A FRAMEWORK FOR ROADMAP-BASED NAVIGATION AND SECTOR-BASED LOCALIZATION OF MOBILE ROBOTS

A Dissertation

by

JINSUCK KIM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2004

Major Subject: Computer Science

A FRAMEWORK FOR ROADMAP-BASED NAVIGATION AND SECTOR-BASED LOCALIZATION OF MOBILE ROBOTS

A Dissertation

by

JINSUCK KIM

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Nancy M. Amato (Chair of Committee)	
Ricardo Gutierrez-Osuna (Member)	John C. Keyser (Member)
Reza Langari (Member)	Valerie E. Taylor (Head of Department)

August 2004

Major Subject: Computer Science

ABSTRACT

A Framework for Roadmap-Based Navigation and
Sector-Based Localization of Mobile Robots. (August 2004)
Jinsuck Kim, B.S., Inha University;
M.S., Purdue University
Chair of Advisory Committee: Dr. Nancy M. Amato

Personal robotics applications require autonomous mobile robot navigation methods that are safe, robust, and inexpensive. Two requirements for autonomous use of robots for such applications are an automatic motion planner to select paths and a robust way of ensuring that the robot can follow the selected path given the unavoidable odometer and control errors that must be dealt with for any inexpensive robot. Additional difficulties are faced when there is more than one robot involved.

In this dissertation, we describe a new roadmap-based method for mobile robot navigation. It is suitable for partially known indoor environments and requires only inexpensive range sensors. The navigator selects paths from the roadmap and designates localization points on those paths. In particular, the navigator selects feasible paths that are sensitive to the needs of the application (e.g., no sharp turns) and of the localization algorithm (e.g., within sensing range of two features). We present a new sector-based localizer that is robust in the presence of sensor limitations and unknown obstacles while still maintaining computational efficiency. We extend our approach to teams of robots focusing on quickly sensing ranges from all robots while avoiding sensor cross-talk, and reducing the pose uncertainties of all robots while using a minimal number of sensing rounds. We present experimental results for mobile robots and describe a web-based route planner for the Texas A&M campus that

utilizes our navigator.

ACKNOWLEDGMENTS

After I finished my master's degree, God led the way to Texas where I devoted six years to study. During this time, I faced several difficulties. Perhaps the greatest one was finding a reason to pursue the Ph.D. degree. To complete the requirements, I needed someone to teach me how to study, and why. My adviser, Dr. Nancy Amato, played the crucial role in the process.

First of all, she allowed me to join her research group, even before transferring from aerospace to the computer science department. Upon successful transfer, which was not possible without her help, I started my own graduate–level research. After that, she provided me with good explanations about the benefits of having a Ph.D. degree. Besides logical reasons, she showed how exciting one's life can be when dedicated to research and development. In advising my research, she allowed me to harmonize theory and application. This is important to me because I didn't have ample theoretical background. Using the three mobile robots that were available to me, I have experienced a good amount of algorithm development, software production, and hardware experimentations.

The office environment (room 407 of the Bright Building) has been a nice place for uninterrupted study and idea sharing with fellow students. All the people in the Parasol lab helped me fulfill the requirements, not to mention maintaining linux systems and being audiences of my practice talks. Finally, I'd like to thank my parents who have been great supporters for such a long time of study. My wife also has been a great helper and has provided me with a peaceful home and a hopeful heart.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

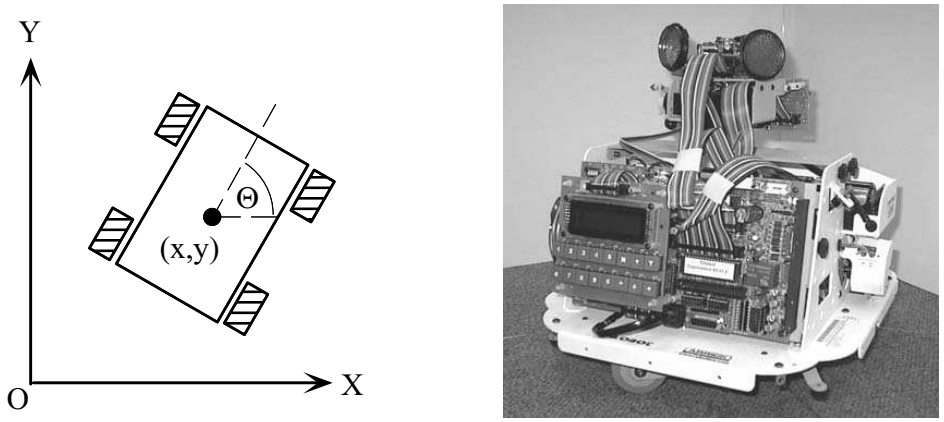
There is an increasing number of potential applications for autonomous mobile robots in indoor environments, ranging from cleaning, to surveillance, to search and rescue operations in burning buildings or hostage situations, to assisting the handicapped or elderly around the home. There are difficulties and challenges that must be addressed before we can realize these applications. In particular, personal robotics applications in indoor environments must be safe, robust, and inexpensive. Two requirements for autonomous use of robots for such applications are an automatic motion planner to select paths, and a robust way of ensuring that the robot can follow the selected path given the unavoidable odometer and control errors that must be dealt with for any inexpensive robot.

The mobile robot navigation problem is to first compute a route from the current position to the destination and then to safely move the robot along that route while interacting appropriately with other objects along the way. Among many issues relevant to autonomous operation, we focus on two main computational issues: path planning and localization. The *path planning* problem is to compute a valid (e.g., collision–free) path connecting the start and the goal configurations. The desired trajectory can be viewed as a sequence of robot configurations, each of which describes the robot's location (x, y) and orientation (Θ) with respect to the world coordinate system (see Figure 1(a)). When following the path, periodic *localization* is necessary to reset odometer errors, e.g., due to wheel slippage. Since GPS signals cannot be relied upon in an indoor environment, one low cost approach to gathering data for

---

The journal model is *IEEE Transactions on Automatic Control.*

localization is to use sensors attached to the robot that find the range (distance) to obstacles. An example of such a robot is shown in Figure 1(b). When there is more than one robot, additional constraints are placed on the path planning and path following problems.



(a) (b)

Fig. 1. (a) Three coordinates are used to represent the configuration of a mobile robot in a planar environment, and (b) mobile robot with three sonar sensors mounted on the pan/tilt head.

A. Global Navigation

Designing robust global navigation techniques for inexpensive mobile robots has been a challenge for scientists for many years. Path planning is known to be NP–hard in all but the most trivial situations [13, 67], and precise path following requires precise localization which can require state–of–the–art sensors that are expensive and sometimes reserved for military uses.

Our strategy is to address these issues simultaneously using a roadmap–based global navigator that has a global understanding of the environment. This understanding will be exploited to select paths which are feasible in terms of localization

costs and requirements. A *roadmap* for mobile robots is analogous to a map of the highway system used by automobile drivers. It contains a representative set of feasible paths, ideally covering the interesting regions of the environment. Using a roadmap, moving a robot from a start to a goal position is composed of three steps: moving from the start to a nearby node in the roadmap, traveling in the roadmap, and exiting the roadmap and moving to the goal. The path planner employs a *local planner* to connect the start (or the goal) to the roadmap, and the roadmap is typically explored using some graph traversal method such as Dijkstra's shortest path algorithm [36].

The main contribution of our work is to propose an integrated approach for path planning and path following, and to design novel methods for path planning and following that are sensitive to this interdependence. We believe an integrated approach is appropriate for many reasons, the most obvious being that there exist some cases where the independent execution of the path planner and path follower could fail to solve the problem. For example, suppose the path planner selects a route on which it is impossible to localize due to the physical limitations of the sensors. Our navigator is based on the philosophy that the "path planner and the path follower (localizer) should assist each other", which we introduced in [38]. Similar concepts were also proposed by Kuipers [47] in his "distinctive places" paper. Namely, the path planner's objective is to select paths that facilitate the localization and other optimization criteria. For example, if there are certain regions in which localization would not be possible, then the navigator should ensure the robot does not need to localize in those areas. To aid this process, useful information for localization is encoded in the roadmap and is used during path selection.

The navigator determines when to execute the path planning and path following modules using its global understanding of the problem and the environment. When a path is selected, the navigator also computes a *subgoal*: a node on the path that

the robot is expected to be able to move to safely from the start, and where it will localize and then wait for its next command. For example, the navigator may choose to localize before approaching a door so that the robot can pass near the center of the narrow region. Also, given a path, the robot could localize frequently or lazily depending on mission requirements. For example, if it is important to stay on the planned trajectory, then the subgoal must be placed so that the robot's error is kept small. Or if large clearances are important, then the path should be selected appropriately and localization should be performed frequently enough to ensure the required clearances are maintained.

An example is shown in Figure 2. Here, we have an artificial environment which represents a room with a few pieces of furniture. Three paths (P_1, P_2, P_3) and five obstacles ($O_1 - O_5$) have been purposefully placed to show that in some circumstances, a seemingly inefficient (long) path might be the only feasible path, and that in that case our navigator should select it. First, a basic distance–prioritized path planner will choose P_1 which is the shortest. The path has little clearance from obstacles O_1 and O_2, and is thus not safe unless the robot has very accurate odometry. Second, P_2 could also be a safe path. However, if the robot has poor odometry and short–range sensors, then it might not be able to follow the path because it might need to localize for safety reasons in a region in the environment where no obstacles or walls can be sensed. So, given these constraints, an intelligent navigator would choose P_3 which provides good scannable obstacles along the path. The robot can localize in the vertical direction when it is close to the upper wall so that its uncertainty region does not grow too much.

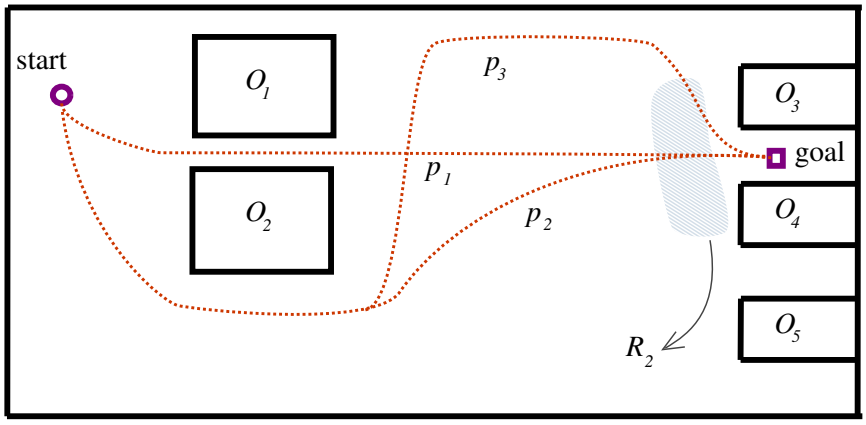


Fig. 2. Navigator chooses a good path for localization.

1. Roadmap–Based Path Planning

Path planning originated in robotics and has also been adapted to other areas such as autonomous transportation systems for automobiles or aircraft, military unmanned vehicles, and computer animations in the entertainment industry [10, 15, 43]. In these applications, paths must be found quickly in large search spaces, often high dimensional C-spaces (many degrees of freedom).

Roadmap-based path planning [49] is ideal for such scenarios [35]. The roadmap can be viewed as a graph in the robot's configuration space, where vertices correspond to robot configurations and edges correspond to paths connecting the configurations in the graph. The strength of roadmap–based planners is that the connectivity of a good roadmap can approximate the connectivity of the planning space. Later, queries can be processed quickly by connecting the start and the goal to the roadmap, and searching the roadmap for a path connecting the two connection points. In this work, we use roadmap–based path planners for our mobile robots, and focus on the issue of extracting a quality path from the roadmap.

There are two main issues that are of concern in searching for and following paths in the roadmap. First, roadmaps contain many possible routes connecting two different nodes. Depending on the graph search algorithm and the criteria applied, different paths connecting the same start and goal nodes can be found. Second, a path extracted from a roadmap is composed of multiple segments, and its quality could likely be improved by "smoothing" it using numerical optimization techniques [45, 48, 84]. These two properties are inherent in roadmap–based approaches. We call the first a global property because the chosen search method can result in large–scale changes in the path. We refer to the second as a local property because typically there are no topological differences between the extracted path and the optimal path. In designing our navigator, we take advantage of the global aspect of roadmaps which enables quick searches in large environments. However, the local property is still useful because it ensures that the robot can go back to the planned path after localization as long as the amount of error is within the granularity of the roadmap.

In this work, we first apply roadmap-based methods that have been developed for robotic manipulators with high degrees of freedom (see Chapter II Section F for previous work [4, 79]) to mobile robot applications and analyze their strengths and weaknesses. After that, we present a new approach for extracting an optimal path in the roadmap. Since we focus on the global planning ability of the roadmap, the paths are not smoothed.

2. Localization

Localization has received a tremendous amount of attention from researchers in mobile robotics. The localization method used primarily depends on the sensors with which the robot is equipped. Common sensor choices for mobile robots are optical sensors such as cameras using CCD arrays, and range sensors such as laser, IR, and

sonar sensors. Though recent research using a camera includes efficient localization methods [8, 82], due to the wealth of information, efficient processing using limited computing power is still not an easy task. In this work we use inexpensive range finders. Compared to vision sensors, an advantage of range sensors is that they are less susceptible to changes in lighting conditions due to, e.g., night, smoke from fire, etc.

To determine what types of *features* in the environment can be used for localization, we need to know the *visibility model* (or perceptual model) of the sensor. If the sensor is perfect, then the visibility model is trivial because it can detect all visible obstacles. Determining the visibility model for realistic sensors requires analyzing physical sensor limitations such as determining their maximum and minimum range. Since the information contained in the features is limited by the visibility model, localization algorithms should be designed to work efficiently given the information provided by the visibility model.

As the robot moves, a certain amount of potential error accumulates in the robot's knowledge of its position. The *uncertainty region* is a representation of the current possible error. In particular, it is the set of all possible positions of the robot at the given time. Localization is the process of reducing the robot's uncertainty region, usually using sensor measurements. There are many design factors such as on-line[1] or off-line localization, statistics-based or geometric-based approaches, topological or bitmap-based representations (maps) of the environment, whether the map of the environment is known, partially known, or unknown[2], the accuracy and resolution of sensors, etc.

---

[1]Localizing while the robot is moving

[2]This is the SLAM (simultaneous localizing and map building) problem.

In this work, we focus on sonar range sensors which are less expensive but also less accurate than laser range sensors. We rely on geometric operations rather than stochastic modeling, assume that the map is partially known (e.g., walls are known but the location of furniture is not), and that the mission does not require on–line localization (localize while moving). Building on previous work that assumed ideal visibility [52], in this research, we propose two localization methods that are applicable for more realistic visibility models. For each method we identify scannable features in the environment, develop algorithms, prove feasibility and robustness, and perform experiments. In addition, since we must deal with imperfect scan data, our methods are inherently robust to unknown obstacles, and have the potential to be used in or with map building algorithms.

B. Contribution

In this research, we make contributions in several areas relevant to mobile robotics.

- We describe a new roadmap-based method for mobile robot navigation that integrates the path planning and path following operations. In particular, the path planner selects feasible paths that are sensitive to the needs of the application (e.g., no sharp turns) and of the localization algorithm (e.g., within sensing range of two features).

- We present a new sector-based localizer that is robust to sensor limitations and unknown obstacles while maintaining computational efficiency. Our method has been verified by experiments with real mobile robots with sonar sensors.

- We describe an augmented version of Dijkstra's shortest path algorithm that is well suited for selecting paths from roadmaps that are sensitive to multiple

optimization criteria. Our path planner has been tested with roadmaps constructed using several strategies and it is used in a web-based route planner for the Texas A&M campus.

- We define and investigate two problems for multiple-robot localization: range sensing without sensor cross–talk, and reducing the pose uncertainties of all robots using a minimal number of sensing steps. We show both problems are NP-complete and then identify approximation algorithms for them.

Portions of this work have been published in the *IEEE International Conference on Robotics and Automation (ICRA)* and in the *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*: multiple robot localization [37], global planning [38], visibility sectors-based localization [39, 41] and path optimization [40].

C. Outline

This dissertation is organized as follows. We begin in Chapter II by discussing previous work related to our work. In Chapter III, our roadmap–based navigation strategy is introduced, and our optimal path extractor is described. Next in Chapter IV, we describe our sector-based localization methods that are sensitive to sensor limitations. We consider two problems related to multiple robot localization in Chapter V. Experimental results are presented in Chapter VI, including hardware experimental results using a mobile robot with sonar sensors in an office environment. In Chapter VII, we describe the 'Campus Navigator' which is a web–based application using our technique for extracting optimal paths from a roadmap subject to multiple optimization criteria. Tables defining the campus navigator's roadmap are described in Appendix A.

CHAPTER II

RELATED WORK

In this chapter, we describe preliminaries and previous work on mobile robot navigation that are related to our work. We begin by describing the physical limitations of odometry and range sensors, which are particular to mobile robotics. Then we discuss the mobile robot localization problem and present a summary of relevant related work. After localization–related topics, previous work related to path planning and roadmap methods is presented.

A. Odometry Error and Uncertainty Regions

The count of wheel revolutions provides positional information relative to the start location. This kind of measurement is called dead–reckoning. Most mobile robots have encoders to measure the rotation of their wheels. In the absence of odometer error, based on its kinematics, the robot's configuration can be precisely calculated from the encoder readings. Unfortunately, encoder readings are error prone. The causes of this error include issues such as wheel slippage, shifting payloads, inaccurate turns, irregular floor surfaces, change of tire radius, etc. These errors render it effectively impossible for a robot to precisely follow a planned path.

There are several algorithms that estimate the position and orientation uncertainty of a dead-reckoning robot [44, 75]. One choice is to represent the robot's position by an *uncertainty ellipse*, which is a simplified version of statistics-based models. The ellipse is centered about the nominal estimate and is the constant probability contour for a multivariate Gaussian distribution [69]. Usually, it is a conservative boundary containing all expected positions of the robot.

For a straight translational path, a simple way to represent the the robot's po-

sition uncertainty is to use two parameters corresponding to longitudinal and lateral deviation to compute minor and major axes of an uncertainty ellipse centered at the end of the path. As illustrated in Figure 3(a), actual parameters to compute an ellipse can be obtained by experiments. The ellipse boundary becomes conservative if the number of experiments are sufficiently large. In reality, it is more complex because the trajectory may be slightly circular and the ellipse grows nonlinearly with the robot's travel distance.

In this work, we take the simple geometric approach by assuming that the ellipse grows linearly. Exact uncertainty modeling is practically impossible, and is not a focus of our work. However, this method is similar to the covariance propagation of a Kalman filter since the elements of the covariance matrix (the square of the expected error) is increased by the square of the state propagator matrix, which is constant [61].

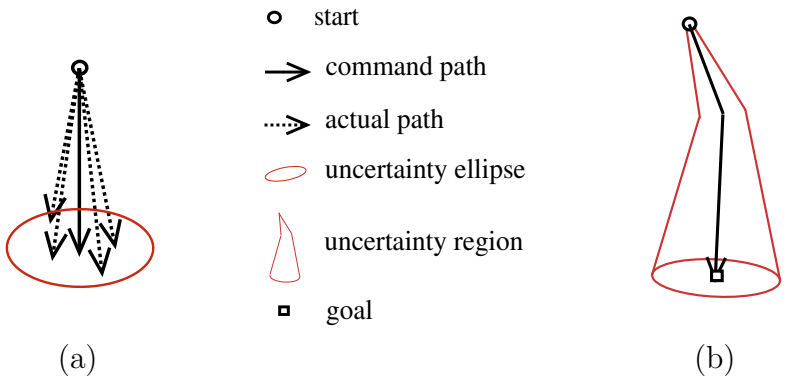


Fig. 3. Modeling robot position uncertainty. (a) Estimating uncertainty ellipse from a translation, and (b) uncertainty ellipses and region.

The *uncertainty region* is the union of the robot's uncertainty ellipses as it moves along a path, as shown in Figure 3(b). If the robot translates after rotation, the final uncertainty ellipse depends on both the rotation angle and the translation distance. This requires an uncertainty compounding technique [69] which is defined as follows.

Given the uncertainties in two coordinate frames, *compounding* collapses them into a single uncertainty that covers its components. This is different from uncertainty *merging* which produces a single uncertainty that is less than any of its components. In our work, we geometrically compound uncertainty ellipses by first identifying ellipses resulting from translation and rotation error, and then by overlapping them and finding an ellipse that includes all.

B. Sonar Sensor Limitations

Ultrasonic range finders detect proximity using the technique of time-of-flight measurement. Polaroid modules are popular devices for inexpensive mobile robots because of their ease of interfacing and relatively low cost. The amplitude profile in Figure 4 shows that the strength of the sonar signal is concentrated in the center of the beam with some side lobes.

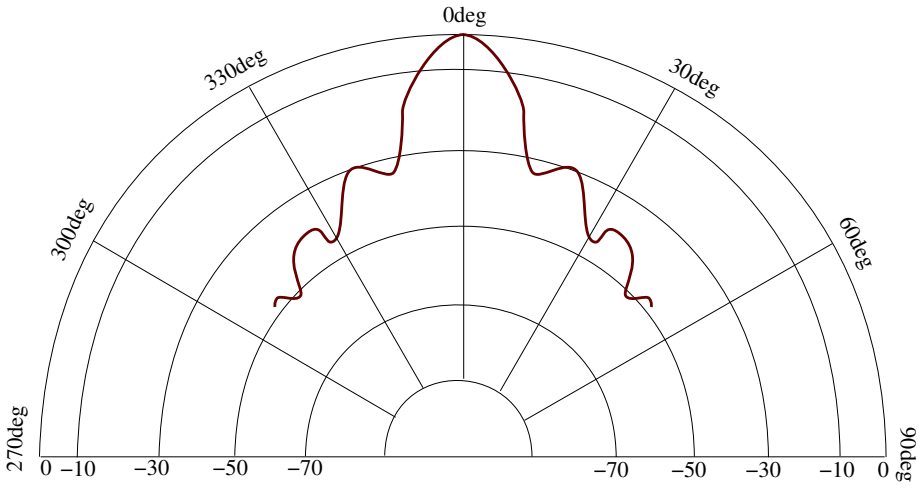
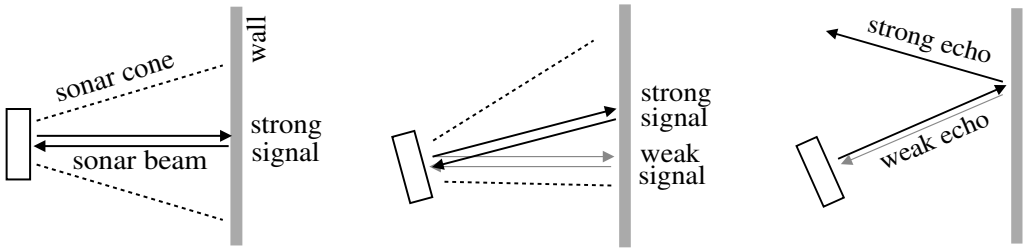


Fig. 4. Typical signal strength pattern for sonar sensors.

The signal emitting from a range sensor is actually like a cone in space. Figure 5 illustrates the signal projected to a plane. Sonar sensors have wider cones than IR



(a) (b) (c)

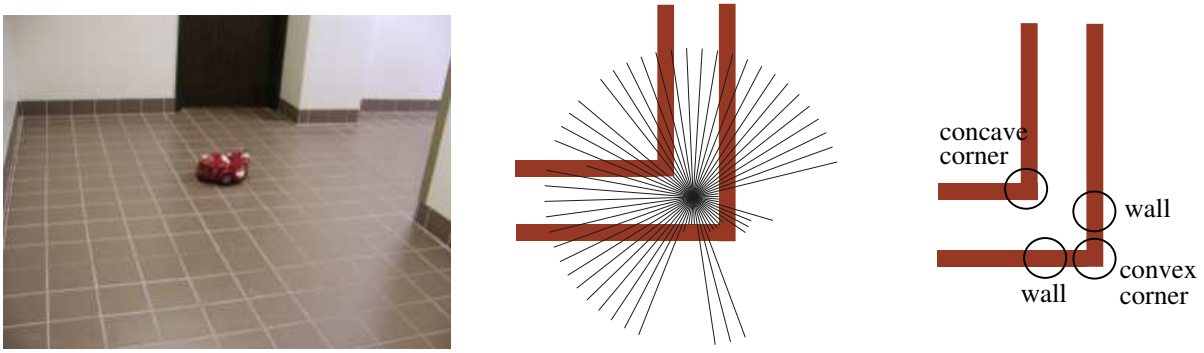
Fig. 5. Beam model of sonar wave. (a) Perpendicular case, (b) slight beam angle returns strong echo and correct measurement, and (c) incidence angle is too large.

sensors. For analytical purposes, we approximate this cone as a beam concentrated on the center axis of the cone and represent it by a line segment as shown by the strong signals in Figure 5(b). When the scan incidence angle is not exactly 90°, two echoes can return (weak first and strong second) to the transmitter. Depending on the calibration of the sensor, either the strong or weak signal can be processed. Our equipment has been tuned to respond to the strong signal only so that the angle of the beam is always roughly perpendicular to the sensor. When the sensor is rotated with a small angular step to measure a wall, in theory, the scan data from an ideal sensor plotted in Cartesian coordinates will align to a line segment.

Unfortunately, sonar sensors have several physical limitations that prevent them from always reliably capturing geometric features from the environment.

Incidence angle limitation. This limitation arises because the sonar beam does not return to the transmitter. For example, for our sensors, experiments indicate that the angle between the beam and wall must be greater than 80° to successfully identify the obstacle. This angle depends on the surface of the obstacles. Figure 5(c)

illustrates the case where the incidence angle is less than 80° and no return signal is received by the sensor. As a result of this limitation, only a small part of the wall can be correctly measured. Figure 6(b) shows an actual scan of the hallway shown in Figure 6(a). Taking these limitations into account, the scannable part of the environment (walls and corners) is indicated in Figure 6(c).



(a) (b) (c)

Fig. 6. Effect of sonar sensor incidence angle limitation. (a) The hallway environment in our lab. (b) Actual scan data from the hallway environment using sonar sensors. (c) Geometric features that can be scanned when taking the incidence angle limitation into account.

Range limitation. These are the limits on the minimum and maximum distances to the obstacle that can be measured by the sensor.

Sensing time. For inexpensive applications, we limit the number of range sensors installed on a robot. If the sensors have pre-determined angles of attachment as opposed to being mounted on a rotating head, the robot must rotate its body to align to a desired direction and take measurements. The time spent for a localization depends on whether the robot must be stationary when scanning, the number of measurements and rotations required to scan the region of interest, and if it is impossible to scan at an arbitrarily high resolution.

In general, the accuracy of measurement is decreased when the distance to the

obstacle is increased. Also, some sensors have an option to adjust the maximum range, such that the sensor resolution varies inversely with the maximum range. Since such properties are hardware–dependent, we do not consider the effect of distance on accuracy in our range sensor model, but we do assume each sensor has a minimum and maximum feasible range.

C. Mobile Robot Localization

Depending on the magnitude of the position uncertainty allowed, two variants of the localization problem have been studied. One is known as "pose maintenance" or "local localization" where *a priori* knowledge of the robot's approximate position is assumed [19]. The other approach is known as the "kidnapped–robot situation" or "global localization" where the pose uncertainty of the robot may not be known. Though the details of the estimation formula vary depending on the type of localization, practical localizers have three components in common: a component for extracting geometric features from sensor data, a high–level robot state estimator and propagator, and information about the environment's spatial data.

Feature extractor. In terms of low-level feature extractors, most previous work focuses on the nonlinear behavior of range sensors and filters the data based on the sensor's perceptual model. These methods recognize the shapes of obstacles such as walls, corners, doors, etc., in known environments [24, 27] or in unknown environments [30]. Such work assumes that the environment is piecewise linear and takes advantage of the knowledge that common geometric features exist in most indoor environments. The filtering methods can range from simple least–squares fit to more sophisticated techniques such as principal component analysis [58], Dempster's rule of combination [86], model matching by coordinate transformation [1], or Kalman filters [7].

Uncertainty reducer. There are several high–level localization methods based on probabilistic modeling of pose uncertainty, motion, and perception. It is commonly assumed that the environment is Markovian (Defined in [74] as "past and future data are conditionally independent if the current state is known"). Due to the soundness in theory and ease of implementation, the Markov assumption is popular for research and practical applications. A well-known implementation is Monte Carlo localization which is based on particle filters [18].[1] One advantage of this approach is that, if global localization of the robot is possible, then particles are allowed to be uniformly spread over the entire free space. Some variants were designed to track multiple targets [65] and multiple robots [21]. Kalman filters are also widely used due to their ability to optimally track the path in the presence of Gaussian noise [61]. It was mentioned in [25] that according to experiments, general grid-based Markov localization is more robust than Kalman filtering while the latter can be more accurate than the former. A fundamental weakness of Kalman filters is that the initial state must be known within a given error range.

Environment subdivider. The third component of the localizer is the knowledge of the environment. Often this information is stored in some data structure representing a spatial decomposition or environment subdivision. The motivation is that (1) the robot's possible configurations in the environment can be mapped from continuous space to a discrete space so that localization is performed in a finite spatial set and (2) data structures representing decompositions are often convenient to search and construct. Examples include uniform meshes (or occupancy grids), non–uniform meshes represented by quadtrees, binary space partitioning trees (BSP), and non–hierarchical exact methods [19]. Evidence grids for sonar sensors [64, 85] have

[1]Each particle represents the expected pose of the robot at a specific time.

been favored for their simplicity, and proved to be useful when the environment is relatively small and simple. These approaches are resolution dependent. An example of an exact method which is not resolution–dependent is the visibility cell decomposition [24]. In this method, each cell is defined based on what region of the environment can be seen by the robot.

To simplify our presentation, we define two classes of localizers. The first group is the *feature–based* methods, which only use two components: the feature extractor and the uncertainty reducer. The second group is the *sector–based* methods, which use all the three components. In general, we note that sector–based methods can be more efficient than feature–based methods because they can take advantage of a convenient representation of the environment.

The *feature–based* localizers improve on brute–force localization by using a geometric or statistical feature extraction module to avoid processing raw scan data. During preprocessing, features in the map that can be identified from sensor data are estimated and stored. A pseudo code description of the localization process is shown in Figure 7(a). In step 3 in the pseudo code, the robot's pose uncertainty is reduced by matching the scanned features and the preprocessed features. If the environment has a large number of features, searching through all preprocessed features can be computationally demanding.

To match features more efficiently, approximate knowledge about the robot's configuration can be utilized to restrict the search to a subset of the preprocessed features. The *sector–based* localization methods do this by combining environment subdivision and feature extraction techniques (see Figure 7(b)). During preprocessing, the environment is subdivided into sectors. Each sector contains a set of features that can be used for reducing the robot's uncertainty. During actual localization, the sectors intersecting the robot's possible positions (represented, e.g., by an uncertainty

Feature–Based Localization
1. Scan the environment
2. Identify features in the scan data
3. Reduce uncertainty using features by matching with map data

(a)

Sector–Based Localization
1. Scan the environment
2. Identify features in the scan data
3. Localize to sectors S using features
4. Reduce uncertainty using features in sectors S

(b)

Fig. 7. Pseudo-code for (a) feature–based localization, and (b) sector–based localization.

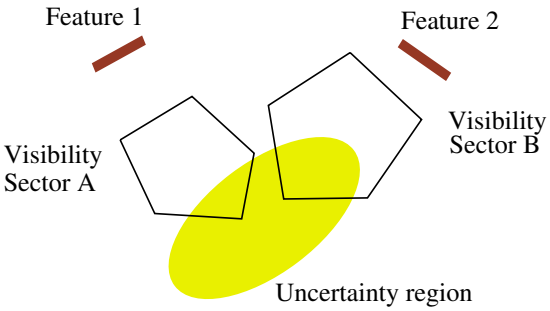
ellipse) are selected. Next, the features associated with those sectors are considered for matching (step 4 in Figure 7(a)). An example of sector–based localization is illustrated in Figure 8.

For ideal range sensors with no physical limitations such as incidence angle restrictions or range limits, Lee et al. [52] proposed a method to construct visibility sectors for efficient localization. That work was inspired by an earlier method of Guibas et al. [24] that used the concept of visibility cells[2] but did not address the issue of efficiently matching the scanned features with the preprocessed features.

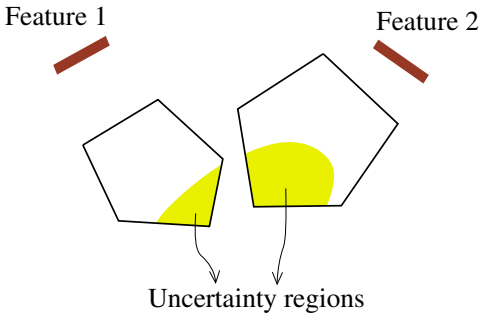
D. Sensor Cross–Talk

Sensor cross–talk is electrical interference caused by electromagnetic or electrostatic coupling between nearby conductors or external sources. An example is shown in Figure 9 where two robots simultaneously scan when they are within sensor range of each other. If the sensing procedure is not carefully coordinated, then the sensor signal fired from one robot might be received by other robots and cause incorrect readings. There are several previous studies that approach this problem from a low–

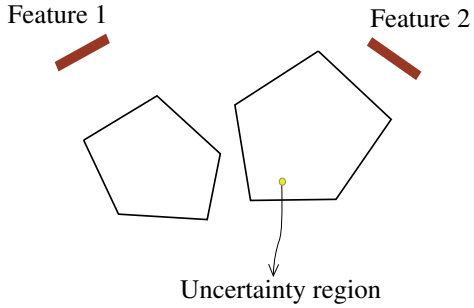
[2]Polygons whose edges delineate visible regions (or features) of the environment for a robot positioned in a cell.



(a)



(b)



(c)

Fig. 8. An example of sector–based localization. (a) Before localization. The sectors A and B correspond to the features 1 and 2, respectively. (c) The result of localization to sectors, and (c) the result of localization to configuration (in the sector). Between the two sectors, the correct one is chosen (explained in Chapter IV).

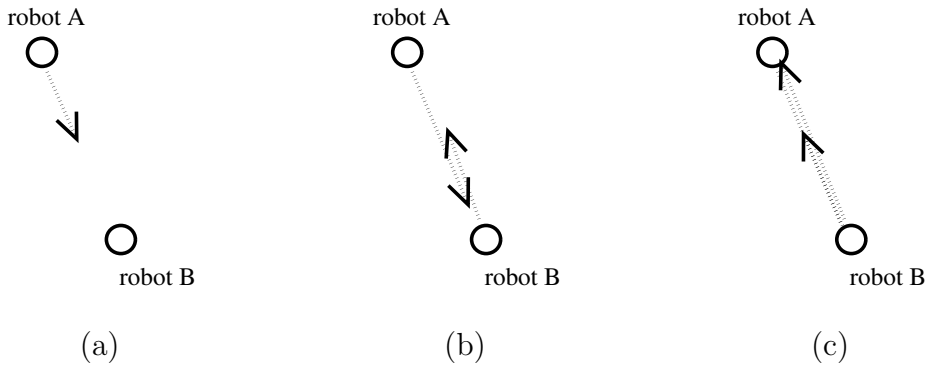


Fig. 9. An example of sensor cross–talk. (a) Robot A fires a sonar beam toward robot B first, (b) Robot B also fires a sonar beam before the signal from robot A gets to robot B, and (c) The signal from robot B is received by robot A, causing robot A to read incorrect distance.

level such as timing emitted bursts [31] and smoothing distance values of sonar sensors [20].

E. Multiple Robot Localization

Most previous work related to multi–robot localization focuses on collaboration among the robots to reduce their uncertainties. A probabilistic approach suggested by Fox et al. [21] is a multi–robot version of Monte Carlo localization. This combines the pose belief of one robot with those of other robots using rules based on statistics. A similar technique has been applied to Kalman filters where other robots are tracked if they are within sensor range [61, 62, 63], and a bounded uncertainty approach using linear programming techniques [72]. Some previous work deals with the issue of synchronizing sensor signal firing to avoid interferences [20, 29, 31]. Our work for multiple–robot localization focuses on high–level coordination, and is different from low–level approaches where localization is combined with trajectory following [60, 71].

F. Roadmap Methods

In terms of spatial representation, existing planners can be divided into two groups: those that search in a continuous space using analytical functions, and those that use a discrete approximation of the environment, e.g., a roadmap. A popular method for searching a continuous space is to use potential fields. Due to the existence of local minima, these methods are most commonly used for local planning [19]. A discrete representation of the environment can be constructed deterministically (e.g., visibility graphs [17, 84] and Voronoi diagrams [17]) or probabilistically (e.g., probabilistic roadmaps [35] or rapidly-expanding random trees (RRT) [50]). A well–known method which minimizes the Euclidean path length is to use a visibility graph and Dijkstra's algorithm [17]. However, this works only in ideal situations where the robot has perfect odometry; an actual robot's position uncertainty renders such roadmaps infeasible.

The particular planners used in our implementation are representatives of a class of roadmap–based planning methods called probabilistic roadmap methods (PRMs) [35]. PRMs have proven to be very successful in efficiently solving high–dimensional problems in complex environments. PRM roadmaps are constructed by randomly sampling configurations and retaining those that are determined to be valid, and then connecting nearby pairs of nodes using so–called local planners.

A basic method to generate sample configurations is uniform sampling. In the narrow passage situation[3], with uniform sampling, it is unlikely that sufficiently many nodes will be sampled in the corridor. Intuitively, narrow corridors in configuration space may be characterized by their large surface area to volume ratio. Several methods have been proposed that attempt to solve this so-called narrow passage

[3] a corridor with small clearance to the robot

problem [4, 12, 28, 79] and PRM variants have been proposed with various sampling schemes, metrics, and local planners [3, 4].

In our experiments, we have used the medial axis PRM (MAPRM) [53, 79, 80] which samples nodes on the medial axis of the free space, and obstacle–based PRM (OBPRM) [4, 83] which samples nodes on or near to the surface of C–space obstacles. The medial axis has been used extensively in motion planning [49]. Paths based on the medial axis are generally considered desirable because they maximize clearance from obstacles. The medial axis is also interesting because the medial axis is a strong deformation retract (SDR) of the free space [81]. This means that the free space can be continuously deformed onto the medial axis while maintaining its topological structure. In general, computing the medial axis is expensive unless the environment is low dimensional, because it is only one dimension lower than the original space. MAPRM computes samples on the medial axis in feasible time, even in high–dimensional space, by random sampling and then moving the nodes to the medial axis [53].

Obstacle–based PRM is another method targeted at the narrow passage problem [4]. It uses several techniques to generate random nodes that are on or near constraint surfaces. Another PRM variant is based on Gaussian sampling [12]. This approach uses a Gaussian sample distribution that is sensitive to the cross-section of the C-space and translational degrees of freedom.

In addition to efforts to solve the narrow passage problem, several techniques have been proposed to improve basic PRMs. A weakness of roadmap-based methods has been noted by Smith et al. [68] that they capture only coarse connectivity information about the environment. To compute the best path in the presence of small unknown obstacles that can be detected during navigation, a fuzzy logic-based local planning method that corrects the path using sensor measurements was suggested [68]. In Vallejo's work [6], the roadmap building and query processing are merged into a

single shot method. This is useful when the preprocessing time has to be as short as possible or the environment is subject to frequent change. To take advantage of human insight for difficult problems, a technique for allowing the user to manually input additional nodes using a haptic input device (a force feedback input device) is proposed in [5]. The results in [34] provide some guidance about how many nodes need to be sampled to guarantee a given probability of success for a particular problem.

