(2.3\ cm, 0.7\ cm)$ and p computed without outliers is at $(0.1\ cm, 0.6\ cm)$. In the second experiment (middle and a blowup at bottom), position estimate p computed with outliers is at $(-12.3\ cm, -3.4\ cm)$ and p computed without outliers is at $(0.1\ cm, 0.1\ cm)$.

the history of position estimates, the commands that make the robot move, and the uncertainties in these commands. Procedure POSITION ESTIMATOR could be incorporated into a navigation algorithm that keeps track of this information. One common approach is Kalman filtering.

Chapter 4

Visual Landmark Recognition

4.1 Introduction

The field of automated object recognition is one of the most complex areas in computer vision and image understanding. Object recognition based on matched filtering has been a very active research area in computer vision for many years. Matched filtering has been used much earlier in the areas of radar, sonar, and signal processing [87]. Valuable information for visual object recognition can be obtained from that literature.

Although template matching has been widely used in computer vision [95, 10, 104], a crucial problem with the method is the size of the search space [81, 77]. There are several approaches published in the literature that either reduce the size of the search space or that direct the search towards areas in the search space for which a match is more likely [51, 52, 86, 81, 7]. In this chapter a new approach is proposed that uses both such techniques. We discuss the problem of how certain landmarks, for example traffic signs, can be recognized by an autonomous vehicle or robot. For this particular application, a five-dimensional search-space is sufficiently large for robust object recognition and small enough for efficient object recognition.

The method presented constructs templates *on-line* during the search. The algorithm uses an efficient local definition of the correlation coefficient to evaluate the match. The algorithm presented correctly finds the location, shape, size, and orientation of objects. If enough independent information is contained in a template image, it can be matched with an object in an image uniquely. False positive matches occur only for objects that have very small information content. To avoid false matches, templates with insufficient information content should not be used for recognition tasks. We describe how to compute the information content of template images.

Although the main objective of this work is to describe a new approach to the general problem of visual object recognition, the solution to the special problem of recognizing traffic signs is significant by itself. Automatically recognizing traffic signs in images is very valuable for mobile robot or autonomous vehicle navigation. A robot that can recognize a traffic sign as a familiar landmark in its map of the environment can then use this information to localize itself in its environment [16, 22]. Our method stands apart from previous approaches to traffic sign recognition because first, it is efficiently applied to real-world landscape images (as opposed to Ettinger's isolated signs [44]), and second, it does not rely on color perception which is very sensitive to lighting changes. This sensitivity limits the approach of May [84] and Zheng et al. [105] who address the problem of recognizing traffic signs using color information.

The optimization technique *fast simulated annealing* is applied to avoid the cost of brute-force search by directing the search successfully. It reduces the search time by orders of magnitude. Recent publications in the sonar literature [32, 71] show that fast simulated annealing has been very successful in coherent signal extraction and localization in noisy environments. We use it in a similar way for incoherent image processing. Kirkpatrick et al. [63] show how to implement a Metropolis algorithm [85] to simulate annealing of combinatorial optimization problems. Szu and Hartley [101] propose an inverse linear cooling schedule for simulated annealing. This version is called "fast simulated annealing." The original slower version of simulated annealing has been applied to segmentation and noise reduction of degraded images by Geman and Geman [49], to represent lobed objects by Friedland and Rosenfeld [46], and to boundary detection by Geman et al. [48]. However, for visual object recognition, fast simulated annealing has not yet been exploited.

There are other filters besides a matched filter (which we use) for detecting target objects in images [59, 80, 58, 70]. These filters have not been shown to perform significantly better on real-world data than the basic matched filter.

This chapter is organized in the following way: An overview of the recognition system is given in Section 4.2. Section 4.3 describes the matched filter. Section 4.4 shows how templates are generated from model images. Section 4.5 examines the search space of the recognition problem and introduces "ambiguity surfaces." Section 4.6 describes our simulated annealing algorithm and Section 4.7 reports our experimental results. Section 4.8 analyzes the error in the correlation and proposes how to avoid false matches. Section 4.9 describes our results on noisy images. We conclude with a summary of this work and suggestions how to apply these results to other problems.

4.2 The recognition system

An object in an image I is defined to be recognized if it correlates highly with a template image T of the hypothesized object. This template image T is a transformed version of the model of the hypothesized object. Model images of objects are stored in a library.

Our recognition system takes as an input the scene image and the library of model images and outputs a description of the recognized object in the scene image (see Figure 4-1).

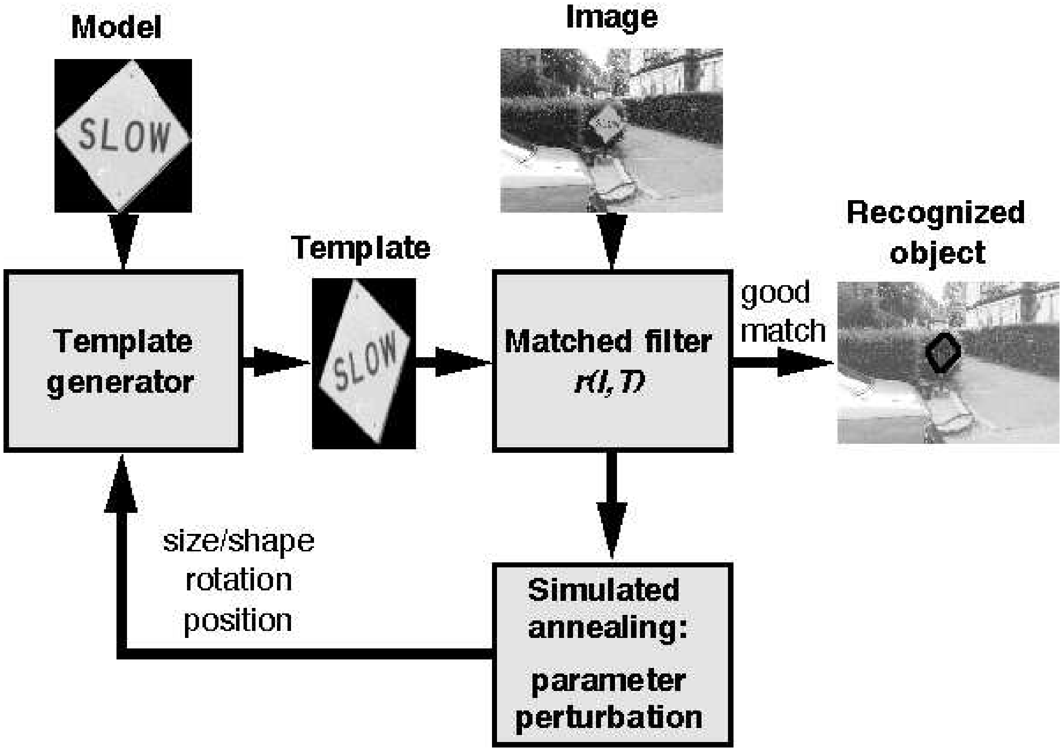

Figure 4-1: The traffic sign recognition system.

The system contains three components: a template generator (see Section 4.4), a matched filter (see Section 4.3), and a parameter perturbation component (see Section 4.6).

An initial guess for the parameters that describe the object is used by the template generator to choose a model image and to transform it into a template image. This template and an initial guess for the position of the object in the scene image is then used by the matched filter to evaluate the match. If the match is good, the system outputs a description of the recognized object. Otherwise, the parameters that describe the hypothesized object and its position in the scene image are perturbed using simulated annealing. With these perturbed parameters, a new template is created and tested at a new position in the scene image. This process is reiterated until a good match is found or until the search has been unsuccessful for a certain amount of time. In the latter case, a different model image is chosen and the process repeated.

4.3 The matched filter

A scene image I and a template image T are compared to evaluate how well a hypothesized object in the template image matches with an object in the scene. A template $T(x,y)$, for $0 \leq x < n_T, 0 \leq y < m_T$, is generally much smaller than the image $I(x,y)$. The template is compared with the part $I_T(x,y)$ of image $I(x,y)$ that contains the hypothesized object. Assuming pixel (x_0, y_0) is at the lower-left corner of the hypothesized object in I, subimage I_T is defined to be

$$I_T(x,y) = I(x_0 + x, y_0 + y) \qquad \text{for} \quad 0 \leq x < n_T, 0 \leq y < m_T.$$

We use the normalized correlation coefficient as a measure of how well images I_T and T correlate or match. For images I_T and T, the normalized correlation coefficient ρ is the covariance of I_T and T normalized by the standard deviation of I_T and T. The correlation coefficient is dimensionless, and $|\rho| \leq 1$. The correlation coefficient measures how accurate image I_T can be approximated by template T. Image I_T and template T are perfectly correlated if $\rho = 1$. We approximate ρ using the sampled coefficient of correlation

$$r = \frac{p_T \sum_{x,y} I_T(x,y)T(x,y) - \left(\sum_{x,y} I_T(x,y)\right)\left(\sum_{x,y} T(x,y)\right)}{\sqrt{p_T \sum_{x,y} I_T(x,y)^2 - \left(\sum_{x,y} I_T(x,y)\right)^2}\sqrt{p_T \sum_{x,y} T(x,y)^2 - \left(\sum_{x,y} T(x,y)\right)^2}}$$

where p_T is the number of pixels in the template image T with nonzero brightness values and $p_T \leq n_T \cdot m_T$. Note this last condition means that not all the pixels in images T and I_T are actually compared but only the nonzero pixels in T with the corresponding pixels in I_T. This is important, for example, if the template contains a circular object. Here pixels in T bordering the circle (or the background) will be zero (black). The computation time of r is proportional to the number of pixels in the hypothesized object, which is usually much smaller than the number of pixels in I. Using the correlation as a measure of successful recognition is also advantageous because it is a very robust measure. That is, it is relatively insensitive to fluctuations in the environment compared to higher resolution methods, as is well documented in spectral, bearing, and range estimation problems [60, 9].