G. Dijkstra's Shortest Path Algorithm.

Dijkstra's algorithm searches for shortest paths in a weighted directed graph. It maintains an explored and an unexplored set of vertices. Figure 10 shows the pseudo code for Dijkstra's algorithm. In the unexplored set, each vertex maintains a *dist* value which is the shortest distance from the start to that vertex known so far. A priority queue PQ keeps unexplored nodes sorted by their *dist* values. Initially, the *dist* value of the start node is 0 and the *dist* value of all other nodes is ∞ and they are inserted into the PQ. At each search step, the algorithm first pops a vertex v from the top of the PQ. The vertices also have *parent* information, which is used to obtain the shortest path by tracing the vertices back from the goal to the start. The vertex v is added to the explored set by setting its parent pointer and updating its *dist* value. A technique to compute the correct shortest path is *relaxation*, which decreases the known shortest distance of the vertices in PQ adjacent to v when a new lower–weighted path is found.

The shortest path from the *start* to the *goal* is computed in the pseudo code shown in Figure 11 where Dijkstra's algorithm is used as a subroutine. The key to computing the correct solution is the *relaxation* in line 8 –9 of Figure 10 which repeatedly decreases an upper bound on the weight of the vertices in PQ when a new

lower–weighted path is found.

```
DIJKSTRA(V, E, start, goal)
1.  for each v ∈ V
2.      dist[v] ← ∞
3.  dist[start] ← 0
4.  PQ ← PriorityQueue of V ordered by dist
5.  while (PQ ≠ ∅)
6.      u ← PQ.dequeue
7.      for each v ∈ PQ adjacent to u
8.          if (dist[v] > (dist[v] + weight(u, v))
9.              dist[v] ← dist[v] + weight(u, v)
10.             parent[v] ← u
11.     end for
12.     PQ.reorder
13. end while
```

Fig. 10. Pseudo code for Dijkstra's algorithm

```
SHORTESTPATH(V, E, start, goal)
1.  parent ← DIJKSTRA(V, E, start, goal)
2.  path ← ∅,   u ← goal
3.  while (suffix of path ≠ start)
4.      append u to path
5.      u ← parent[u]
6.  end while
7.  reverse path
```

Fig. 11. Pseudo code for shortest path algorithm

H. Robot Path Optimization

Previous research shows that applying common optimization techniques to robotics is not straightforward because the collision–free requirement renders it difficult to solve optimization constraints analytically or numerically [2, 77]. In particular, the discontinuity of the search space makes it difficult to find the optimal path. Figure 12(a) shows a path extracted from a roadmap (p_2) and paths generated by general

optimization techniques (p_1, p_3, p_4). Figure 12(b) shows two regions separated by an obstacle. To solve two–point boundary value optimization problems, an initial guess of the solution must be given [42]. If the initial guess is p_4, then the solution cannot be improved beyond p_3 without understanding the discontinuity of the search space. However, the suboptimal path p_2 can be transformed to the optimal p_1.

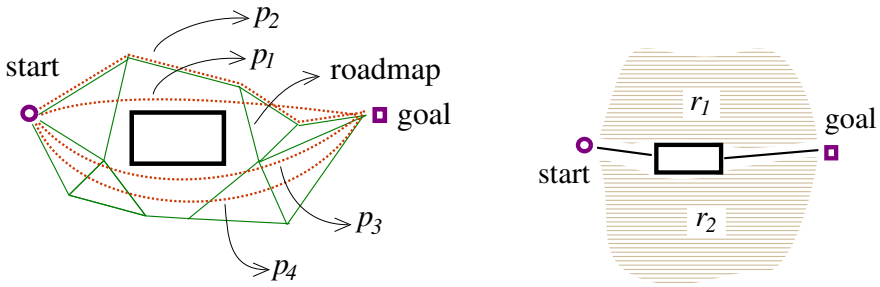


Fig. 12. Applying optimization techniques to robot path planning. (a) Optimizing path with initial guesses, with or without using roadmap, and (b) two separate regions that initial guess can select.

Many recent methods for motion planning are based on roadmaps, and paths extracted from roadmaps often need to be improved because they are composed of many line segments. Recently, two different approaches have been developed to obtain optimal paths for robotics applications: motion planning-based and optimal control-based.

The first approach is based on motion planning in robotics, and relies on techniques to improve the quality of the path. In Konkimalla's work [45], the optimal path of a nonholonomic robot is found by iteratively growing the computed region of optimal control points from the goal configuration using a cost–to–go function. At each step of the iteration, a cost–to–go function is computed in barycentric coordinates which evenly subdivide the environment. To find optimal motions for human

figures, Shiller's work [66] uses Dijkstra's shortest path algorithm in grids with edge weights reflecting the clearance and rotation of the body parts. A dynamics filter was used to compute feasible human postures while maximizing safety and minimizing unnecessary movements. For 2D environments with polygonal obstacles, in Yamamoto's work [84], a roadmap from the visibility graph is computed by computing the connectivity graph (connecting middle points of the edges in the visibility graph) and finding combinations of the edges from start to goal. The final path is optimized using B–splines for kinematic constraints and driving torque.

All of the approaches above use deterministic roadmaps and do not address the issue of computational feasibility when the search space is large. Probabilistic roadmaps encoding physical constraints have been studied by Song et al. where the roadmap is customized for various applications, and paths are improved by iterative refinement in the query step [70]. This relies on the global property explained in Chapter I Section 1. Many techniques have been proposed for locally improving path quality, often called smoothing. Common approaches for path smoothing include converting to a curve, moving existing nodes, or adding additional nodes to the suboptimal path [48, 84].

The second approach is to modify general optimization or optimal control techniques for robot motion planning. Because the methods are not based on roadmaps, collision checking needs to be geometrically and/or mathematically formulated, and is relatively complex and inefficient. The constraints of the optimization problem are extended to AND and OR logic, which are referred to as generalized constraints and deal with polygonal obstacles [77]. Modification of genetic algorithms was attempted in [26] to improve the path using using Gram–Schmidt orthogonalization.

To optimally coordinate multiple robots with specified trajectories, Akella et al. used MILP (mixed integer linear programming) where the collision between two

robots is formulated as a δ function [2]. It is difficult to apply these techniques to the paths extracted from the roadmap due to the discontinuities in the search space. Our work extracts an optimal path from among the paths in the roadmap using a cost function from optimal control theory.

CHAPTER III

ROADMAP-BASED NAVIGATION

This chapter describes our roadmap-based global navigator which integrates the path planning and path following components. We first present the navigator's general strategy, and then describe the details of our methods for computing the best path among the candidate paths stored in the roadmap.

A. General Strategy

There are two main components to mobile robot navigation: path selection and path following. Path selection is assisted when the environment is represented by a simplified data structure such as a roadmap. A naive navigator would call the path selector and then the follower, and expect that the path follower will ensure the robot reaches the goal. In this case, the robot should localize as frequently as necessary to follow the pre-determined path from the start to the goal. This works if the robot can localize anywhere along the path. Unfortunately, determining this a priori would require complete knowledge about the environment before the path follower is executed. Since we assume that the environment is partially known, it is not possible to select a path which is guaranteed to be feasible or on which it is guaranteed that localization can be done everywhere. So, it must be possible to dynamically re-plan after localization.

Our global navigator has the same general strategy as the method proposed in [52], and its pseudo code is described in Figure 13. The first step in the main loop is to extract a path from the roadmap connecting the start and the goal nodes. In this step, the planner is responsible for performing any global optimization. For example, if sensing and localization is a time consuming operation, and speed is a concern, then

the navigator must consider both the length of the path and the number of localization operations required when selecting a route. It will aid the navigator's task if the path planning module is capable of providing multiple paths from the start to the goal. This is one reason we favor roadmap–based path planning algorithms.

When planning a new path from the current position, the nominal position which is the center of the robot's uncertainty region is used as a new start position. This is one way to take advantage of the assumption that the robot's position is roughly known and bounded. The size of the ellipse is assumed to be reasonably small, and its maximum size can be controlled by the navigator depending on the situation. If the robot's initial position is not known, then a global localizer would need to be executed to determine it; this is beyond the scope of this work.

Next, a subgoal is determined. A simple way is to use the longest "safe" prefix of the path, i.e., the subpath such that the uncertainty region does not intersect any obstacles as the robot moves along the subpath. This implements a basic requirement — the robot must be able to move to the subgoal without collision. In addition, if the localization algorithm has any special requirements, then the navigator must also be sure that the uncertainty regions on the trajectory could not place the robot in a situation where it needs to localize and localization is not possible. Conditions making localization difficult are discussed in Chapter IV Section B.4. Furthermore, the subgoal might need to satisfy mission–specific requirements such as "do not localize in a certain area", "localize before entering a certain area", or "position uncertainty must be smaller than certain bound". The main intelligence required by the navigator is to determine a good subgoal.

Once the subgoal is computed, the robot is commanded to move, stop there, scan and then localize. While the robot is moving, it has to use its sensors to detect dynamic or unknown obstacles. The robot has to wait or plan a detour if unexpected

obstacles emerge. After successful localization, the uncertainty ellipse at the subgoal is reduced and the path is dynamically re-planned. These steps are repeated until the robot reaches the goal. The step 5 calls the sector-based localizer that was described in Figure 7(b).

NAVIGATOR(*start*, *goal*, *roadmap*, *sectors*)
1. while *goal* is not reached
2. extract the best path from *start* to *goal*
3. determine localizable *subgoal* in the path
4. drive robot to *subgoal* and stop
5. scan and localize using *sectors*
6. set *start* to current configuration
7. end while

Fig. 13. Pseudo-code for our navigator

In the following sections, our method for extracting optimal paths from the roadmap (line 2 in the pseudo code) is presented. We first describe it at a conceptual level, and then present our new techniques, and finally show several examples. Simulation results are shown in Chapter VI.

B. Selecting Optimal Paths from a Roadmap

Our approach to designing an intelligent navigator is based on cost function–based optimization techniques and is different from logic–based methods popular in the artificial intelligence community. From the mathematical point of view, there is no fundamental difference from shortest-distance path planning since the process of minimizing path cost is the same. The difference is that we facilitate the application of multiple optimization criteria depending on mission requirements.

Our goal is to apply the cost function for general optimization theory to mobile robotics without losing the computational efficiency of roadmap–based methods. This requires some trade offs between exactness and efficiency. The remainder of this

chapter presents the following: the general cost function that we desire to minimize, the manner in which we discretize the cost function so that is it applicable to the roadmap, issues with the Markov property for robot paths, and finally, an augmented version of Dijkstra's algorithm that finds paths according to our cost function. In addition, we discuss our techniques for flexibly representing the goal state as a subset of configurations as opposed to a unique goal configuration. We formally define the problem using both mathematical and graph–specific terms (see Section C.1). The resulting path is composed of a discrete set of segments in the roadmap, and can be post-processed to improve path quality if desired. Path smoothing is not considered in this work.

1. Standard Cost Function

The optimization of certain values for a physical system that moves from an initial state at time 0 to a final state at time T_f, while subject to constraints, is described by the problem of minimizing a cost function. The standard cost function J in optimal control theory [42] is described by

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) \tag{3.1}$$

where $g()$ is the cost at time t, $x(t)$ is the state at time t, and $u(t)$ is the control input at time t. The necessary condition at the final time T_f is described by $h(x(T_f))$.

This form of the cost function has been used in [45] and optimizes the path of a car–like robot by subdividing configuration space and linearly interpolating. In general, an optimal path satisfying Equation 3.1 with initial and final boundary conditions can be computed using several numerical methods [42].

2. Non–Markov Optimization Criteria

Compared to our approach, previous work based on roadmaps lacks two properties needed for real applications. The first is the need for non–Markovian states, i.e., states which depend on information from a range of previous states.

For example, to maximize clearance, it is clear that a cost function g will contain the reciprocal of the clearance if the optimizer minimizes J. We denote the reciprocal of the clearance as $\frac{1}{cl(x(t))}$. If we let $g(x(t), u(t)) = \frac{1}{cl(x(t))}$ in Equation 3.1, then the resulting path will maximize the accumulated $\frac{1}{cl(x(t))}$ from the start to goal. In most cases, the objective is to optimize the path for maximum safety and the proper criterion is to maximize the minimum path clearance, not to maximize the accumulated clearance. This requires a modified cost function

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) + \frac{1}{cl(x(t_m))} \tag{3.2}$$

where $t_m \in [0, T_f]$ such that $cl(x(t_m))$ is minimum, and maximizing $cl(x(t_m))$ is equivalent to minimizing $\frac{1}{cl(x(t_m))}$. The term $cl(x(t_m))$ is non–Markov, and we force the state to be Markovian.

3. Goal Sets – Flexible Final States

The second issue that has not been addressed explicitly in previous mobile robotics is a flexible definition of the final goal configuration. Describing the final condition at T_f using an equality condition changes Equation 3.1 to

$$\begin{aligned} J &= \textstyle\int_0^{T_f} g(x(t), u(t))dt \\ h(x(T_f)) &= 0 \end{aligned} \tag{3.3}$$

where the problem is now minimizing J with $h(x(T_f)) = 0$ satisfied. This is identical to one of the boundary conditions of the optimal control formulation where T_f is free and $x(T_f)$ is moving on the surface, $h(x(t)) = 0$. In a graph–search based path planner such as Dijkstra's algorithm, it is difficult to find a node that satisfies $h(x(T_f)) = 0$ unless some of the nodes are generated exactly on the surface where $h(x(t)) = 0$. So, we modify the surface to be more inclusive by using an inequality condition.

$$\begin{aligned} J &= \textstyle\int_0^{T_f} g(x(t), u(t))dt \\ h(x(T_f)) &\leq c_f \end{aligned} \tag{3.4}$$

The final necessary condition $h(x(T_f)) \leq c_f$ is used to terminate the graph search if any node satisfying $h(x(T_f)) \leq c_f$ is reached. We call this set of nodes a *goal set*, and its size is determined by the constant c_f. Note that Dijkstra's algorithm requires two cost functions corresponding to g and h in Equation 3.4.

C. Augmenting Dijkstra's Shortest Path Algorithm

Our path optimization system is based on a roadmap-based path planner and Dijkstra's shortest path algorithm. To address the issues mentioned in the previous section, we design an augmented version of Dijkstra's algorithm and cost computation.

1. Problem Formulation

Before explaining the details of our framework, we reformat the mathematical description (in Equation 3.4) to a pseudo–code friendly version. Figure 14 describes our path optimization problem of minimizing the cost of a given path p. Operators $start(e_i)$ and $end(e_i)$ denote the start and end vertex of edge e_i, respectively, and the cost functions $cost_g$ and $cost_h$ denote the functions g and h in Equation 3.4, respec-

Given environment, $start$, $cost_h$, and c_f,

find a path $p = \{e_1, e_2, \ldots, e_f\}$ such that

$$\text{minimize } cost(p)$$

where $cost(p) = \sum cost_g(e_i)$ under the constraints

$$start(e_1) = start$$

$$clearance(e_i) > 0, \qquad i = 1 \ldots n$$

$$\text{others (e.g., time, energy, } \ldots)$$

and the final condition

$$end(e_f) \in goal_{set}$$

where $goal_{set} = \{end(e_i)) \mid cost_h(end(e_i)) \leq c_f\}$

Fig. 14. Path optimization problem

tively. *Start* is a node in the roadmap, and the final condition specified by a constant c_f is internally transformed to a goal set, $goal_{set}$, that will terminate the search when reached. In Section 4, pseudo code is used to describe this in detail.

Note that we do not use the approach of iterative improvement of J, such as hill climbing and steepest descent. Like dynamic programming methods in optimal control, we compute the solution in one shot using Dijkstra's algorithm.

2. Markov–like Optimization

Ideal Markov Function. The issue of maximizing minimum clearance was introduced in Section B. The cost function including a non–Markovian state is

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) + m(x(t_m)), \quad t_m \in [0, T_f] \tag{3.5}$$

where $m(x(t_m))$ is a general non–Markovian cost function. In Equation 3.2, $m(x(t_m))$ was $\frac{1}{cl(x(t_m))}$ with t_m the time when the clearance is lowest. This formulation is not tractable for common optimization solvers. Our approach to this problem is to modify $g(x, u)$ or $cost_g(e_i)$ in Figure 14 so that $m(x(t_m))$ is eliminated in the cost function. An example of such a cost function is shown in Figure 15(c) where the cost depends on the path clearance (Figure 15(b)).

Discretization. Since we are using a graph search algorithm which is similar to dynamic programming in classic optimization theory, Equation 3.5 can be represented by a discretized version

$$\begin{aligned} &J = \textstyle\sum_{i=0}^{i \leq N_f} g(x_i, u_i) + h(x_{N_f}) + m(x_{i_m}), \quad i_m \in \{0, 1, \ldots, N_f\} \\ &x_i = a(x_{i-1}, u_{i-1}) \end{aligned} \tag{3.6}$$

where N_f is the total number of time steps, i_m is the time step corresponding to t_m, and a is a discrete time state update equation of the system dynamics. An example illustrating the edge/node weights along a path is shown in Figure 15.

Using the Previous State. Now, we replace $g(x_i, u_i)$ with $g(x_i, x_{i-1}, u_i)$ so that both previous and current states are used for computing the cost. The previous state is obtained by using the *parent* data structure in the search tree of Dijkstra's algorithm. The vertex corresponding to x_{i-1} can be quickly obtained from the *parent* data structure and the vertex corresponding to x_i. We note that this is similar to converting a continuous time state $\dot{x}$ to a discrete time state composed of x_i, x_{i-1} and Δt using Taylor's series expansion. Many optimization values such as the turning angle can be computed from $\dot{x}$ (or x_i, x_{i-1} and Δt if in discrete time). In this case, using $g(x_i, x_{i-1}, u_i)$ in Dijkstra's algorithm can be regarded as applying a standard discrete time optimization to a graph search technique. This does not exhaust the possible applications of our optimizer.

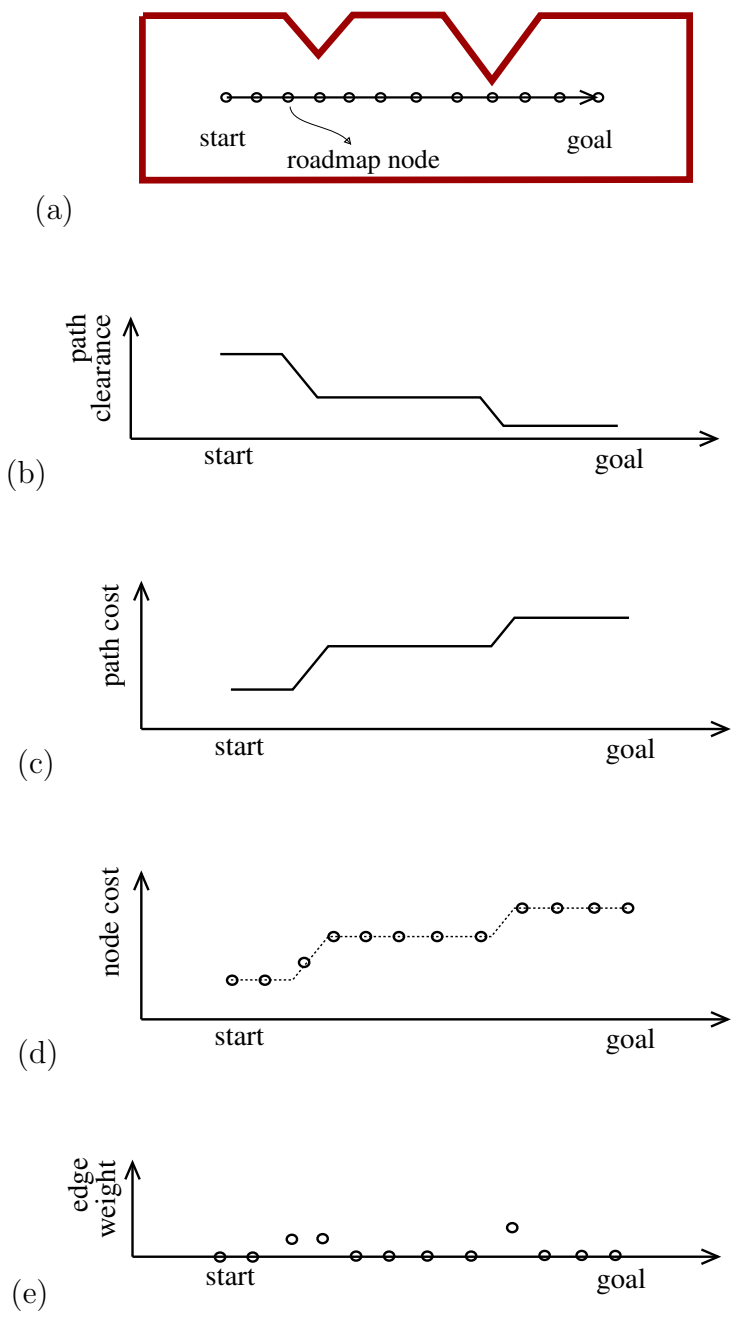


Fig. 15. Edge weights for path clearance. (a) A straight line path from the start to the goal and roadmap nodes, (b) the path clearance, (c) the path cost (inversely proportional to the path clearance), (d) the cost of the roadmap nodes, and (e) the edge weight (Equation 3.10) at the nodes are shown.

Markov–like Cost Function. An example of an optimization value that cannot be computed from $\dot{x}$ is minimum clearance, which will be computed from x_i and x_{i-1}. There are other optimization values such as a localization success ratio that can be formulated using x_i and x_{i-1}. So, our motivation for using $g(x_i, x_{i-1}, u_i)$ is not from discretizing $g(x, \dot{x}, u)$, but to extend the ability of the graph search based path optimizer using current and previous states. We call this approach Markov–like because x_{i-1} is not Markov in a strict sense but x_i and x_{i-1} can be considered as a compound state $\mathbf{x}_i$. The general cost function is

$$\begin{aligned} &J = \textstyle\sum_{i=1}^{i \le N_f} \mathbf{g}(\mathbf{x}_i, u_i) + h(x_{N_f}) + m(x_{i_m}), \quad i_m \in \{1, 2, \ldots, N_f\} \\ &\mathbf{x}_i = \begin{pmatrix} x_i \\ \\ x_{i-1} \end{pmatrix} \end{aligned} \tag{3.7}$$

New State Update Equation. We added x_{i-1} to the cost function with the intention of eliminating $m(x_{i_m})$, and the state equation $a(x_{i-1}, u_{i-1})$ needs to be changed accordingly. The idea is that $\mathbf{x}_i$ should contain the entire history of the non–Markovian property. For example, to maximize the minimum path clearance, an element in $\mathbf{x}_i$ will indicate the minimum clearance from start to time step i. Now, we denote the minimum clearance state by x_i^{cl} and add it to $\mathbf{x}_i$.

$$\begin{aligned} &J = \textstyle\sum_{i=1}^{i \le N_f} \mathbf{g}(\mathbf{x}_i, u_i) + h(x_{N_f}) \\ &\begin{pmatrix} x_i \\ \\ x_i^{cl} \end{pmatrix} = \begin{pmatrix} a(x_{i-1}, u_{i-1}) \\ \\ a^{cl}(x_i, x_{i-1}^{cl}) \end{pmatrix} \\ &\mathbf{x}_i = [x_i \;\; x_i^{cl} \;\; x_{i-1}^{cl}]^T \end{aligned} \tag{3.8}$$

The state equation a^{cl} returns x_i^{cl} which is lower than x_{i-1}^{cl} only if $cl(x_i)$, the clearance of x_i, is smaller than x_{i-1}^{cl}. Otherwise, x_i^{cl} must equal x_{i-1}^{cl} because the

clearance of the current state is not smaller than the minimum clearance discovered so far (see Figure 21). It is clear that a^{cl} must contain a Boolean operator.

$$a^{cl}(x_i,\ x_{i-1}^{cl}) = \begin{cases} cl(x_i) & \text{if } cl(x_i) < x_{i-1}^{cl} \\ x_{i-1}^{cl} & \text{otherwise} \end{cases} \tag{3.9}$$

New Cost Function. Next, we focus on $g^{cl}(x_i^{cl}, x_{i-1}^{cl})$ which is a part of $\mathbf{g}(\mathbf{x}_i, u_i)$ and corresponds to the state x^{cl}. It compares the difference between x_i^{cl} and x_{i-1}^{cl}, and should return a nonzero positive value if $x_i^{cl} < x_{i-1}^{cl}$. Otherwise, it returns zero so that J does not increase. So, we have

$$g^{cl}(x_i^{cl},\ x_{i-1}^{cl}) = \begin{cases} c \cdot (x_{i-1}^{cl} - x_i^{cl}) & \text{if } x_i^{cl} < x_{i-1}^{cl} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

where c is a constant. This technique for minimum clearance can be applied to other non–Markovian optimization values with the superscript ^{cl} changed in Equations 3.8, 3.9 and 3.10. Figure 15(e) shows the node weights corresponding to the environment in Figure 15(a). The edge weight is same as the node weight at the edge's endpoint.

3. Flexible Final Condition

We apply the modified final condition shown in Equation 3.4 to our new cost function in Equation 3.8, which is the final form of the cost function that we seek.

$$\begin{aligned} &J = \textstyle\sum_{i=1}^{i \le N_f} \mathbf{g}(\mathbf{x}_i, u_i), \\ &\mathbf{x}_i = \mathbf{a}(\mathbf{x}_i, u_{i-1}) \\ &\mathbf{h}(\mathbf{x}_{N_f}, u_{N_f}) \le c_f \end{aligned} \tag{3.11}$$

4. Augmented Dijkstra's Algorithm

Dijkstra's algorithm is augmented to reflect the changes in Equation 3.11, and its pseudo code is shown in Figure 16. To use Markov–like states, the *weight* function that corresponds to $cost_g$ in Figure 14 is changed so that three adjacent vertices are used. The cost function $cost_h$ checks if a node is in the goal set using c_f.

```
AUGMENTED DIJKSTRA(V, E, start, c_f)
1.  for (each v ∈ V)  dist[v] ← ∞
2.  dist[start] ← 0
3.  PQ ← PriorityQueue of V ordered by dist
4.  while (PQ ≠ ∅)
5.     u ← PQ.dequeue
6.     for each v ∈ PQ adjacent to u
7.        if (dist[v] > (dist[v] + weight(u, v, parent[u]))
8.           dist[v] ← dist[v] + weight(u, v, parent[u])
9.           parent[v] ← u
10.       if (cost_h[v] < c_f) return
11.    end for
12.    PQ.reorder
13. end while
```

Fig. 16. The augmented Dijkstra's algorithm

D. Mobile Robot Applications

In this section we provide some robotic examples that benefit from the path optimization methods described. They utilize our roadmap–based mobile robot system described in [38, 39, 51]. A T–shaped environment and roadmap are shown in Figure 17 where five nodes in the goal set are marked. Since the mobile robot moves in 2-dimensional space, the nodes in a goal set comprise a set of robot positions (x, y) and orientation (Θ). The nodes $parent[u], u$, and v in Figure 16 correspond to the left vertex of e_{i-1}, the vertex connecting e_{i-1} and e_i, and the right vertex of e_i in Figure 17, respectively.

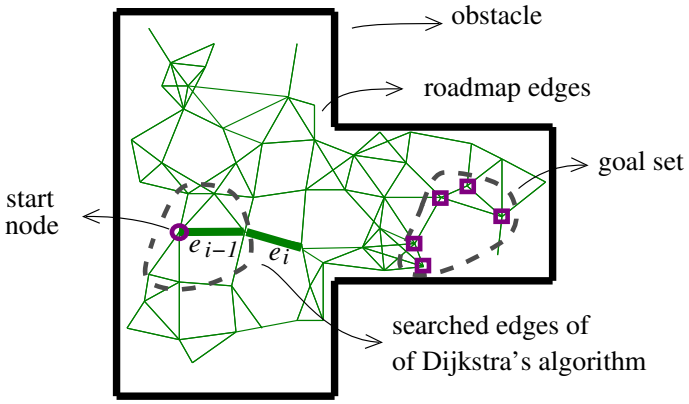


Fig. 17. Environment, roadmap and path searching.

The diagram shown in Figure 18 has two components, Dijkstra's algorithm and weight computation. In this section, we will show that various optimization values can be computed by using different weight computations in the common framework. In the diagram shown in Figure 18, we compute the shortest distance path using

$$cost(e_i) = length(e_i) \tag{3.12}$$

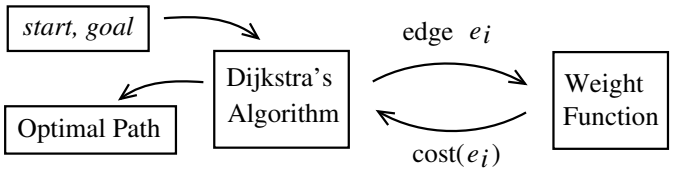


Fig. 18. Diagram of shortest path computation.

1. Minimizing Travel Time

The path extracted from a roadmap consists of a series of translations and rotations (unless converted to a curve). The time for travel is approximated by the time required

for translation and rotation. In this section, we assume that the rotation time can be approximated by a constant value and the translation time is proportional to the length of the edge. In Figure 18, travel time is minimized by using

$$cost(e_i) = c_1 \cdot length(e_i) + c_2 \tag{3.13}$$

where c_1 and c_2 are constants. To compute the rotation time as a function of the rotated angle, the technique explained in Section 3 is needed.

2. Avoiding Localization Failure

In this case, we assume that the robot's sensors have range limits and always fail to localize if no feature exists within the range. The locations of all features in the environment are assumed to be known. In Figure 18, we use

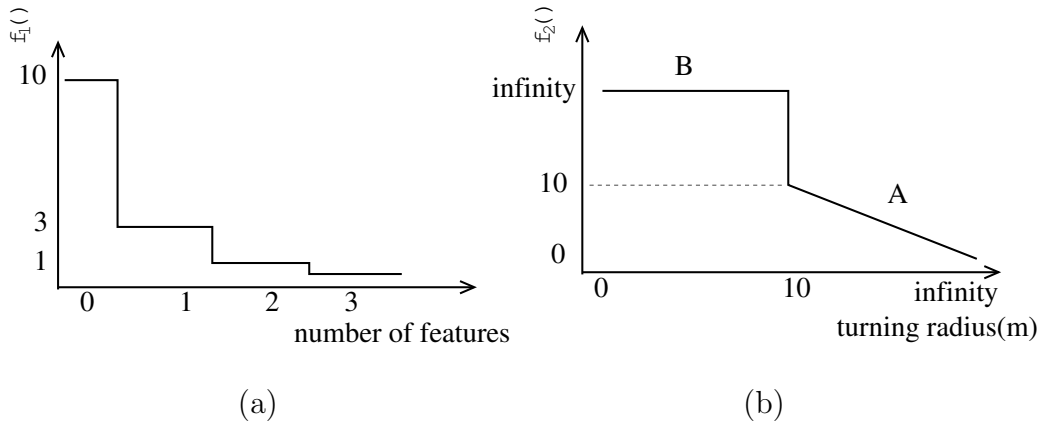
$$cost(e_i) = c_3 \cdot f_1(\text{visibility of } e_i) \tag{3.14}$$

where 'visibility of e_i' determines if the robot can successfully scan one or more feature(s) on the edge e_i. The function $f_1(e_i)$ converts the visibility of edge e_i into a scalar as shown in Figure 19(a). Note that the optimal path can traverse a region with no features if necessary.

3. Kinematic Constraints

If the robot has constraints on its turning radius, two adjacent edges e_i and e_{i-1} are needed to compute the required turning radius to obtain the cost of e_i. The weight function now uses two edges (or three vertices) as shown in the pseudo code in Figure 16. In Figure 20, which reflects the modified weight computation, we use

$$cost(e_i) = c_4 \cdot f_2(\text{turn radius of } e_{i-1} \text{ and } e_i) \tag{3.15}$$



(a) (b)

Fig. 19. Cost functions, (a) for features and (b) for turning radius.

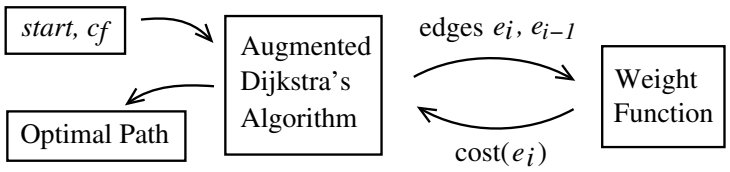


Fig. 20. Weight function with two adjacent edges.

where f_2 is an appropriate linear or nonlinear function.

Figure 19(b) shows an example of a nonlinear function that maximizes the turning radius (region A) and prohibits e_i from being used if it violates the kinematic constraint of a turning radius of less than 10 meters (region B).

4. Maximizing Minimum Clearance

As discussed in Section 2, the minimum clearance x^{cl} is a non–increasing variable and is shown as a solid line in Figure 21. To implement this in the augmented Dijkstra's algorithm framework, we add the new variable as auxiliary data as in Figure 22. The data is maintained according to the rule shown in Equation 3.9. The edge cost

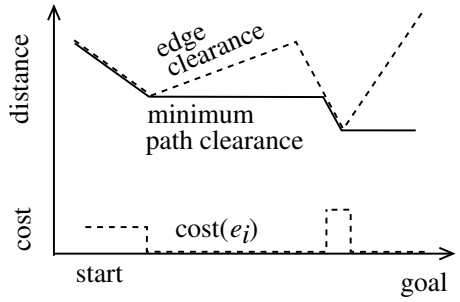


Fig. 21. Cost function for edge clearance

computation equivalent to Equation 3.10 is described by

$$cost(e_i) = \begin{cases} c_5 \cdot (cl_{min} - cl(e_i)) & \text{if } cl(e_i) < cl_{min} \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

where cl_{min} is the auxiliary data and $cl(e_i)$ is the clearance of edge e_i. Initially, cl_{min} is set to the clearance of the start node.

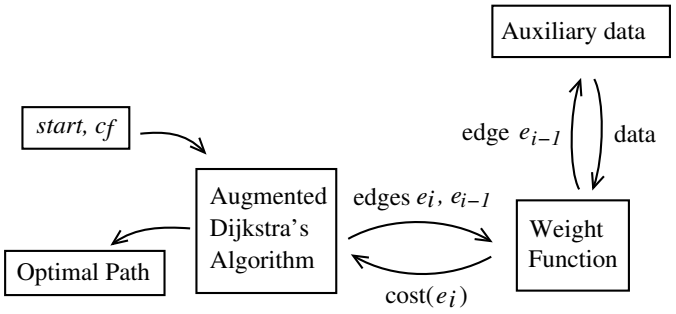


Fig. 22. Weighting with two adjacent edges and related data.

5. Minimizing Localization Attempts

We assume that each localization takes a significant amount of time due to slow–rotating sensors, etc., and that the number of localizations should be minimized. For simplicity, we assume the following. First, the robot localizes only when its

uncertainty ellipse intersects an obstacle. In other words, no preventive localization is performed. Second, the uncertainty ellipse grows linearly as the robot moves and rotates. Third, after attempting to localize, the ellipse shrinks to a point. Our approach is to compute uncertainty ellipses for all nodes in the roadmap during path planning. This is computationally expensive compared to other path optimization cases, and an alternative is to approximate ellipses with simple polygons such as boxes.

The cost of edge e_i depends on the history from the start, but the ellipse of e_i can be obtained from the ellipse of e_{i-1} only. In Figure 22, we use

$$cost(e_i) = \begin{cases} c_5 & \text{if collision detected} \\ 0 & \text{otherwise} \end{cases}$$

6. Combination of Criteria

Combining various costs into one function results in an optimization for multiple values, and is useful in many applications. The combined edge cost is expressed by

$$cost(e_i) = \sum_j w_j \cdot cost_j \tag{3.17}$$

where w_j is an appropriate weight and $cost_j$ is $cost(e_i)$ in Equations 3.12–3.16.

The issue of how to select weight values to obtain desired paths is not covered in this dissertation. Automatic determination of weight values might be achieved using machine learning techniques. In this case, training data must provide a characterization of the environment, desired path, and working weight values. Recent work [68] used a fuzzy logic-based technique to adaptively modify a path locally while satisfying multiple optimization values.

CHAPTER IV

SECTOR–BASED ROBUST LOCALIZATION

In this chapter, we describe our method for localization of a mobile robot using realistic range sensors in partially known indoor environments. Since our work builds on previous work considering ideal sensors, each section first describes the ideal case before discussing our new methods. Simulation and hardware experimental results are presented in Chapter VI.

A. Overview of Sector–Based Method

This section describes our methods conceptually, without formal definitions. The feature–based and the sector–based localization methods have been introduced in Chapter II Section C. It was mentioned that visibility sectors subdivide the environment and contain preprocessed feature information. We assumed that the robot's pose is roughly known. Our method constructs resolution–independent sectors in two steps — identifying primitive sectors and performing geometric operations.

A *primitive sector* is a planar region from which a particular feature can be scanned, and a *visibility sector* is a region from which a particular set of features can be scanned. These two concepts will be formally defined in the following sections. Figure 23 shows the relation among geometric features, primitive sectors, and visibility sectors. Sector A is the subtraction of the polygon for feature 2 from the polygon for feature 1, and only feature 1 is visible. In sector C which is the intersection of two primitive sectors, two features are visible.

The pseudo code shown in Figure 24 provides a high-level overview of sector–based localization. In this approach, the robot's pose uncertainty is reduced in a hierarchical way, making it more efficient than non–sector–based approaches. The

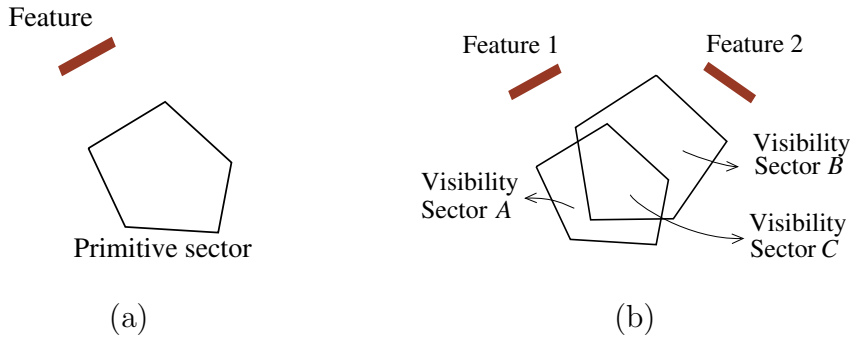


Fig. 23. Conceptual view of sectors. (a) A primitive sector corresponding to a geometric feature, and (b) visibility sectors for two features.

step of *localization to a sector* is added which uses the robot's uncertainty region (line 4) and scanned features (line 6) to select the sector containing the robot. The robot's uncertainty region often intersects more than one sector, and our algorithm selects the correct one by matching features. Note that geometric features are used for two purposes, for localizing to a sector and for localizing in a sector.

```
SECTOR–BASED LOCALIZATION
1.  scan the environment
2.  F_scan ← features in the scan data
3.  for each sector ∈ all sectors
4.      if sector matches the robot's uncertainty
5.          F_sector ← features in sector
6.          if F_sector matches F_scan
7.              reduce uncertainty using F_scan and F_sector
8.          end if
9.      end if
10. end for
```

Fig. 24. Pseudo-code for localization based on geometric features and visibility sectors.

B. Constructing Visibility Sectors

We organize the following subsections according to the level of abstraction — from low level components focusing on geometric features to high level components requiring

global knowledge. Design of each component depends on the sensor measurement models and the output requirements for localization. In this section, we consider two sensor models and two output requirements that produce three combinations. Finally, the time complexity of sector computation is analyzed to show the feasibility of our methods.

1. Visibility Models

The visibility information from a single range sensor measurement is the distance from the sensor to a part of the environment. A set of distance values are converted to a geometric feature if they form a pattern, providing a means to connect hardware (sensors) and software (localizers). For a polygonal environment in which all obstacles are represented by simple polygons, there are three kinds of features that can be extracted and used for reducing the uncertainty region: walls, convex corners, and concave corners. Convex corners are identified by a *local maximum* in the scanned data. Figure 25(a) shows an ideal scan with local minimum (`m`) and maximum points (`M`). Based on the local minimum and maximum points, we define two instances of features for two visibility models.

First, if the sensors are perfect and have no range limits, then both local minimum and maximum points can be scanned and identified. It is also possible to distinguish between the local minimum points from the walls and from the corners. Convex corners (local maximum) are attractive features for localization because their coordinates are fixed in space and the robot's coordinates can be computed [52]. In this case, it is possible to perform *localization to a configuration.*

In practice, however, range sensors have physical limitations which result in restricted visibility, and it is important that localization methods are designed to work with these. In this case, local maximum points cannot always be scanned and

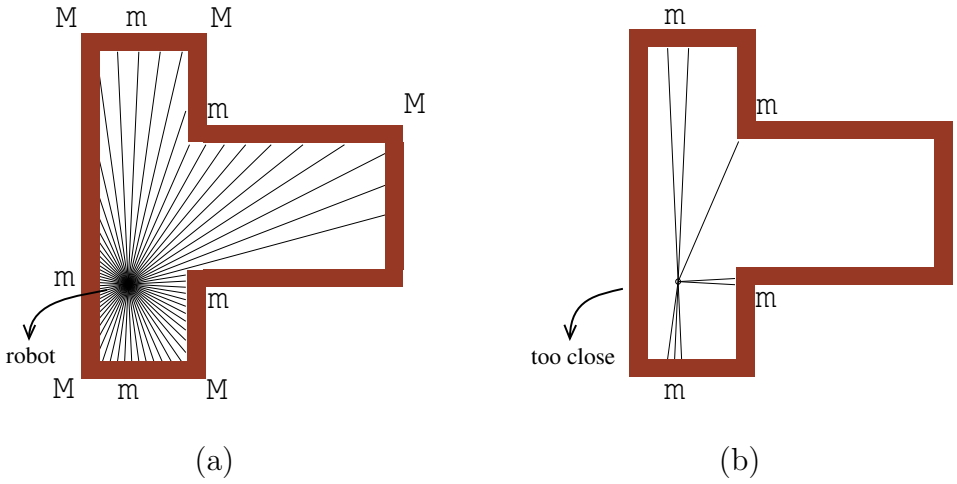


Fig. 25. Range sensor data for (a) perfect visibility and (b) limited visibility.

two possibilities are suggested: estimating invisible features (local maximum points in this case) using visible features (local minimum points in this case), or reducing the uncertainty region using local minimum points.

A simple way of estimating the coordinates of a local maximum point is to extract features from scanned adjacent local minimum points and then to intersect scanned walls. For example, in Figure 25(a), two walls on the left and the bottom can be scanned and the corner `M` is obtained by intersecting the scanned walls. This process is described in the pseudo code in Figure 26.

Another method employs local minimum points as features. Localization using a local minimum feature does not precisely determine the robot's configuration, but it might still reduce the region known to contain the robot. This reduced uncertainty region might be sufficient to enable navigation to the goal if the navigator is designed to incorporate this situation. We call this *relaxed localization*. In Figure 25(b), five features can be potentially scanned given the position of the robot. However, because the sensor's minimum range is larger than the distance to the wall on the left in this case, the `m` to the left cannot be scanned.

```
ESTIMATE CORNER
1.  for 3 ~ 5 times (depending on sensor calibration)
2.      MS ← unprocessed scan, minimum range
3.      check a few scans around MS
4.      if they form line segment (L)
5.          repeat
6.              LS ← left scan adjacent to MS
7.              if LS is farther than MS
8.                  move LS to be on the line L
9.                  MS ← LS
10.             end if
11.         until no farther LS found
12.         do the same for right direction
13.     end if
14. end for
```

Fig. 26. Pseudo code for estimating a corner from two walls.

In reality, not all walls are linear. For curved walls, it is possible to define a corresponding feature using its length and curvature. Figure 27 shows arc features where the robot can scan local minimum or maximum features depending on its distance to the wall. Such features are not considered in this work since the linear wall assumption holds for most actual indoor spaces and can be extended if needed. Also, designing features in three dimensional space is not prohibitive.

2. Primitive Sectors

To compute visibility sectors, a data structure that relates the robot's position and expected visible features needs to be constructed first. This is fundamentally different from previous approaches like evidence grids [64] which answer "given a robot position, what part of the environment is visible?" This is not only inefficient but also resolution–dependent. Instead, visibility sectors are generated based on the query "given a part of the environment (i.e., a feature), from which region can the robot see this feature?"

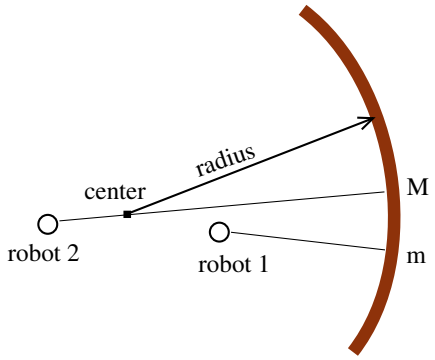


Fig. 27. An arc–shaped wall can be a feature (depending on the distance to the wall, local maximum or minimum feature is scanned).

A *primitive sector* represents a region in the free space from which the robot can scan a particular feature, i.e., there is one sector for each feature. For range sensors and polygonal environments, primitive sectors were introduced in [24] as "visibility polygons" which could be scanned by a robot with ideal or perfect sensors. We describe two kinds of primitive sectors for the features discussed previously.

For a local maximum feature with perfect visibility, we have a primitive sector as shown in Figure 28(a). To compute the exact position of the corner using a finite number of scans, in [52], four scans (p_1 – p_4) were used as shown in Figure 28(b). The sensor maximum range limitation is illustrated in Figure 28(c). With incidence angle limitations, as shown in Figure 28(c), the area close to a corner cannot be measured if the robot is close to the adjacent walls. However, such primitive sectors are not well–defined, and directly scanning a convex corner with restricted visibility is often impossible.

Using realistic sensors with min/max range and incidence angle limitations, only

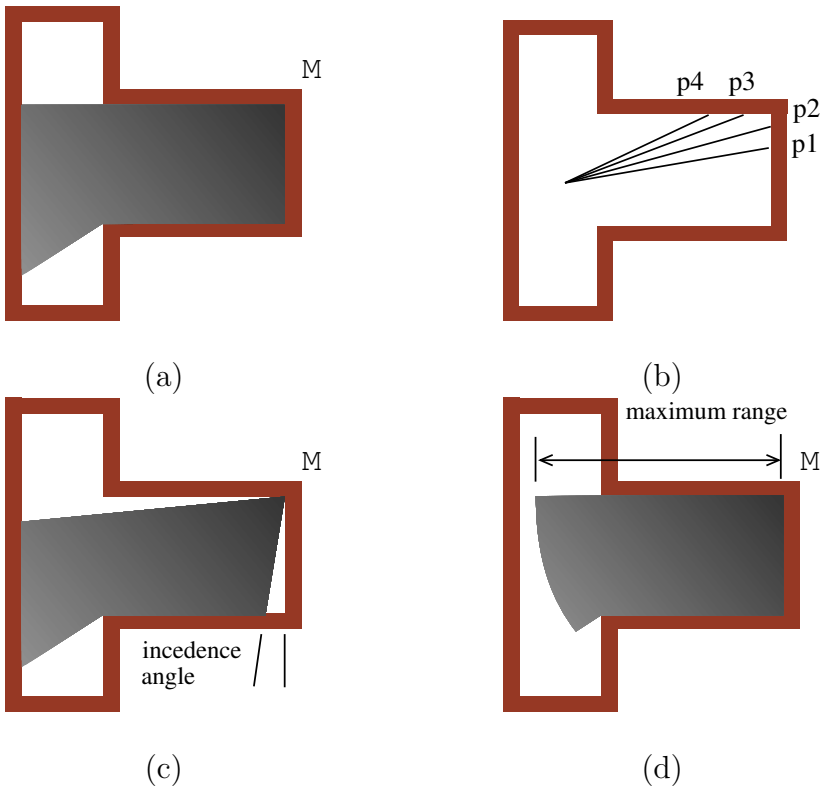
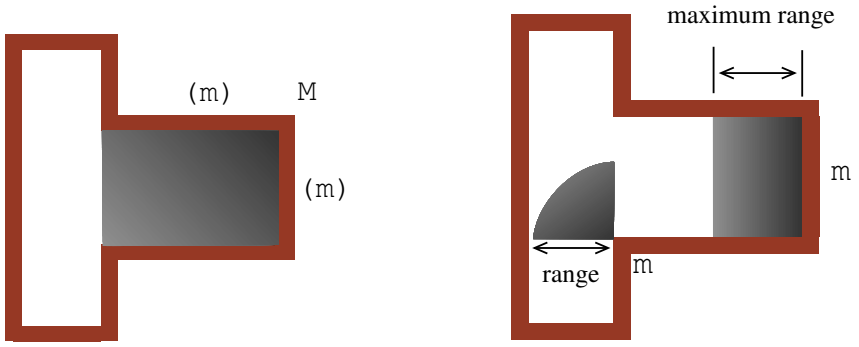


Fig. 28. Primitive sectors for range sensors. (a) Sectors with perfect visibility. (b) Computing M with four adjacent scans. (c) Sectors with incidence angle limitation, and (d) Sectors with maximum range limitation.

local minimum points can be reliably scanned. One solution estimates the corner feature using two local minima as mentioned in Section 1. A primitive sector is shown in Figure 29(a) where two walls (upper and right) must be scannable to obtain the corner feature. If local minimum points serve as features, and relaxed localization is allowed, then the corresponding primitive sectors are shown in Figure 29(b). Using the visibility model that has been experimentally obtained (see Figure 6 in page 14), two features are used — concave corners and walls.



(a) (b)

Fig. 29. Finding features with restricted visibility. (a) Estimating M from two m's if only local minimum are visible. (b) Visibility polygons for two local minimum points (wall and concave corner).

3. Identifying Sectors

A *visibility sector* is a planar region in the environment in which the same set of features is visible from every point in the sector. A basic requirement of valid sector subdivision is that there are no neighboring sectors with the same feature set. This is the same concept as visibility cells in [24]. Based on the primitive sectors, we present three different sector subdivision methods.

In [52], ideal sensors with perfect visibility are assumed and the environment is subdivided into sectors by intersecting all primitive sectors for local maximum features and identifying simple polygons. In Figure 30, nine sectors are created from six primitive sectors. Each line segment between sectors denotes a difference in the visible feature set. For example, sectors 6 and 9 are differentiated by the top left corner in the environment, which is same for sectors 7 and 8.

In the second method, we improve on the previous approach by allowing for sensor limitations which result in limited visibility. If a localization algorithm requires that a feature provide two dimensional information, then one way is to use primitive

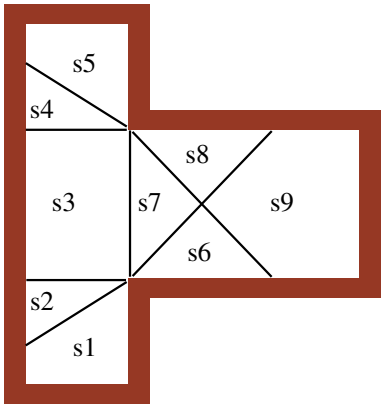


Fig. 30. Visibility sectors for perfect sensors

sectors that enable estimating a corner, as shown in Figure 29(a). Though it might be possible to compute visibility sectors from such primitive sectors in simple cases, we propose a more general method that is based on a sector merging technique. The idea is that sector subdivision for ideal sensors is a superset of the subdivision for limited sensors. In other words, not all sector boundaries for perfect visibility are valid for restricted visibility.

Our approach for removing unnecessary sector boundaries is to merge adjacent sectors. Figure 31 shows the pseudo code for computing sectors based on this merging technique. To merge two adjacent sectors, we propose two heuristics.

```
MERGING SECTORS
1.  compute visibility sectors for perfect sensors
2.  for adjacent pairs of sectors
3.      if differentiated by un-scannable features
4.          undo the sector division (merge)
5.      end if
6.  end for
```

Fig. 31. Computing sectors for limited visibility and incomplete feature set.

Merge rule #1. Merge adjacent sectors that can see the same closest feature.

The result is shown in Figure 32(b) which was obtained from the visibility sectors in (a). The only feature in this case is `M` that can be estimated from two `m` points. Note that it resembles a Voronoi diagram where the regions are centered around the corner features. Rule #2 is the counterpart of rule #1, and the result is shown in Figure 32(c).

Merge rule #2. Merge adjacent sectors that share some common visible feature.

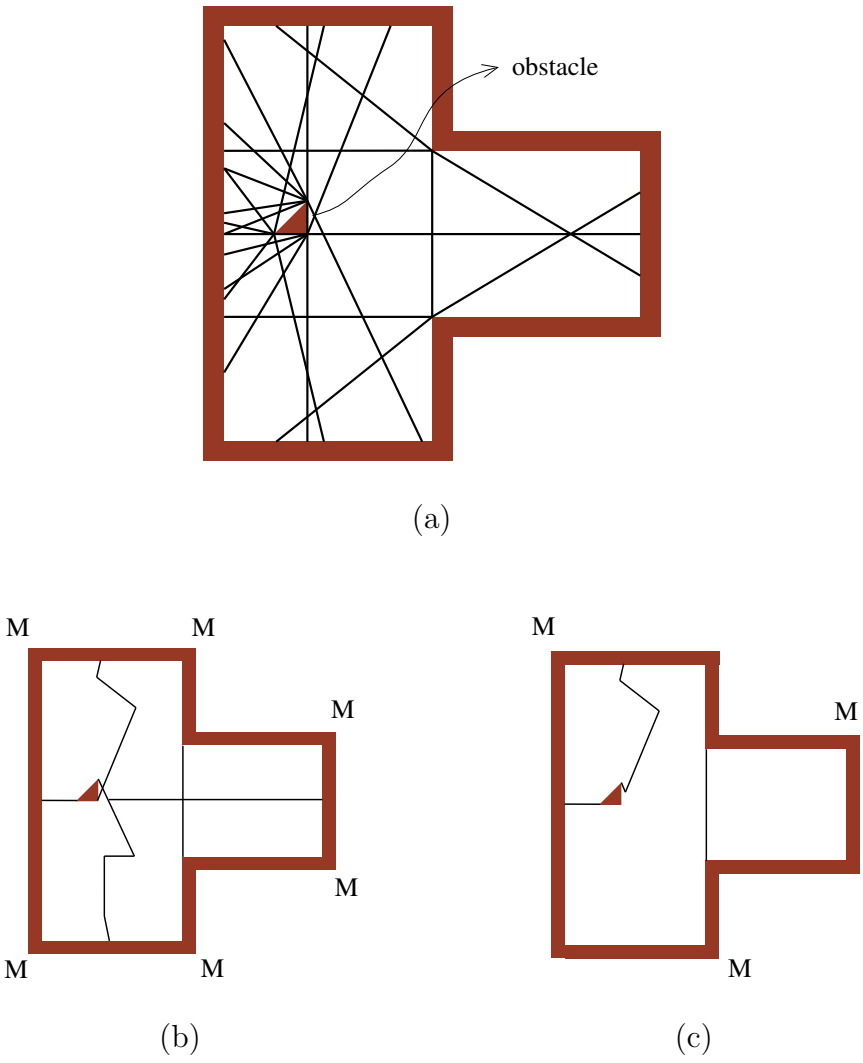
This rule is an extension of rule #1 and provides flexibility in choosing the localization feature, which may enable a greater reduction in uncertainty. In real situations where the the closest feature may not be scanned due to sensor noise, merge rule #2 should be used.

Finally, if relaxed localization is preferred, then sectors are identified from local minimum points. With sensor maximum and minimum range limitations, primitive sectors have different shapes than for ideal sensors; see Figure 33(a) and 33(b). From a computational point of view, generating visibility sectors in this case is essentially the same as the perfect sensors case. The resulting visibility sectors are shown in Figure 33(c).

4. Information in Sectors

We need to store enough information in the sectors so that localizing to a sector is possible: the selectivity of a sector depends on the uniqueness of the feature set with which it is associated. Also, the features are used for reducing the robot's uncertainty, but often a smaller number of features can be used for this task.

For perfect sensors, in this section, we define two features in addition to `M` and `m`. Previous work [52] suggested using `D` and `c` (discontinuity and connection points) which are always used in conjunction. As shown in Figure 34, features `Dc` and `cD`



(a)

(b)

(c)

Fig. 32. Sector merging heuristics. (a) Visibility sectors (before merging), (b) after merging with merge heuristic #1, and (c) after merging with merge heuristic #2.

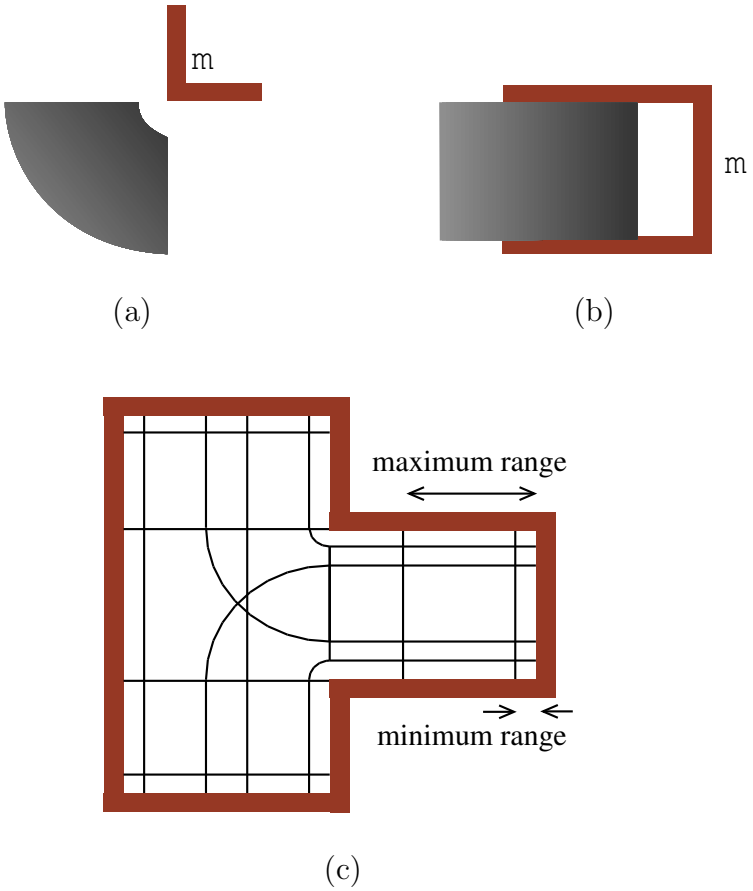


Fig. 33. Visibility sectors for local minimum point features. (a) Primitive sector for a concave corner and (b) a wall. (c) Visibility sectors.

correspond to a large increases and decreases in the sensed range, respectively. The

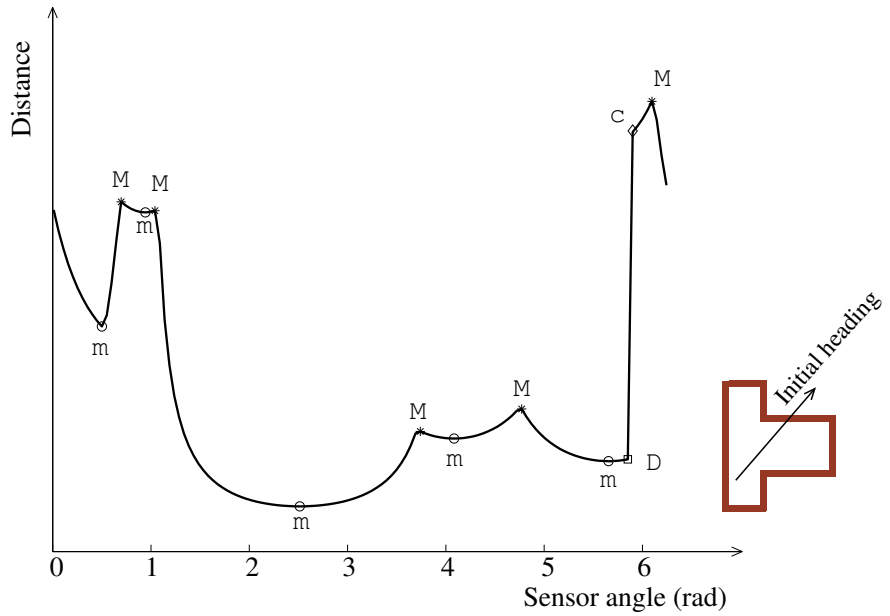


Fig. 34. Range measurements for perfect sensors.

scan label is a string composed of the features visible from a sector, and it is computed and stored for all sectors during preprocessing. Figure 35 shows the pseudo-code for constructing scan labels. In the pseudo code, R denotes range measurements, $\circ$ denotes string concatenation, and ℓ_{R} is the scan label. The threshold value for determining discontinuity is denoted by ε_T. In addition to the labels, the global coordinates of features (`M` points in this case) are stored because they are needed for the localization. For the visibility sectors shown in Figure 30, scan labels are shown in Figure 36.

In certain situations, the pseudo code in Figure 35 does not generate the sector boundaries that we desire. Figure 37(a) shows an example where sector boundaries perpendicular to obstacles (such as the segment labeled a in Figure 37) would be required as well as the sector boundary b. With such sector boundaries, labels differ

CONSTRUCT SCAN LABEL(R)
1. $\ell_{\mathrm{R}} := \emptyset$
2. for $i := 0$ to N
3. if $(r_i - r_{i+1} > \varepsilon_T)$ then $\ell_{\mathrm{R}} := \ell_{\mathrm{R}} \circ$ `cD`
4. else if $(r_{i+1} - r_i > \varepsilon_T)$ then $\ell_{\mathrm{R}} := \ell_{\mathrm{R}} \circ$ `Dc`
5. else if $(r_{i-1} > r_i$ and $r_i < r_{i+1})$ then $\ell_{\mathrm{R}} := \ell_{\mathrm{R}} \circ$ `m`
6. else if $(r_{i-1} < r_i$ and $r_i > r_{i+1})$ then $\ell_{\mathrm{R}} := \ell_{\mathrm{R}} \circ$ `M`
7. end if
8. end for

Fig. 35. Pseudo-code for scan label construction.

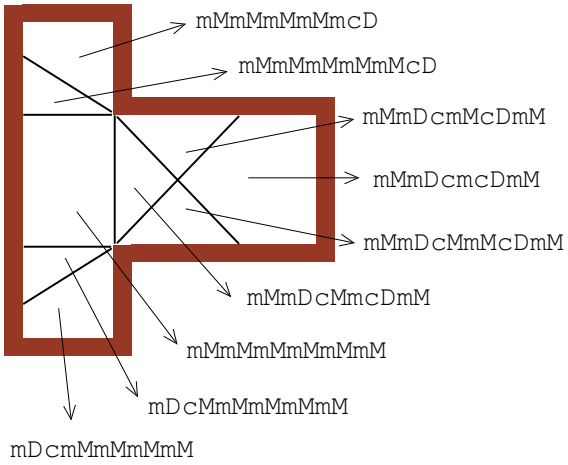
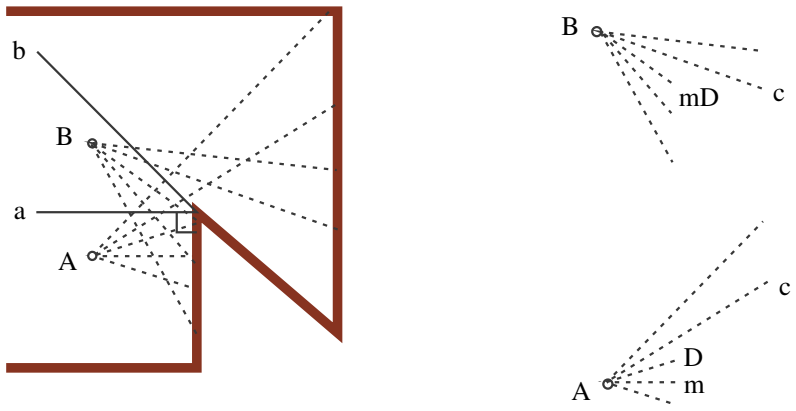


Fig. 36. Visibility sector labels (Robot's initial heading is to the right).

only by inclusion or exclusion of local minimum characteristic points. Since local minimum points are not useful for precise localization, we choose to eliminate such sector boundaries by modifying CONSTRUCTSCANLABEL (see Figure 38).

In the restricted visibility case, we suggest a merge–based method where the robot can estimate a local maximum using two local minima. A merged sector has at least one common feature `M` visible from all points in the sector. However, there might be additional `M` features which are not guaranteed to be scannable in the sector.



(a) (b)

Fig. 37. Sector boundary with local minimum point. (a) Using the pseudo code shown in Figure 35, scan labels obtained from the positions A and B are `mDc` and `Dc`, respectively. Because they are different, a sector boundary a is needed. (b) Using the modified pseudo code shown in Figure 38, the same scan label is obtained from the positions A and B.

CONSTRUCT SCAN LABEL(R)

1. $\ell_{\mathrm{R}} \leftarrow \emptyset$
2. for $i \leftarrow 0$ to N
3. if $(r_i - r_{i+1} > \varepsilon_T)$
4. if $(r_{i+1} < r_{i+2})$ $\ell_{\mathrm{R}} \leftarrow \ell_{\mathrm{R}} \circ$ `cDm`
5. else $\ell_{\mathrm{R}} \leftarrow \ell_{\mathrm{R}} \circ$ `cD`
6. else if $(r_{i+1} - r_i > \varepsilon_T)$
7. if $(r_i < r_{i-1})$ $\ell_{\mathrm{R}} \leftarrow \ell_{\mathrm{R}} \circ$ `mDc`
8. else $\ell_{\mathrm{R}} \leftarrow \ell_{\mathrm{R}} \circ$ `Dc`
9. else if $(r_{i-1} > r_i$ and $r_i < r_{i+1}$ and $r_{i-1} - r_i < \varepsilon_T)$ $\ell_{\mathrm{R}} \leftarrow \ell_{\mathrm{R}} \circ$ `m`
10. else if $(r_{i-1} < r_i$ and $r_i > r_{i+1}$ and $r_i - r_{i-1} < \varepsilon_T)$ $\ell_{\mathrm{R}} \leftarrow \ell_{\mathrm{R}} \circ$ `M`
11. end if
12. end for

Fig. 38. Modified pseudo-code for scan label construction.

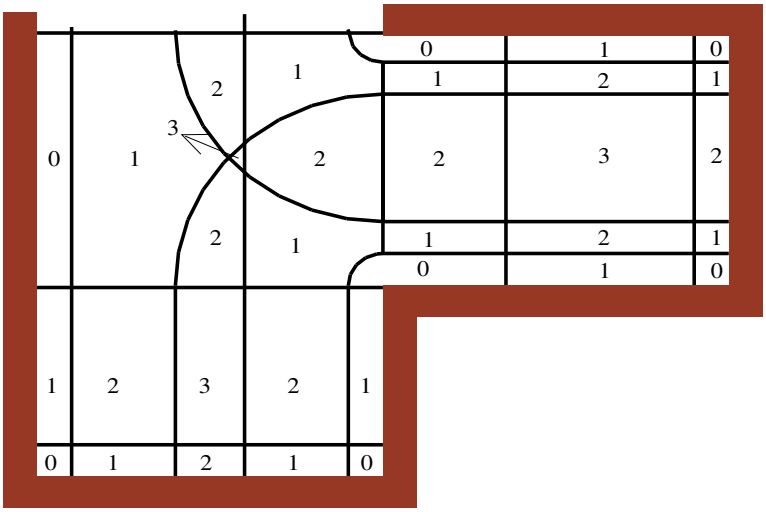
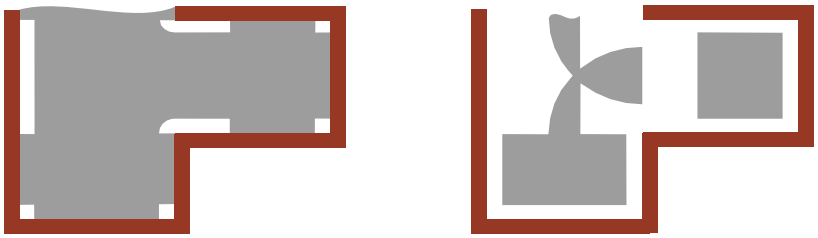


Fig. 39. Visibility numbers for each sector.

In this case, we prefer retaining all possible features since it maximizes the chance of localization in the presence of sensor noise and unknown/moving obstacles.

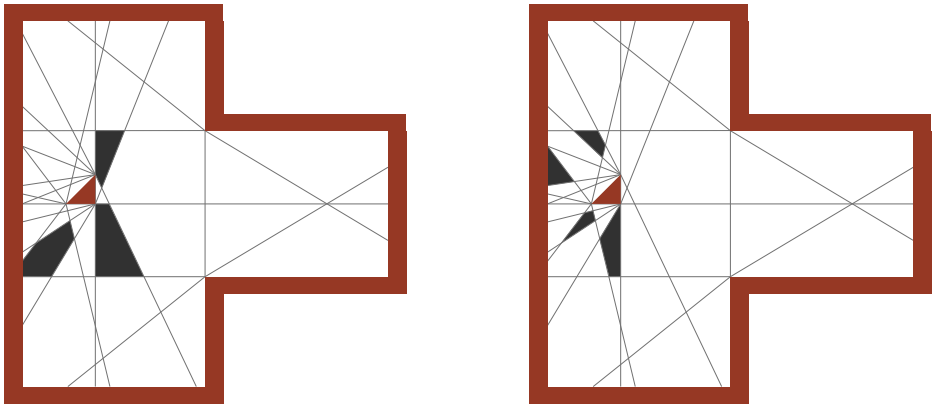
If only `m` points are used as features, then sector merging is not required because the features for sector subdivision and localization are the same. The *visibility number* of a sector is the number of features visible from the sector (Figure 39). With this information, we can easily see where the robot can effectively localize with sensor limitations. Figure 40(a) shows the region that the robot's uncertainty can be reduced to a smaller region, and (b) shows the region where the exact pose of the robot can be computed.

Ambiguous sectors are a set of sectors that have cyclically identical labels [52]. Two examples are shown in Figure 41 where (a) and (b) have labels `mMmMmMmcDmDcmMmM` and `mMmMmMmMcDmDcmM`, respectively. These sectors cannot be distinguished by labels if the robot's heading or some other additional information is not known.



(a) (b)

Fig. 40. Regions with different numbers of visible features. (a) One or more, and (b) two or more local minimum points for limited visibility.



(a) (b)

Fig. 41. Two examples of ambiguous sectors

5. Geometric Operations

In Figure 23(b), two primitive sectors yield three visibility sectors. Let s and p denote visibility sectors and primitive sectors, respectively. Computing s from p requires three Boolean polygon operations.

- $s_A = p_1 \setminus p_2$
- $s_B = p_1 \cap p_2$

- $s_C = p_2 \setminus p_1$

Computing the visibility sectors s_A, s_B and s_C from primitive sectors p_1 and p_2 is described in the pseudo code in Figure 42. During the computation, generalized polygons are used because s_A, s_B and s_C can be empty, or consist of one or more components. Examples are shown in Figure 43 where (a) s_A and s_C have two components, (b) s_B has two components, (c) s_A is empty and s_C has a hole and (d) s_B is empty.

```
COMPUTE VISIBILITY SECTORS(p1, p2, SA, SB, SC)
1.  sA ← p1,   sB ← p1 ∩ p2,   sC ← p2
2.  if sB = ∅
3.      SA ← SA ∪ {sA}
4.      SC ← SC ∪ {sC}
5.  else
6.      for each component sBi ∈ sB
7.          sA ← sA \ sBi
8.          sC ← sC \ sBi
9.          SB ← SB ∪ {sBi}
10.     SA ← SA ∪ sA  (for each component of sA)
11.     SC ← SC ∪ sC  (for each component of sC)
12.     end for
13. end if
```

Fig. 42. Pseudo-code to compute visibility sectors from two primitive sectors.

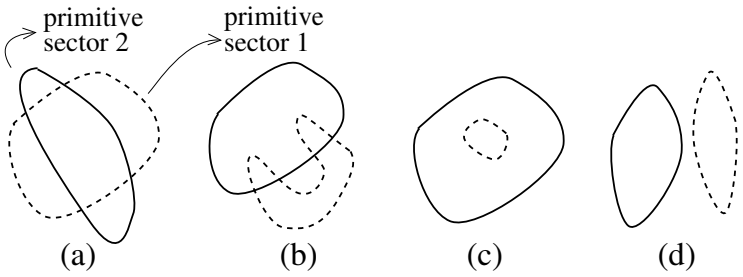


Fig. 43. Sectors represented by generalized polygons.

The algorithm sketched in Figure 44 constructs all the visibility sectors in the environment. It uses the COMPUTEVISIBILITYSECTORS subroutine shown in Figure

42. The input is a set of primitive sectors $P = \{p_1, p_2, \ldots, p_{n_p}\}$, and the output is a set of visibility sectors $S = \{s_1, s_2, \ldots, s_{n_s}\}$ where n_p and n_s are the number of primitive sectors and visibility sectors, respectively. In the pseudo code, t is the primitive sector that will be partitioned into visibility sectors in each iteration. For example, in Figure 45, t is initially set to p_7 and it is eventually divided into three components (s_8, s_9, s_{10}).

```
CONSTRUCT VISIBILITY SECTORS(P, S)
1.  S = {p_1}
2.  for i = 2 to n_p
3.      t ← p_i,   S_t ← ∅
4.      for each component s_i ∈ S
5.          COMPUTEVISIBILITYSECTORS(s_i, t, S_A, S_B, S_C)
6.          S_t ← S_t ∪ S_A  (for each component of S_A)
7.          S_t ← S_t ∪ S_B  (for each component of S_B)
8.          t ← S_C
9.      end for
10.     S ← S ∪ t   (for each component of t)
11.     S ← S ∪ S_t  (for each component of S_t)
12. end for
```

Fig. 44. Pseudo-code to compute visibility sectors from primitive sectors.

(a) (b)

Fig. 45. Six visibility sectors are added by a new primitive sector. (a) Before subdividing and (b) after subdividing.

6. Time Complexity Analysis

The basic strategy for accelerating the actual localization is to do as much computation as possible during preprocessing. Nevertheless, it must be shown that constructing the sectors is feasible, i.e., the time and storage requirements are reasonable. In this section, we analyze the time required for constructing the sector boundary. In [24], it was shown that the number of visibility cells (n_{vs}) is $O(n^2r)$ and that the number of line segments composing a sector boundary is $O(nr)$, where n is the number of line segments in the environment and r is the number of reflex vertices.[1] It has been proven in [24] that the visibility cell decomposition can be computed in time $O(n^2r)$.

In the following we analyze the time complexity for merging sectors. A polygonal environment can be represented by a planar graph $G = (V, E)$. To merge, we eliminate the edge(s) common to the sectors and take the union of the remaining edges. The adjacent sectors can be found in linear time using G's adjacency list. In most cases, the number of adjacent sectors is small, and the search takes almost constant time. Thus, a merge operation will take $O(m_{be})$ time, where m_{be} is the number of bounding edges of a sector.

The value of m_{be} varies as merging progresses, and we consider best and worst cases for our analysis. In the worst case, sectors share only one or very few edges. As a result, m_{be} is almost doubled after merging. The best case is when the sectors share as many edges as possible so that m_{be} does not increase much or even decreases. An example of the latter case is shown in Figure 46 where the number of edges of sector 1 is decreased from 16 to 4.

Whether a merging process is close to the best or the worst case does not mainly depend on environment or sensor properties; for any merging process, m_{be} might

[1] a vertex which subtends an angle greater than 180^o inside primitive sector

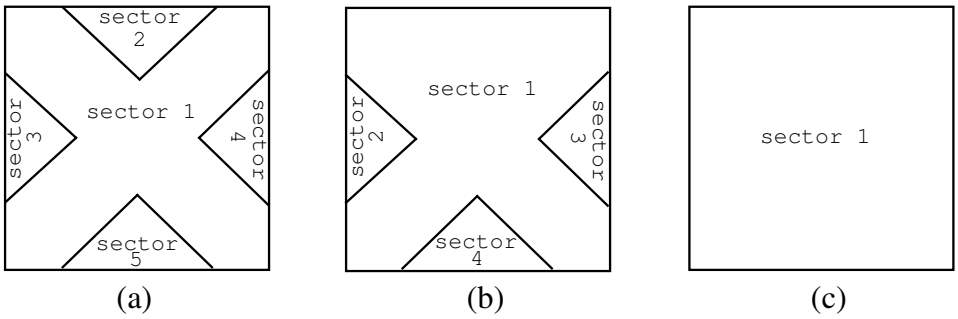


Fig. 46. Merging sectors resulting in smaller number of edges. (a) Before merging, (b) after merging sector 2 into sector 1, and (c) merging all sectors into one.

increase and then decrease, or vice versa. The worst case usually occurs in the beginning part of a merging process because the (original) visibility sectors are convex and they share only one edge. In the middle or in the later part of the merging process, the situation may be closer to the best case, as illustrated in Figure 46.

To express the upper bound on m_{be} in terms of n and r, we assume that the worst case dominates the merging process. This means that after merging k times, m_{be} is $O(nr\,2^k)$ since $m_{be} = O(nr)$ in the beginning. If all the sectors are involved in merging at each step, i.e., the sectors are finally merged into one, then there are a total of $n_{vs} - 1$ merges. In this case, we refer to a *merging step* as finding all pairs of sectors to be merged and then merging all the pairs, resulting in doubled m_{be} for all sectors and halved total number of sectors. The total number of merging steps n_k can be as large as $\lceil \log(n_{vs} - 1)) \rceil$, but in practice, n_k is smaller because we don't always obtain a single sector after merging.

In the first merging step, at most $n_{vs}/2$ sectors are merged, and at the kth merging step, at most $n_{vs}/(2^k)$ sectors are merged. So, the total time for merging

sectors is

$$\begin{aligned} T_{rs} &= 2nr\tfrac{n_{vs}}{2} + 4nr\tfrac{n_{vs}}{4} + \cdots + 2^k nr\tfrac{n_{vs}}{2^k} \\ &= nr\,\lceil \log(n_{vs}-1)) \rceil \\ &= O(nr\,\log(n^2 r)) \end{aligned}$$

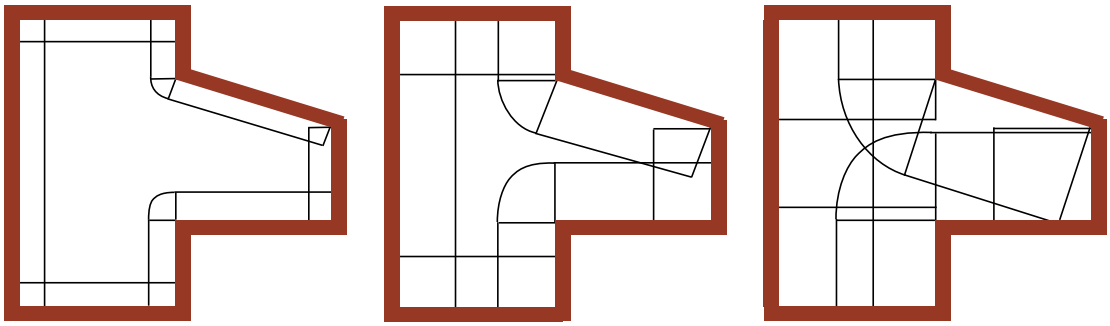
7. Space Complexity Analysis

In this section, we analyze the size of the sector data structure when the sensors have a range limitation. Based on the previous result for ideal sensors, we compute the upper bound on the number of visibility sectors for realistic sensors.