4.4 Generating templates from model images

A template $T(x,y)$ is generated from a model image $M(x,y)$ by choosing three parameters that describe a transformation from M into T. The parameters determine how the model is sampled, and if necessary, how it is interpolated to generate the template. The parameters used are a rotation parameter τ and two sampling parameters s_x and s_y.

For notational convenience, we define the origin of a coordinate system for model image $M(x,y)$ to be in the middle of the image, i.e., $M(x,y)$ is defined for $-(n_M - 1)/2 \leq x \leq (n_M - 1)/2$ and $-(m_M - 1)/2 \leq y \leq (m_M - 1)/2$ for n_M, m_M odd. Then the rotation parameter τ determines how the x and y axes of $M(x,y)$ are rotated to define the x and y axes of $T(x,y)$. More precisely, given vectors

$$\mathbf{m_x} = \left(\frac{n_M - 1}{2}, 0\right) \quad \text{and} \quad \mathbf{m_y} = \left(0, \frac{m_M - 1}{2}\right),$$

which lie on the coordinate axes of M, and model radius $R_M = \sqrt{\left(\frac{n_M-1}{2}\right)^2 + \left(\frac{m_M-1}{2}\right)^2}$, we compute vectors

$$\mathbf{t_x} = R_M\,(\cos\tau, \sin\tau) \quad \text{and} \quad \mathbf{t_y} = R_M\,(-\sin\tau, \cos\tau)$$

which define the coordinate axes of the template image T in continuous space. The axes of T always span the model object as show in Figure 4-2.

The sampling parameters s_x and s_y determine how many samples along vectors $\mathbf{t_x}$ and $\mathbf{t_y}$ are used for the template image, respectively. The spacing between the samples along $\mathbf{t_x}$ is $((n_M - 1)/2)/s_x$. If there is a pixel in $M(x, y)$ after every $(n_M - 1)/(2s_x)$ step along $\mathbf{t_x}$, its brightness is used to define T along its x-axis. For example this scenario may occur if $\tau = 45$ degrees, and $s_x = (n_M - 1)/2$. As shown in Figure 4-2, if $s_x = (n_M - 1)/4$ the model is down-sampled and transformed into a template that is about one-quarter the size of the model. Pixels of zero brightness are added where necessary as shown in Figure 4-2.

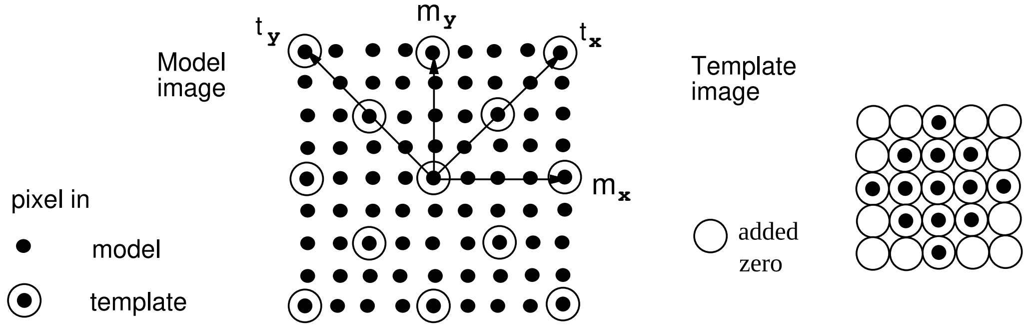

Figure 4-2: A 5×5 template image is obtained from a 9×9 model image using parameters $s_x = s_y = 2$ and $\tau = 45$ degrees.

In general, there may not be a pixel in M at the sampling point on vector $\mathbf{t_x}$. If this is the case, we use a four-point interpolation to define the brightness for the template at that point. Similarly, M is sampled (and if necessary interpolated) along vectors $\mathbf{t_y}, -\mathbf{t_x}$, and $-\mathbf{t_y}$ to obtain the brightness of the template pixels along the template coordinate axes. The rest of the template is now determined from M along the grid that is defined by the samples on the template coordinate axes.

Since the sampling rates s_x and s_y in the template coordinate system are different in general, the template is a rotated, scaled, and uniformly deformed version of the model. More parameters would be needed to describe more general non-uniform and non-linear deformations of the model. A straightforward extension would be to add a fourth parameter to obtain a non-uniform linear deformation of the model. However, for our purposes, the transformation described is sufficient because the objects to be recognized are usually flat, normal to the viewing direction and far away from the camera compared to the object size. Our method computes the template very quickly by sweeping over the model image only once. The time for creating a $n_T \times m_T$ template image is $O(n_T m_T)$.

Examples of a model and corresponding transformed templates are shown in Figure 4-3. The first two templates are scaled by $s_x = s_y$ and are not rotated. The remaining templates in Figure 4-3 are defined by more general transformations with $s_x \neq s_y$.

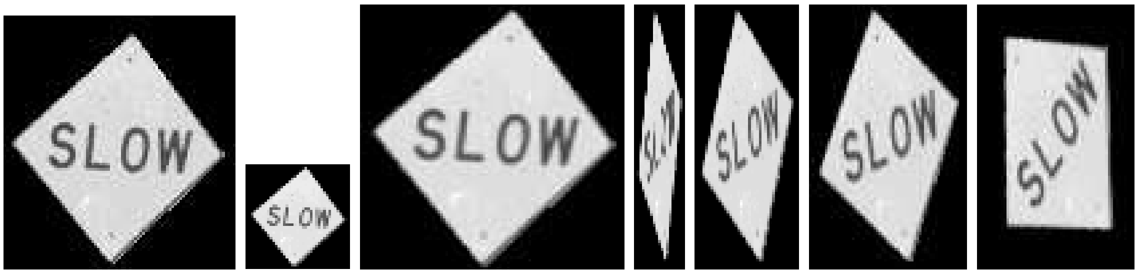

Figure 4-3: Model of slow sign with 101×111 pixels, and six templates of slow sign. Templates are obtained by sampling model sign at various sampling rates and degrees of rotation.

4.5 The parameter search space

The space of possible solutions of the recognition problem is extremely large, even if a particular object is known to be in the image a priori. The dimension of the search space is determined by the number of possibilities for position, size, shape, and orientation of the object. The number of possibilities for the position of the centroid of the object in the image is $O(n^2)$ for a $n \times n$ image. Assuming that the size and shape of the object can be approximated by sampling the model along two perpendicular axes as described in the previous section, the number of possibilities to approximate the size and shape of the object is also $O(n^2)$. Even with this assumption, the number of possible angles is still very large; since the image is discrete, we assume that the number of possible angles is $O(n)$. Thus, the size of the search space is $O(n^5)$ for an $n \times n$ image. For a typical image of size 256×256, the search space has a size of order 10^{14}. An exhaustive search of this space would take too long to find a good match between templates and images.

We use terminology from the radar and sonar literature to describe the search space. We call the space an *ambiguity surface.* A peak in the ambiguity surface means that the correlation coefficient is high for a particular set of parameters. Figure 4-4 shows an example of a two-dimensional ambiguity surface with a peak shown in black. There may be several peaks in an ambiguity surface. If the template and the object in the image match perfectly, the cross-correlation between template and image results in a peak in the ambiguity surface which is the global optimum. Due to noise and reduction of the search space by our template transformation, we do not expect a perfect match. However, in most cases the global optimum corresponds to a correct match or recognition.

As we can also see in Figure 4-4, an iterative search for a peak in the ambiguity