To obtain the upper bound on n_{vs}, the first step in [24] was to show that there are $O(nr)$ lines introduced in the interior of the environment. Though an obvious upper bound on the number of sectors is $O(n^2r^2)$, the structure of the sector subdivision problem was exploited and a tight upper bound on n_{vs} was shown to be $O(n^2r)$. With limited sensor maximum range, the interior lines become shorter but additional lines are introduced due to the sector boundaries representing the range limit.

If the sensor range is shorter than the walls (see Figure 47(a) as an example), then each wall introduces between 1 and 3 interior line segment(s). Also, each concave corner (a reflex vertex) introduces two interior line segments and a curve. The lower bound on the number of interior lines introduced is $\Omega(n+r)$ which is the same as the upper bound on the number of primitive sectors for wall and corner features. The lower bound on n_{vs} is also $\Omega(n+r)$ since each pair of overlapping primitive sectors generates a new visibility sector. When the sensor range is increased, the number of sectors is increased accordingly (Figures 47(b) and (c)). However, the upper bound on $O(n_{vs})$ is same as the ideal sensor case, $O(n^2r)$; the number of interior lines can be at most three times larger and $O(3nr) = O(nr)$.

Now that n_{vs} is shown to be bounded by $\Omega(n+r)$ and $O(n^2r)$, we are interested



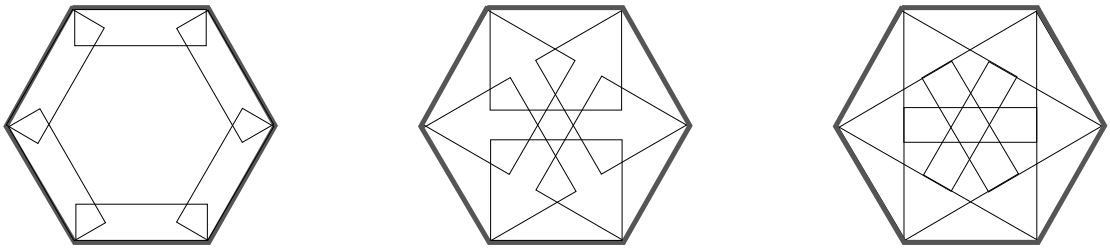
(a) (b) (c)

Fig. 47. Visibility sectors with maximum sensor range limitation.

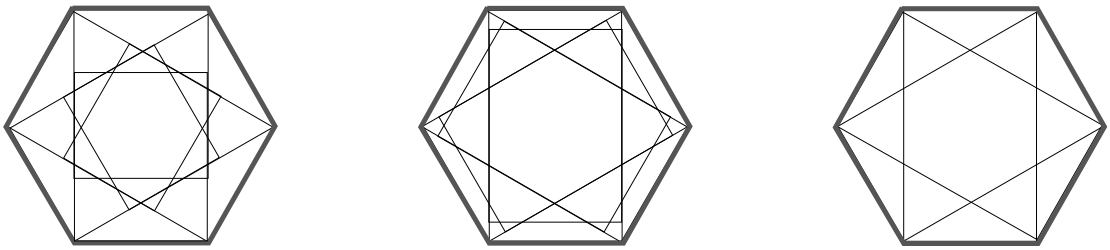
in describing n_{vs} as a function of the sensor maximum range (M) as well as n and r. The exact value of n_{vs} depends on the shape of the environment, which is hard to parameterize. Our approach is to consider a simple case for analysis and to compare with the result from a real environment. In this section, we use an environment that is symmetric, convex, and can be represented by a simple polygon.

In Figure 48(a), only adjacent sectors overlap and n_{vs} is 12 or $2n$ where $n = 6$ (hexagon). If the sensor maximum range M is longer than $R\arccos(\theta)/2$ (see Figure 49(a)), then n_{vs} is increased to 24 (see Figure 48(b)). If M is longer than $R\cos(\pi/n)$ (see Figure 49(a)), then primitive sectors on opposite walls overlap and n_{vs} is 31 (see Figure 48(c)).

Equation 4.1 shows five different numbers of sectors depending on the sensor range in the hexagon environment. The corresponding graph is shown in Figure 50, which is similar to the simulation results for our real office environment shown in Chapter VI Section B.2. From these, we observe that n_{vs} tends to increase with the sensor range, and reaches its maximum value when the sensor maximum range M is between the radius and the diameter of the environment, and then n_{vs} drops if it nears or exceeds the size of the environment.

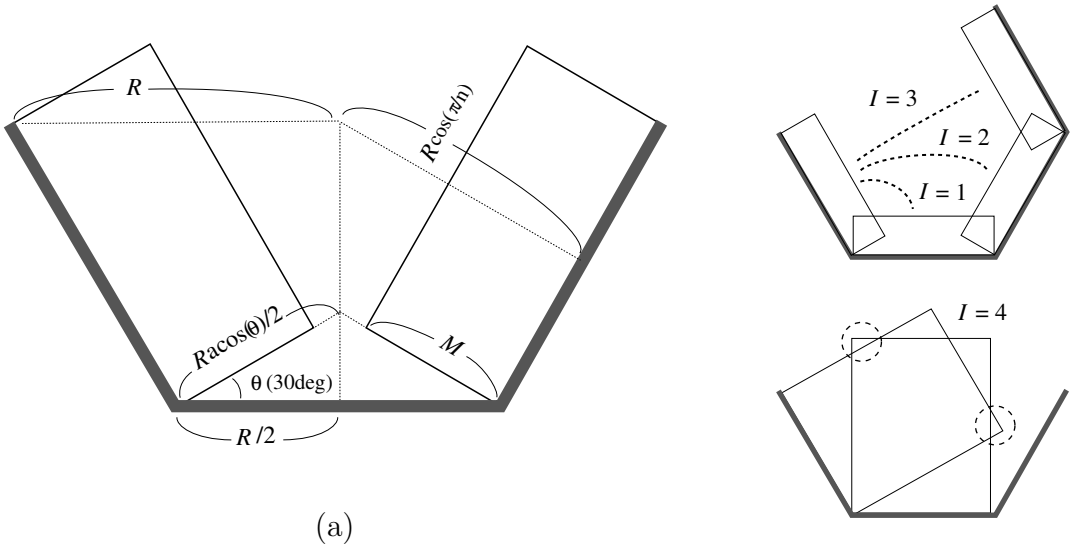


(a) $n_{vs} = 12 = 2n$ (b) $n_{vs} = 24 = 4n$ (c) $n_{vs} = 31 = 5n + 1$



(d) $n_{vs} = 37 = 6n + 1$ (e) $n_{vs} = 37$ (same as (d)) (f) $n_{vs} = 13 = 2n + 1$

Fig. 48. Visibility sectors for hexagon environments.



(a)

Fig. 49. Conditions for sector intersections. (a) Definitions of R, θ, and M. (b) A primitive sector intersects with its adjacent sector ($I = 1$), its next adjacent sector, and next ($I = 2, 3 \ldots$). (c) A primitive sector passes through its adjacent sector ($I = 4$).

$$n_{vs} = \begin{cases} 2n & \text{if} \quad M < \frac{R \ \arccos(\theta)}{2} \\ 4n & \text{if} \quad \frac{R \ \arccos(\theta)}{2} < M < R\cos(\pi/n) \\ 5n+1 & \text{if} \quad R\cos(\pi/n) < M < R \ \arccos(\theta) \\ 6n+1 & \text{if} \quad R \ \arccos(\theta) < M < 2R\cos(\pi/n) \\ 2n+1 & \text{if} \quad 2R\cos(\pi/n) < M \end{cases} \tag{4.1}$$

where $\theta = 30^{\circ}$ and $n = 6$

Though it is difficult to capture the parameters for computing n_{vs} based on the shape and size of general environments, we can investigate the relation between n_{vs} and M in more detail. A general observation is that n_{vs} increases as more and more primitive sectors overlap. Figure 49(b) introduces a new variable I which denotes the number of primitive sectors that simultaneously intersect. The smallest I is 1 where only adjacent primitive sectors overlap ($M < R\arccos(\theta)/2$). The value is increased

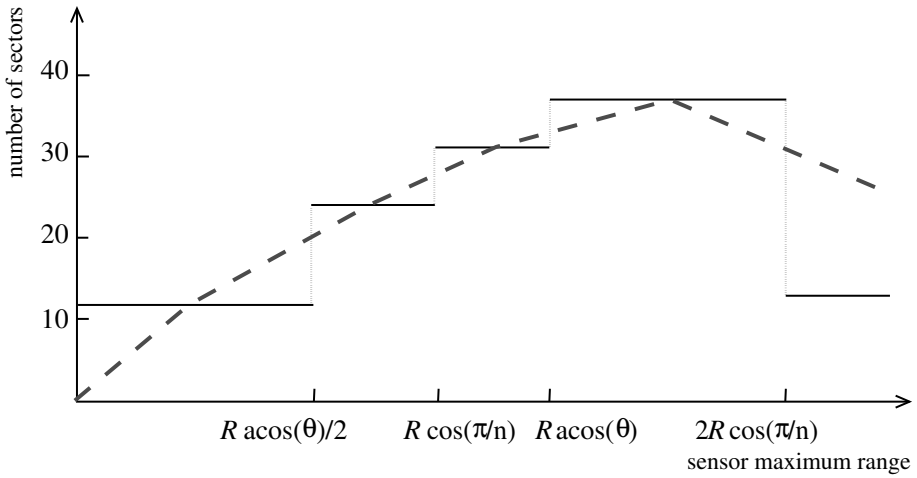


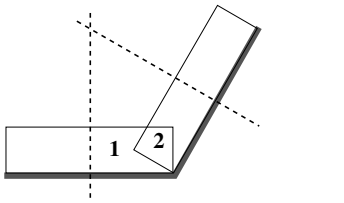
Fig. 50. Number of sectors for M (sensor maximum range) in five regions.

to 2 if the primitive sectors overlap the sectors adjacent to their neighboring sectors. In the hexagon case, I becomes 3 if a sector intersects the opposite primitive sector, and 4 if it passes through its adjacent sector.

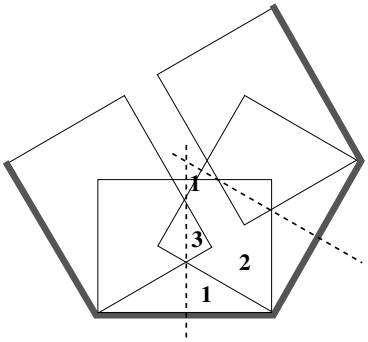
The number of features in the sectors is another factor that affects the size of the sector data structure. In general, the average number of features in the sectors increases with I. In Figure 51, the average number of features is 1.5, 1.75, 3.09, 3.08 for $I = 1, 2, 3, 4$, respectively. In particular, if the opposite primitive sectors meet, the maximum number of features jumps from 3 to 6.

C. Localization Algorithm

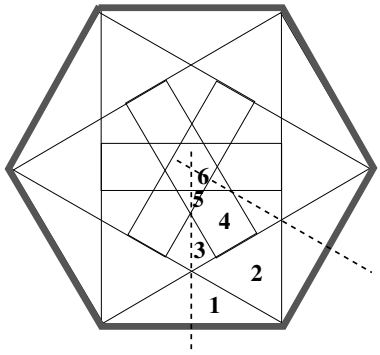
In this section, our methods for localization using visibility sectors are presented. The application of visibility sectors to particle filters and Kalman filters is conceptually described. After that, we analyze the properties of our sector-based methods from a theoretical point of view. This, together with the experimental results presented in Chapter VI, supports our claim that our sector-based localization methods are



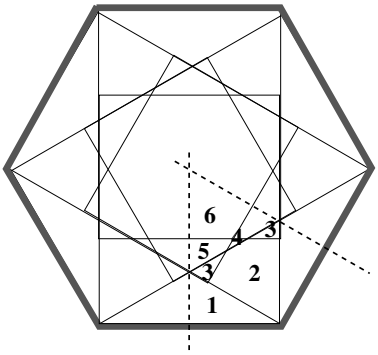
(a) $I = 1$



(b) $I = 2$



(c) $I = 3$



(c) $I = 4$

Fig. 51. Visibility numbers for sector intersection patterns

efficient and robust.

1. Localizing to a Sector

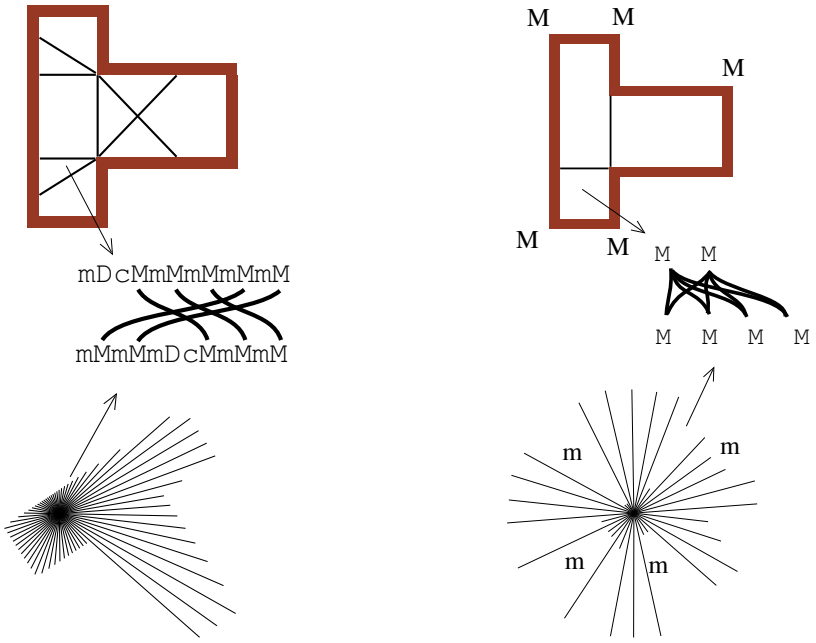
Figure 52 shows the pseudo code for sector–based localization which implements the concept introduced in Section A. The first step narrows down the search space using the information from the robot's odometry. After that, steps from 3 to 6 attempt to reduce the number of possible sectors to one. In line 5, the Cartesian product of the scanned and preprocessed features is computed to form *feature pairs*. The pair is stored in F_{match} only if the two features in the pair are matching. The subroutine REDUCE UNCERTAINTY is called when F_{match} is not empty for a sector s.

```
SECTOR–BASED LOCALIZATION(unc_reg, scans)
1.  S ← sectors intersecting with unc_reg
2.  F_scan ← features in scans
3.  for each s ∈ S
4.      F_sector ← features in s
5.      for each (f_scan ∈ F_scan)
6.          F_sector−match ← F_sector matching f_scan
7.          REDUCE UNCERTAINTY(unc_reg, F_sector−match, f_scan)
8.      end for
9.  end for
10. return unc_reg
```

Fig. 52. Pseudo-code for localization based on visibility sectors.

When visibility is perfect with complete environment knowledge, the process of matching a feature pair is a simple string comparison as in [52]. Since the robot's heading is not known in general, cyclic matching is needed. Figure 53(a) shows an example where five `M`'s are 1–to–1 matched after rotating the scanned label to the left four times. The robot's orientation can be roughly computed by counting the rotations, but a more precise value can be computed when reducing the uncertainty.

When only partial sector labels are obtained due to restricted visibility, exact



(a) (b)

Fig. 53. Matching features after localized to a sector. (a) using complete scan label [52], and (b) with limited visibility, using local maximum points estimated from local minimum points.

cyclic string matching does not work. One way to overcome this difficulty is to get additional information from the robot's odometry. If the robot's orientation is known approximately, then the scanned label can be rotated accordingly before the string matching is performed. Depending on the angular range of the robot's known heading, there might be 1–to–1 or 1–to–many matching. Figure 54 shows the geometric process of feature matching that was used in our navigator implementation (Chapter VI). The robot's position uncertainty is shown in Figure 54(a) and its intersection with the sectors results in three pieces (Figure 54(b)). The expected feature boundary for Figure 54(c) is shown in Figure 54(d), and is characterized by three measures: two distances A, B and an angle C. The A and $A + B$ are the minimum and maximum distances between the uncertainty region and the feature,

respectively. These determine the minimum and maximum possible distances to the scanned feature. The angle C corresponds to the orientation uncertainty of the robot. In this case, the robot's angular range is small enough to allow 1–to–1 matching. If the angle C is as large as 180 degrees, then the upper and lower walls cannot be distinguished.

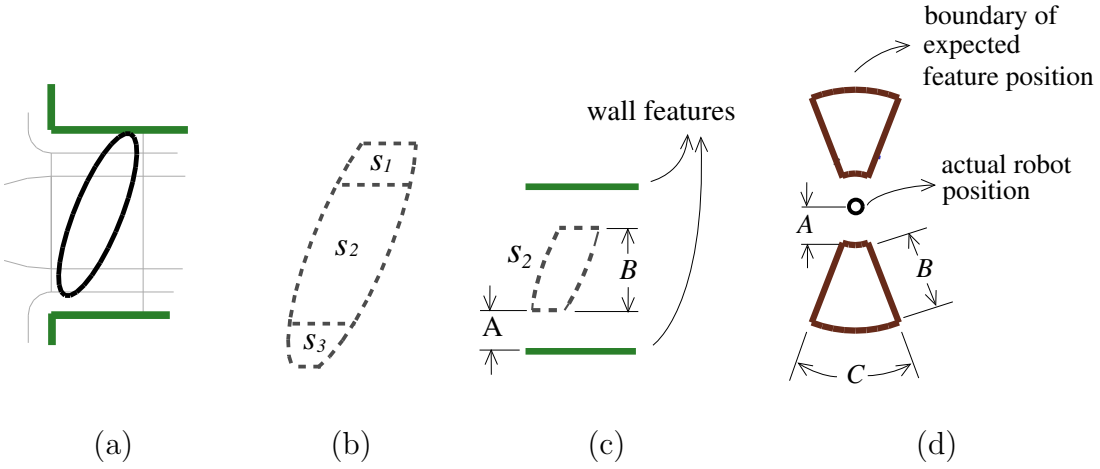


Fig. 54. Matching scanned and preprocessed features using the robot's pose uncertainty. (a) An uncertainty ellipse and wall features with corresponding sectors. (b) Three pieces of an ellipse intersecting sectors with feature(s). (c) A portion of a sector intersection and features. (d) Expected boundaries for two wall features.

Another method would be to use the "minimum string edit distance" to find a close match. This works even if the lengths of the scanned label and the preprocessed label are different. A unit string edit adds a new character, deletes a character, or changes a character in the label. Each editing operation adds one to the distance value. The total edit distance of two strings can be quickly computed using dynamic programming [16].

Due to many reasons including sensor limitations and unknown/moving obstacles, it is often impossible to get 1-to-1 matching of scanned and preprocessed features.

A practical localizer must try all possible pairs of features and verify later. (A verification method is discussed in the next section.) Note that trying multiple feature pairs is still more efficient than raw scan data matching because the union of all scanned features is the subset of the entire scan data. An example for this case is shown in Figure 53(b).

2. Reducing the Uncertainty Region

In this section, three ways of using visibility sectors for localization are presented. First, using a particular position uncertainty representation (an uncertainty ellipse), we explain a complete localization algorithm. The *uncertainty ellipse* is centered about the nominal estimate and is the constant probability contour for a multivariate Gaussian distribution [69]. In our framework, it is a conservative boundary containing all expected positions of the robot.

If we explicitly compute the precise configuration of the robot, then the ellipse is reduced to a point. We call this *localizing to a configuration*. In [52], it was assumed that local maximum points are scanned or correctly estimated. Let (X_M, Y_M) and (x_M, y_M) denote the coordinates of the scanned local maximum points in the global and the local coordinate systems. The coordinates in the real scan (in the robot's local coordinates) and in the map data (in global coordinates) are equated to compute the robot's configuration. The robot configuration (x_o, y_o, Θ_o) denotes the linear transformation from global to local with translation (x_o, y_o) and rotation (Θ_o) using Equation 4.2.

$$\begin{bmatrix} -\cos\Theta_o & \sin\Theta_o & X_M \\ -\sin\Theta_o & -\cos\Theta_o & Y_M \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_M \\ y_M \\ 1 \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ 1 \end{bmatrix} \tag{4.2}$$

If a local minimum point is scanned, then we have only one dimensional information and Equation 4.2 is undetermined. In this case, we can shrink the uncertainty region to an uncertainty line segment whose direction is perpendicular to the sensor beam (Figure 55(b)). If the sensor's Gaussian noise is not small enough to be neglected, then instead of using an uncertainty line segment we use a thin ellipse or a thin rectangle instead (Figure 55 uses a thin ellipse). This is relaxed localization using a local minimum feature.

With two or more local minimum features, we can localize to a configuration by combining the information from each feature. One way of combining two ellipses is by geometrically intersecting the ellipses; a rigorous method based on grids is explained in [32]. However, if the walls are parallel to each other, then the feature information is linearly dependent and cannot be used for localizing to a configuration.

After attempting to reduce the uncertainty region, a confirm step is required to choose the correct result. Using the robot's pose uncertainty information, a necessary (but not sufficient) condition is that the computed robot's configuration must be inside the uncertainty ellipse. In Figure 55(c), if the two ellipses do not overlap, then it indicates that the result is invalid. A sufficient condition to verify the localization is to use a *synthetic scan* which is a software scan of the model environment. After a configuration is determined from the localization, if the synthetic scan and the actual scan are close enough to each other, then the verification is complete. This step is computationally expensive and should be used after computationally simpler checks

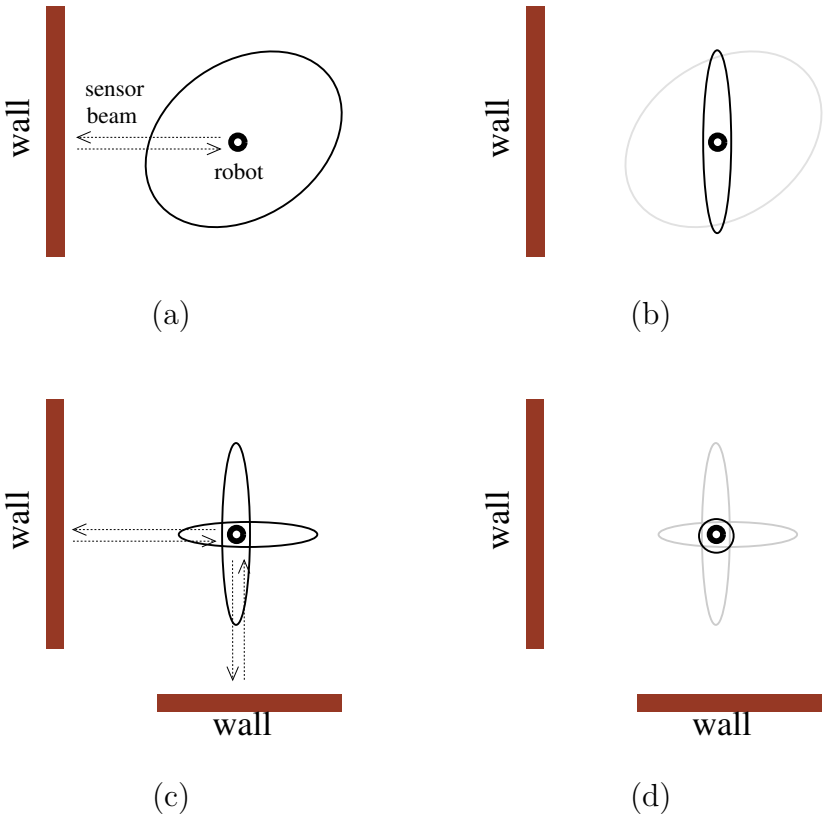


Fig. 55. Relaxed and precise localization. (a) Sensing a wall. (b) Uncertainty ellipse is reduced to a thin ellipse or a line segment. (c) Sensing two walls. (d) Uncertainty ellipse is reduced to a small circle or a point.

have been satisfied. The pseudo code in Figure 56 shows the algorithm for reducing an uncertainty ellipse and confirming the result.

Second, we consider feature–based uncertainty reduction for Monte Carlo localization. Particle filters use sets of samples to represent the belief of the state $x_t = < x, y, \theta >$ [22]. After computing the next state x_t from odometry and the previous state x_{t-1}, we need a perceptual model describing the likelihood of an observation y_t given that the robot is at location x_t. In [22], the probability $p(y_t \mid x_t)$ is computed by ray–tracing using a sensor uncertainty model. Using features such as local minimum and maximum points, an easy way to obtain $p(y_t \mid x_t)$ is to compute

```
REDUCE UNCERTAINTY(unc_reg, F_sector−match, f_scan)
1.  for each (f_sector−match ∈ F_sector−match)
2.      unc_reg_scan ← reduce with f_scan, f_sector−match
3.      if unc_reg_scan ⊂ unc_reg
4.          if confirm unc_reg_scan with synthetic scan
5.              unc_reg ← unc_reg ∩ unc_reg_scan
6.          end if
7.      end if
8.  end for
```

Fig. 56. Pseudo-code for reducing uncertainty region.

a reduced uncertainty ellipse and then to check if each particle is inside the ellipse or not. More sophisticated methods of computing $p(y_t \mid x_t)$ based on a Gaussian distribution are possible.

A benefit of visibility sectors is that the probability can be obtained quickly. For instance, in Figure 23, x_t in sector A will attempt to match y_t with Feature 1, sector C with Feature 2, and sector B (which is A $\cap$ C) with Features 1 and 2. Visibility sectors also help to decide how many actual and synthetic scans are required to compute $p(y_t \mid x_t)$ with a specified confidence.

Third, feature–based methods can help to efficiently compute extended Kalman filters. For sensors with limited visibility, the uncertainty must be encoded as a nonlinear function in the measurement model. In discrete extended Kalman filters, the measurement model $\mathbf{z}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{w}(k)$ describes the sensor observation $\mathbf{z}(k)$ computed by a nonlinear function $\mathbf{h}$ and the state $\mathbf{x}(k)$ [61]. A valid linearization of $\mathbf{h}$ at the estimated $\mathbf{x}$(k) can be modeled depending on the features expected from the preprocessed data. Or, if explicit linearization of $\mathbf{h}$ is not necessary as in [32], then visibility sectors can help decide which feature(s) should be used for computing the mean and variance of $\mathbf{z}(k)$.

3. Efficiency and Robustness

The primary advantage of sector–based methods is computational efficiency: the sector information enables one to restrict the search for matching scanned and known features. Visibility sectors enable localizers to exploit spatial locality when matching features in the environment. Once localized to a sector, only a small set of features needs to be processed when reducing the uncertainty region. Moreover, since they are resolution independent, visibility sectors do not sacrifice accuracy as do some other subdivision methods.

The secondary benefit of sector–based localization is related to robustness. First, after scanning, the localizer can easily estimate if the scanned data contains enough information for successful localization. The robot might need to scan again, or move a short distance to look for other features. In this process, searching neighboring sectors is more efficient than searching the entire environment.

Due to sensor limitations which sometimes render areas in the environment featureless, it has been observed that determining good subgoals is sometimes more important than sophisticated feature identification methods. For this purpose, we have suggested an integrated roadmap-based path planner and localizer. Given a path selected from the roadmap, a good subgoal is the farthest node from the start at which successful localization is expected. This requires knowledge of the visible features for each node on the path, which is facilitated by the sectors data structure which records the feature visibility from a sector.

One way to guarantee feature visibility along any path is, when constructing the roadmap, to generate random nodes in sectors that have one or more visible features. Such a roadmap may provide increased robustness for localization, but it may not be good for efficient navigation. Given a roadmap that can satisfy robust localiza-

tion and efficient navigation, our path planner selects a path that simultaneously, e.g., maximizes the number of visible features and minimizes the distance traveled. Visibility sectors help the path planner to quickly evaluate feature visibility and the roadmap provides a way to determine path length.

Additionally, visibility sectors help estimate the chance of successful global localization. Here, we assume that given perfect visibility and no ambiguous sectors, the robot can localize to a configuration without any prior knowledge of the robot's pose. As the relative area of the ambiguous region increases, global localization becomes difficult. Figure 57(a) shows an ambiguous region whose area is 26% of the entire environment. Such regions are ambiguous due to non-unique sector labels, which we can decrease by extending feature sets. A new feature `l`, which means 'long distance between two features', has been added. The result is shown in Figure 57(b) where all sectors have unique labels. In this case, the ambiguity was caused by the vertical symmetry of the environment. It has been resolved since the new feature determines if the discontinuity is before or after a long wall. Note that the role of visibility sectors is not to create robustness but to facilitate localization when one has some idea of the correct location.

Finally, visibility sectors can deal with dynamic environments where new features can appear and known features can disappear. If changes are not very significant, then it may be sufficient that many original features are visible so that navigation can proceed using only those features. Otherwise, when feature information is changed, the corresponding sector should be inserted or deleted. The most expensive step in this process might be updating sector boundaries, which involves dividing or merging simple polygons. Updating scan labels is straight forward. This is more efficient than grid–based methods where the number of affected grid cells will often be larger than the number of sectors that need to be modified. So, in addition to localization,

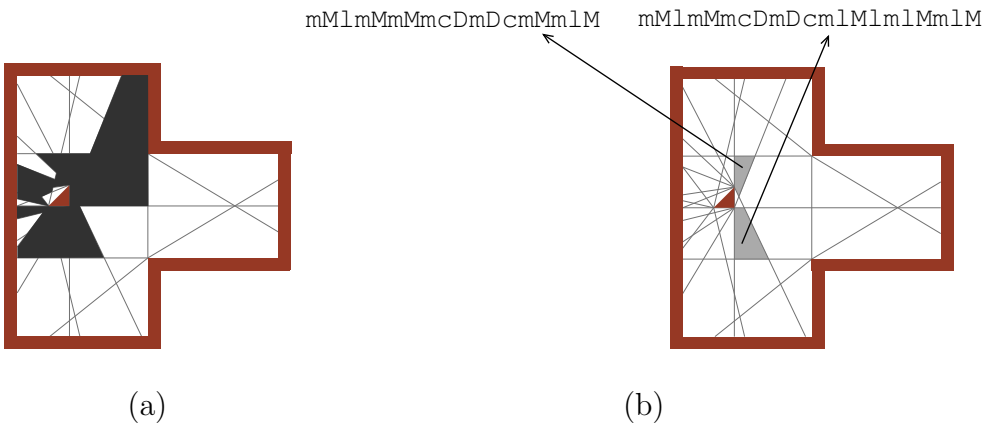


Fig. 57. Removing ambiguous regions. (a) Ambiguous region (union of 8 sets of ambiguous sectors). (b) New feature `l` added, and the ambiguity is resolved.

visibility sectors have potential application in map building algorithms (or SLAM).

The two major issues — efficiency and robustness — are competing and represent trade-offs that may not be achieved at the same time. For example, defining complex features and accurately modeling the environment results in a large sector data structure with ample feature information. A general rule in designing features and sectors is that robustness should be maximized to the extent that is required, i.e., sectors should have adequate size and an appropriate number of features to satisfy mission requirements. An example of such a decision is presented in Section 2 using simulation results.

CHAPTER V
MULTIPLE ROBOT LOCALIZATION

In designing practical mobile robot systems, we often desire scalable systems that can be dynamically extended and configured to address the current needs. For example, a mobile robot designed to carry a small and light payload might be teamed together with additional robots to carry heavier items. Or, a surveillance robot that patrols a designated area might need to coordinate with other surveillance robots to effectively cover an entire shopping mall. In such cases, the algorithms used to coordinate the system should be easily extensible to accommodate different numbers of robots.

In this chapter, we define, analyze the complexity, and propose approximate solutions for two problems related to multi–robot localization. The difficulty of solutions for this problem depends on the capabilities of the robot and the sensors. For example, Kalman filters and particle filters focus on state estimation and often require omni-directional sensing capability. We aim to effectively localize multiple robots using a minimum number of sensings, assuming that the robots have a small number of inexpensive range sensors. We formulate this problem using a particular position uncertainty representation (uncertainty ellipses), but our approach can be applied to other methods as well.

A. General Assumptions

We investigate the localization problem for systems of multiple mobile robots that satisfy the following assumptions.

Indirect localization. When multiple robots are localizing, some robots may reduce their position uncertainty by sensing known obstacles (*direct sensing*) or other

robots (*indirect sensing* or also called cooperative localization [60]). Localizing multiple robots with known obstacles can be handled simply by applying a traditional localization algorithm for each robot. Our interest here is on indirect localization only.

Shared information. Each robot has a unique ID and can communicate with the other robots. This can be realized by peer–to–peer messaging or by a client–server network configuration with wireless communication devices.

Stop all and localize. All robots must stop moving while any robot is scanning features. This is to ensure that there is no discrepancy in the positions of the other robots obtained by sensing and information sharing. In fact, this can be relaxed so that only the robots within sensor range must remain stopped.

Interaction among robots. We consider scenarios where robots interact with other robots by scanning other robots, as well as by direct communication. One condition for this is that the distances between the robots are loosely bounded so that there always exists at least one robot within sensor maximum range. One example is flocking where robots move together.

No omni-directional sensor. Each sensor has a limited angular range of view, and sensing the entire environment requires rotating the sensor. To measure the distance to the other robots, since the direction to the feature is not exactly known, several measurements must be taken. The time required to sense a feature depends on the sensor angular range, the time required to rotate the sensor, the pose uncertainty of the robots, the diameter of the robots, and the time needed to take one measurement.

Fast communication. The time spent communicating with other robots is insignificant and can be neglected. In our algorithm, the majority of the communication

among robots involves sharing uncertainty region information.

Sensor crosstalk model. Depending on the type of sensors, many different factors can cause cross–talk. We assume that such reasons are well modeled and can be computed, i.e., in this work, we do not explain how to model cross–talk.

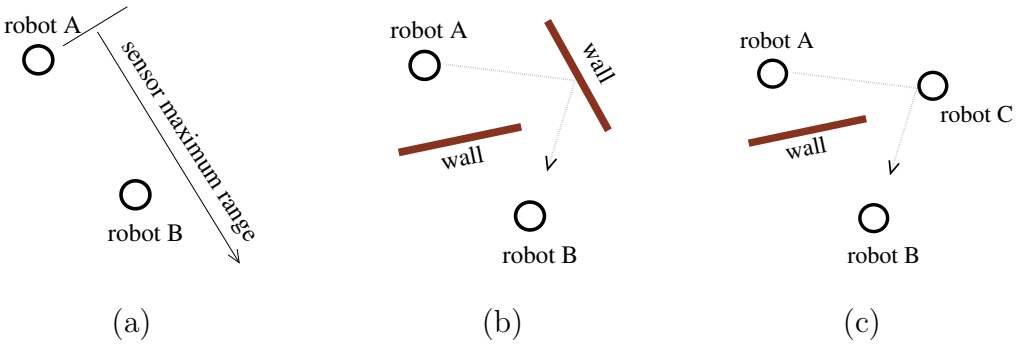
No Gaussian noise. We do not consider the effect of Gaussian sensor noise when sensing ranges. In reality, several measurements of a fixed object increases confidence, and might overcome the effect of Gaussian noise. However, for clarity in analysis, we assume that the uncertainty region is reduced to the smallest size after scanning once.

Sequence–independent uncertainty reducing. We assume that uncertainty reducing is associative, i.e., the order of processing the features does not matter. For example, localizing with feature A and then feature B results in the same reduced uncertainty as localizing with feature B and then feature A.

The first step in localization is sensing the distances (ranges) from the robot to objects in the environment. If the robot does not have enough sensors to provide a 360^{o} view, it might rotate (its sensors or itself) so that it can sense all necessary directions.

B. Avoiding Crosstalk

One difficulty in localizing multiple robots in the same environment is to avoid cross–talk while sensing. One solution is to allow only one robot to sense at a time. We attempt to improve this by assigning the robots to groups which can safely sense simultaneously. We will show that minimizing the total number of sensing rounds is NP–complete by reducing a known NP–complete scheduling problem to it.



(a) (b) (c)

Fig. 58. Examples of sensor cross–talk. (a) Two robots are located within maximum sensor range and can sense each other, (b) the signal from a robot bounces from a wall, and (c) the signal from a robot bounces from another robot.

1. Problem Definition

Definition: A *cross–talk graph* (CTG) is an undirected graph where the vertices are robots, and each edge connects two robots that can cross–talk if they sense simultaneously. That is, robots connected by an edge in CTG cannot safely sense at the same time.

To construct the graph, we need to know the possible robot positions bounded by their uncertainty region, the maximum sensor range, and other objects that can bounce sensor signal. Ray–tracing based techniques can be used to compute the possibility of cross–talk between two robots, and this determines if an edge for the robots exists in the graph. Figure 58 shows three possible robot configurations that can experience sensor cross–talk. For five robots, an example cross–talk graph is shown in Figure 59.

Definition: The *unit sensing time* t_s is the time spent by a robot sensing its environment. We assume that t_s is constant and is the same for all robots. The sensing process consists of *sensing rounds*.

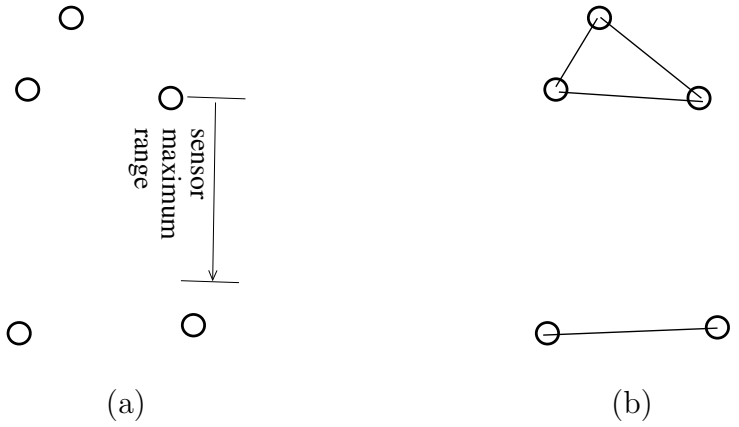
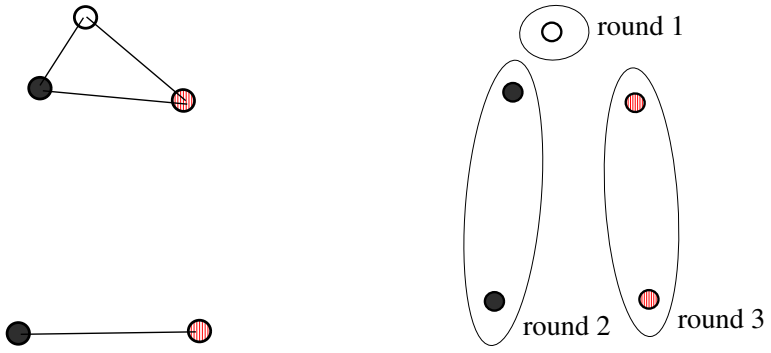


Fig. 59. An example of cross–talk graph. (a) An environment with five robots and no wall, and (b) the corresponding cross–talk graph.

Any group of robots between which there are no edges in the CTG form an independent set and can sense simultaneously without cross–talking. So, if we can partition the robots into k such groups, then we can sense in k rounds, which takes $k \cdot t_s$ time.

Definition: Given a cross–talk graph and a positive integer k, the *Range Sensing* problem (RS) is to schedule the robots so that they can all safely sense the environment in k rounds.

A solution of RS is a sensing schedule which assigns the robots to rounds so that no two robots adjacent in the CTG are assigned to the same round. An optimal schedule will minimize the number of rounds. The decision problem for RS is to determine if all robots can sense in k rounds. An optimization version of the problem is to minimize the number of sensing rounds, which has the same complexity as the decision version in terms of membership in P and NP.



(a) (b)

Fig. 60. A solution to a simple RS problem. (a) A solution of graph coloring problem, and (b) a valid cross–talk–free sensing schedule.

2. Graph Coloring

Given a undirected graph $G = (V, E)$ and an integer k, the graph coloring problem is to find, if one exists, a coloring of V using at most k colors so that no two vertices with the same color are joined by an edge in E. A valid coloring for the graph in Figure 59(b) is shown in Figure 60(a).

This problem is known to be NP–complete [23]. The complexity of NP–complete problems is unknown but in general they are believed to be intractable, i.e., that it is unlikely that they can be solved in polynomial time.

3. Proof that RS is NP–complete

We claim that RS is NP–complete. The proof follows the standard reduction technique which reduces a known NP–complete problem to RS.

RS is in NP. To verify a solution to RS, we check that each sensing round does not contain any two robots that share an edge in the CTG. This takes $O(n^2)$ time, where n is the number of robots.

RS is NP–hard. We show this by reducing graph coloring to RS. Given a graph coloring instance, a graph $G = (V, E)$, and an integer k, we construct an RS instance that has cross–talk graph G and integer k.

Figure 60(b) shows a solution that has been transformed from a valid colored G. If we can solve RS, then we obtain at most k groups of robots such that no two robots in any group are adjacent in G. Hence, an assignment of a different color to each group will result in a valid k–coloring of G. Thus, since k–graph coloring is known to be NP–Complete, RS is also NP–Complete.

C. Coordinating Reducing Ellipses

When localizing multiple robots in the same environment, the next step after sensing is to reduce our uncertainty in the previous positions of the robots (the ellipses). We aim to use a minimal number of total sensings, and this is shown to be NP-complete.

1. Issues

An example of ellipse reducing using a feature is shown in Figure 61. Robot 1 is localized using robot 2 because $d_1 > d_2$. Robot 1 attempts to sense in several directions since the relative position of the robot 2 is not exactly known. Here, d_1 and d_2 are measured along the line that connects the two robots. The reduced ellipse in Figure 61(c) is shaped using the four points intersecting two line segments and the large ellipse.

Each robot can sense all its visible features after a round of sensing. In this section, we define *a round of localization* as the process of reducing uncertainty using only one feature. Some localizers reduce the uncertainty region using several features simultaneously, but we observe that such localizations can usually be divided into

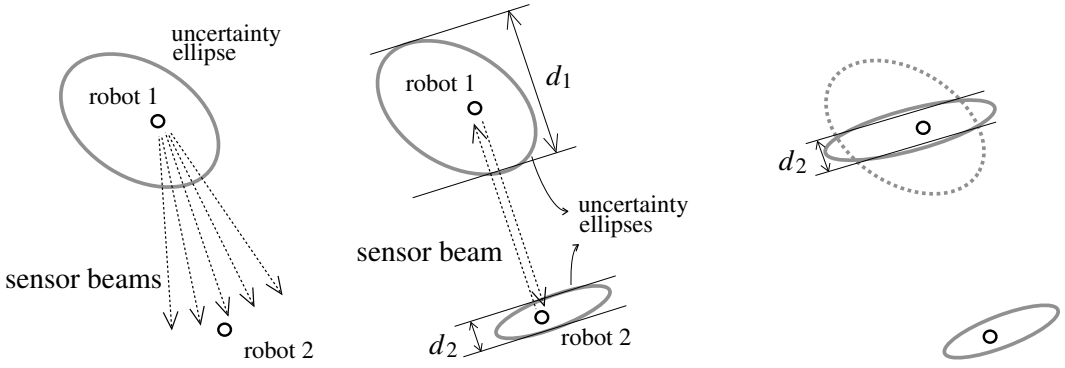


Fig. 61. Reducing ellipse after range sensing. (a) The robot 1 senses the robot 2. (b) Reducing ellipse: before reducing, and (c) the ellipse of robot 1 is reduced.

several atomic rounds, each dealing with one feature.

We assume that reducing the position uncertainty of a robot using one feature takes time t_r, and we assume that it is the same for all robots. Our interest is in efficiently localizing multiple robots, and the efficiency can be measured in several ways. We aim to reduce the total time needed to reduce the ellipses, which is the same as minimizing the number of required rounds. To achieve this, parallelism must be increased (we must allow more than one robot to localize simultaneously in each round) and the number of localizations for each robot must be decreased.

A simple approach to reduce the ellipses of all robots is to repeatedly localize a random robot using a random feature until no further improvement is obtained. This is not necessarily the worst, but has room for improvement. An intuitive approach for efficient localization might be, at each round of localization, to find a robot with the smallest uncertainty and localize its neighboring robot(s). Our approach is to use ideas similar to those used in job scheduling.

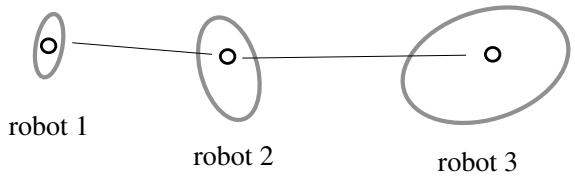
2. Problem Definition

Definition: The *feature–visibility graph* (FVG) is an undirected graph where vertices correspond to robots, and an edge connects two robots if they can sense each other. We store the uncertainty ellipse for the corresponding robot in the node.

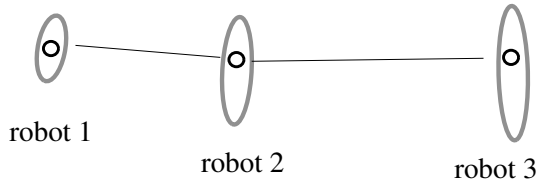
Definition: The *Uncertainty Reducing* problem (UR) is, given a feature–visibility graph and a positive integer l, compute a schedule so that all robots' position uncertainties are completely reduced in l rounds.

Here, we call a position uncertainty *completely reduced* if $d_1 = d_2$ for all robots adjacent to each other in FVG. A trivial lower bound on l is the number of robots that have neighboring robots with smaller uncertainty regions. In the ideal case, every robot completely reduces its uncertainty region by sensing only one feature. In practice, robots often need more than one round of localization to completely reduce their uncertainty.

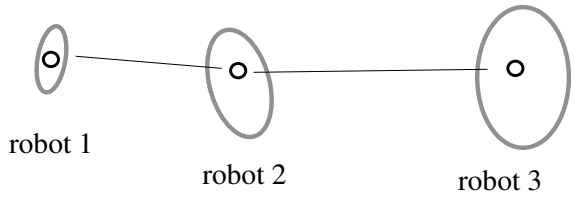
After localizing a robot using a feature, it is clear that an efficient solution of UR should avoid localizing the robot using the same feature again. This condition is illustrated by the three robots shown in Figure 62. If we localize in the order of increasing ellipse size, i.e., robot 2 and then robot 3, then all robots are localized in two rounds (Figure 62(b)). However, if robot 3 is localized first (Figure 62(c)), then robot 2 and 3 need to be localized again. From this we observe that if the robots can be arranged in a tree-like structure according to increasing ellipse size, then an upperbound on l is the height of the tree. Indeed, it is not hard to show that in the special case in which the FVG forms a tree, and if it further satisfies the monotonically increasing ellipse size, then the optimal l is equal to the height of the tree. In an optimal solution to UR, a robot is often localized more than once. Figure 63 shows an example where the robots are arranged in an alpha shape.



(a)



(b)



(c)

Fig. 62. Localizing three robots takes two steps. (a) Before localization, (b) localizing robot 2 and then 3, and (c) localizing robot 3 first (need to localize robot 2 and 3 again).

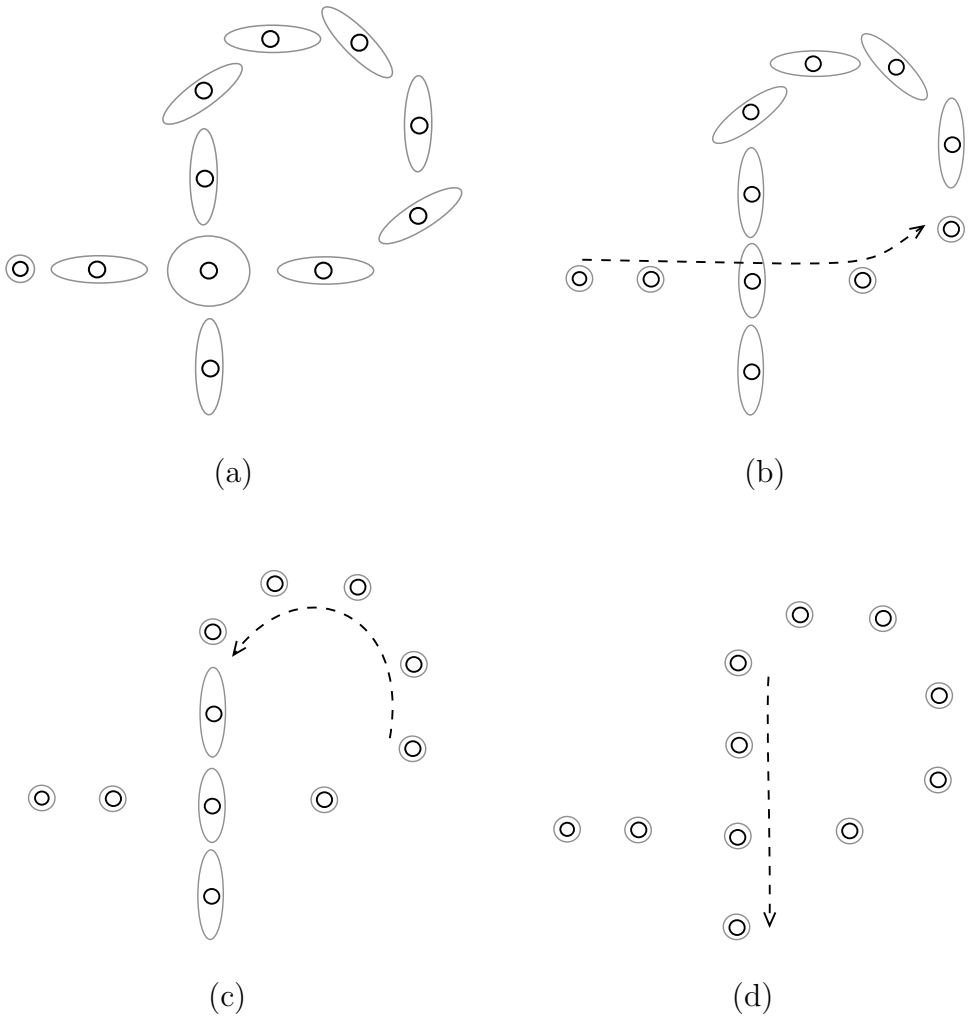


Fig. 63. An optimal solution to UR requires one robot to be localized twice (horizontally and then vertically).

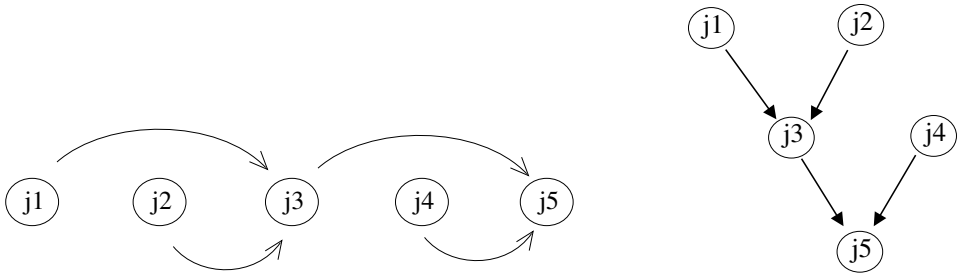


Fig. 64. Precedence–constrained job scheduling problem. (a) Five jobs with precedence constraints, and (b) DAG representation of scheduling problem.

3. Precedence–Constrained Job Scheduling

The precedence–constrained job scheduling problem is often studied in the areas of operating systems, compiler design, production/project management, etc. Given n jobs under m precedence constraints and p identical parallel processors, the *precedence–constrained job scheduling* problem is to assign jobs to processors while satisfying the constraints [9] (also called DAG scheduling). In a feasible schedule, each processor executes only one job at a given time, and the execution is non–preemptive. A simple case is illustrated in Figure 64 where five jobs with precedence constraints are represented by a directed acyclic graph with n nodes and m edges. Jobs $\{1,\ 2\}$ and $\{3,\ 4\}$ can be executed simultaneously if two processors are available.

The *makespan* of a schedule is the elapsed time between the time when the first task starts execution and when the last task finishes execution. Determining the optimal (minimal) makespan for a job scheduling problem is NP–hard for arbitrary p, but becomes feasible if the constraints are tree–structured or if $p = 2$ [9].

4. Proof that UR is NP–complete

UR is in NP. To verify a solution to UR, we can verify that each round is feasible and that the number of rounds in the schedule is at most l.

UR is NP–hard. We reduce from the NP–complete precedence–constrained scheduling problem. Given an instance of job scheduling represented by a directed graph $G = (V, E)$, we can transform it into an instance of UR as follows.

1. Each job in the job scheduling problem corresponds to a robot.

2. The size of the uncertainty regions for the robots increases according to the edge directions in G so that precedence constraints are mapped to the UR problem.

3. Because robots corresponding to starting jobs in the scheduling problem don't have any feature, additional features are needed. For this, sensor–visible obstacles or new robots with very small uncertainty regions are added in appropriate positions.

An example of the UR problem corresponding to the job scheduling problem shown in Figure 64(b) is shown in Figure 65. Robots 6, 7, and 8 have been added so that five robots corresponding to jobs 1 — 5 are all localized. Though the FVG is undirected, we added arrows in the figure so that the relative size of the uncertainty ellipses for each pair of robots can be easily seen.

The solution to UR provides an optimal localization sequence. It can be transformed into a job execution sequence (solution of job scheduling) by reversing the transformation explained above. That is, jobs can be created for the robots to be localized, and precedence constraints can be determined according to the relative size of uncertainty regions. This takes linear time in the number of nodes and edges in FVG. Thus, since job scheduling is known to be NP–complete, UR is also NP–complete.

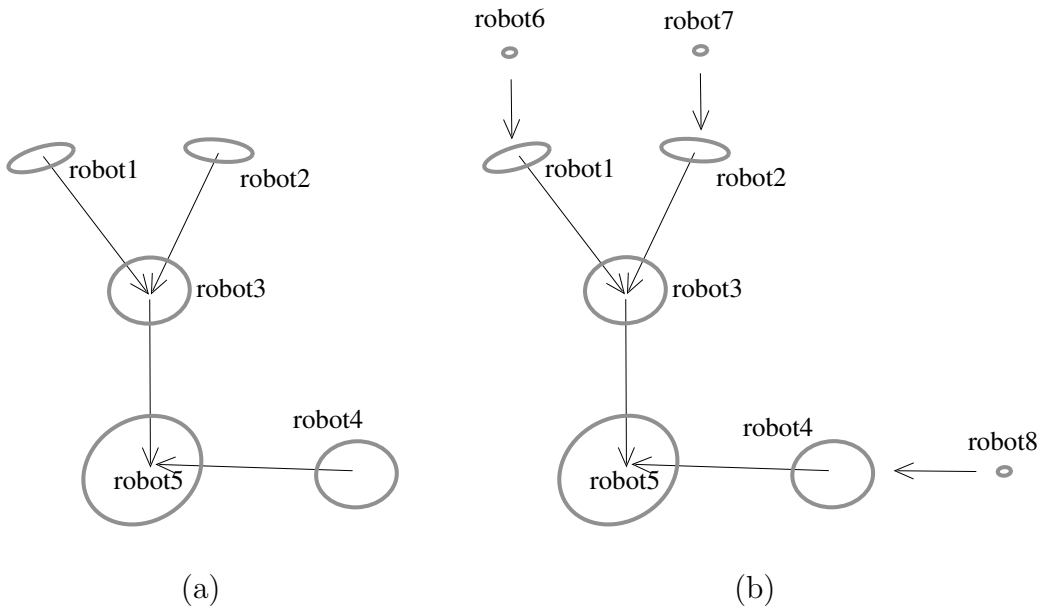


Fig. 65. Transforming into an instance of UR. (a) An instance of uncertainty reducing (UR) at an intermediate step, and (b) three robots are added according to rule 3 of transformation.

D. Approximate Solutions

Though it is believed to be impossible to compute optimal solutions for the two multi-robot problems in polynomial time, heuristics for graph coloring and job scheduling can be employed to obtain approximate solutions for the two multi–robot problems. Our goal is to reduce the number of total steps in RS and UR given the CTG and FVG, respectively, using reasonably simple algorithms.

1. Range Sensing

Many polynomial–time approximation algorithms for graph coloring have been proposed, e.g., Wigderson [78], Blum [11], Karger et al. [33], and Zwick et al. [87]. The problem has been applied to several other research areas. For example, the method for efficient channel assignment for mobile cellular communication systems [73]. For

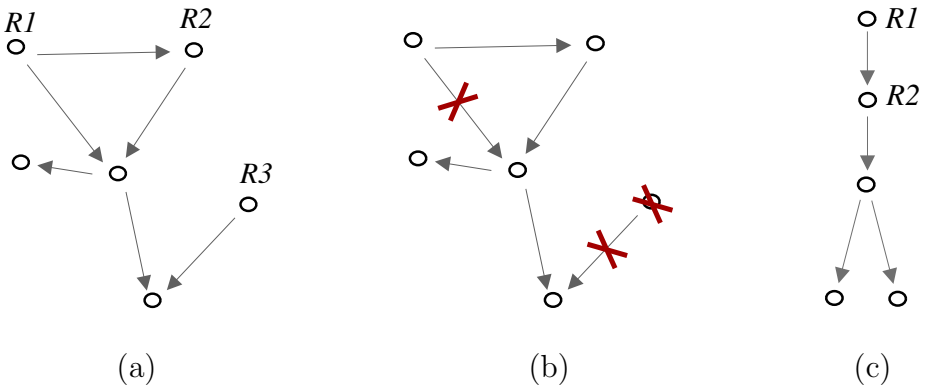


Fig. 66. Choosing the robot to localize using embedded tree. (a) Given a FVG, (b) removing two edges and a node R_3 that do not belong to the embedded tree, and (c) the nodes are rearranged to be seen like a tree.

range sensing, any existing approximation algorithm for graph coloring can be applied without further modification. We do not discuss the details here.

2. Uncertainty Reducing

Approximate solutions for job scheduling have been researched for a number of applications. For parallel database and scientific applications, [14] considered two constraints — the volume of the resource time product and the critical path.

For the UR problem, we propose two heuristic approaches. The first approach is based on *rooted embedded tree* which is defined to be any maximal tree structure that is a subset of the FVG. After a rooted embedded tree is found, the first robot to localize is the child vertex of the root in the tree; the root robot has no feature and its uncertainty cannot be reduced. An example is shown in Figure 66 where the robot to be localized first is R_2. We call this the *rooted tree* method.

The second approach is an extension to the first approach and useful when the problem size is large. In case there are several embedded trees with similar size, this method selects the robot with the smallest uncertainty region to localize first.

This is based on the concept of the *critical path* that is often used to calculate the lowerbound on total execution time. However, we do not compute the tree height to identify a critical path but estimate it using the robot's uncertainty region. There are two reasons for this. First, choosing the tallest embedded tree in the FVG does not guarantee the minimum total time. This is because critical path–based scheduling does not always yield the best performance when the task graph is dynamic and the FVG in UR is dynamic, i.e., subject to change during localization. Second, explicitly computing the height of embedded trees takes longer than comparing the size of uncertainty regions. We call this approach the *uncertainty–based* method. Simulation results in a more complex environment are presented in Chapter VI Section F.

CHAPTER VI

EXPERIMENTAL RESULTS

In the previous chapters, we have described our methods for path planning and localization. In this chapter, we examine the performance of our methods through simulation and test the entire navigation system using a real mobile robot in our office environment. Some simulations were performed in simple artificial environments to clearly evaluate particular aspects of our methods. In other cases, including the hardware experiments, we used office and hallway areas of our lab which are meant to represent typical indoor environments for personal mobile robot applications.

This chapter is organized as follows. First, we describe implementation issues and assumptions related to our experimental setup. Then we present experiments that were designed to enable us to study the following issues.

1. Localization in Partially Known Environments: How robust and fast is the localizer?

2. Path Planning: How safe and efficient are the paths provided by our path planner?

3. Global Navigation System: How do the localizer and the path planner work together?

4. Practicality: How does our navigator work with a real mobile robot?

5. Multiple Robots: How effective are our methods for multi–robot localization?

Simulation results for multi–robot localization are presented in a separate section since the algorithm is different from the single robot case.

A. Implementation Details

1. Assumptions

We assume that a model of the environment is provided *a priori*, that the portion of the workspace relevant for navigation can be considered to be planar, and that obstacles can be modeled by simple polygons. Walls, doors, and big stationary objects such as tables and book shelves are recorded in the map while small uncertain objects such as chairs or trash cans may not be charted. The height of the robot is ignored and any obstacle that lies above the robot is not considered.

2. Code and System

We implemented our navigator in C++ using the LEDA [57] library for geometric primitives. Simulation and hardware experiments were done using Pentium 4 PCs running Linux and the SAPHIRA [46] package was used for hardware communication with an AmigoBot [54].

The path planner employs several strategies for building probabilistic roadmaps including basic PRM [36], obstacle–based PRM [4, 12, 28, 79] and medial axis PRM [53, 79, 80] To generate roadmap edges, each node was attempted to the k closest nodes in the roadmap; generally we used $k = 7$. For local planning, we used only straight–line planners.

3. Robot

The AmigoBot has eight sonar sensors — four in the front, two in the back, and one each on the left and right. We have used only the left and the right side sensors for localization. The amount of odometry error depends on the speed of moving, and we limited its speed to be similar to a human's walking speed. This is slower than the

robot's maximum speed but this allowed us to safely operate the robot with some people moving around.

To represent the position uncertainty of the robot, ellipses were used where the major and minor axis represent lateral and longitudinal errors, respectively. Our method for compounding ellipses after translation and rotation has been explained in Chapter II.

When performing simulations, we tried a variety of robot parameters including varying sensor minimum/maximum ranges and the accuracy of odometry measurement. Small sensor maximum range and large odometry error, which characterize inexpensive systems, were used to test the robustness of our methods. By incrementally relaxing the limitations, i.e., by increasing the sensor maximum range and the odometry accuracy, we observed their effects on the efficiency of navigation.

4. Environments

Environments were represented with polygons, with one describing the boundary of the environment, and additional polygon(s) used to represent other obstacles. For geometric computations such as intersecting polygons, we have used the LEDA library [57]. Since LEDA does not support the representation of curves, curves were approximated with a sequence of line segments.

Two real environments have been measured to create map data (see Figure 67). One is the lab environment. It has a total area of 24.2 square meters (260 square feet) and contains several desks and room doors. The other is the hallway environment outside our lab. Obstacles inside the environment often consists of small tables, chairs, and other small objects that are stationary and whose positions are known.

We tried adding varying number of obstacles to the lab environment to see its effect on localization. Depending on the number of obstacles present, such envi-

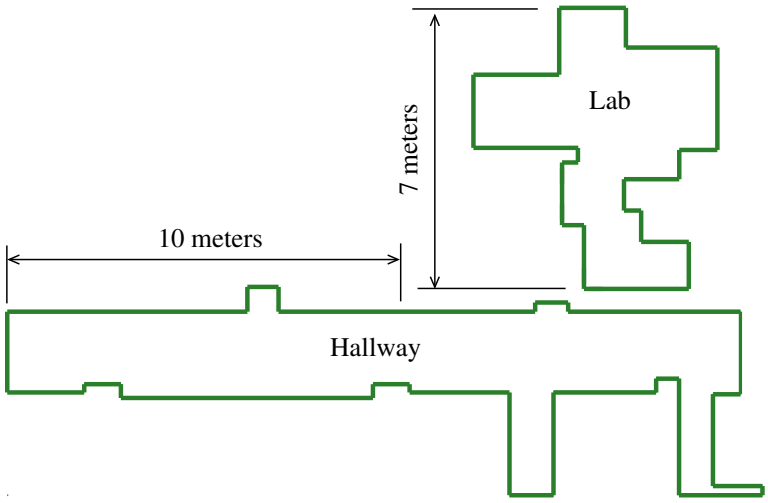


Fig. 67. Environments for simulation and hardware experiments.

ronments can be grouped to free, mostly free, moderately cluttered, and cluttered. Cluttered environments might not be often used in real applications but they are useful in studying visibility sector properties.

B. Localization in Partially Known Environments

This section describes localization simulation results focusing on exploring the robustness of our methods in the presence of imperfect map knowledge and limited feature visibility. First, we will analyze the effects of varying the maximum range of the sensors. We consider how this affects the sector properties and also our general ability to perform global localization. Second, we will show how we can augment the localization algorithm so that it can handle additional scenarios.

Limited storage and computing power are other design constraints that are important for inexpensive mobile robot systems. We analyze the storage and computation requirements of our methods for different subdivision methods and for varying sensor limitations.

1. Sector Construction Time

Depending on how features will be defined and identified, there are two ways to compute sectors. The first method was explained in Section 4 and uses a "synthetic" scan around a point in each sector. This is often used for perfect visibility cases because some of the features can be identified only by synthetic scanning (e.g., `Dc` and `cD`).

The simulation results presented in Table I show that the feature identification step is responsible for essentially the entire sector construction time (3.3 seconds). Visibility sectors in this case are convex simple polygons and very efficient geometric operations are permitted. When generating synthetic scans, the angular step of sensor rotation must be small enough to detect all features in the environment. The minimum necessary angular step depends on the maximum distance to a feature, the minimum feature size (e.g., wall length), etc.

We have suggested a way to extend the synthetic scan approach to compute features while accounting for limited visibility. Table I shows results for our merging based method. It is seen that the reduced visibility case requires additional time over the perfect visibility case. During the merge process, sectors need to be represented by generalized polygons because they may have holes inside. Since the size and shape of visibility sectors are variable, our approaches have advantage over grid-based methods in efficiently subdividing the environment.

Another method of computing sectors avoids synthetic scans and instead adds a feature (a character) to the label at each step of sector subdivision. This is possible only if each primitive sector is related to a unique feature and no additional features are used. For the same lab environment, the time for identifying primitive sectors is significantly longer than the perfect sensor case (5.7 sec vs. 0.1 sec) because the prim-

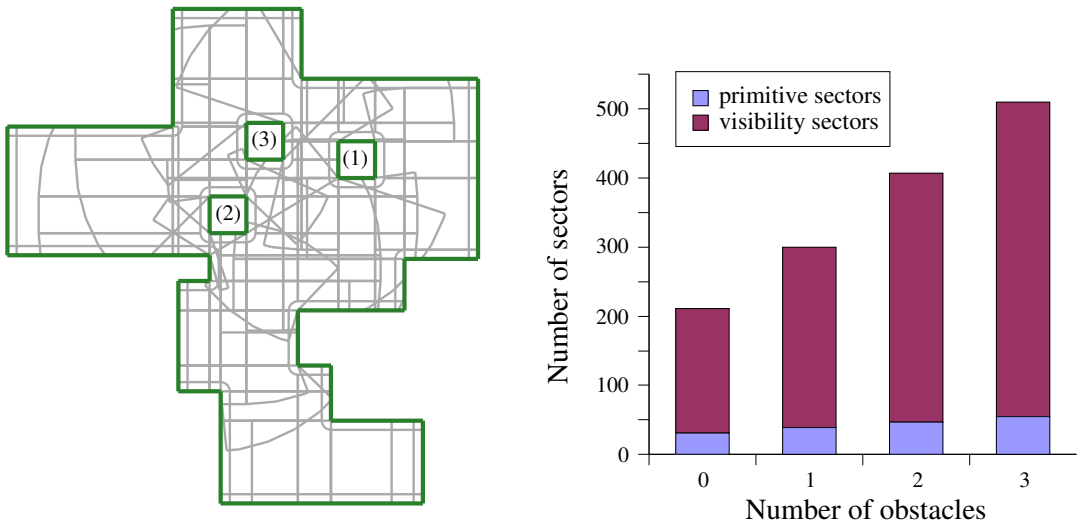
TABLE I
SECTOR CONSTRUCTION TIME

	perfect visibility	limited visibility	
	`m,M,D,c`	`M` only	`m`
number of sectors	254	13	122
primitive sector	0	0	2.2 sec
sector boundary	0.1 sec	0.1 sec	3.5 sec
computing features	3.3 sec	3.3 sec	0
merging	N/A	2.1 sec	N/A
total time	3.4 sec	5.5 sec	5.7 sec

Sensor maximum range was set to 50 inches for the limited visibility case. The lab environment was used.

itive sectors are more complex, having concave portions and curved edges. Since they often cause numerical instability, our implementation includes additional routines to deal with degeneracies.

The sector construction time increases with the sensor range and the number of obstacles. Figure 68 shows that when an obstacle is added, the number of sectors is increased significantly. Though only a small number of primitive sectors are introduced with each additional obstacle (eight in this case), they intersect a large number of existing sectors. Similarly, the sector construction time increases when additional obstacles are added. Figure 69 shows that in the lab environment, additional obstacles or a longer sensor range can double the total construction time.



(a) (b)

Fig. 68. Simulation in the lab environment. (a) Visibility sectors with three obstacles and sensor maximum range of 95 inches, and (b) Number of sectors increases with number of obstacles (each obstacle is added according to the sequence numbered in Figure 68(a)).

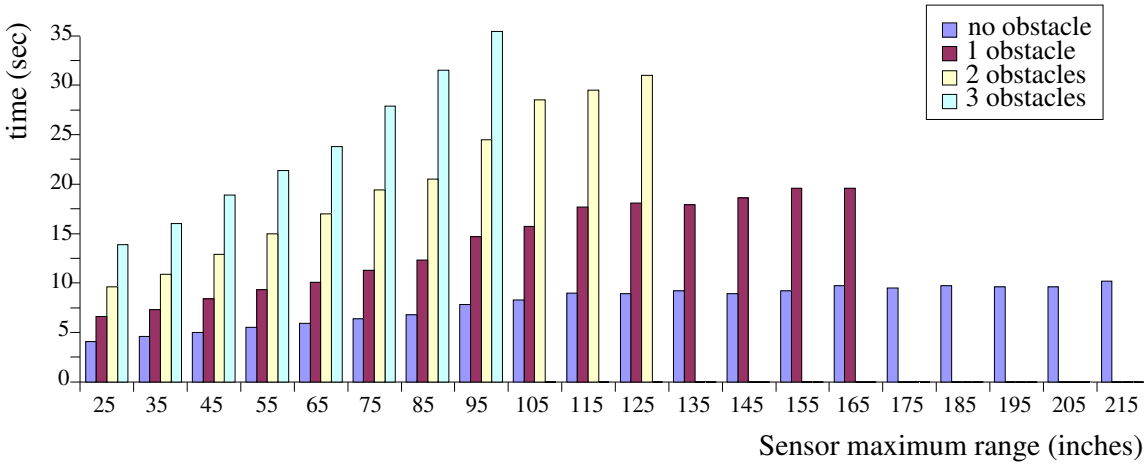


Fig. 69. Time of constructing sectors with varying sensor range and obstacles.

2. Varying Sensor Maximum Range

Among the many sensor properties, perhaps the maximum range affects the attributes of the visibility sectors the most. In Figure 70, among five different sensor maximum ranges (50, 100, 150, 200, 250), the 200 inch case shows the largest total number of sectors. The largest visibility number (the number of visible features) is ten when the maximum range is 200 or 250. Figure 70(c) shows that as the sensor maximum range is increased, the area with high feature visibility becomes larger. For example, if the sensor range is increased from 150 to 200 inches, then the area with visibility number ten is increased almost ten times. Sector boundaries with different sensor ranges are illustrated in Figure 71.

Figure 72 shows the regions where at least (a) two or (b) four features are visible with a sensor maximum range of 100 inches. With two or more features, the robot can precisely localize if all features are successfully identified. This may not be robust enough if some of the features are not scannable. In the marked area in Figure 72(b), the robot can localize even if it fails to recognize two features in the map data. Such information can be utilized by our path planner so that it will select paths from the roadmap that are robust for the localizer.

Though a larger sensor range means increased robustness in general, it demands more storage. As the sensor range becomes larger, Figure 70(b) shows that the majority of the sectors have larger visibility numbers. When the maximum range is 200 inches, the average number of features is approximately 5.5. However, depending on mission requirements, shorter–ranged sensors may be preferred to save storage (or we could elect to record only close features). If the sensor range is reduced to 100 inches, then the number of sectors is decreased by 15% and the sectors have approximately four features on average. Such information about an environment can

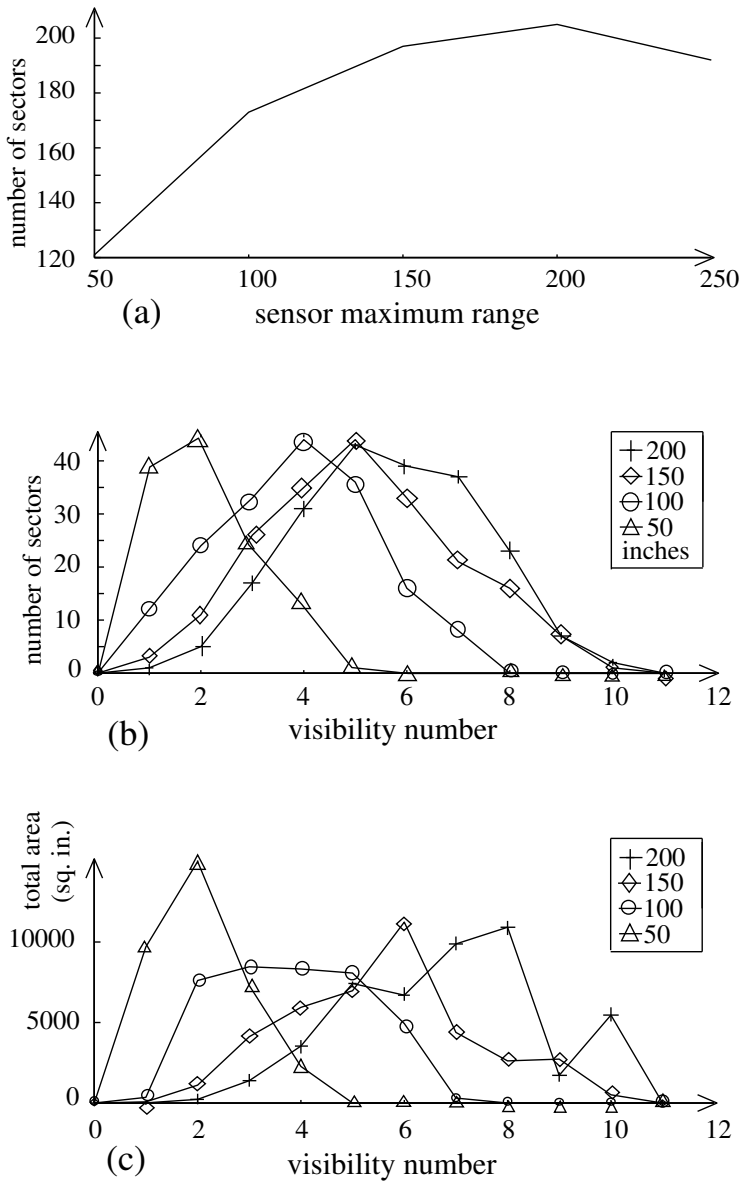


Fig. 70. Properties of visibility sectors with different sensor maximum ranges.

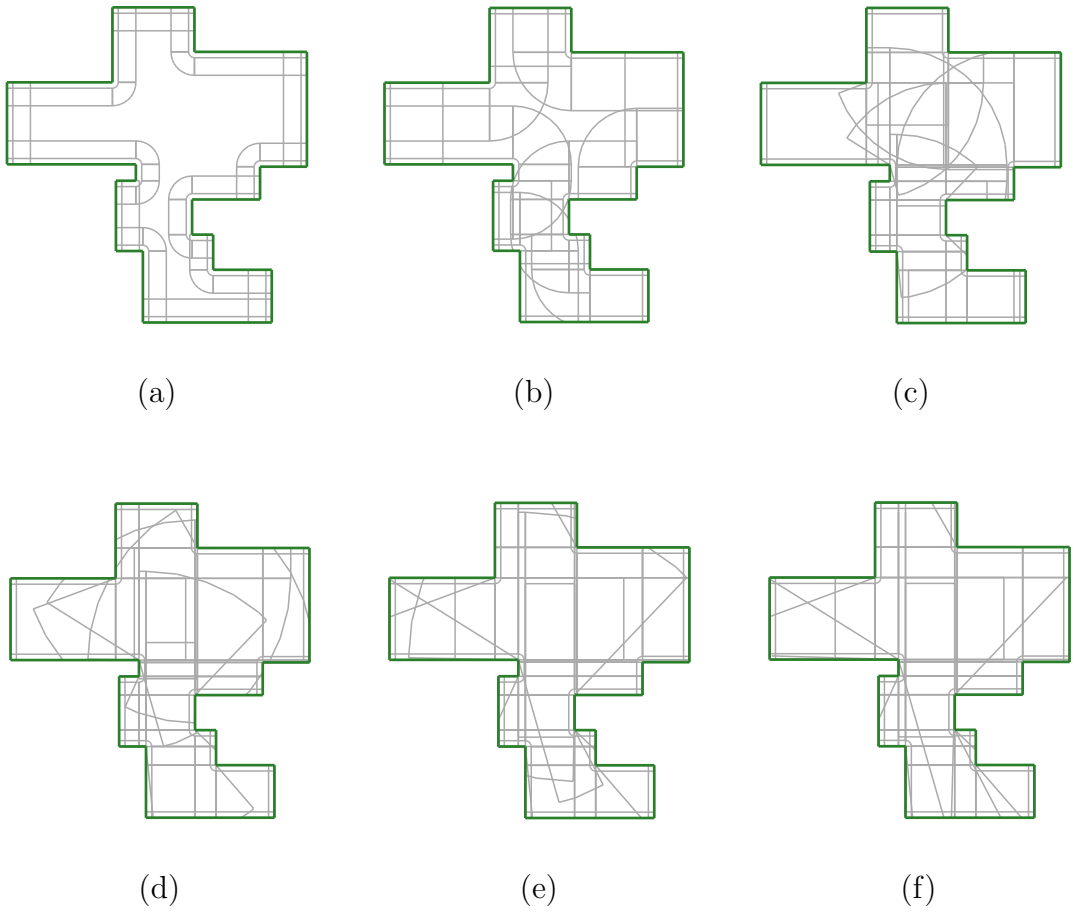
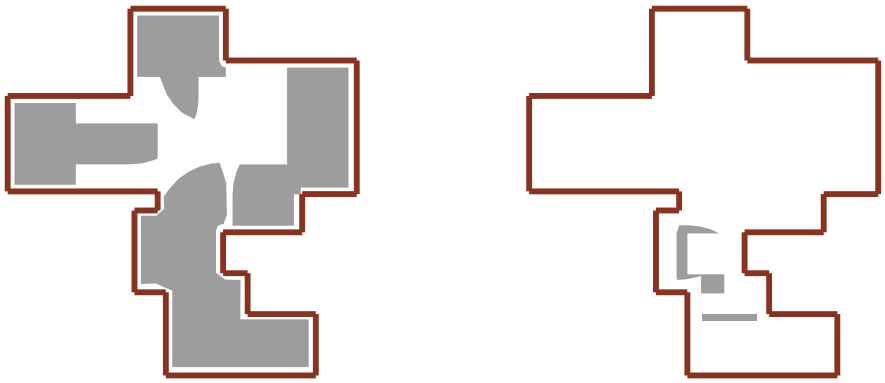


Fig. 71. Visibility sectors with sensor minimum range of 5 inches and maximum ranges of (a) 20 inches, (b) 50 inches, (c) 100 inches, (d) 150 inches, (e) 200 inches, and (f) 300 inches.