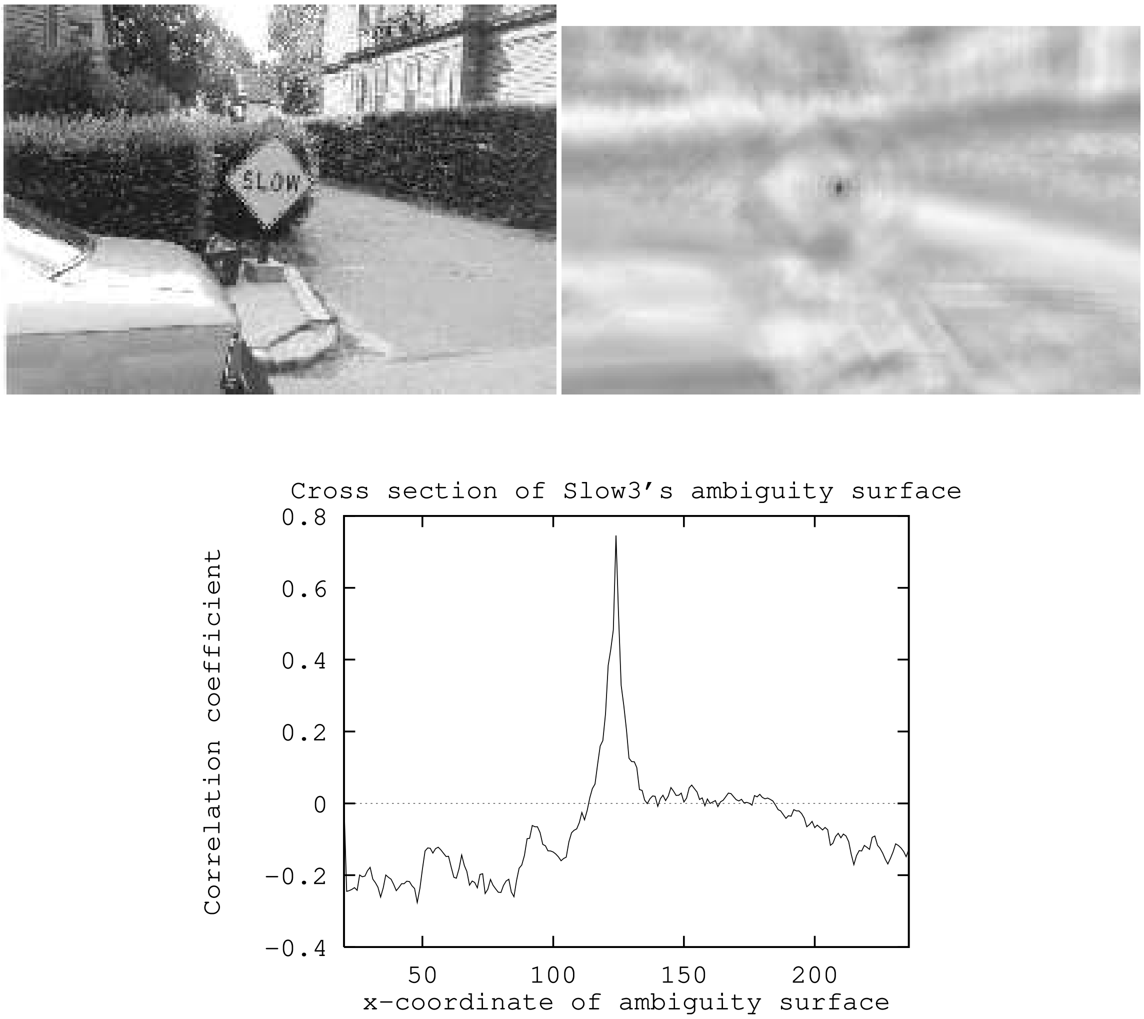

Figure 4-4: On the top left, image Slow3. On the top right, the ambiguity surface of image Slow3 computed for all possible translations given fixed angle and scaling parameters. On the bottom, a slice of the ambiguity surface of image Slow3 for a fixed y-coordinate. A deterministic search would compute each value on the ambiguity surface. A steepest ascent procedure would fail because of local maxima. Therefore, a stochastic search is used to find the best correlation value (here the darkest pixel value).

surface such as steepest ascent would fail because it would get "stuck" in local maxima. Simulated annealing, however, is able to "jump" out of local maxima and find the globally best correlation value.

4.6 The simulated annealing algorithm

In this section we describe our algorithm for finding an optimal match between images and templates. Our algorithm is based on a fast version of simulated annealing. Simulated annealing has become a popular search technique for solving optimization problems. Its name originates from the process of slowly cooling molecules to form a perfect crystal. The cooling process and its analogous search algorithm is an iterative process, controlled by a decreasing temperature parameter. At each iteration, our algorithm generates templates on-line as described in Section 4.4. New test values for the location, sampling, and rotation parameters of the template are randomly perturbed from current values. If the correlation coefficient r_j increases over the previous coefficient r_{j-1}, the new parameter values are accepted in the j-th iteration (as in the gradient method). Otherwise, they are accepted if

$$e^{-(r_{j-1}-r_j)/T_j} > \xi \tag{4.1}$$

where ξ is randomly chosen to be in $[0,1]$, T_j is the temperature parameter, and r_j is the sampled correlation coefficient in the j-th iteration. For a sufficient temperature this allows "jumps" out of local maxima. We choose

$$T_j = T_0/j \qquad\qquad 1 \leq j \leq L$$

as the cooling schedule for the j-th update of the temperature parameter where T_0 is the initial temperature and L is the number of iterations during the search. Note that the rate at which the temperature decreases is inverse linear and converges faster than an often used logarithmically inverse cooling schedule [101, 49]. As a criteria for stopping the annealing process, we simply put a limit on the search length L. Although this does not ensure convergence to the optimal correlation coefficient, the solutions we obtain for the parameters are generally sufficient and solve the recognition task.

As Kuperman et al. [71] point out, if the search problem involves different kinds of parameters the annealing algorithm is rather analogous to the cooling of a mixture of liquids, each of which have different freezing points. Therefore, at a specific temperature we do not combine the test for the choice of the location, sampling, and rotation angle. Instead, we first evaluate Equation (4.1) after having perturbed the location parameters, then after having perturbed sampling rates s_x and s_y, and finally after having perturbed rotation angle τ. We also obtain good results using simulated annealing only for the location parameters, and a gradient ascent with large enough random perturbations [32] for the remaining parameters.

To properly deal with image boundaries of an image $I(x,y)$ for which $0 \leq x < n_I$ and $0 \leq y < m_I$, we use the following formula to perturb the x-coordinate c_x of the

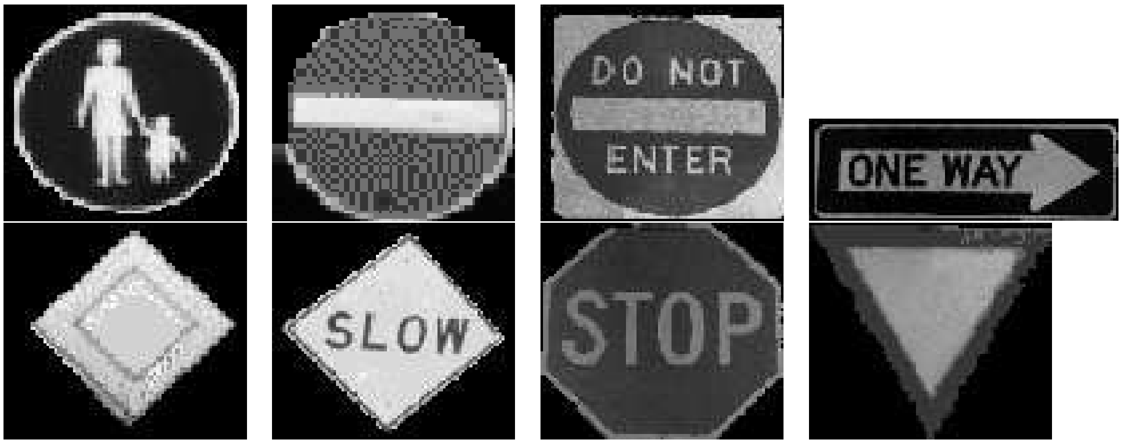

Figure 4-5: Model images used in experiments: Footpath, E-no-entry, No-entry, One-way, Priority, Slow, Stop, and Yield.

centroid position of a template with radius R_T in image $I(x, y)$

$$c_x = \begin{cases} c_x & \text{if } c_x - R_T \geq 0 \text{ and } c_x + R_T \leq n_I \\ -c_x & \text{if } c_x + R_T < 0 \text{ and } c_x - R_T \geq -n_I \\ 2n_I - c_x & \text{if } c_x - R_T > n_I \text{ and } c_x + R_T \leq 2n_I \\ n_I/2 & \text{otherwise (unlikely perturbation).} \end{cases}$$

The y-coordinate c_y of the centroid of the template is perturbed similarly. This formula avoids attracting the centroid position to the rim or corners of the image.

4.7 Experimental results

The algorithm described above was implemented on a Sun workstation and on a Silicon Graphics Iris. We used the model images shown in Figure 4-5 to find templates that correlate optimally with the scene images shown in Figures 4-6–4-9. The images are quantized using 256 grey levels. The size of the model images is 122×117 pixels (except for the one-way sign, which has 178×60 pixels.) The size of the scene images varies between 100×70 and 516×365 pixels.

For all scene images, the shape, size, orientation, and location of any traffic sign is found if it is known a priori what kind of sign to look for. For example, using the stop sign model shown in Figure 4-5 the algorithm finds the stop sign in a complicated scene image like image Stop5. (This is the last image in the second row of images in Figure 4-9; see also Figure 4-10). The stop sign in scene image Stop5 is recognized although the stop sign model was constructed from a picture of a completely different stop sign. Note that the stop sign in image Stop5 has graffiti, while the model sign does not.