(a) (b)

Fig. 72. Sectors with (a) two or more local minimum features, and (b) four or more features for sensor maximum range of 100 inches.

be used to help decide how long the sensor's maximum range should be for that environment.

Next, we show simulation results that illustrates that a larger sensor maximum range can assist efficient navigation. We restrict the robot to stay close to the wall so that at least one feature is visible while moving. In the hallway environment, Figure 73 shows that at least five localizations are necessary if the sensor range is 15 inches. The second and the third cases use maximum sensor ranges of 25 and 35 inches, respectively, and they require fewer localizations. Also, we show that a large sensor maximum range and accurate odometry measurements contribute to efficient navigation in a similar way. Figures 74(a) and 74(b) show almost the same subgoal positions with different combinations of parameters. However, this does not always hold. For example, subgoals in Figure 74(c) and 74(d) are far apart though the parameters do not vary significantly. This is because with a sensor maximum range of 45 inches, the upper and lower sectors overlap and the robot can move along the medial axis of the environment.

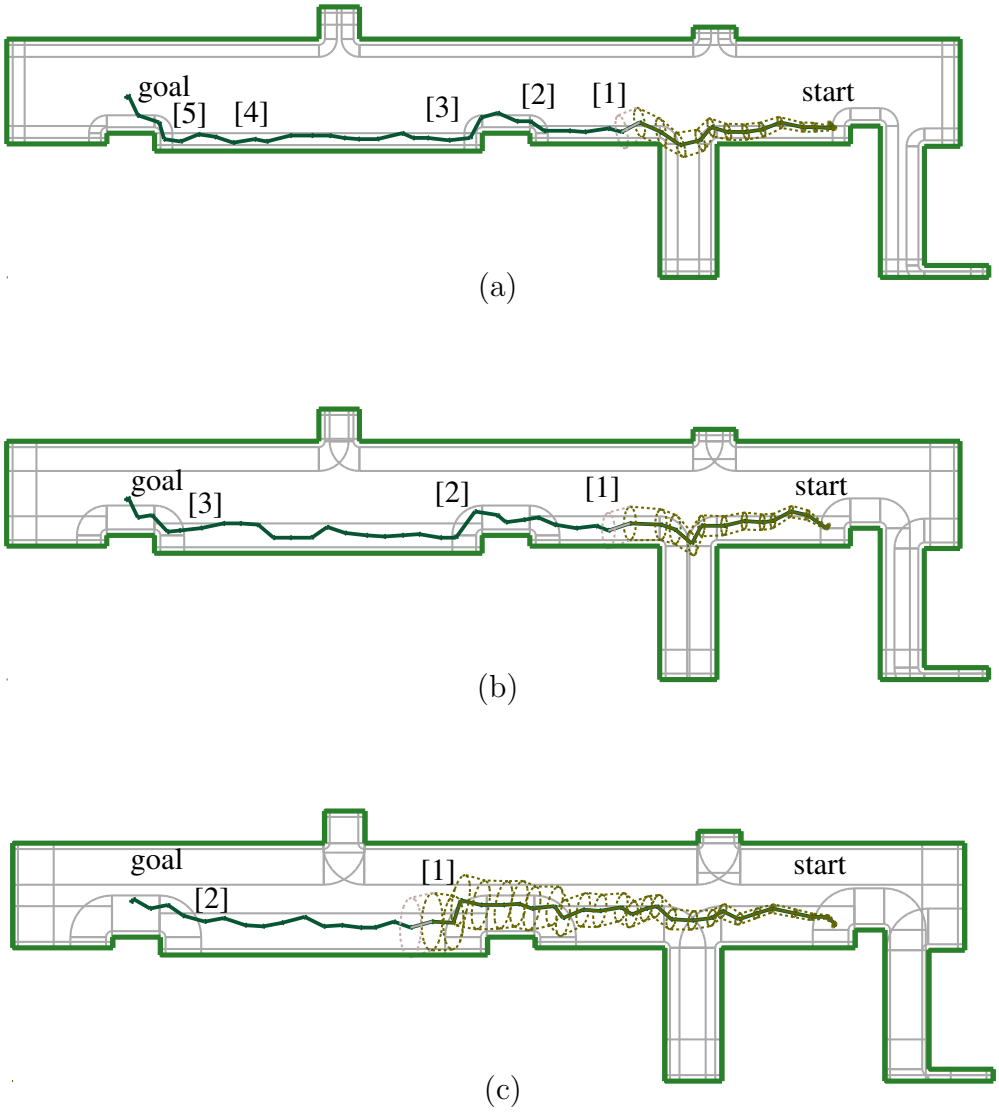


Fig. 73. Hallway environment with sensor maximum range (a) 15 inches, (b) 25 inches, and (c) 35 inches. Localization position are labeled [1] – [5].

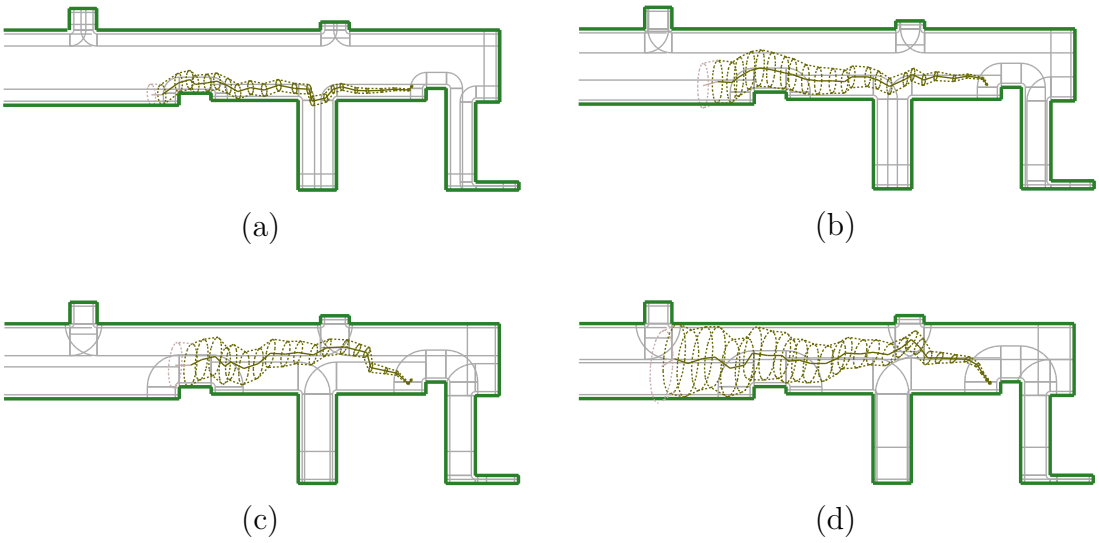


Fig. 74. Hallway environment with sensor maximum range and lateral deviation: (a) 20 inches and 2% of travel distance, (b) 30 inches and 4%, (c) 40 inches and 5%, and (d) 45 inches and 6%.

3. Varying Number of Obstacles

To test our methods in the environments cluttered with small obstacles, up to fifteen box-shaped obstacles were added to the lab environment. Figure 75 shows the order of adding each obstacle. The first three obstacles are same as those in Figure 68. For restricted visibility, sensor maximum range was set to 50 inches.

First, we investigate visibility sectors with local maximum features (perfect visibility and merging sectors cases). Figure 76 shows the time of construction and number of sectors in the lab environment. In Figure 76(a), the time of merging sectors explodes when a few obstacles are added. This is because our implementation does not take advantage of a sector adjacency list. When computation is optimized, our analysis results in Chapter IV Section B.6 expected that the time will be bounded by $O(nr \log(n^2 r))$. With five obstacles present, the number of sectors is increased to 16 times the free environment case with perfect visibility (Figure 76(b)) and 2.5 times

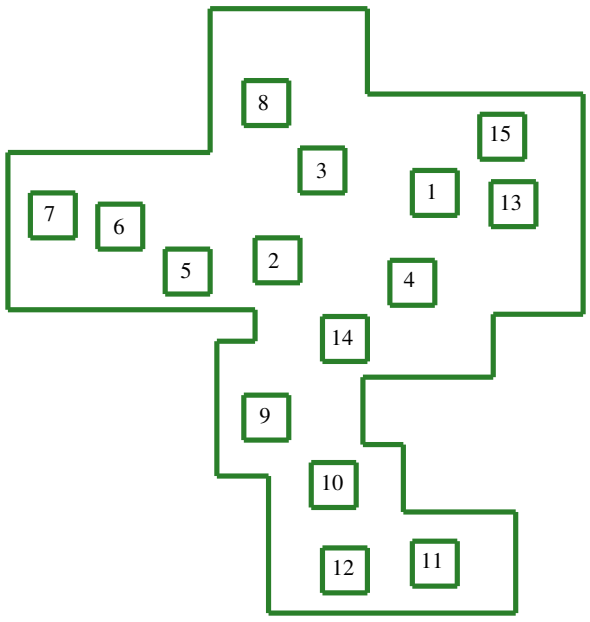


Fig. 75. Lab environment cluttered with fifteen obstacles.

the free environment case (with restricted visibility (Figure 76(c)). This shows that using sensors with perfect visibility might render the computation infeasible when the environment is complex.

Next, we show the results for local minimum point features. In this case, synthetic scanning is not necessary and the sector construction time depends on the process of environment subdivision. Figure 77 shows the number of sectors and time of construction with 0 – 15 obstacles present. Though cluttered environments take a long preprocessing time, they prove to provide good feature visibility. With four obstacles, 72.3% of randomly sampled nodes have feature visibility of two or higher. The ratio is increased to 99.3% when fifteen obstacles are present.

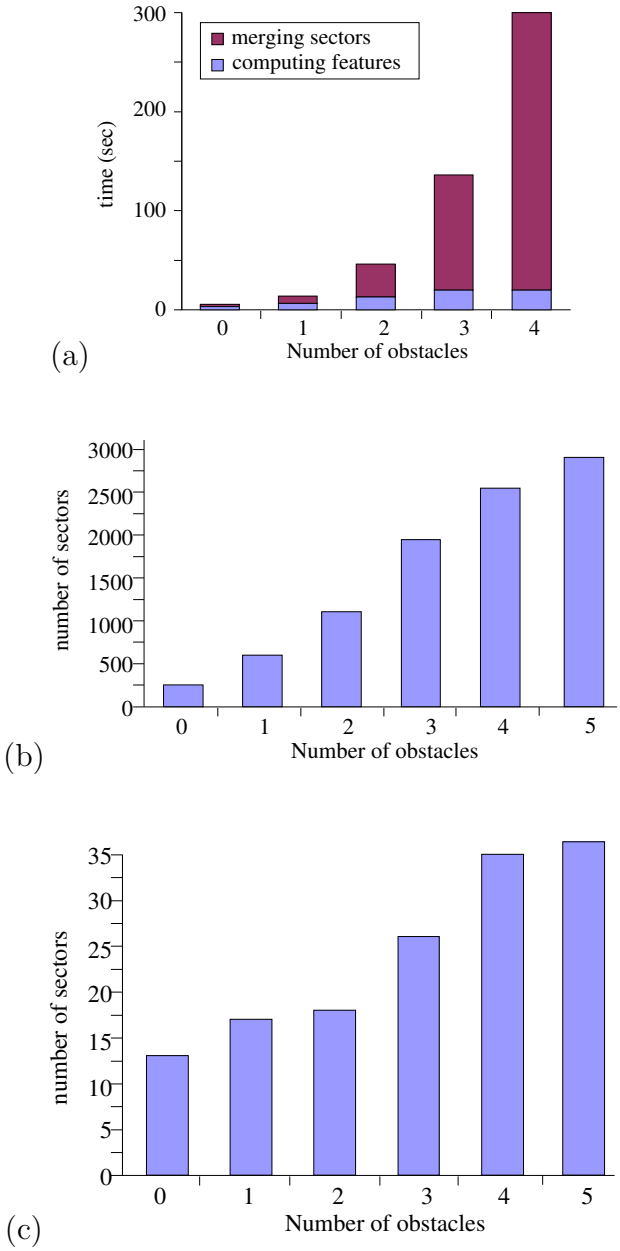
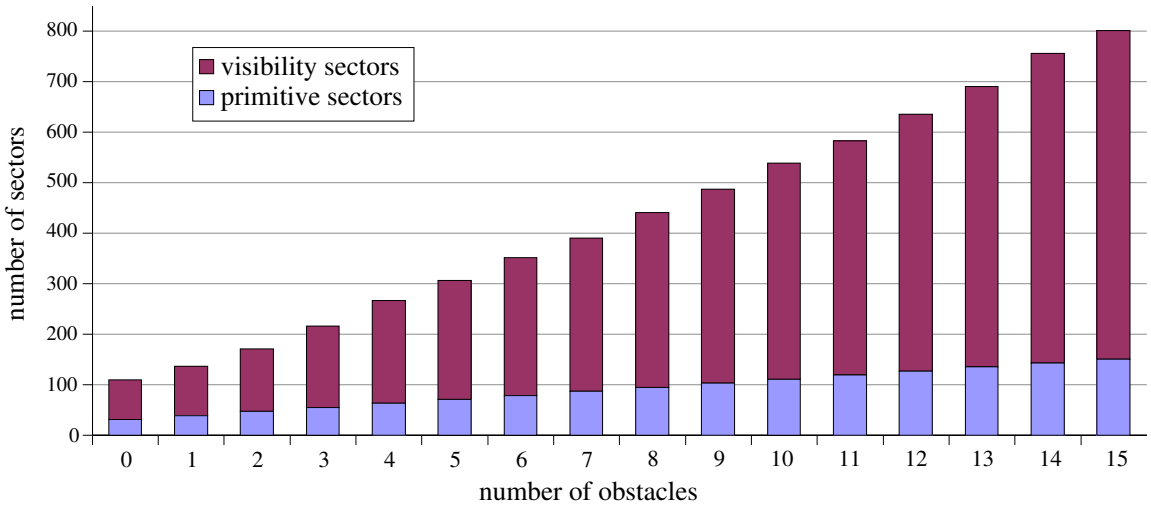
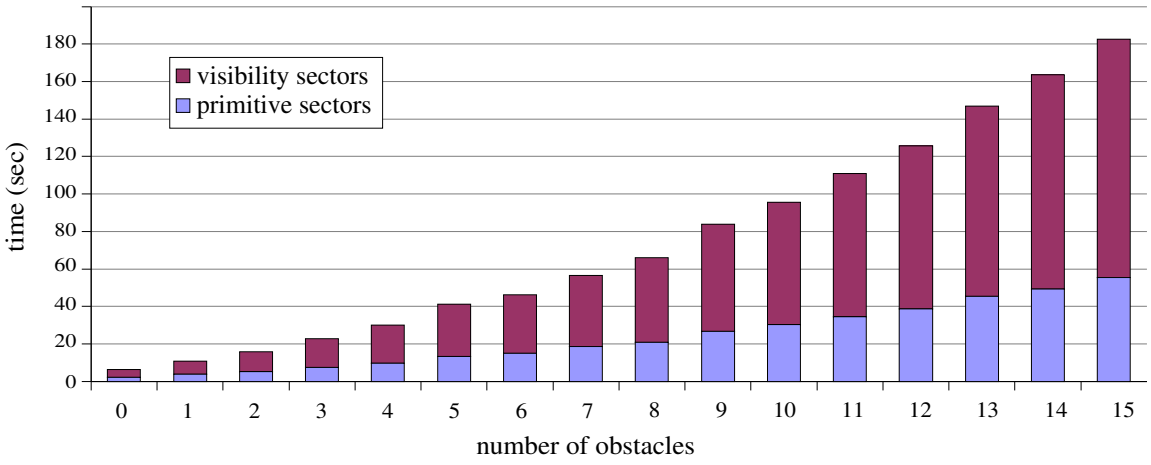


Fig. 76. Merging sectors with varying number of obstacles. (a) Time of computing visibility sectors for perfect visibility and restricted visibility (needs merging). (b) Number of visibility sectors for perfect visibility. (b) Number of visibility sectors after merging.



(a)



(b)

Fig. 77. Sector properties in cluttered environment. (a) Number of primitive and visibility sectors, and (b) time for constructing sectors.

4. Ambiguity in Global Localization

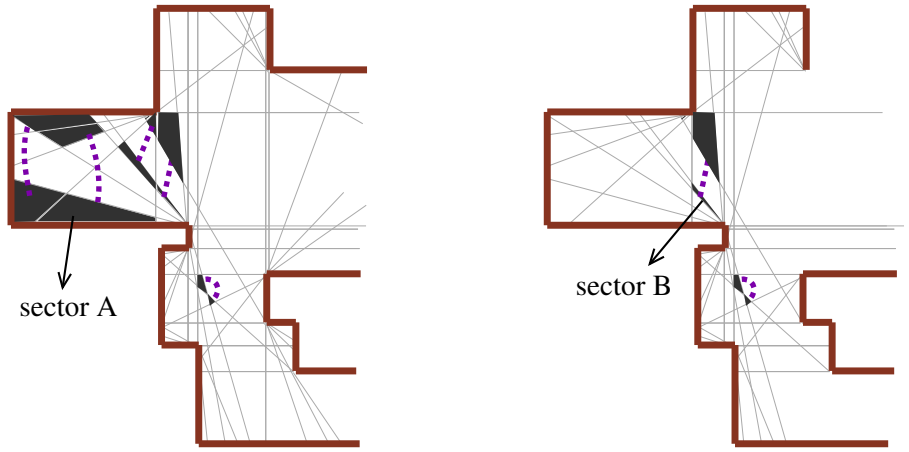
Our strategy for overcoming the issue of ambiguous regions is to avoid selecting paths which might place the robot in two sectors with indistinguishable labels. However, in global localization, the robot should be able to localize virtually anywhere in the environment. In this case, to prevent localization failure we need to minimize the areas where ambiguities could occur.

With the four features (`m, M, D, c`) defined in Chapter IV, there are five pairs of ambiguous sectors in the lab environment as shown in Figure 78(a). The total area of such sectors is 8% of the free space. Global localization could fail in these regions. To improve the differentiability of sector labels, we can add additional feature (feature `l` in this case) as explained in Chapter IV Section C. The ambiguous region is reduced to two pairs of sectors and their area is 1.5% of the environment. If one small obstacle shown in Figure 68(a) is added, then the ambiguity is removed without using feature `l`. As mentioned earlier, the main role of visibility sectors is not to create robustness but to help to quickly assess the effectiveness of a feature set.

In practice, such small ambiguous regions are of little importance since robustness depends on many other factors such as the resolution and noise of the sensors, the size of the robot, unknown obstacles, the accuracy of the map, etc.

5. Reducing Uncertainty When No Features Are Scanned

This section describes techniques to increase the performance of our localization methods without modifying the sector data structure. An example shown here is that the robot's position uncertainty can be reduced even when no feature is identified from the scan data. This is possible if a part, but not all, of the uncertainty region intersects visibility sectors from which features should be visible. Figure 79(a) shows that



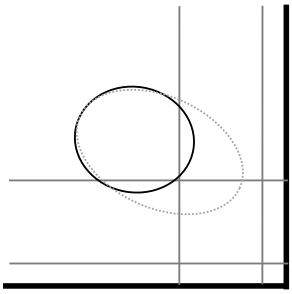
(a) (b)

Fig. 78. Ambiguous regions (same cyclic scan labels) for perfect visibility (a) with four features and (b) with five features (pairs of sectors with indistinguishable cyclic labels are linked with dotted lines).

an uncertainty ellipse (dotted line) is reduced to cover the featureless region (solid line). The pseudo code in Figure 79(b) was used to compute the reduced ellipse. This process can be integrated into the uncertainty reducing algorithm shown in Figure 56, Chapter IV.

C. Path Planning

We now examine the performance of our path planner. First, we show how to select feasible paths given the robot's physical constraints. We first address kinematic and dynamic constraints for nonholonomic robots and then discuss how the methods can be extended to compute efficient paths in general. Second, we consider the issue of safe navigation. We will show by simulation results that passive (increasing path clearance) and active (increasing feature visibility) measures can be used. For practical applications, we will also consider path clearance and length together. Finally,



(a)

LOCALIZATION($unc\_reg, scans$)
1. S ← sectors intersecting with unc_reg
2. F_{scan} ← features in $scans$
3. if $F_{scan} = \emptyset$ then
4. for each $s \in S$
5. $unc\_reg \leftarrow unc\_reg \setminus s$
6. end for
7. end if
8. return unc_reg

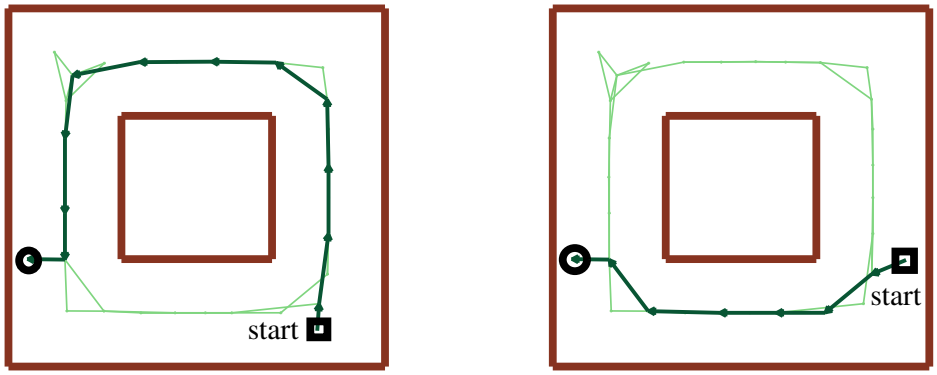
(b)

Fig. 79. Reducing position uncertainty when no feature is scanned. (a) The dotted gray ellipse shows the uncertainty ellipse before reduction and the black ellipse shows the uncertainty ellipse after reduction. (b) Pseudo code for reducing the ellipse.

a flexible final condition will be formulated and simulated to illustrate how this can make it easier for the robot to reach the goal.

1. Kinematic Constraints

Nonholonomic robots, such as car–like robots, have kinematic constraints that prevent them from turning with an arbitrary radius. Feasible paths for such robots are computed by avoiding selecting edges that violate such constraints. Our approach is to use an infinite edge weight for such unrealizable turns as explained in Figure 19(b) in page 42, part B of f_2. In Figure 80, different start positions were tried to obtain different valid paths which respected the robot's turning angle limitation. So, our path planning methods can be extended to the applications where kinematic constraints often exist.



(a) (b)

Fig. 80. Kinematic constraints — feasible path for nonholonomic robots. Different start positions in (a) and (b) result in different paths.

2. Minimizing Turning Effort

For holonomic and nonholonomic robots, reducing turning angles makes it easier to follow paths. In this section, two methods for minimizing turning effort are simulated. First, edge costs are computed based on the turning angle (only part A of Figure 19(b)) and they are accumulated using Equation 3.15. The result is shown in Figure 81(b). Because the edge cost is accumulated, the path is smooth and its length is reasonably short. Second, the path's maximum turning angle is minimized using a technique similar to maximizing path clearance. The result is shown in Figure 81(c) which is the smoothest path, in theory. Note that the resulting paths can be arbitrarily long because there is no control over the path length. Our last method is to combine path length (Figure 81(a)) and maximum turning angle (Figure 81(c)) using the cost function in Equation 6.1.

$$cost(e_i) = \begin{cases} cost_{dist} \cdot length(e_i) + cost_{tr} \cdot f_2(\text{turning radius of } e_{i-1} \text{ and } e_i) \\ \text{or} \\ cost_{dist} \cdot length(e_i) + cost_{ta} \cdot (\text{turning angle of } e_{i-1} \text{ and } e_i) \end{cases} \tag{6.1}$$

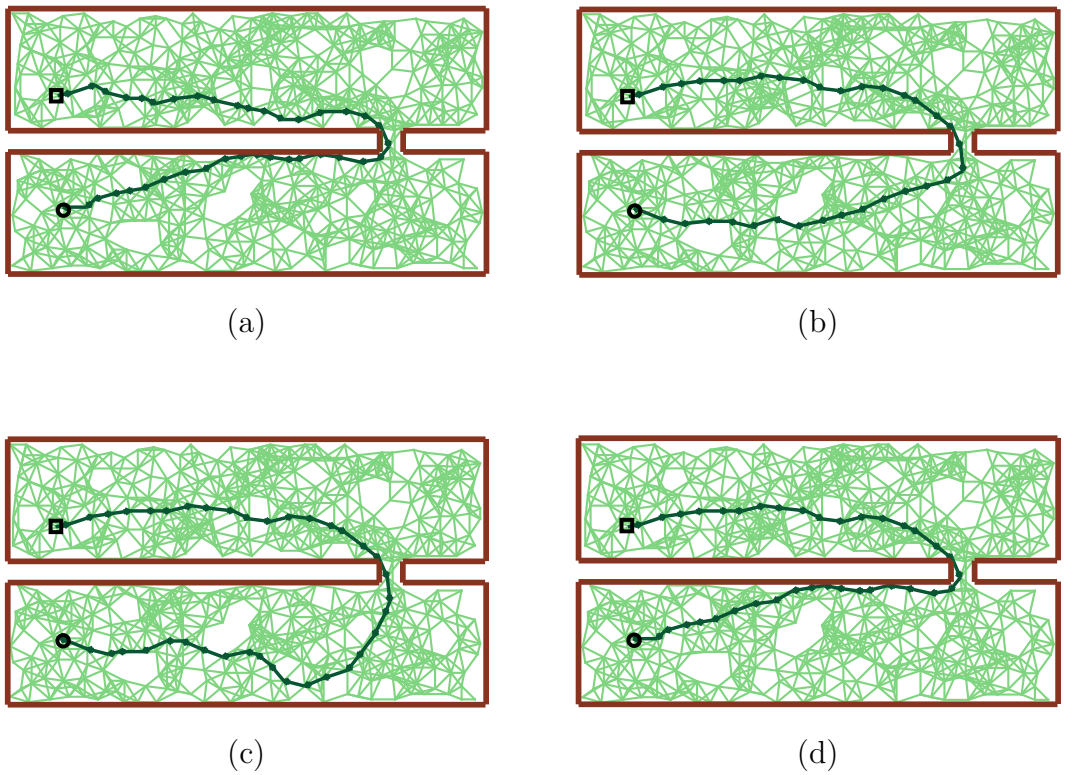


Fig. 81. Kinematic constraints — smoothing paths. (a) A shortest–distance path, (b) path selected by minimizing accumulated turning angle, (c) path selected by minimizing maximum turning angle, and (d) path selected using a combination of turning angle and distance.

For the simulation result shown in Figure 81(d), the values $cost_{dist} = 1$ and $cost_{ta} = 10$ were used. This shows an example that balances the competing goals of minimizing path length and maximizing turning radius.

3. Dynamic Constraints

Dynamic constraints become important as speed increases. To show how our path planner deals with increased speed, we used a mobile platform with a three–link arm as illustrated in Figure 82. The wall in the middle has a passage, and each node's position in the roadmap (Figure 82(a)) indicates the mobile platform's center of mass.

The start configuration is shown in Figure 82(b), and the final condition is that the end effector of the robotic arm should touch the wall opposite to the start position. Also, the mobile platform must come to a stop at the final position. Note that such a final condition should be formulated using a set of goal configurations (a goal set) instead of a unique goal configuration.

The optimization value is to minimize the time required from the start to the goal. We use bang–bang control logic (move at full speed until the end effector touches the wall, and apply the brake as hard as possible) to compute the minimum time of each path using Dijkstra's algorithm. Depending on the weight (including payload) of the robot, the final position varies because the stopping distance depends on the weight. Three different simulations show three different final configurations depending on the mass of the robot.

4. Path Clearance and Length

Path clearance is defined to be the minimum clearance of all edges in the path from the start to the goal, and is one way of representing the safety of the path. Three passages exist in the environment shown in Figure 83(a), where the start and the goal are shown in Figure 83(c). Paths going through corridor A or C are obtained by maximizing the minimum clearance or minimizing path length, respectively. Figure 83(c) shows the path going through corridor B; this is the result of using Equation 6.2 which combines the two conditions depicted in Equations 3.12 and 3.16 using $cost_{dist} = 0.03$ and $cost_{clr} = 0.97$.

$$cost(e_i) = cost_{dist}\ length(e_i) + \begin{cases} cost_{clr}\ (cl_{min} - cl(e_i)) & \text{if } cl(e_i) < cl_{min} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

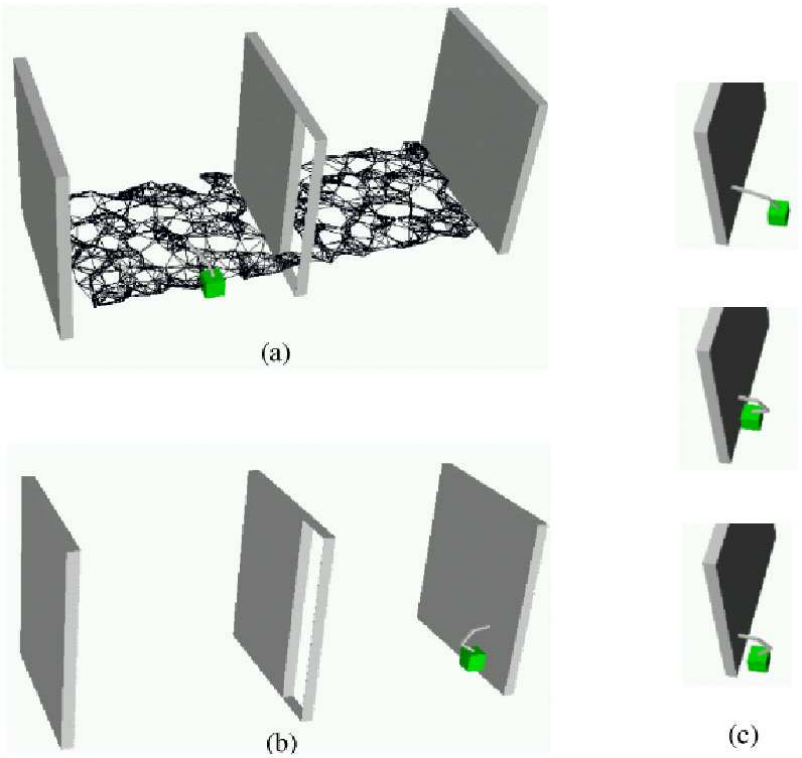
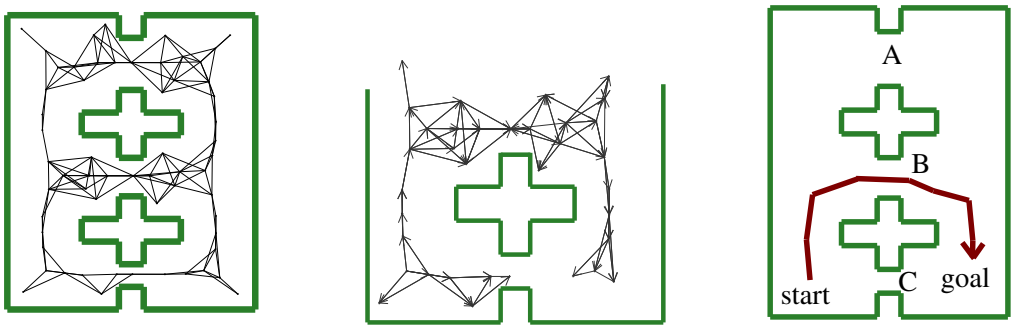


Fig. 82. A mobile platform with an robotic arm. (a) roadmap, (b) start position, and (c) three different configurations in the goal set.



(a) (b) (c)

Fig. 83. Maximizing minimum path clearance. (a) Roadmap, (b) visited edges in the middle of the search, and (c) the shortest path computed using the two optimization values are shown.

The search tree edges of Dijkstra' algorithm are illustrated in Figure 83(b) by arrows representing the direction of the search from the start node.

If the turning radius is considered together with the clearance and distance values, then the resulting path is short, efficient, and safe. For this, we add $cost_{tr}$ to the formula computing $cost(e_i)$. Simulation results are shown in Table II and Figure 83(c). The fifth and sixth rows show that different combinations of weight values can result in similar paths.

5. Goal Sets

This section shows the benefit of using goal sets instead of a unique goal configuration. A simple way of connecting the start and the goal configurations to the roadmap is to try to connect them both to nearby nodes in the roadmap. This sometimes causes difficulty in obtaining quality paths. For example, the path shown in Figure 84(a) connects the goal and the roadmap so that there is little room between the path and the wall. Unless the connection method is changed, the path clearance cannot be

TABLE II
THREE DIFFERENT PATHS

Corridor	$cost_{dist}$	$cost_{clr}$	$cost_{tr}$
A	0	1	0
B	0.03	0.97	0
C	1	0	0
B	0	0	1
B	0.08	0.84	0.08
B	0.03	0.32	0.65

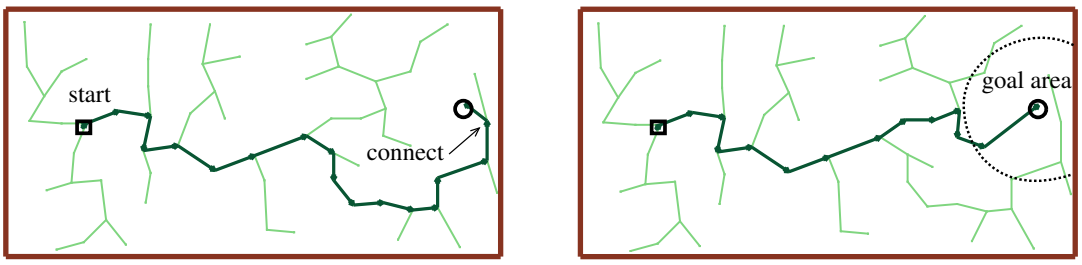
These weights were used in the environment shown in Figure 83(c).

increased any further. So, we have used the final condition $\mathbf{h}()$ described by Equation 3.11 which computes the distance between the current search node and nearby nodes. Figure 84(b) shows the improved path using the goal area marked by a circle.

There are other cases where the goal sets can be used to obtain good paths. Examples include the disassembly problem where the goal is to move the various parts far from each other, the flocking problem where relative distances among the robots are important, and car–like robots where the final orientation of the robot depends on the path taken.

D. Global Navigation: Integrated Path Planning and Path Following

The third issue we study considers how well we integrate the localizer and the path planner in our global navigation framework. We observe that building multi–purpose roadmaps is a prerequisite for high-quality navigation, and we will evaluate several roadmap construction methods for this purpose. Next, we demonstrate benefits that



(a) (b)

Fig. 84. Using goal sets improves the path quality. (a) A node closest to the goal is found first, and then used to compute the shortest path. (b) The goal set consists of the nodes in the goal area.

can be obtained where feature information is shared by the localizer and the path planner. Finally, we explain our techniques for improving paths during postprocessing. In this step, the path processor is also aware of the needs for localization.

1. Feature–Based Roadmap

Table III compares three methods for computing probabilistic roadmaps using a hallway environment outside our lab. Figure 85(a) shows that at the center of the large square room, there are regions with zero visibility. If we use basic PRM which uses only uniform sampling, then some nodes will be created in the featureless area. For the benefit of localization, this must be improved.

First, two existing PRMs are tested (see Chapter II Section F). The nodes generated by medial axis PRM (MAPRM) [79] are useful for traveling across large areas. Simulation results for MAPRM show that 85.4% of the nodes have nonzero visibility. If obstacle-based PRM (OBPRM) [4, 83] is used, simulation results show that all the nodes have nonzero feature visibility.

Next, we propose a new PRM variant that uses sector information when sampling

TABLE III
ROADMAP NODES IN LOCALIZABLE AREAS

method	number of nodes/edges	number of nodes with visibility 1	number of nodes with visibility 2 or higher
(b)MAPRM	150/424	1(0.7%)	127(84.7%)
(c)OBPRM	150/262	18(12.0%)	132(88.0%)
(d)FBPRM	300/817	69(23.0%)	231(77.0%)

These weights were used in the environment shown in Figure 85.

nodes. This *feature-based PRM* (FBPRM) generates nodes that are evenly distributed in the sectors with nonzero visibility (Figure 85(b)). Depending on mission requirement, we may choose to use sectors where two or more feature are visible. Or, it is also possible to vary the probability of creating nodes depending on the number of features visible. For other mission–specific requirements, new node generation methods and connection methods can be employed. For example, if the robot frequently visits a recharging station, then the area may be connected to other important areas with straight line segments.

Next, we simulate in the lab environment while varying the number of obstacles and the sensor range. When additional obstacles are added, as shown in Figure 68(a), the number of roadmap nodes that can see multiple features increases as illustrated in Figure 86(a). A similar result is obtained when the sensor maximum range is increased as shown in 86(b). In this environment, MAPRM almost always showed the best result.

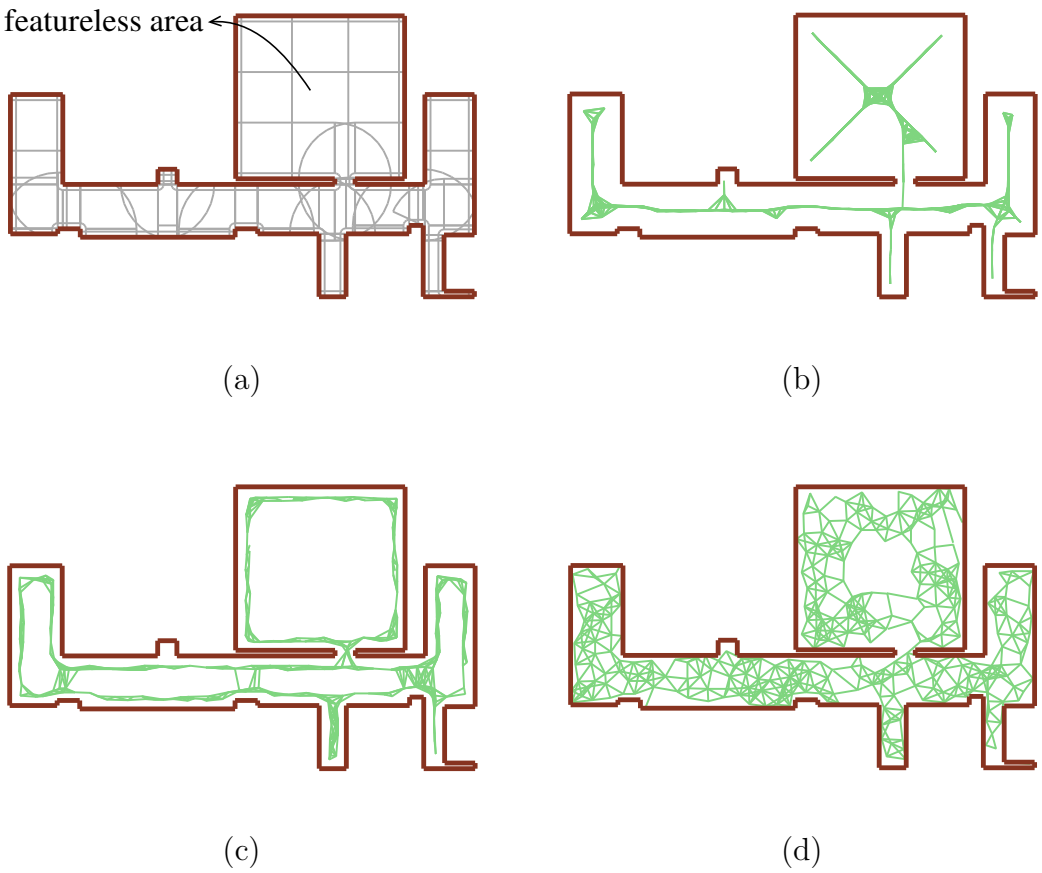


Fig. 85. Properties of roadmap nodes. (a) Visibility sectors in a hallway environment. Roadmaps were constructed using: (b) medial axis PRM, (c) obstacle based PRM, (d) feature based PRM.

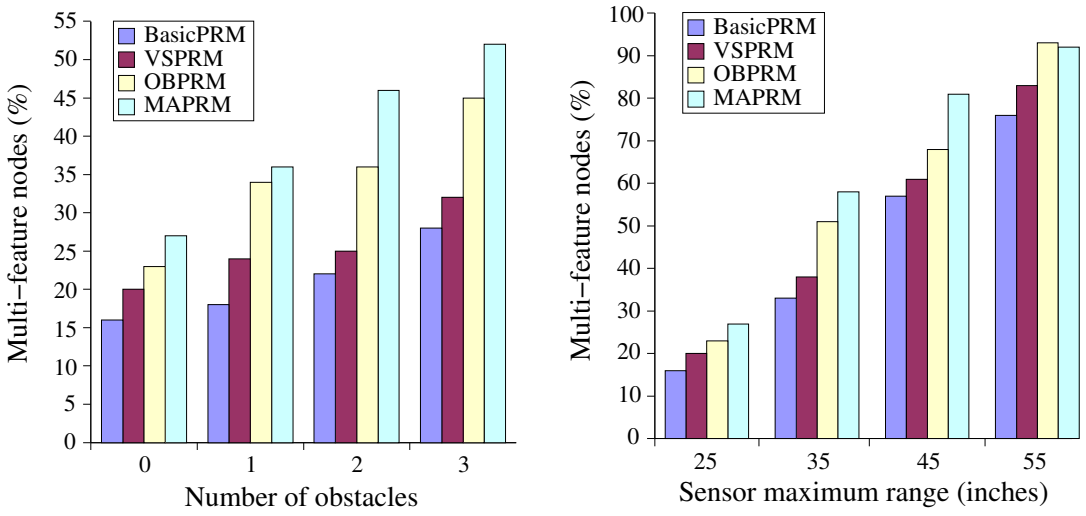


Fig. 86. Multi–feature nodes in the roadmap. (a) Varying obstacles (sensor range is 25 inches), and (b) varying sensor maximum range (without obstacle).

2. Dynamic Path Replanning

In this section, we show how a different route is selected when re-planning after a localization operation. One obvious example is shown in Figure 87 where three different paths could be selected in the second iteration depending on the result of the localization at the first subgoal. The path in the first iteration uses the medial axis, which enables the robot to move as far as possible before the first localization operation. Note that in the second iteration, different cost functions could be used that might move the path away from the medial axis to ensure good feature visibility and accurate navigation around the goal.

Dynamic path replanning can sometime cause a big difference in the selected route for a small change in the robot's position. In Figure 88(a), the global planner initially selects the path `start-b-a-f-goal` by finding the shortest path in the roadmap from `b` to `f`. The node `b` is selected as the `subgoal` since the uncertainty

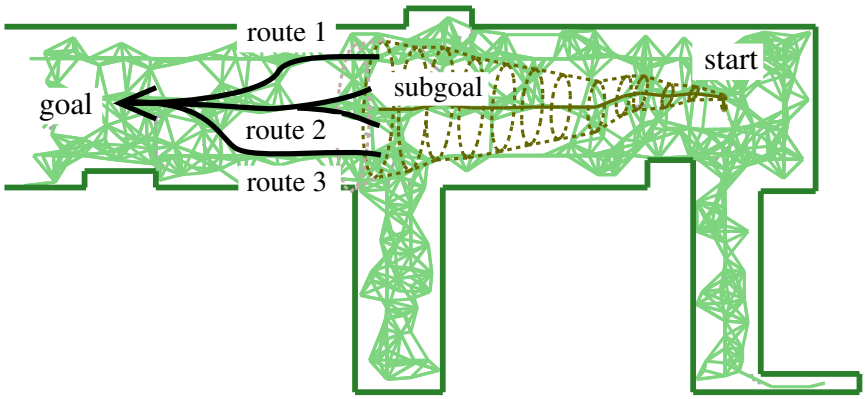


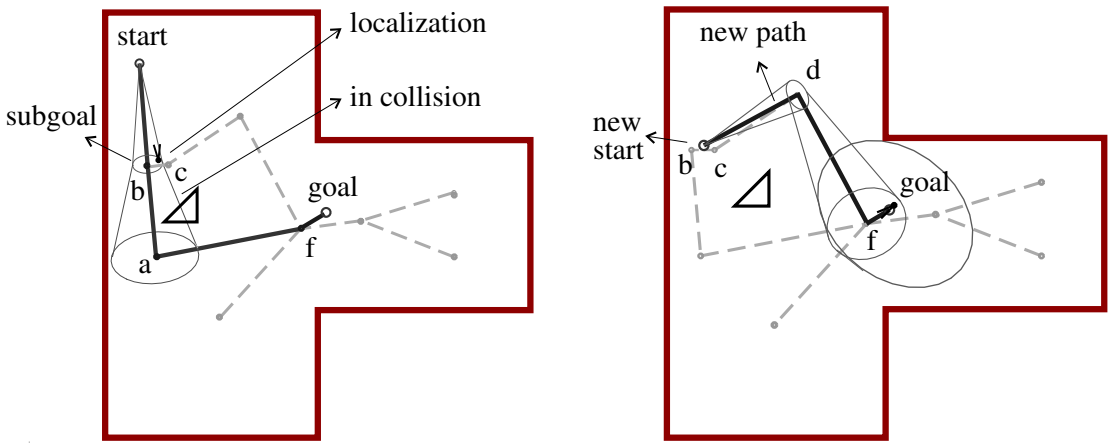
Fig. 87. After localizing at the subgoal, three routes can be taken depending on the robot's actual position.

ellipse along the segment `b-a` intersects an obstacle. After the first robot movement and subsequent localization (see Figure 88(b)), the actual robot position was close enough to the node `c` so that the (re)planning by the path planner selected a different path `(new)start-c-d-f-goal`. Given the new start position, this new path is more efficient than the originally computed path.

Now, the actual `goal` is selected as the `subgoal` since the uncertainty ellipses indicate a collision-free path exists that does not encounter ambiguous sector labels. The goal is reached, within the specified tolerance, after the second iteration. This example shows the difference between our navigator and a simple navigator where the entire path is planned and fixed in the beginning.

3. Avoiding Featureless Areas

A basic requirement for successful localization is that the robot should avoid attempting to localize in featureless areas. One way of ensuring this is to prevent the path from passing through such area. This may result in very inefficient paths. So, we need to allow some portion of the path to be in a featureless area and ensure that no



(a) (b)

Fig. 88. Simulation of path replanning. (a) first iteration and (b) second iteration.

subgoal is placed there. In computing good paths, we aim to achieve feature visibility and efficiency together. Equation 6.3 is used to compute edge weights where $f_1()$ was described in Figure 19(a). This is simulated in the environment shown in Figure 89(a).

$$cost(e_i) = cost_{dist} \cdot length(e_i) + cost_{vis} \cdot f_1(\text{visibility of } e_i) \tag{6.3}$$

In Figure 89(b), $cost_{vis}$ was set small and the resulting path is close to the shortest–distance path. This allows the last two nodes to localize using the upper wall. In Figure 89(c), all the nodes in the path are in sectors with non–zero visibility numbers due to increased $cost_{vis}$. However, in the last node before the goal, the robot cannot see the wall to the right and precise localization is not possible there. The robot might not be able to safely approach the goal. To improve this, $cost_{vis}$ was increased and the result is shown in Figure 89(d). Though this path is longer than

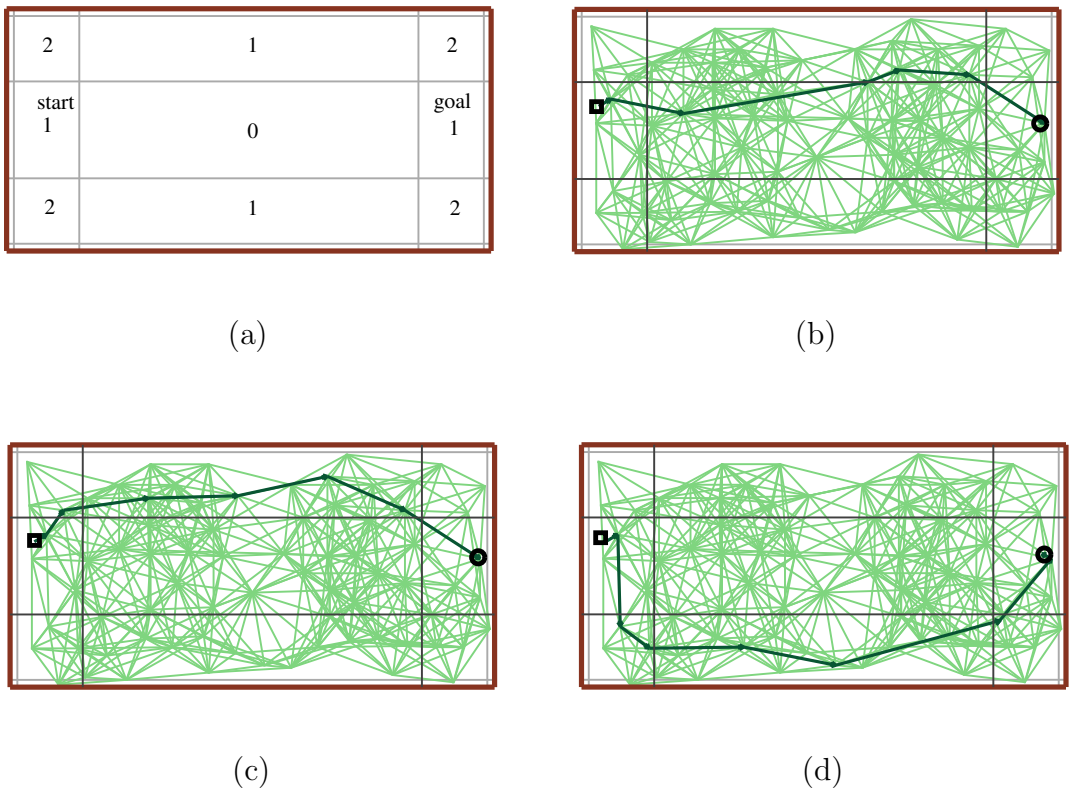


Fig. 89. Avoiding featureless areas. (a) Sectors with visibility numbers (0 – 2), and start and goal are shown. (b) Path selected using a combination of distance and visibility, (c) path selected when visibility gets more weight than distance, and (d) path selected when visibility is maximized.

the previous cases, three nodes are in areas that have two visible features.

Next, we simulate in the lab environment using the same edge weight function. The sensor maximum range has been set to be short (20 inches) to create a large featureless area in the middle of the environment. Depending on the combination of edge weight constants as shown in Table IV, four different paths are obtained as shown in Figure 90. Path 1 strictly avoids the featureless area, and paths 2 and 3 have one and two node(s) in the featureless area, respectively. Path 4 is the shortest-distance path (roadmap is not shown).

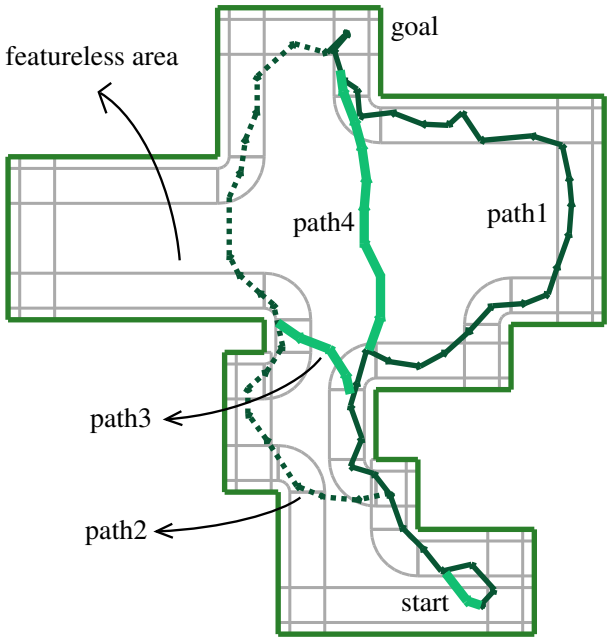


Fig. 90. Avoiding featureless areas in the lab environment. Four paths are obtained using the edge weight combinations shown in Table IV.

TABLE IV
FOUR DIFFERENT PATHS

path number	$cost_{dist}$	$cost_{vis}$
path 1	0	1
path 2	1	50
path 3	1	15
path 4	1	0

These weight constants were used for the lab environment shown in Figure 90.

4. Reducing the Number of Localizations

Though localization helps avoid collision, unnecessary localization should be avoided because it takes time and resources. Like many previous cases where the boundary between optimal and inefficient is not clear, the minimum number of necessary localizations cannot be estimated. Our approach is to assume that the robot always makes a certain amount of error and that it needs to be localized whenever a collision would be possible if no localizations were performed. Also, we assume that each localization is successful and reduces the uncertainty to zero, i.e., the uncertainty ellipse a point after localization.

A simple environment with two corridors was made for simulation. Figure 91(a) shows the roadmap in the environment. Subsequent figures show snapshots of searching a path and do not reflect the robot's actual movement. A collision is detected in Figure 91(b), and as a result, the upper corridor is explored faster in Figure 91(c). Finally, the goal is reached from the upper corridor first as shown in Figure 91(d).

Next, we simulate in the hallway environment. If we consider the number of localizations only, then the path will have large uncertainty regions and thus approximately follow the medial axis of the environment. This is shown in Figure 92 where the dotted line was obtained using $cost_{dist} = 1, cost_{loc} = 4, cost_{vis} = 0$ in Equation 6.4. The resulting path minimizes the estimated navigation time.

$$\begin{aligned} cost(e_i) = \quad & cost_{dist} \cdot length(e_i) \\ & + cost_{vis} \cdot f_1(\text{visibility of } e_i) \\ & + cost_{loc} \cdot (\text{if collision in } e_i) \end{aligned} \tag{6.4}$$

Finally, we improve the path's safety by employing nonzero $cost_{vis}$. The result is shown by the solid line in Figure 92 which visits an area that has multiple features. In general, the path cost functions must consider efficiency and safety measures of

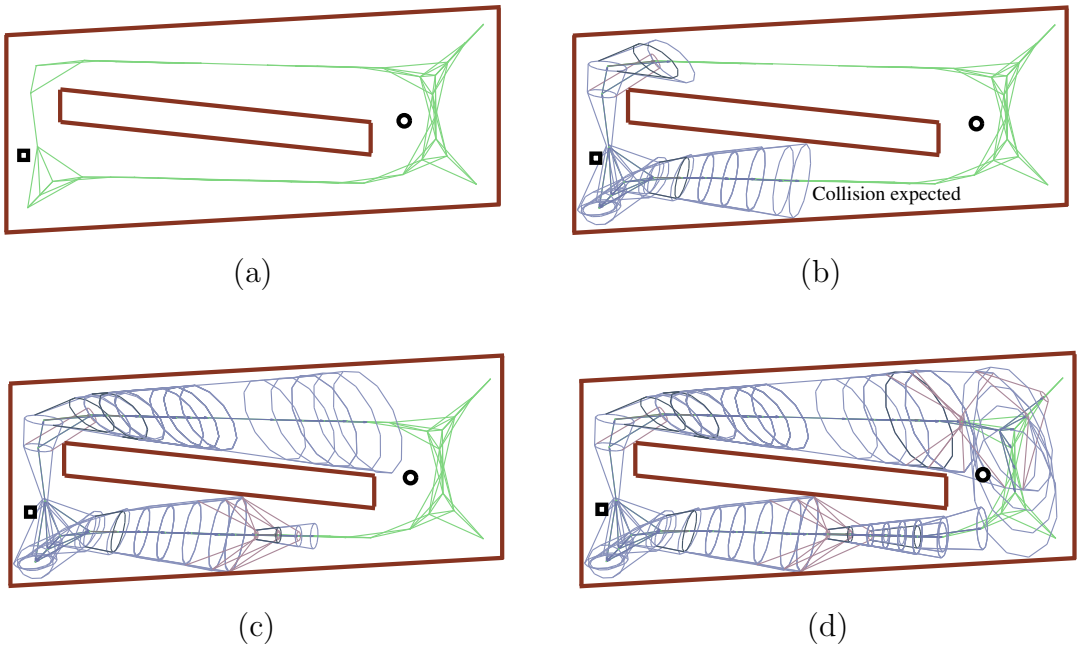


Fig. 91. Minimizing total time by choosing a smaller number of localizations.

path planner and path follower, which are summarized in Table V.

TABLE V
SUMMARY OF EDGE WEIGHTS

	Safety	Efficiency
Path planner	$cost_{clr}$	$cost_{dist}$
Path follower	$cost_{vis}$	$cost_{loc}$

5. Path Postprocessing

The path extracted from a probabilistic roadmap is generally composed of a number of straight line segments as shown in Figure 93(a). Such paths often make unnecessary short translations and turns. The quality of the paths can be improved by

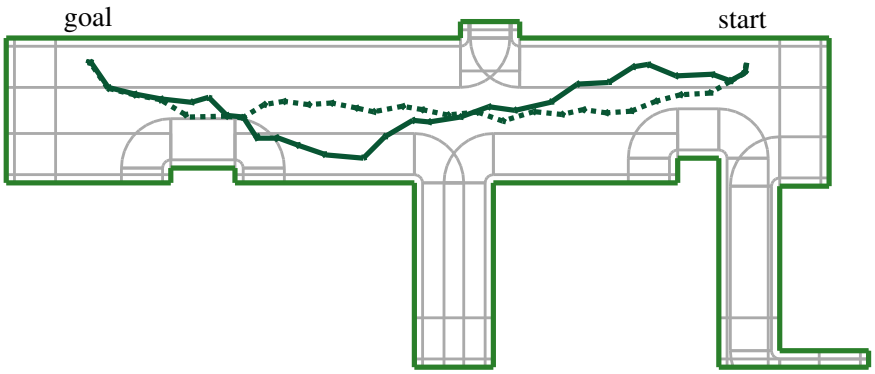


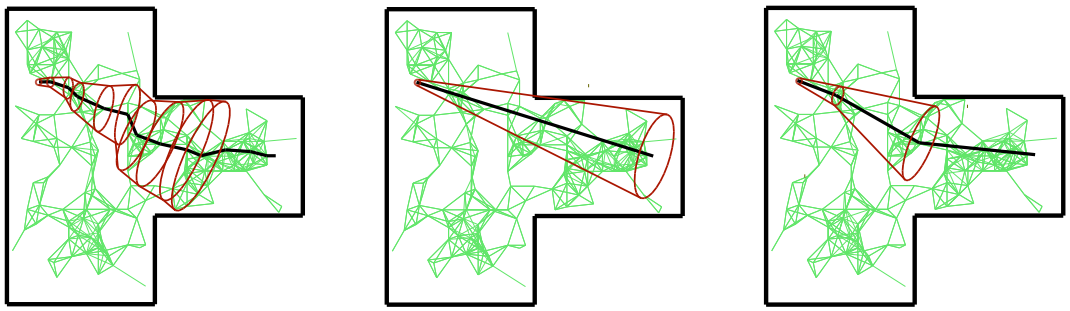
Fig. 92. Optimizing for number of localizations and feature visibility. The dotted line is the shortest path, and the solid line accounts for feature visibility while minimizing the number of localizations.

additional processing. One can convert to a spline curve, which allows the robot to move continuously. Though there are several ways to generate such high–quality paths, they often require long processing times.

Our approach to this problem is to reduce the number of segments by removing a node in the path whenever the angle between two adjacent edges is smaller than a certain value. Figure 93(b) shows the result of removing all possible nodes using the turning angle condition. This path has been maximally straightened and no valid subgoal can be found. So, we check that resulting path meets the necessary criteria before removing a node. The result is shown in Figure 93(c) where the path is shortened so that it contains two subgoals.

E. Hardware Experiments

In this section, we demonstrate the effectiveness of our navigation methods with a real mobile robot, focusing on the localization process. The first experiment shown in Figure 94 used a simple small environment to test the reliability of localization. The



(a) (b) (c)

Fig. 93. Path postprocessing. (a) Uncertainty ellipses shown for each line segment in the roadmap. (b) The path is over-straightened, and (c) a subgoal is determined after reducing the path to three segments.

environment's outside dimension is 2.25 by 2.25 meters, and the width of the robot is 26cm. It required two localizations to move from the start to the goal. After the goal was reached, the next goal selected was the original start location. In other words, the start and the goal were exchanged and the path was reversed. This experiment was repeated ten times and they were all successful with a small variance in trajectory due to odometry error and sensor noise.

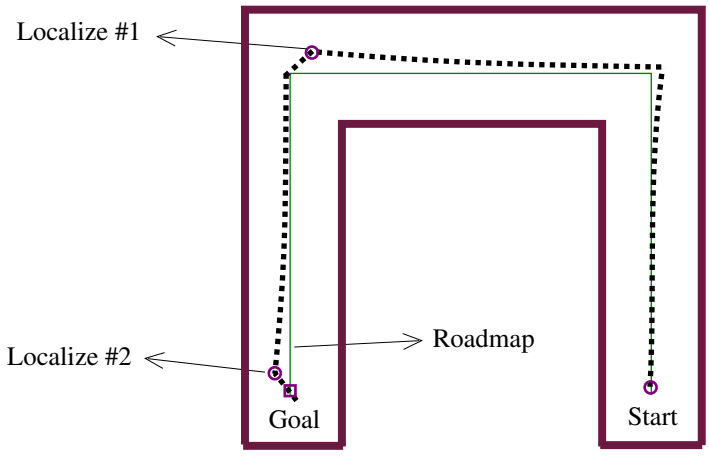


Fig. 94. A small environment for repeated experiment.

Next, we used our lab environment for experiments. Figure 95 shows a part of our lab that has doors, desks, cabinets, and trash cans. The minimum and maximum sensor range was set to 5 and 100 inches, respectively. This range was large enough to cover all the nodes generated by MAPRM and to ensure that all of them have visibility two or greater. Figure 96(a) shows the robot's start and goal configurations.

After a path is extracted and the first subgoal is chosen, the robot travels toward the subgoal and stops. Figure 96(a) also depicts the robot's growing uncertainty and actual position at the subgoal. Stopped at the subgoal in Figure 96(b), the robot scans the environment and potential features are gathered. The features observed by the sonar scans are matched to known features in the environment and the robot's pose is updated, Figure 96(c). Continuing, a new path is computed from the robot's actual position to the goal and the process iterates, Figure 96(d).

In the experiment shown in Figure 96, the robot's position at the first subgoal was off by 6 inches to the west and 6 inches to the north. The localization step corrected this but the sonar scans also included error because of an unknown obstacle, a trash can, shown in Figure 96(a). After localization the robot's position continued to be off target by 4 inches to the east and 2 inches to the south. The robot was able to reach the goal area after replanning the path from the localized configuration.

F. Multiple Robot Localization

For the uncertainty reducing problem described in Chapter V Section C, two environments were used to simulate the proposed heuristic methods. Also, a random selection method was used to compare with our methods.

Fig. 95. A part of the lab environment with an AmigoBot.

1. Tree Root–Based Method

In the environment shown in Figure 97, a random selection and our rooted tree methods were simulated.

Although we would not expect it to perform well, we compare with a naive random ordering method because we are not aware of any other multiple robot localization methods that address the issue of determining the best order in which to localize the robots. As illustrated in Figure 98, the random selection took ten sequential rounds to completely localize all robots. This was reduced to seven rounds with the rooted tree method as shown in Figure 99. If we allow simultaneous localization of multiple robots in each round, then rounds {3, 4} and {5, 6} can be processed in parallel, and the total rounds required are reduced to five.

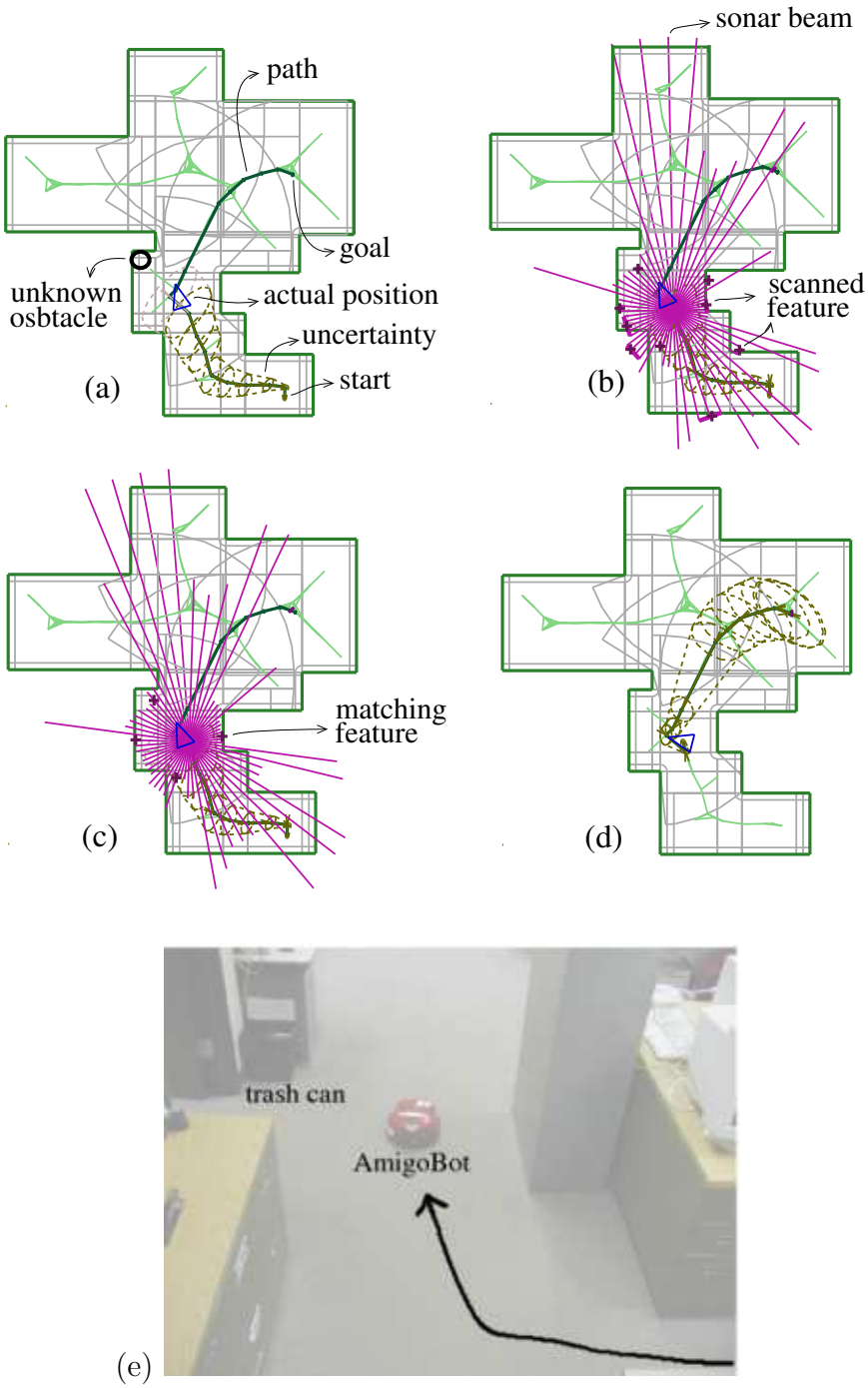


Fig. 96. Hardware experiment using AmigoBot in lab environment. (a) the first iteration, (b) local minimum features identified, (c) matching features, (d) new path after localizing, and (e) trajectory of the robot.

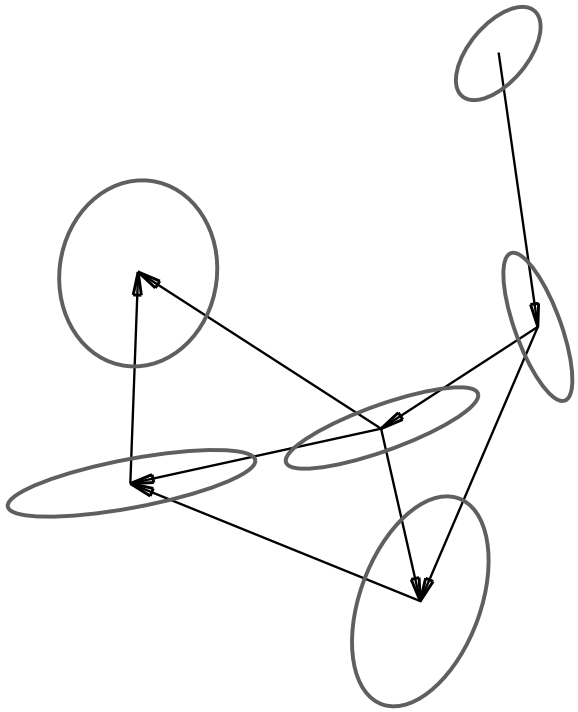


Fig. 97. Environment for approximate solution (uncertainty ellipses and FVG with directed edges are shown).

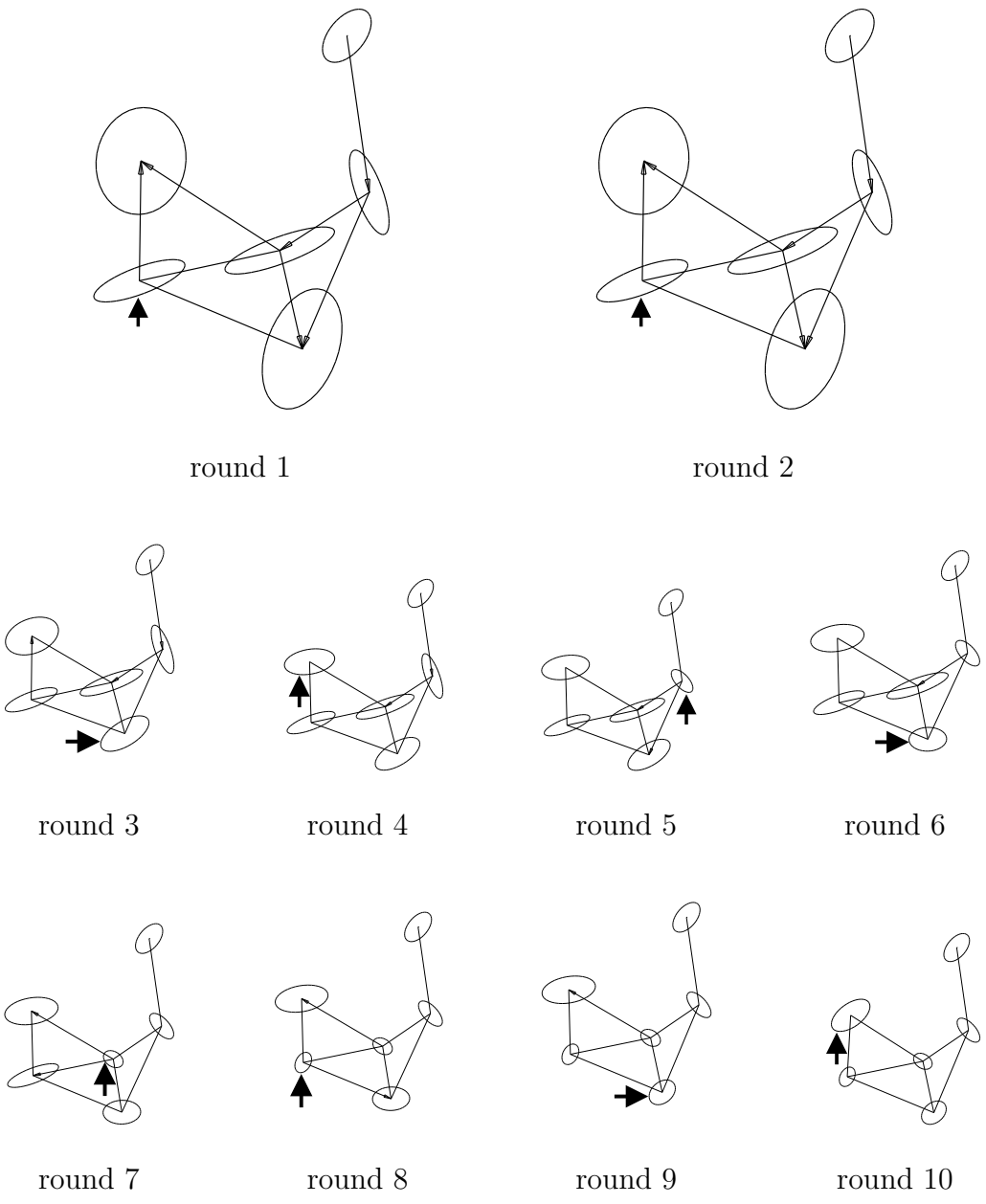
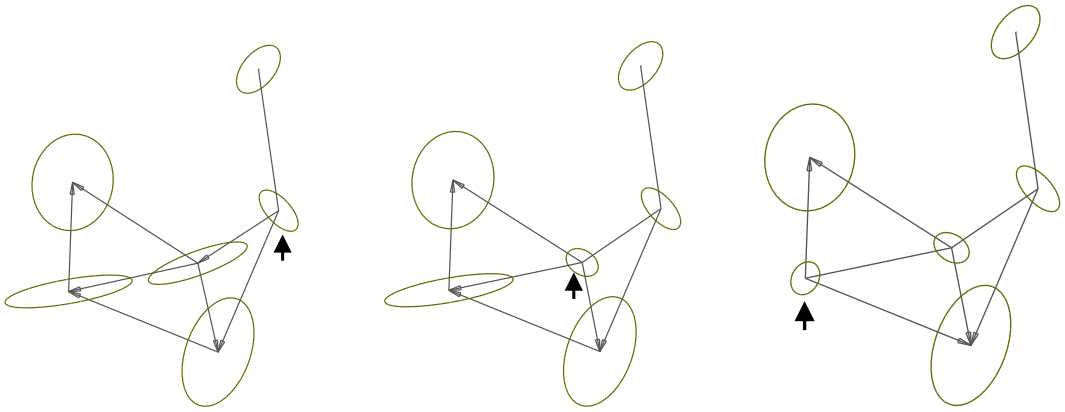
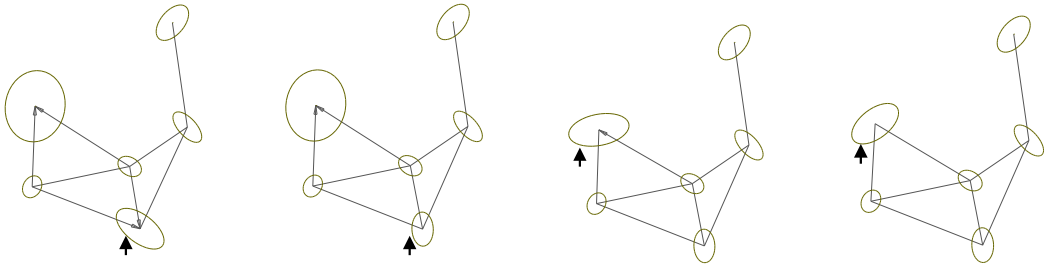


Fig. 98. Randomly selecting a robot. A total of ten rounds were taken to completely localize all robots. At each round, in this figure, only one robot is localized (sequential case)

.



round 1 round 2 round 3



round 4 round 5 round 6 round 7

Fig. 99. Selecting a robot at the top of tree. The first round selects a robot that is adjacent to the smallest ellipse. The 4th and the 5th rounds localize the same robot using different features.

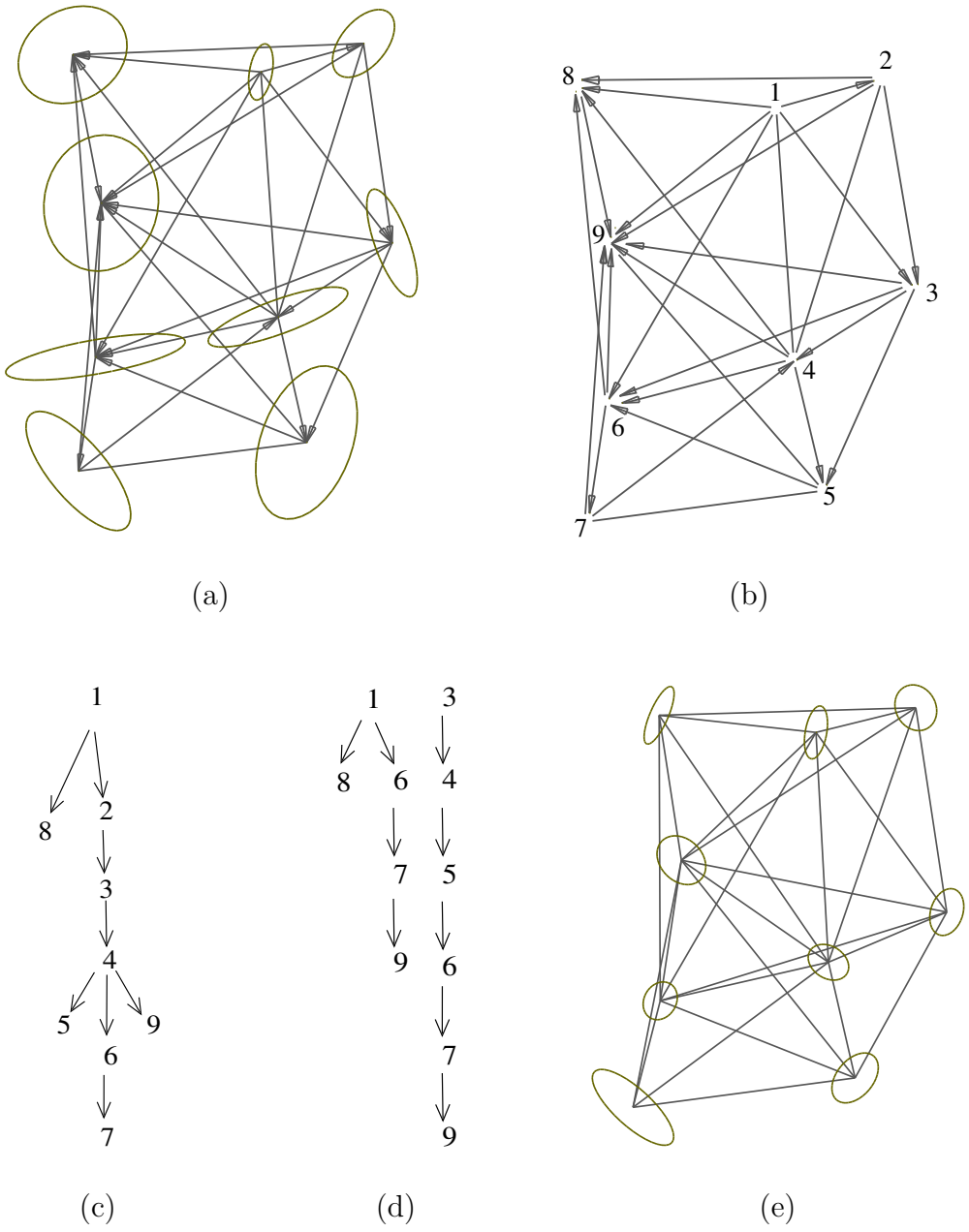
TABLE VI
THREE HEURISTICS FOR UR

	random	rooted tree	uncertainty-based
sequential rounds	15.8	15.2	14
parallel rounds	11.3	10.3	10

2. Uncertainty-Based Method

To demonstrate the uncertainty–based method, a more complex environment shown in Figure 100(a) was used. An embedded tree for this environment is shown in Figure 100(c). After localizing robots 1 and 2, two trees are formed (Figure 100(d)). Robot 6's ellipse is larger than the ellipse of robot 4, and the robot with a smaller ellipse belongs to the taller tree in this case. This environment was simulated using three methods and the results are shown in Table VI.