For the more general problem of recognizing *which* object is in a scene image (i.e., not knowing the kind of traffic sign a priori), we ran 144 experiments with 18 scene

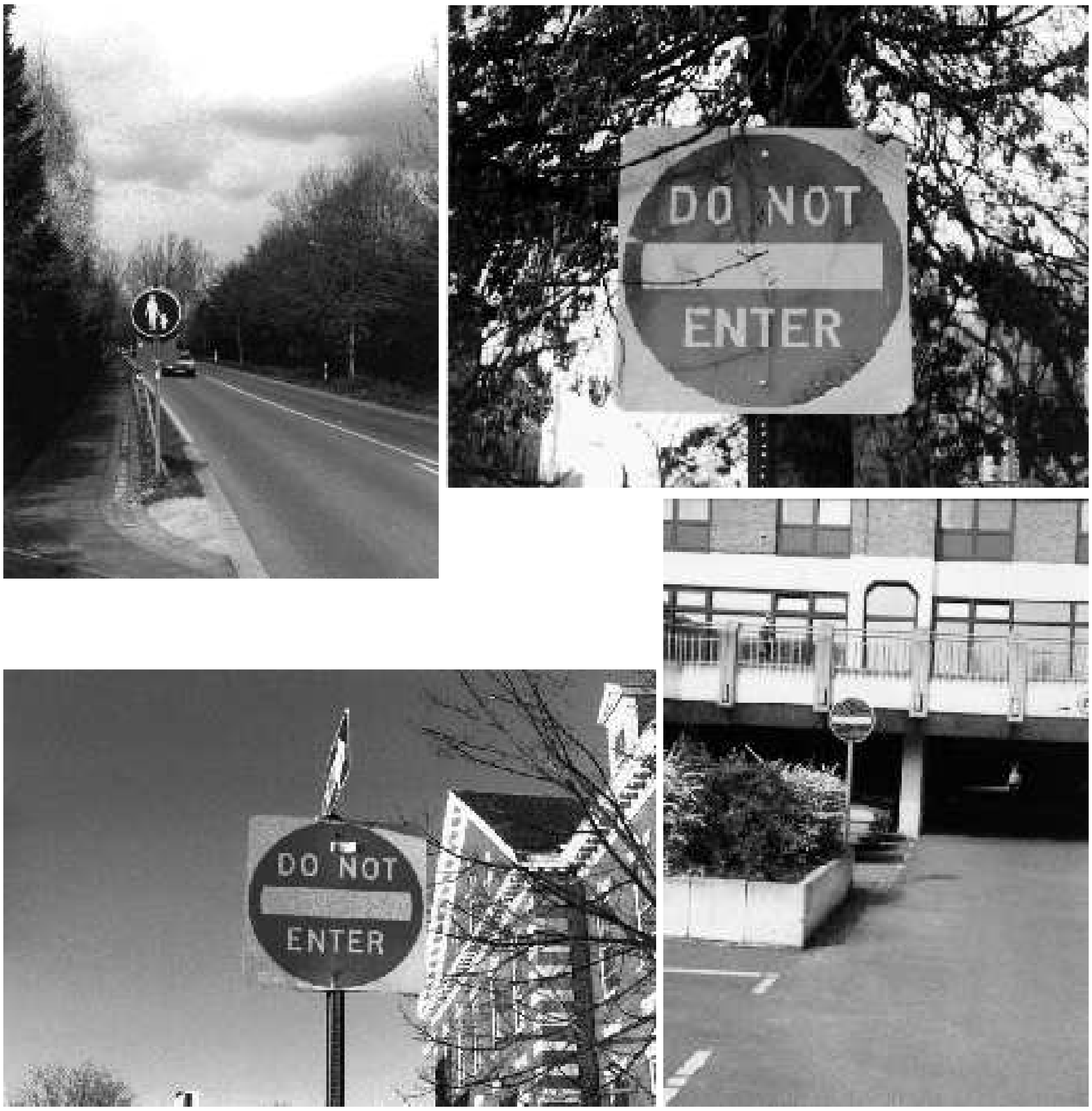

Figure 4-6: Scene images Footpath, No-entry1, No-entry2, and E-no-entry.

Figure 4-7: Scene images One-way, Priority1–Priority3, Slow3 and Slow4.

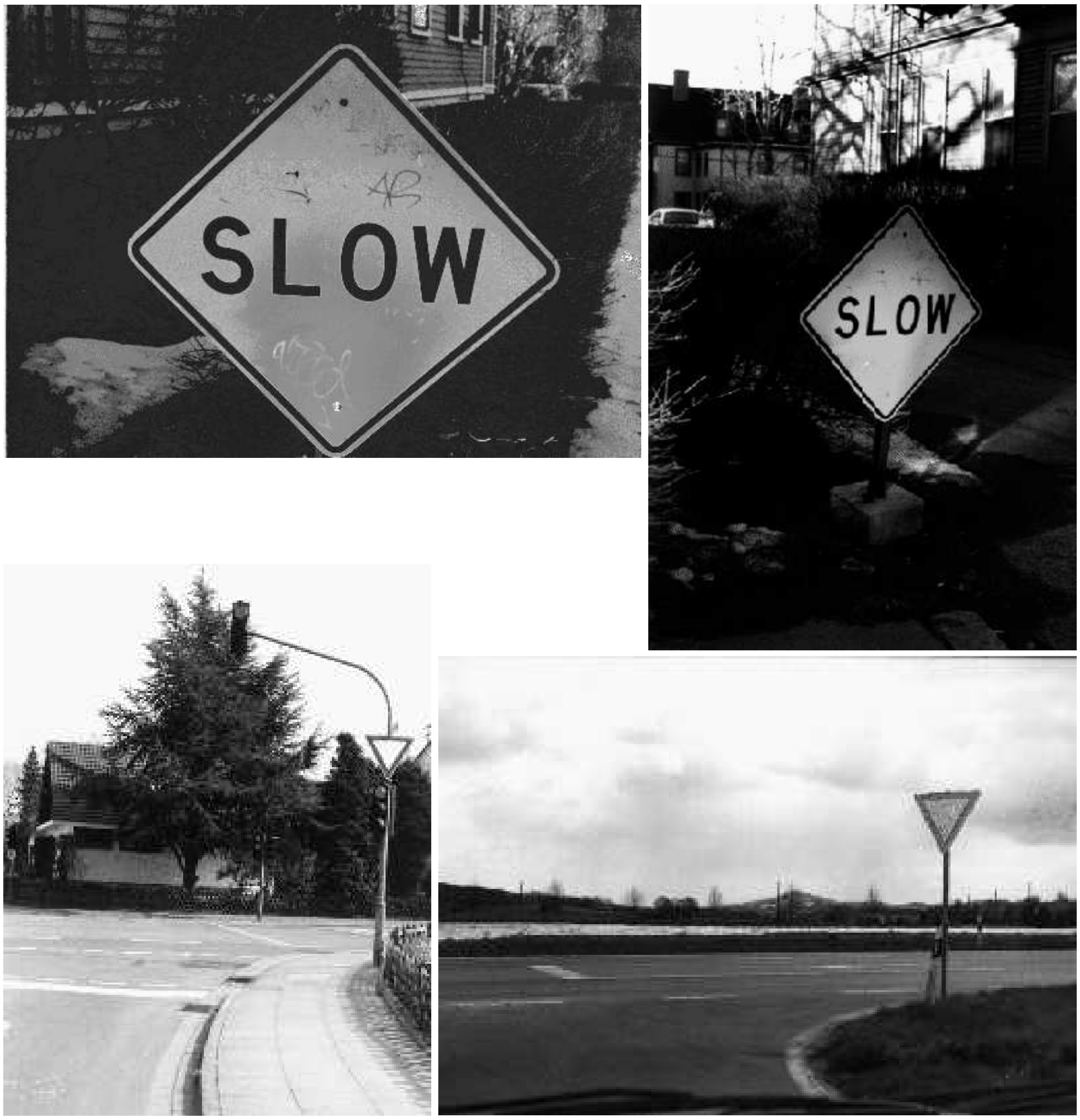

Figure 4-8: Scene images Slow1, Slow2, Yield3 and Yield2.

Figure 4-9: Scene images Stop1–Stop5.

images and 8 model images. Table 1 contains the correlation values obtained in the experiments. For each scene image, our algorithm computes the highest correlation coefficient among the set of values obtained for each model (boldface values in Table 1). The model corresponding to the maximum correlation value is selected as the sign recognized in the scene image. For most scene images, the correlation coefficient is highest if a match between a sign in the image and its corresponding template occurs. Only for three images, Slow2, Stop4, and Stop5, a false positive match occurs because the best correlation coefficient is not the one for the corresponding model. We show correct and false positive matches in Figure 4-10.

There are two facts that contribute to the false positive matches. First, some models do not have enough structure by themselves and match easily with arbitrary parts of the images. For example, the European no-entry sign's white middle bar matches with the roof of a car in image Stop5, as shown in Image 5 of Figure 4-10. In Section 4.8.2 we analyze this problem quantitatively. Second, some models look quite different from the actual landmark in the scene image. For example, as mentioned before, the stop sign model does not have any graffiti while the signs in Stop4 and Stop5 do. The templates constructed from the model stop sign do not match the stop signs in images Stop4 and Stop5 well enough to result in a correlation coefficient larger than the one obtained with the model E-no-entry (see Image 4 and 5 of Figure 4-10). One could try to solve this problem by making a model of each traffic sign (including its graffiti) in the environment. However, this would result in

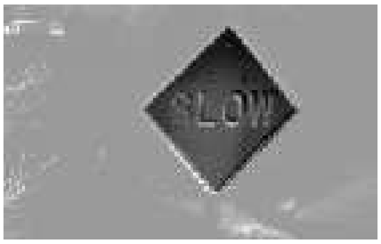

Image 1 Image 2 Image 3

Image 4

Image 5

Figure 4-10: Images 1 and 2 show templates constructed from models Slow and Yield overlying the sign in image Slow2 (correlation values 0.56 and 0.58, respectively.) Image 3 is a cropped image of Stop4 illustrating the best match with a template made from the Stop model. For images Stop4 and Stop5, templates generated from model image E-no-entry yield better correlation values than templates generated from model image Stop. A correct and a false positive match in scene image Stop5 are shown in Images 4 and 5 (correlation values 0.51 and 0.73, respectively.).

TABLE 1
Correlation Values for Recognition Task

Images	Models							
	Foot-path	E-no-entry	No-entry	One-way	Priority	Slow	Stop	Yield
Footpath	**0.77**	0.59	0.38	0.37	0.46	0.29	0.35	0.62
E-no-entry	0.49	**0.73**	0.39	0.43	0.46	0.26	0.38	0.62
No-entry1	0.22	0.21	**0.67**	0.31	0.24	0.18	0.17	0.40
No-entry2	0.29	0.18	**0.84**	0.37	0.14	0.26	0.23	0.35
One-way	0.37	0.55	0.24	**0.70**	0.40	0.38	0.31	0.58
Priority1	0.36	0.49	0.34	0.35	**0.58**	0.32	0.30	0.44
Priority2	0.46	0.54	0.40	0.45	**0.66**	0.29	0.32	0.31
Priority3	0.37	0.57	0.40	0.39	**0.62**	0.34	0.37	0.56
Slow1	0.25	0.29	0.25	0.25	0.45	**0.74**	0.15	0.38
Slow2	0.38	0.48	0.39	0.39	0.32	0.56 2nd	0.21	**0.58**
Slow3	0.39	0.58	0.41	0.38	0.40	**0.75**	0.30	0.59
Stop1	0.41	0.47	0.42	0.30	0.22	0.25	**0.69**	0.58
Stop2	0.23	0.16	0.27	0.25	0.18	0.11	**0.38**	0.30
Stop3	0.26	0.20	0.33	0.19	0.13	0.00	**0.34**	0.19
Stop4	0.42	**0.73**	0.46	0.50	0.43	0.32	0.56 3rd	0.66
Stop5	0.43	**0.73**	0.44	0.48	0.29	0.31	0.51 3rd	0.65
Yield1	0.45	0.75	0.39	0.50	0.53	0.32	0.37	**0.78**
Yield2	0.42	0.73	0.39	0.50	0.43	0.32	0.36	**0.82**

a huge library of signs which would increase the search time substantially. Moreover, the environment may change and outdate the library quickly. Therefore, we instead propose to select a small number of model images with *high information content* (see Section 4.8.2) so that false positive matches are avoided.

4.7.1 Illumination changes

The correlation coefficient $\rho(I_T, T)$ measures not only how accurate image I_T can be approximated by template T, but also how accurate image I_T can be approximated by a linear function of T, since $\rho(I_T, T) = \rho(I_T, aT + b)$ for some constants a, b. Therefore, the correlation coefficient is invariant to constant scale factors in brightness. Thus recognition is not affected by new lighting conditions that mainly result in such brightness changes.

4.7.2 Simulated annealing vs. exhaustive search

We also implemented an exhaustive search of the entire parameter space to compare its running time to our fast simulated annealing algorithm. The comparison of our simulated annealing algorithm and exhaustive search demonstrates the advantage of simulated annealing. We used image Noentry2 which has 112×77 pixels. The search space had about 6.8×10^7 sets of parameters. It took 15 seconds to recognize the sign using our simulated annealing algorithm. In contrast, exhaustive search found the sign after more than 10 hours of computation time.

Figure 4-11 illustrates how fast our simulated annealing algorithm recognizes a sign in a scene image.

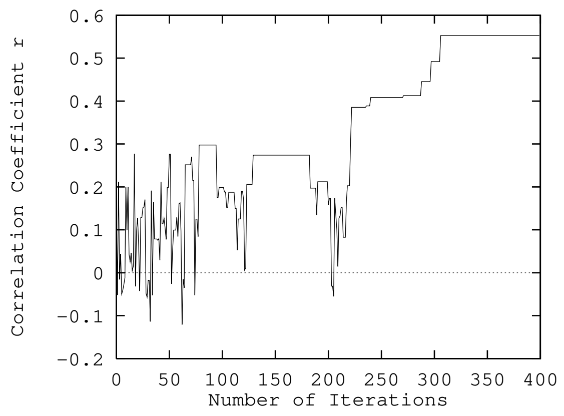

Figure 4-11: A typical run of our simulated annealing algorithm. The sign is found after about 300 iterations (ca. 15 s).

4.8 Avoiding false matches

4.8.1 Digitization error

The error in the sampled coefficient of correlation r increases if the number of pixels p_T in the image window considered decreases. For large samples of p_T pixels the error of r can be expressed as the mean squared error (MSE)

$$E[(r-\rho)^2] = \frac{1-\rho^2}{\sqrt{p_T}}$$

(see Figure 4-12 and Weatherburn [103]). As Weatherburn points out, the sampling distribution of r is never even approximately normal. The probability curve is very skewed in the neighborhood of $\rho = \pm 1$, even for large samples.

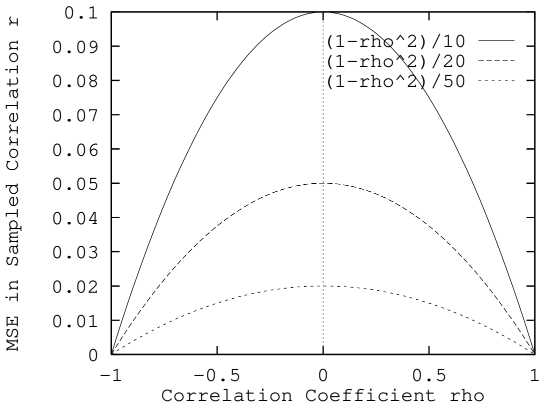

Figure 4-12: Mean squared error of r for $p_T = 100, 400$ and 2500.

4.8.2 Resolution and degrees of freedom

The *normalized auto-correlation* of model image $M(x, y)$ is

$$R(\tau_x, \tau_y) = (\sum_x \sum_y M(x, y)M(x - \tau_x, y - \tau_y)) / \sum_x \sum_y (M(x, y))^2.$$

Examples of auto-correlation images are shown in Figure 4-13. For notational convenience, we define $a = \sum_x \sum_y (M(x, y))^2$. Note that

$$R(x, y) = \frac{1}{a} M(x, y) * M(-x, -y).$$

Therefore, the Fourier transform

$$F_R(\omega_1, \omega_2) = \frac{1}{(2\pi)^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} R(x, y) e^{-j\omega_1 x} e^{-j\omega_2 x} d\omega_1 d\omega_2$$

of the auto-correlation $R(x, y)$ can be written as

$$F_R(\omega_1, \omega_2) = \frac{1}{a} F_M(\omega_1, \omega_2) F_M(-\omega_1, -\omega_2)$$

where $F_M(\omega_1, \omega_2)$ is the Fourier transform of $M(x, y)$. Since $M(x, y)$ is real, we can relate the magnitude $|F_M(\omega_1, \omega_2)|$ to the spectrum $F_R(\omega_1, \omega_2)$ of the auto-correlation:

$$F_R(\omega_1, \omega_2) = \frac{1}{a} F_M(\omega_1, \omega_2) F_M^*(\omega_1, \omega_2) \quad (4.2)$$

$$= \frac{1}{a} |F_M(\omega_1, \omega_2)|^2. \quad (4.3)$$

Therefore, the frequency content of the model image $M(x, y)$ can be expressed by the magnitude $|F_M(\omega_1, \omega_2)|$ as well as the spectrum $F_R(\omega_1, \omega_2)$.

Our goal is to measure the resolution, or *coherence area* A, of model $M(x, y)$

without having to compute Fourier transforms. Therefore, we compute the coherence area A by using Parseval's Theorem to relate $R(x,y)$ and $F_M(\omega_1,\omega_2)$:

$$A = \sum_x \sum_y |R(x,y)|^2 = \frac{1}{(2\pi)^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} |F_R(\omega_1,\omega_2)|^2 d\omega_1 d\omega_2$$
$$= \frac{1}{(2\pi a)^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} |F_M(\omega_1,\omega_2)|^4 d\omega_1 d\omega_2$$

Given the coherence area A and the number of pixels n of $M(x,y)$, the number of *coherence cells* is

$$c = n/A.$$

The number of coherence cells is equivalent to the number of degrees of freedom of the model image. It can be used as a measure of the information content of the model image.

We examine the information content of each model image to evaluate how useful the model image is for the recognition task. All our model images $M(x,y)$ have the same number of pixels n. Model images with low resolution (little structure) such as the European No-entry and Yield signs, do not have enough information content for robust object recognition. This, and the mean squared error in r for small p_T, are responsible for the false positive matches reported in Table 1. In order to avoid false matches, we need to avoid using such model images with low information content.

The models that contribute to the false positive matches, E-no-entry and Yield, have a coherence area of 313 and 197, respectively. This is much higher than the coherence area for models with more reliable matching results. For example, the Footpath and Stop signs' auto-correlation falls off much faster; their coherence areas are 148 and 56, respectively. The number of coherence cells in E-no-entry is 297 and in Yield 473, but in Footpath it is 628 and in Stop, even 1641.

Thus, the number of coherence cells is a quantitative measure for determining if a model has enough information content to be useful as a template. Most of the models we use have a large enough number of coherence cells for robust detection, but subsequent downsampling in generation of templates may corrupt this.

4.9 Results on noisy images

Gaussian noise is added to the brightness values of some of the scene images to examine the robustness of our algorithm. The algorithm is able to find the sign even in strongly degraded pictures. The signal-to-noise ratio (SNR) of a noisy image is defined as $10 \log$ of the variance of the noiseless image over the variance of the noise.

Several noisy images are obtained by corrupting image Slow3 by zero-mean Gaussian noise with various signal-to-noise ratios. Our results for image Slow3 are summarized in Figure 4-14. Note that the correlation increases as the signal-to-noise ratio increases.

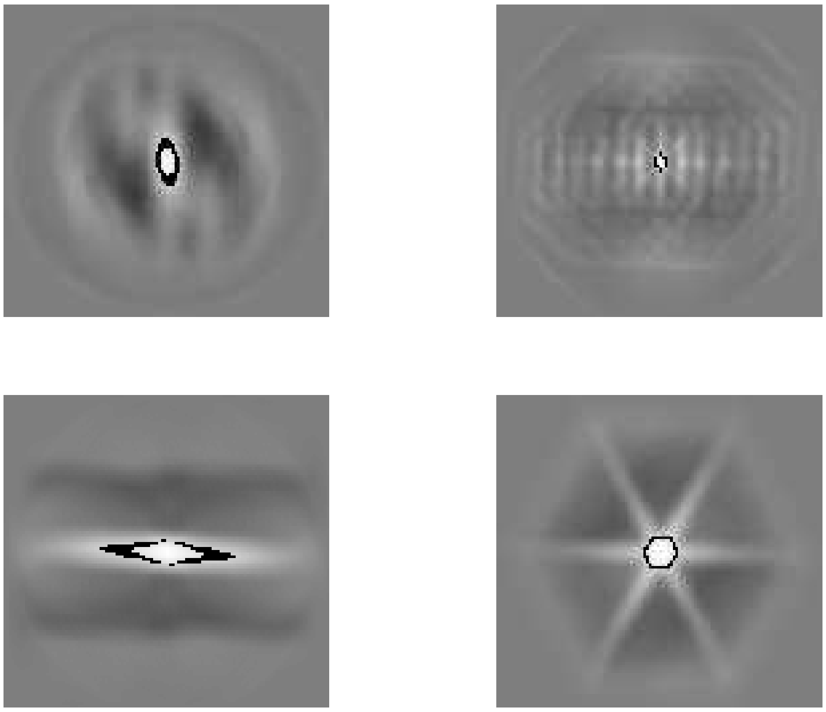

Figure 4-13: On the top, auto-correlation of model images Footpath and Stop. The numbers of coherent cells are $c = 628$ and $c = 1,641$ respectively. On the bottom, auto-correlation of model images E-no-entry and Yield. The numbers of coherent cells are $c = 297$ and $c = 473$ respectively. To illustrate how fast the auto correlation falls off, the e-folding lengths, i.e., pixels (x, y) with $R(x, y) \approx 1/e$, are shown on a dark contour.

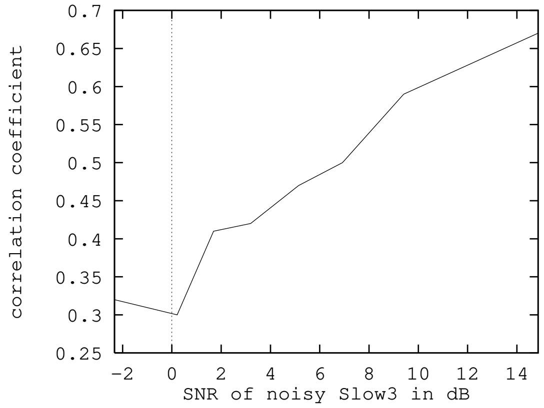

Figure 4-14: Correlation coefficient for sign recognition in noisy versions of image Slow3.

Figures 4-15 and 4-16 show images Slow3 and Slow4 corrupted by Gaussian noise with zero mean and SNR 0 dB and 3 dB, respectively. Matches for pictures with lower SNR are possible for templates with greater information content than those presented. (In radar and sonar, signals with negative SNR are commonly extracted given sufficient information content.)

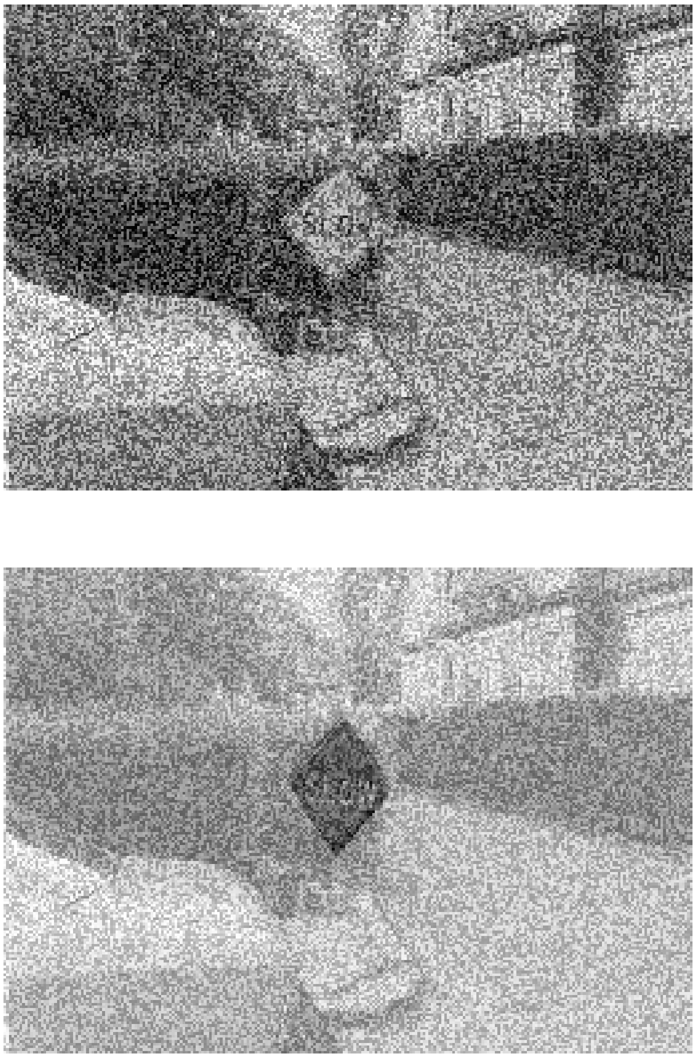

Figure 4-15: On the top, image Slow3 degraded by Gaussian noise with zero mean and SNR 0 dB. On the bottom, an image illustrating that the object is recognized where the template computed is shown overlying the recognized sign in the scene. (This image is shown brighter so that the overlying template can be illustrated better.)

4.10 Conclusions

Our method has been shown to efficiently recognize objects in complicated landscapes in the presence of noise. To our knowledge, our work is the first to apply fast simulated

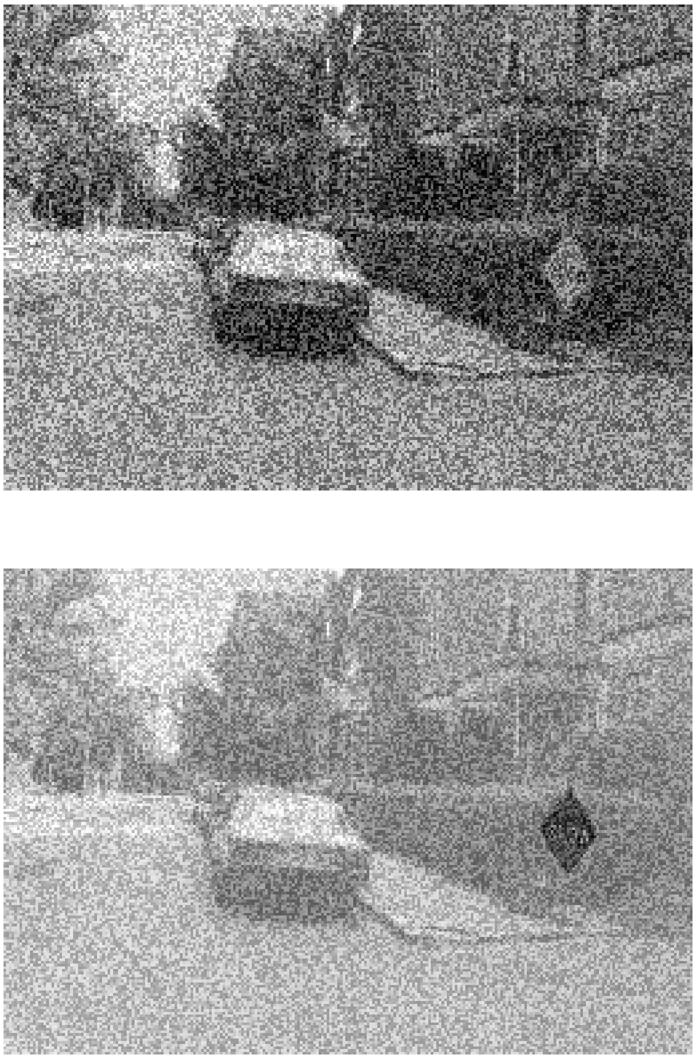

Figure 4-16: On the top, image Slow4 degraded by Gaussian noise with zero mean and SNR 3 dB. On the bottom, an image illustrating that the object is recognized where the template computed is shown overlying the recognized sign in the scene. (This image is shown brighter so that the overlying template can be illustrated better.)

annealing to object recognition. Our results show that it makes the parameter search of object recognition feasible.

We strongly advocate the use of template matching in recognition tasks and provide quantitative techniques to analyze its limits. We show how to measure the information content of templates as a way to make the recognition algorithm robust.

For the application of traffic signs, we have shown that the search space can be successfully reduced by using a three parameter transformation from model image to template. This method is well suited for recognition tasks that involve objects with scale and shape variations. The method is so efficient that templates can be constructed on-line during the search.

For future work, severe illumination variations within the object and occlusion problems can be addressed. Other applications of our method, for example in medical computer vision and in face recognition, are being investigated. We believe that with our method we can generalize common approaches to recognize faces [26] by using images that are not normalized but contain more general scenes with varied backgrounds.

Chapter 5

Conclusions

The central question addressed in this thesis is how a navigating robot can cope with uncertainty. This thesis provides both answers to short-term implementation issues, as well as solutions to difficult long-term problems.

Uncertainty during mobile robot navigation occurs for many reasons. There can be uncertainty in the robot's environment. The environment may be unknown to the robot. Or, the robot's model of its surroundings is an inaccurate description of a dynamic world. In an uncertain environment the robot's actions have an uncertain outcome which makes planning a navigation route difficult.

Another source of uncertainty is the robot's hardware. A robot cannot accurately execute its commands. Sensors and actuators never work perfectly.

The thesis has addressed how to reduce uncertainty in visual sensor information, so that the robot can recognize objects in its surroundings. The object recognition method presented in the thesis searches for a template of an object that matches with the image taken by the robot. Template matching has been widely used in computer vision. However, a crucial problem with the method is the size of the search space. Therefore, our approach uses "fast simulated annealing" to reduce the search time. The method works well for noisy real-world images with complicated backgrounds.

We have several ideas for future work. Severe illumination variations within the object to be recognized and occlusion problems can be addressed. Our system should be tested on a physical robot. Our method uses only a single image. It could be extended to a system that recognizes objects in a robot's environment, making use of sequences of images. The expectation is that using a sequence of images of the robot's surroundings not only speeds up recognition, but also eliminates false recognition. Thus, uncertainty in the robot's understanding of its environment can be reduced.

In another part of the thesis we have shown how sensory feedback and understanding of the robot's environment can reduce uncertainty due to actuator inaccuracies. Errors in executing the robot's commands accurately lead to uncertainty in the robot's position and orientation in its environments. The thesis has presented a method to estimate the location of the navigating robot in its environment. The robot's environment consists of landmarks that are represented on a 2-D map of the environment. The robot measures bearings of the landmarks in the environment and uses these measurements to estimate the robot's position and orientation with respect to its map of the environment. Our method is very efficient and handles measurement errors well. For future work, the history of position estimates during navigation could be incorporated. The method should also be tested on a physical robot.

The location method presented in the thesis is based on prior knowledge of the robot's environment. The most difficult scenario for a navigating robot is the one in which it does not know its environment a priori. For this scenario we introduced the "confident explorer model."

For this model, there is no uncertainty in the position of the robot in its environment. Sensors and actuators are assumed to work accurately enough, so that the robot is able to explore its unknown environment and, at all times, knows its location in the explored part of the environment.

Within this model we developed algorithms for "piecemeal learning" - the robot explores its unknown environment in pieces. Our best algorithm for piecemeal learning of environments that can be modeled as arbitrary graphs runs in almost linear time. For special environments that can be modeled as "city-block graphs," our algorithms run in linear time. Further research is needed either to show that arbitrary graphs cannot be explored in linear time, or to provide a linear time algorithm for piecemeal exploration of arbitrary unknown graphs.

The part of the thesis on piecemeal learning is of a theoretical nature since we wanted to focus on the algorithmic design of efficient piecemeal exploration methods. However, the "piecemeal constraint" is motivated by practical restrictions, for example, finite battery power. Real robots cannot be "confident explorers." They have to deal with uncertainty in the world and uncertainty in their perception of the world. The piecemeal learning algorithms we presented are aimed at robots of the future that have such precise sensors that they can indeed be confident in their sensory input. However, for piecemeal exploration of certain environments, for example, exploration of a huge system of pipes in a factory [24], current technology should be sufficient. It would be interesting to implement our piecemeal algorithms on a physical robot to explore such an environment.

Bibliography

[1] Arthur Albert. *Regression and the Moore-Penrose Pseudoinverse.* Academic Press, New York, 1972.

[2] David Avis and Hiroshi Imai. Locating a robot with angle measurements. *Journal of Symbolic Computation*, 10:311–326, 1990.

[3] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. In *Proceedings of the 1995 Conference on Computational Learning Theory*, Santa Cruz, California, July 1995.

[4] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. Technical Report MIT/LCS/TM-516, MIT, January 1995.

[5] Baruch Awerbuch and Robert G. Gallager. Distributed BFS algorithms. In *Proceedings of the 26th Symposium on Foundations of Computer Science*, pages 250–256, October 1985.

[6] Baruch Awerbuch and Robert G. Gallager. A new distributed algorithm to find breadth first search trees. *IEEE Transactions on Information Theory*, IT-33(3):315–322, 1987.

[7] Nicholas Ayache and Olivier D. Faugeras. HYPER: A new approach for the recognition and positioning of two-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54, January 1986.

[8] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, October 1993.

[9] Arthur B. Baggeroer, William A. Kuperman, and Peter N. Mikhalevsky. An overview of matched field methods in ocean acoustics. *IEEE Journal of Oceanic Engineering*, 18(4):401–424, 1993.

[10] Dana H. Ballard and Christopher M. Brown. *Computer Vision.* Prentice-Hall, 1982.

[11] E. Bar-Eli, Piotr Berman, A. Fiat, and Peiyuan Yan. On-line navigation in a room. In *Proceedings of the Symposium on Discrete Algorithms*, pages 237–249, 1992.

[12] Kenneth Basye, Thomas Dean, and Leslie Pack Kaelbling. Learning dynamics: system identification for perceptually challenged agents. Technical report, Brown University.

[13] Reinhold Behringer and Stefan Hötzel. Simultaneous estimation of pitch angle and lane width from the video image of a marked road. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pages 966–973, Munich, Germany, September 1994.

[14] Michael A. Bender and Donna K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proceedings of the Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 75–85, November 1994.

[15] Margrit Betke. Algorithms for exploring an unknown graph. Master's thesis, MIT Department of Electrical Engineering and Computer Science, February 1992. (Published as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-536, March, 1992).

[16] Margrit Betke and Leonid Gurvits. Mobile robot localization using landmarks. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pages 135–142, Munich, Germany, September 1994. To appear in *IEEE Transactions on Robotics and Automation*.

[17] Margrit Betke and Nicholas C. Makris. Fast object recognition in noisy images using simulated annealing. Technical Report AI-Memo 1510, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, December 1994.

[18] Margrit Betke and Nicholas C. Makris. Fast object recognition in noisy images using simulated annealing. In *Proceedings of the Fifth International Conference on Computer Vision*, Cambridge, Massachusetts, June 1995.

[19] Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. In *Proceedings of the 1993 Conference on Computational Learning Theory*, pages 277–286, Santa Cruz, California, July 1993. (Published as MIT AI-Memo 1474, CBCL-Memo 93).

[20] Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2/3):231–254, February/March 1995.

[21] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. In *Proceedings of Twenty-Third ACM Symposium on Theory of Computing*, pages 494–504. ACM, 1991.

[22] David J. Braunegg. MARVEL: a system for recognizing world locations with stereo vision. Technical Report 1229, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1990.

[23] Rodney A. Brooks. Aspects of mobile robot visual map making. In Hideo Hanafusa and Hirochika Inoue, editors, *Robotics Research*. MIT Press, 1985.

[24] Rodney A. Brooks. Private communication, 1995.

[25] Rodney A. Brooks, Jonathan H. Connell, and Peter Ning. Herbert: a second generation mobile robot. Technical Report AI-Memo 1016, MIT Artificial Intelligence Laboratory, January 1988.

[26] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1042–1052, October 1993.

[27] Philippe Burlina, Daniel DeMenthon, and Larry S. Davis. Navigation with uncertainty: I. Reaching a goal in a high collision risk region. Technical Report CAR-TR-565, CS-TR-2717, University of Maryland at College Park, June 1991.

[28] Raja Chatila and Jean-Paul Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 138–145, St. Louis, Missouri, March 1985.

[29] Frédéric Chenavier and James L. Crowley. Position estimation for a mobile robot using vision and odometry. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2588–2593, Nice, France, 1992.

[30] Wei-Pang Chin and Simeon Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[31] Ruel V. Churchill. *Introduction to Complex Variables and Applications*. McGraw-Hill, 1948.

[32] Michael D. Collins, Jonathan M. Berkson, William A. Kuperman, Nicholas C. Makris, and John S. Perkins. Applications of optimal time-domain beamforming. *Journal of the Acoustical Society of America*, 93(4):1851–1865, April 1993.

[33] Jonanthan Connell. A colony architecture for an artificial creature. Technical Report AI-TR 1151, MIT AI Laboratory, 1989.

[34] Simon Cooper and Hugh Durrant-Whyte. A Kalman filter model for GPS navigation of land vehicles. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pages 157–163, Munich, Germany, September 1994.

[35] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[36] J. J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, Reading, Massachusetts, 1989.

[37] Jill D. Crisman and Charles E. Thorpe. SCARF: a color vision system that tracks roads and intersections. *IEEE Transactions on Robotics and Automation*, 9(1):49–58, February 1993.

[38] James Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, volume 2, pages 674–680, Scottsdale, Arizona, May 1989.

[39] Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. How to learn an unknown environment. In *Proceedings of the 32nd Symposium on Foundations of Computer Science*, pages 298–303, San Juan, Puerto Rico, October 1991.

[40] Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, volume I, pages 355–361, 1990.

[41] Michael Drumheller. Mobile robot localization using sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(2):325–332, March 1987. Also published as MIT AI-Memo 826, January 1985.

[42] Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese Postman. *Mathematical Programming*, 5:88–124, 1973.

[43] Alberto Elfes. A sonar-based mapping and navigation system. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[44] Gil J. Ettinger. Large hierarchical object recognition using libraries of parameterized model sub-parts. In *IEEE Proceedings of Computer Vision and Pattern Recognition*, pages 32–41, June 1988.

[45] John Evans. HelpMate^{TM}: An autonomous mobile robot courier for hospitals. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pages 1695–1700, Munich, Germany, September 1994.

[46] Noah S. Friedland and Azriel Rosenfeld. Lobed object delineation using a multipolar representation. Technical Report CS-TR-2779, Center of Automation Research, University of Maryland, October 1991.

[47] Gellert, Küster, Hellwich, and Kästner, editors. *Mathematik.* VEB Verlag Enzykopädie, Leipzig, 1967.

[48] Donald Geman, Stuart Geman, Christine Graffigne, and Ping Dong. Boundary detection by constrained optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):609–628, July 1990.

[49] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, November 1984.

[50] Russell Greiner and Ramana Isukapalli. Learning useful landmarks. Technical report, Siemens Corporate Research, Princeton, 1993.

[51] W. Eric L. Grimson. On the recognition of parameterized 2D objects. *International Journal on Computer Vision*, 3:353–372, 1988.

[52] W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press, 1990.

[53] Leonid Gurvits and Margrit Betke. Robot navigation using landmarks. In preparation, 1995.

[54] Thomas R. Hancock and Stephen J. Judd. Hallway navigation using simple visual correspondence algorithms. Technical Report SCR-94-479, Siemens Corporate Research, Princeton, 1993.

[55] Christian Hofner and Günther Schmidt. Path planning and guidance techniques for an autonomous mobile cleaning robot. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 1, pages 610–617, Munich, Germany, September 1994.

[56] Berthold K. P. Horn. *Robot Vision*. MIT Electrical Engineering and Computer Science Series. The MIT Press/McGraw-Hill, 1986.

[57] Ian D. Horswill and Rodney A. Brooks. Situated vision in a dynamic world: chasing objects. In *AAAI 88, The Seventh National Conference on Artificial Intelligence*, volume 2, Saint Paul, Minnesota, August 1988.

[58] Bahram Javidi, Philippe Refregier, and Peter Willett. Optimum receiver design for pattern recognition with nonoverlapping target and scene noise. *Optics Letters*, 18(19):1660–1662, 1993.

[59] Bahram Javidi and Jun Wang. Optimum filter for detection of a target in nonoverlapping scene noise. *Applied Optics*, 33(20):4454–4458, 1994.

[60] J. H. Johnson. The application of spectral estimation methods to bearing estimation problems. *Proceedings of the IEEE*, 70(9):1018–1028, 1982.

[61] Stephen Judd. Private communication, 1993.

[62] Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 441–447, Austin, Texas, 1993.

[63] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, pages 671–680, May 1983.

[64] Lindsay Kleeman. Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2582–2587, Nice, France, 1992.

[65] Rolf Klein. Walking an unknown street with bounded detour. *Computational Geometry: Theory and Applications*, 1(6):325–351, June 1992.

[66] Jon M. Kleinberg. On-line search in a simple polygon. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Arlington, Virginia, 1994.

[67] Akio Kosaka and Avinash Kak. Fast vision-guided mobile robot navigation using model-based reasoning and reduction of uncertainties. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 56(3):271–329, 1992.

[68] Eric Krotov. Mobile robot localization using a single image. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 978–983, Scottsdale, Arizona, May 1989.

[69] Benjamin J. Kuipers and Tod S. Levitt. Navigation and mapping in large-scale space. *AI Magazine*, 9(2):25–43, 1988.

[70] B. V. K. Vijaya Kumar and J. D. Brasher. Relationship between maximizing the signal-to-noise ratio and minimizing the classification error probability for correlation filters. *Optics Letters*, 17(13):940–942, 1992.

[71] William A. Kuperman, Michael D. Collins, John S. Perkins, and N.R. Davis. Optimal time-domain beamforming with simulated annealing including application of a priori information. *Journal of the Acoustical Society of America*, 88(4):1802–1810, October 1990.

[72] Mei-ko Kwan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[73] Anthony Lazanas and Jean-Claude Latombe. Landmark-based robot navigation. Technical Report STAN-CS-92-1428, Stanford University, May 1992.

[74] John J. Leonard and Hugh F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE International Conference on Intelligent Robot Systems*, Osaka, Japan, November 1991.

[75] Tod S. Levitt, Daryl T. Lawton, David M. Chelberg, Kerry V. Koitzsch, and John W. Dye. Qualitative navigation II. In *Proceedings of the DARPA Image Understanding Workshop*, pages 319–326, Cambridge, Massachusetts, 1988.

[76] Tod S. Levitt, Daryl T. Lawton, David M. Chelberg, and Philip C. Nelson. Qualitative navigation. In *Proceedings of the DARPA Image Understanding Workshop*, volume 2, pages 447–465, Los Angeles, California, February 1987.

[77] Z.-Q. Liu and Terry M. Caelli. Multiobject pattern recognition and detection in noisy backgrounds using a hierarchical approach. *Computer Vision, Graphics, and Image Processing*, 44:296–306, 1988.

[78] Tomas Lozano-Pérez, Matthew W. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[79] Vladimir J. Lumelsky and Alexander A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(4):403–430, 1987.

[80] Abhijit Mahalanobis, B. V. K. Vijaya Kumar, Sewoong Song, S. R. F. Sims, and J. F. Epperson. Unconstrained correlation filters. *Applied Optics*, 33(17):3751–3759, 1994.

[81] Avraham Margalit and Azriel Rosenfeld. Using probabilistic domain knowledge to reduce the expected computational cost of template matching. *Computer Vision, Graphics, and Image Processing*, 51:219–234, 1990.

[82] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report AI-TR-1228, MIT, 1990.

[83] Larry Matthies and Steven A. Shafer. Error modelling in stereo navigation. Technical Report CMU/CS/86-140, Carnegie-Mellon University, 1986.

[84] Franz May. Vision system for safe driving. Presented at the Center for Biological and Computational Learning, MIT, September 1994.

[85] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Physics*, C–21, 1953.

[86] Roger N. Nagel and Azriel Rosenfeld. Ordered search techniques in templates matching. *Proceedings of the IEEE*, 60(2):242–244, 1972.

[87] A. V. Oppenheim, editor. *Applications of Digital Signal Processing.* Englewood Cliffs: Prentice Hall, 1978.

[88] Christos H. Papadimitriou. On the complexity of edge traversing. *Journal of the Association for Computing Machinery*, 23(3):544–554, July 1976.

[89] Christos H. Papadimitriou and Mihalis Yanakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[90] Nagewara S. V. Rao, Srikumar Kareti, Weimin Shi, and S. Sitharama Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, July 1993.

[91] Uwe Regensburger and Volker Graefe. Visual recognition of obstacles on roads. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pages 982–987, Munich, Germany, September 1994.

[92] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceeding of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 78–87, Los Angeles, California, October 1987.

[93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, April 1993.

[94] Azriel Rosenfeld. *Picture Processing by Computer*. New York, Academic Press, 1969.

[95] Azriel Rosenfeld and Avinash C. Kak. *Digital Picture Processing. Second Edition, Volume 2*. New York, Academic Press, 1982.

[96] Karen B. Sarachik. Characterising an indoor environment with a mobile robot and uncalibrated stereo. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 984–989, Scottsdale, Arizona, May 1989.

[97] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[98] Gilbert Strang. *Linear Algebra and its Applications*. Academic Press, New York, 1976.

[99] Kokichi Sugihara. Some location problems for robot navigation using a single camera. *Computer Vision, Graphics, and Image Processing*, 42:112–129, 1988.

[100] Karen T. Sutherland and William B. Thompson. Inexact navigation. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, volume 1, pages 1–7, Atlanta, Georgia, May 1993.

[101] H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122(3–4):157–162, June 1987.

[102] Yutaka Watanabe and Shin'ichi Yuta. Position estimation of mobile robots with internal and external sensors using uncertainty evolution technique. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 2011–2016, Cincinnati, Ohio, May 1990.

[103] C. E. Weatherburn. *A First Course in Mathematial Statistics.* Cambridge University Press, 1962.

[104] Leonid P. Yaroslavsky. *Digital Picture Processing.* Springer-Verlag Berlin, 1985.

[105] Yong-Jian Zheng, Werner Ritter, and Reinhard Janssen. An adaptive system for traffic sign recognition. In *Proceedings of the Intelligent Vehicles Symposium*, Paris, France, 1994.