If random numbers are used, the value reported is the average of five simulations that were performed for each problem using different random number seeds. The maximum number of tree–like portions is at most three in this case. So, localizing more than three robots in one round does not help improve the total time.



(a) (b)

(c) (d) (e)

Fig. 100. Ellipse-based method for UR. (a) The FVG and uncertainty ellipses before localizing, (b) the FVG with numbers ordered by the size of the ellipses, (c) an embedded tree before localization, (d) two major embedded trees after localizing robot 1 and 2, and (e) the FVG after localizing all robots.

CHAPTER VII

CAMPUS NAVIGATOR

Our roadmap-based path planning techniques have been applied to a web-based route planner that covers the Texas A&M campus. This is intended to be a practical application that can potentially be used by people to navigate on our campus. The idea was conceived originally as a practical form of the technique for optimal path extraction for mobile robots, and received attention from the people in related fields including the TAMU Administrative GIS office.[1] There are many important issues when implementing such systems; such as generating precise paths and minimizing the effort for software development. There are two major components in our system: path planner (called campus-query) and visualizer (called campus-vizmo), and they are modified versions of query and vizmo that have been developed by the Algorithms and Applications group in the Parasol lab [59]. Four graduate students (Jinsuck Kim, Jyh-Ming Lien, Aimee Vargas E., Roger Pearce) and two former undergraduate students (Nick Downing and Bharatinder Sandhu) have participated in this project.

A. Features

Our system is more sophisticated than Yahoo! Map or MapQuest, etc. It includes a path planner and hand–made roadmaps.

- It supports several modes of transportation. Currently we have implemented pedestrian (walking and wheelchair), individual vehicle (bicycle, motorcycle and automobile), and bus. Paths utilizing multiple modes are considered to obtain

[1]Maintains the official map of Texas A&M University campus at College Station. http://www-agis.tamu.edu

the best result. For example, most paths will start and/or end with walking but they might use a bus in the middle to minimize time.

- The path can be optimized and individually customized with respect to several criteria such as shortest time, shortest distance, or shortest walking distance. For example, the traveler can choose not to take a bus even if it results in faster travel.

- When visualizing the path, unlike other software that shows 2D images, our program generates 3D images with realistic building models. These 3D images can be projected to 2D (see Figure 101). Our system supports graphics and text-based user interaction. For example, when selecting building query points, the user can click on the screen where buildings and their names are displayed, or they can type in a building name or select from a building list (see Figure 102).

B. Components

The major components of the Campus Navigator are illustrated in Figure 103. User requests over the internet are received by a web server and the corresponding query data are inserted into the queue. The query data includes start, goal, and other parameters such as transportation methods. For each query, a path is computed by the path generation module (campus-query) and sent to the image generator (campus-vizmo). After the environment and the path are drawn in 3D (and projected to 2D), the resulting webpage is generated and sent back to the user. Our system runs in linux with Apache and PHP for web serving, OpenGL and QT [76] for graphics, and MySQL for roadmap storage. The campus-query, campus-vizmo, and other modules

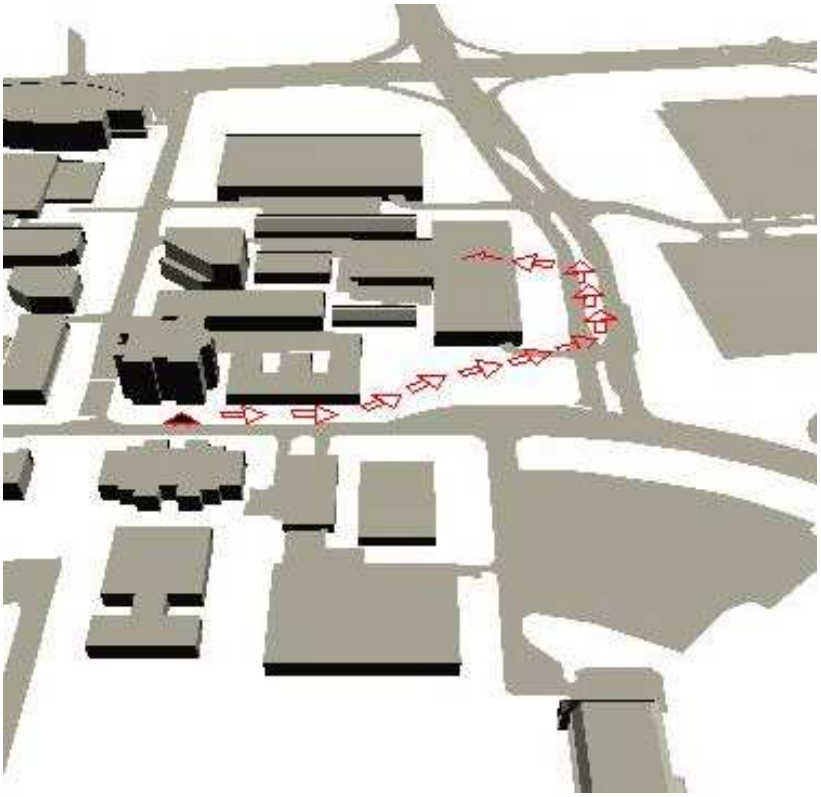


Fig. 101. A part of result webpage of the Campus Navigator. Buildings, streets and a path from Bright building to Wisenbaker building are shown.

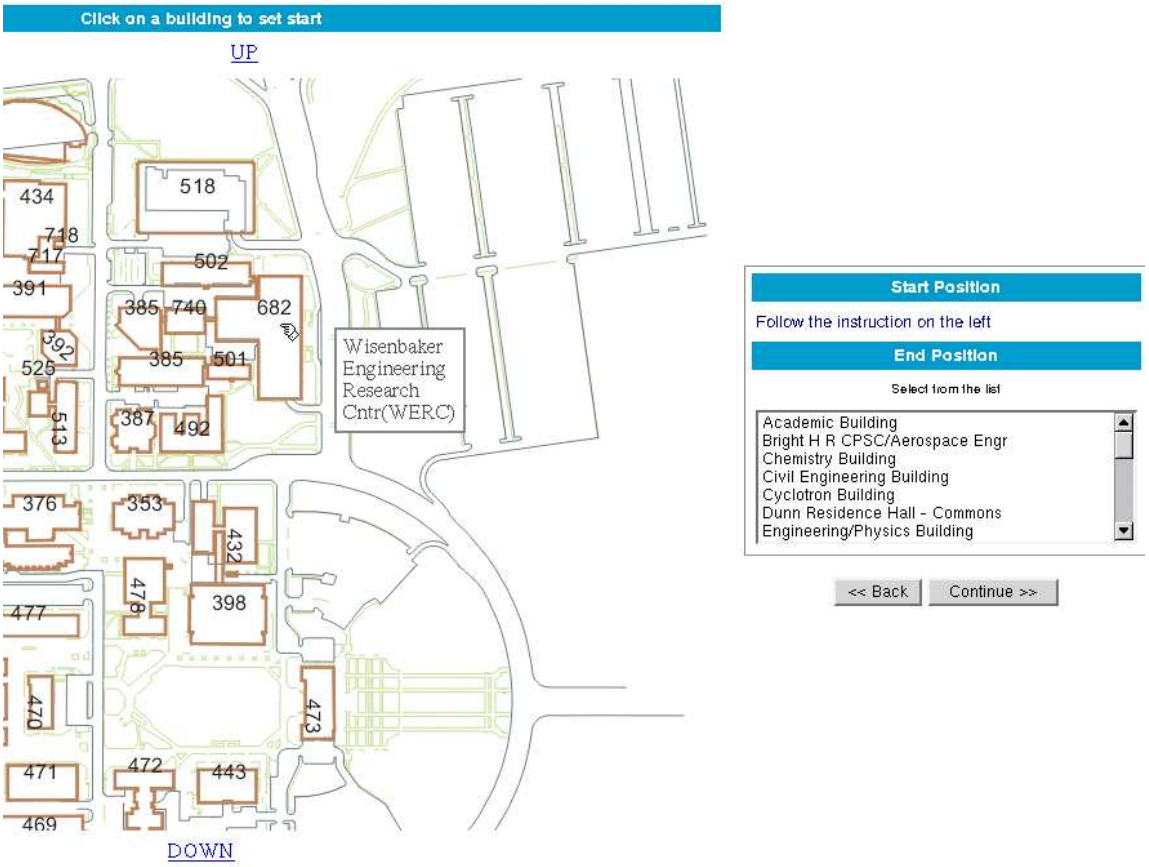


Fig. 102. Two ways to choose the start location: clicking on the screen or selecting from a list of buildings.

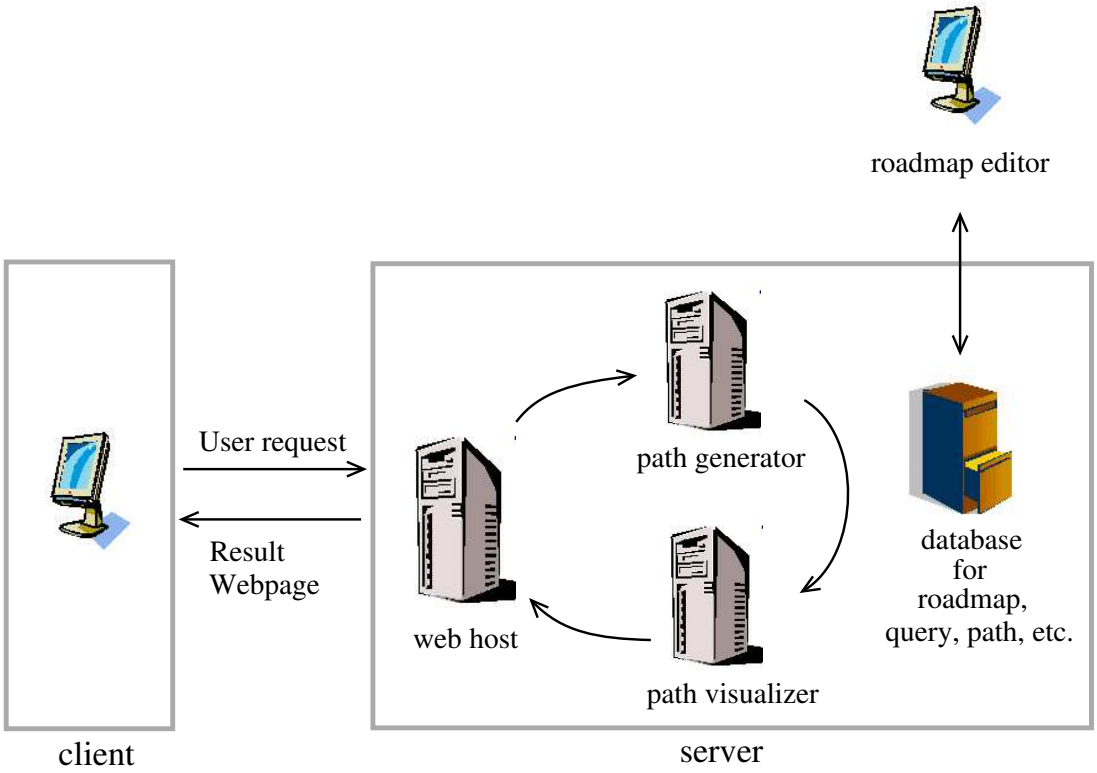


Fig. 103. System architecture of the Campus Navigator.

are written in C++ with MySQL API. Figure 104 shows additional components that handle multiple user requests, generate the text guide, and support zoom/pan of the resulting image. The query and path queue are actually database tables which are convenient for maintenance and monitoring.

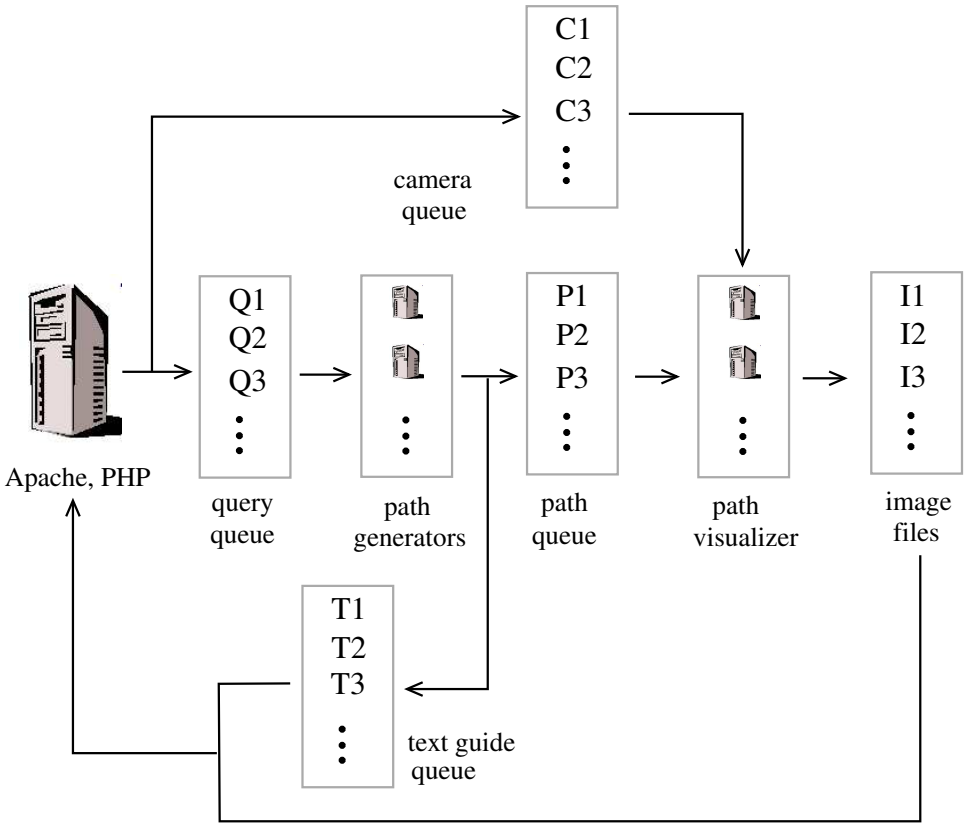


Fig. 104. Detailed architecture of Campus Navigator server. Each queue is actually a table in the database, and image files are stored in a network file server system.

Roadmap. The roadmap is manually generated and contains real–world information. For example, each vertex in the graph is related with a building or street. Some buildings represent departments and services in the university, and some street vertices are bus stops, parking lots, etc. Bus stops are related to bus routes in the database. For these reasons, we have chosen to use a relational database system

to store and maintain our roadmap. Constructing such roadmaps involves modifying graph topology and editing edge/vertex properties, which ultimately results in editing database tables. Appendix A shows the tables used for our campus navigator.

Roadmap Editor. Text–based database table editing tools such as MySQL client or phpMyAdmin [55] are not suitable for creating large, complex roadmaps. For this purpose, we have developed a dedicated graphical roadmap editor. Our roadmap editor is based on vizmo [56] and has been extended to have specialized functions such as node/edge editing dialogs and database connection. Screen shots of the editor are shown in Appendix B.

Path Generation. During initialization, campus-query first reads the roadmap from the database to memory and converts it to a graph–based roadmap. The roadmap uses a vertex for a single location with a single transportation method. For example, if a node in the database roadmap is a bus stop which is also a part of a street, then two vertices are used to represent 'in the bus' and 'walking on the street'. The motivation for this roadmap conversion is that our path planning software can be used without modification on these roadmaps.

Converting the roadmap to a graph takes two steps. First, for each vertex in the database, a vertex is created as a seed vertex. This step generates a graph whose topology resembles the database roadmap (see Figure 105(a)). Valid modes of transportation for each vertex are identified and recorded in the seed vertex. Second, for all modes of transportation, vertices are created and grouped with the seed (see Figure 105(b)). After that, all pairs of vertices in a group or in adjacent groups are connected if they share a transportation method. (see Figure 105(c)).

Web interface. The web interface module uses HTML and PHP to communicate with the user. In the first screen, it asks the user how start and goal positions

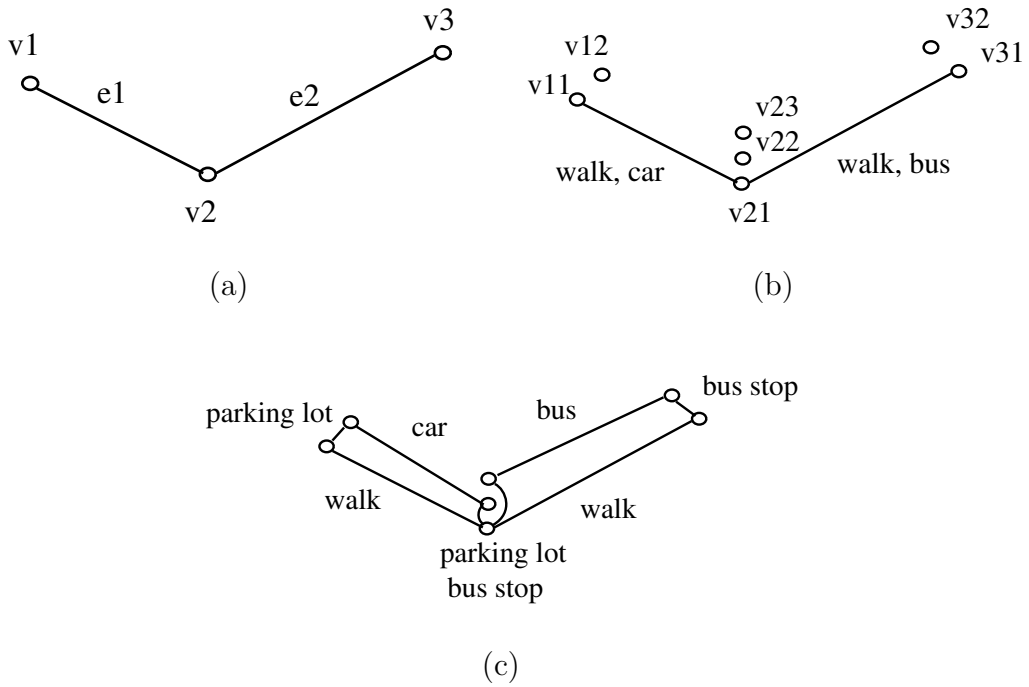


Fig. 105. Constructing graph-based roadmap from database during initializing campus-query. (a) Three vertices and two edges with different transportation methods. (b) In the first step, vertices are duplicated according to valid transportation methods in the edges connected. (c) In the second step, for each method, edges are created to connect all the vertices.

will be specified. Currently available options include typing a building name in a text box, clicking with a mouse on the map, or selecting from a list of building names. In the next page, the user performs actions to specify the start and the goal. If the mouse click is selected, since the entire campus map is large and contains many buildings, the image is enlarged (zoom in) before the user can click on a building.

Next, the user may request that a route be computed or they can input more options. If more options are chosen, the user is allowed to specify, e.g., where a bike or a car is parked. Also, specific transportation modes can be specified which will then override automatic selection of the transportation methods. After the resulting image and text guide have been displayed in the user's screen, the image can be zoomed or

panned if desired. For this, the query/path queue data contains special fields that indicate the horizontal and vertical position of the camera.

Image generation. No significant changes were required in making campus-vizmo from vizmo [56]. We added methods to connect to the database and an automatic camera positioning function. To speed up image generation, the entire building and street model files are loaded only once when campus-vizmo starts.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

In this dissertation, we proposed methods of navigation for personal mobile robotic applications that are safe, robust, and inexpensive. Our navigator uses a roadmap–based path planner and sector–based localizer. We first discussed limitations of previous methods and described our general strategy which integrates the path planner and localizer. We proposed improvements for path planning, by augmenting Dijkstra's graph traversal algorithm, and we developed new sector subdivision techniques for localization. Also, our localization approach has been extended to handle multiple robots. To show the effectiveness of our methods, experimental results were obtained assuming partial knowledge about the environment, a constrained driving mechanism, modest data storage and computing power, and limited sensor visibility. To demonstrate the applicability of our techniques to other areas, a web–based campus route planner was developed.

In Chapter III, we explained our approach for integrating the path planner and localizer. This was facilitated by the use of roadmaps which contain candidate paths for various purposes such as efficient navigation and robust localization. Our path planner extracts the best path in our roadmap using an augmented version of Dijkstra's algorithm that can handle a (i) Markov–like cost function and (ii) a relaxed goal definition. We discussed useful cost function formulations such as minimizing travel time, avoiding featureless areas, dealing with kinematic constraints, and maximizing minimum clearance. Customized combination of these priorities enables the selection of paths that balance various criteria such as safety and efficiency. As future work, the path planner can be extended to deal with multiple robots.

In Chapter IV, we discussed our new localization methods that combine advan-

tages of previous approaches for feature identification and space subdivision. We defined primitive sectors that account for physical sensor limitations. Our approach for dealing with sensor range and incidence angle restrictions was to (i) merge perfect–visibility sectors and (ii) use only scannable features. A complete localization algorithm and simulation results have been presented. Hardware experiments to show the practicality of our methods were performed in our lab environment using an AmigoBot equipped with sonar sensors. Future work related to visibility sectors includes proving their effectiveness if used with other localization methods such as particles filters.

Our localization methods have been extended to handle multiple robots in Chapter V. First, we investigated and formulated two problems for efficiently localizing multiple robots in close proximity: (i) minimizing total sensing time while avoiding sensor cross–talk, and (ii) reducing the uncertainty regions using a minimum number of sensings. We showed that both these problems are NP–complete by reduction from the graph coloring and task scheduling problems, respectively. To compute approximate solutions for the second problem efficiently, we have proposed an algorithm that estimates the size of the critical path based on uncertainty ellipse size. Simulation results showed that our proposed algorithm reduces the total localization time compared to a completely random approach. As future work, we are interested in enhancing the methods to deal with cycles in the graph representation of the robots.

To demonstrate that our navigation techniques are not limited to mobile robots, we have developed a web-based route planner for the Texas A&M campus. Major components are (i) a path planner, (ii) a path visualizer, (iii) a roadmap representing the campus stored in database, and (iv) a user interface. The first two components are slight modifications of software that has been developed by the Parasol Algorithms and Applications group. This is an on–going group project and we are working on enhancing the visual output, supporting more complex query options, and building a

more realistic roadmap.

REFERENCES

[1] F. Ababsa and N. Bouguechal, "Environment modeling and localization technique for an autonomous mobile robot," in *IEEE Internation Conference on Control Applications*, vol. 2, pp. 1338–1342, 1998.

[2] S. Akella and S. Hutchinson, "Coordinating the motions of multiple robots with specified trajectories," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 624–631, 2002.

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo, "Choosing good distance metrics and local planners for probabilistic roadmap methods," Tech. Rep. 98-010, Dept. of Computer Science, Texas A&M University, May 1998. A preliminary version of this paper appeared in *ICRA'98*.

[4] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Robotics: The Algorithmic Perspective*, (Natick, MA), pp. 155–168, A.K. Peters, 1998. Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[5] N. M. Amato, O. B. Bayazit, K. Kim, W. Son, and G. Song, "Providing haptic 'hints' to automatic motion planners," Tech. Rep. 98-026, Dept. of Computer Science, Texas A&M University, Nov 1998.

[6] N. M. Amato, C. V. Jones, and D. Vallejo, "An adaptive framework for 'single shot' motion planning," Tech. Rep. 98-025, Dept. of Computer Science, Texas A&M University, Nov 1998.

[7] E. Araujo and R. Grupen, "Feature detection and identification using a sonar-array," in *ICRA*, pp. 1584–1589, 1998.

[8] S. Atiya and G. Hager, "Real-time vision-based robot localization," *IEEE Trans. Robot. Automat.*, vol. 9, no. 6, pp. 785–800, 1991.

[9] I. Baev, W. Meleis, and A.Eichenberger, "Lower bounds on precedence-constrained scheduling for parallel processors," *International Conference on Parallel Processing*, pp. 549–553, 2000.

[10] O. B. Bayazit, J.-M. Lien, and N. M. Amato, "Probabilistic roadmap motion planning for deformable objects," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 2126–2133, May 2002.

[11] A. Blum, "New approximation algorithms for graph coloring," *Journal of the ACM*, vol. 41, no. 3, pp. 470–516, 1994.

[12] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 1018–1023, 1999.

[13] M. Cesati and H. Wareham, "Parameterized complexity analysis in robot motion planning," in *IEEE International Conference on Intelligent Systems for the 21st Century*, vol. 1, pp. 880–885, 1995.

[14] S. Chakrabarti and S. Muthukrishnan, "Resource scheduling for parallel database and scientific applications," *ACM Symposium on Parallel Algorithms and Architectures*, pp. 329–335, 1996.

[15] Y. Chen and Y.-L. Gu, "Optimal path generation of a (6+1)-axis robot for auto-industrial applications," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1,

pp. 388 – 395, 1993.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: MIT Press, McGraw-Hill, 1998.

[17] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer-Verlag, 2000.

[18] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 1322–1328, 1999.

[19] G. Dudek, *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000.

[20] T. Emaru and T. Tsuchiya, "Avoid crosstalk of sonar by using sliding mode system," in *2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation (IECON-2000)*, vol. 4, pp. 2377–2382, 2000.

[21] D. Fox, B. Wolfram, H. Knruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robots on Heterogeneous Multi-Robot Systems*, vol. 8, no. 3, 2000.

[22] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," Technical Report UW-CSE-01-08-02, University of Washington, Seattle, WA, 2001.

[23] M. Garey and D. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[24] L. J. Guibas, R. Motwani, and P. Raghavan, "The robot localization problem," *SIAM Journal on Computing*, vol. 26, pp. 1120–1138, Aug. 1997.

[25] J. Gutmann and D. Fox, "An experimental comparison of localization methods continued," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 1, pp. 454 – 459, 2002.

[26] C. Hocaoglu and A. Sanderson, "Planning multiple paths with evolutionary speciation," in *IEEE transactions on evolutionary computation*, vol. 5, pp. 169–191, 2001.

[27] J. Howell and B. R. Donald, "Practical mobile robot self-localization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3485–3492, 2000.

[28] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pp. 141–153, Natick, MA: A. K. Peters, Ltd., 1998.

[29] G. Hueber, T. Ostermann, T. Bauernfeind, R. Raschhofer, and R. Hagelauer, "New approach of ultrasonic distance measurement technique in robot applications," in *5th International Conference on Signal Processing Proceedings*, vol. 3, pp. 2066 – 2069, 2000.

[30] I. S. Jeong and H. S. Cho, "Self-localization for mobile robot by matching of two consecutive environmental range data," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1603–1608, 2001.

[31] K. Jorg and M. Berg, "First results in eliminating crosstalk and noise by applying pseudo-random sequences to mobile robot sonar sensing," in *Proceedings of the*

First Euromicro Workshop on Advanced Mobile Robot, pp. 40–45, 1996.

[32] G. Kantor and S. Singh, "Preliminary results in range-only localization and mapping," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 1818 –1823, 2002.

[33] D. Karger, R. Motwani, and M. Sudan, "Approximate graph coloring by semidefinite programming," *Journal of the ACM*, vol. 45, pp. 246–265, 1998.

[34] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 4, pp. 3020–3025, 1996.

[35] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," in *IEEE Trans. Robot. Automat.*, vol. 14, pp. 166–171, 1998.

[36] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, pp. 566–580, August 1996.

[37] J. Kim and N. Amato, "Complexity analysis and approximate solutions for two multiple-robot localization problems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2004. to appear.

[38] J. Kim, N. Amato, and S. Lee, "An integrated mobile robot path (re)planner and localizer for personal robots," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 4, pp. 3789–3794, 2001.

[39] J. Kim, R. Pearce, and N. Amato, "Robust geometric–based localization in indoor environments using sonar range sensors," in *Proc. IEEE Int. Conf. Intel.*

Rob. Syst. (IROS), 2002.

[40] J. Kim, R. Pearce, and N. Amato, "Extracting optimal paths from roadmaps for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 2424 – 2429, 2003.

[41] J. Kim, R. Pearce, and N. Amato, "Feature-based localization using scannable visibility sectors," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 2854 – 2859, 2003.

[42] D. E. Kirk, *Optimal Control Theory: An Introduction*. Prentice Hall, 1970.

[43] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida, "3-d path planning in a dynamic environment using an octree and an artificial potential field," in *IEEE/RSJ International Conference on Human Robot Interaction and Cooperative Robots*, vol. 2, pp. 474 – 481, 1995.

[44] K. Komoriya and E. Oyama, "Position estimation of a mobile robot using optical fiber gyroscope," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 1, pp. 143–149, 1994.

[45] P. Konkimalla and S. LaValle, "Efficient computation of optimal navigation functions for nonholonomic planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 187–192, 1999.

[46] K. Konolige, A. Saffiotti, E. Ruspini, K. Myers, and D. Guzzoni, "Saphira." [Online]http://www.ai.sri.com/∼konolige/saphira/.

[47] B. J. Kuipers and Y. T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Journal of Robotics and Autonomous Systems*, vol. 8, pp. 47–63, 1991.

[48] F. Lamiraux and J.-P. Lammond, "Smooth motion planning for car-like vehicles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 498–501, 2001.

[49] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[50] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: progress and prospects," in *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pp. SA45–SA59, 2000.

[51] S. Lee, N. M. Amato, and J. P. Fellers, "Fast localization of mobile robots using visibility sectors," tech. rep., Department of Computer Science, Texas A&M University, 1999. Appeared in ICRA 2000.

[52] S. Lee, N. M. Amato, and J. P. Fellers, "Localization based on visibility sectors using range sensors," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3505–3511, 2000.

[53] J.-M. Lien, S. L. Thomas, and N. M. Amato, "A general framework for sampling on the medial axis of the free space," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 3, pp. 4439–4444, 2003.

[54] ActiveMedia Robotics, Peterborough, NH, "Amigobot." [Online] http://www.amigobot.com/.

[55] phpMyAdmin project in the SourceForge.net. [Online]http://www.phpmyadmin.net/.

[56] The Parasol lab, "Vizmo." [Online]http://parasol.tamu.edu/groups/amatogroup/intranet/vizmo++/.

[57] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*. New York: Cambridge University Press, 1998.

[58] T. Nakamura, S. Takamura, and M. Asada, "Behavior-based map representation for a sonar-based mobile robot by statistical methods," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pp. 276–283, 1996.

[59] The Parasol Lab, Department of Computer Science, Texas A&M University, College Station, TX [Online]http://parasol.tamu.edu/.

[60] M. Rekleitis, G. Dudek, and E. Milios, "Multi-robot cooperative localization: a study of trade-offs between efficiency and accuracy," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 3, pp. 2690 –2695, 2002.

[61] S. Roumeliotis, "Robust mobile robot localization: from single-robot uncertainties to multi-robot interdependencies," Ph.D. dissertation, University of Southern California, Los Angeles, CA, 2000.

[62] S. Roumeliotis and G. Bekey, "Collective localization: a distributed kalman filter approach to localization of groups of mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 3, pp. 2958 – 2965, 2000.

[63] S. Roumeliotis and M. Rekleitis, "Analysis of multirobot localization uncertainty propagation," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 2, pp. 1763 – 1770, 2003.

[64] A. C. Schultz and W. Adams, "Continuous localization using evidence grids," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 4, pp. 2833–2839, 1998.

[65] D. Schulz, W. Burgard, D. Fox, and A. Cremers, "Tracking multiple moving targets with a mobile robot using particle filters and statistical data association,"

in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 1665–1670, 2001.

[66] Z. Shiller, K. Yamane, and Y. Nakamura, "Planning motion patterns of human figures using a multi-layered grid and the dynamics filter," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1, pp. 1–8, 2001.

[67] J. Shwartz, *Planning, Geometry, and Complexity of Robot Motion*. Norwood, NJ: Ablex Publishing Corporation, 1989.

[68] E. B. Smith and R. Langari, "Fuzzy multiobjective decision making for navigation of mobile robots in dynamic, unstructured environments," in *Journal of intelligent and Fuzzy Systems*, vol. 14, 2003. Issue 2, to appear.

[69] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *Int. J. Robot. Res.*, vol. 5, no. 4, pp. 56–88, 1986.

[70] G. Song, S. L. Miller, and N. M. Amato, "Customizing PRM roadmaps at query time," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1500–1505, 2001.

[71] J. Spletzer, A. Das, R. Fierro, C. Taylor, V. Kumar, and J. Ostrowski, "Cooperative localization and control for multi-robot manipulation," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pp. 631–636, 2001.

[72] J. Spletzer and C. Taylor, "Analysis of multirobot localization uncertainty propagation," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 2, pp. 1258 – 1265, 2003.

[73] C. Sung and W. Wong, "A graph theoretic approach to the channel assignment problem in cellular systems," *Vehicular Technology Conference, IEEE*, vol. 2, pp. 604–608, 1995.

[74] S. Thrun and W. Burgard, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Machine Learning and Autonomous Robots (joint issue)*, vol. 31, no. 5, pp. 1–25, 1998.

[75] Y. Tonouchi, T. Tsubouchi, and S. Arimoto, "Fusion of dead-reckoning positions with a workspace model for a mobile robot by bayesian inference," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 2, pp. 1347–1354, 1994.

[76] Trolltech, "Qt." [Online]http://www.trolltech.com/products/qt/.

[77] Y. Wang and D. Lane, "Solving a generalized constrainted optimization problem with both logic AND and OR relationships by mathematical transformation and its application to robot motion planning," in *IEEE transactions on systems, man and and cybernetics, part C*, pp. 525–536, 2000.

[78] A. Wigderson, "Improving the performance guarantee for approximate graph coloring," *Journal of the ACM*, vol. 30, pp. 729–735, 1983.

[79] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 1024–1031, 1999.

[80] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "Motion planning for a rigid body using random networks on the medial axis of the free space," in *Proc. ACM Symp. on Computational Geometry (SoCG)*, pp. 173–180, 1999.

[81] S. A. Wilmarth, "A probabilistic method for rigid body motion planning using sampling from the medial axis of the free space," Ph.D. dissertation, Texas A&M University, College Station, TX, 1999.

[82] J. Wolf, W. Burgard, and H. Burkhardt, "Robust vision-based localization for mobile robots using an image retrieval system based on invariant features," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1, pp. 359–365, 2002.

[83] Y. Wu and N. M. Amato, "A randomized roadmap method for path and manipulation planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1, pp. 113–120, 1996.

[84] M. Yamamoto, M. Iwamura, and A. Mohri, "Quasi-time-optimal motion planning of mobile platforms in the presence of obstacles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 4, pp. 2958—2963, 1999.

[85] B. Yamauchi, "Mobile robot localization in dynamic environments using dead reckoning and evidence grids," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1401–1406, 1996.

[86] Z. Yi, H. Khing, C. Seng, and Z. Wei, "Multi-ultrasonic sensor fusion for multiple robots," in *IEEE Intelligent Vehicles Symposium*, pp. 387–391, 2000.

[87] U. Zwick, R. Nathaniel, and E. Halperin, "Coloring k-colorable graphs using smaller palettes," *Symposium on Discrete Algorithms*, pp. 319–326, 2001.

APPENDIX A

DATABASE FOR CAMPUS NAVIGATOR

The database tables listed below serve three purposes in the campus navigator: to represent the roadmap, to serve as queues for user requests, and to maintain campus-query and campus-vizmo servers running simultaneously.

- bike: if bikes can move on the edge, edge ID and time of travel

field	type
eid	int(10)
time	mediumint(8)

- building: the building's full name, abbreviation, and building number

field	type
id	int(10)
number	smallint(6)
abbrv	varchar(10)
name	varchar(50)

- busEdge: if there is a bus route on the edge, time and direction

field	type
route_id	int(10)
eid	int(10)
time	mediumint(9)
if_directed	char(4)

- busRouteStopMapping: bus stop ID for bus routes

field	type
bus_route_id	int(10)
bus_stop_id	int(10)

- busRoutes: name for bus routes

field	type
id	smallint(6)
name	varchar(40)
number_buses	smallint(6)

- busStop: name for bus stops

field	type
id	int(10)
name	varchar(50)

- car: if cars can move on the edge, time and direction

field	type
route_id	int(10)
eid	int(10)
time	mediumint(9)
if_directed	char(4)

- deptServices: department (type is 'D') or service (type is 'S') for some buildings

field	type
id	int(10)
name	varchar(50)
building_number	smallint(6)
type	char(1)

- edge: start and end vertices, direction, and length

field	type
eid	int(10)
street_id	smallint(10)
start_vid	int(10)
end_vid	int(10)
distance	double

- motorcycle: if motorcycles can move on the edge, time and direction

field	type
eid	int(10)
time	mediumint(8)
if_directed	char(4)

- parkingLot: name and number of parking lots

field	type
id	int(10)
name	varchar(50)
number	varchar(10)

- path: result of query (a series of robot configurations in text format), campus-vizmo snapshot image name, camera zoom, and camera move information

field	type
id	int(11)
path	text
image_name	varchar(100)
zoom	int(11)
move_ud	int(11)
move_lr	int(11)
time	timestamp(14)

- pedestrian: if (handicapped) people can walk on the edge, time of moving

field	type
eid	int(10)
time	mediumint(8)
wheelchair	char(4)

- placeType: valid places (building, parking lot, department, bus stop)

field	type
place_type_id	int(10)
table_name	varchar(50)
type_name	varchar(50)

- query: user query queue generated by the web interface and consumed by campus-query, some information sent to campus-vizmo.

field	type
id	int(10)
time	timestamp(14)
start_vid	int(11)
goal_vid	int(11)
start_method	int(11)
goal_method	int(11)
park_vid	int(11)
park_method	char(1)
camera_zoom	int(11)
camera_move_ud	int(11)
camera_move_lr	int(11)
image_name	varchar(100)
query_server_id	int(11)

- queryServers: query server(s) running over NFS (network file server system)

field	type
id	int(11)
start_time	timestamp(14)
campus_dir	char(100)

- street: name of streets

field	type
id	smallint(6)
name	varchar(35)

- textGuide: the text guide generated by the path planner and used by the web interface

field	type
id	int(11)
guide	text
image_name	varchar(100)

- vertex: coordinates of vertices

field	type
vid	int(10)
x	int(11)
y	int(11)

- vertexPlaceMapping: place(s) for vertices defined in the table placeType

field	type
vid	int(10)
type_id	int(10)
place_id	int(10)

- vizmoServers: visualization server(s) running over NFS

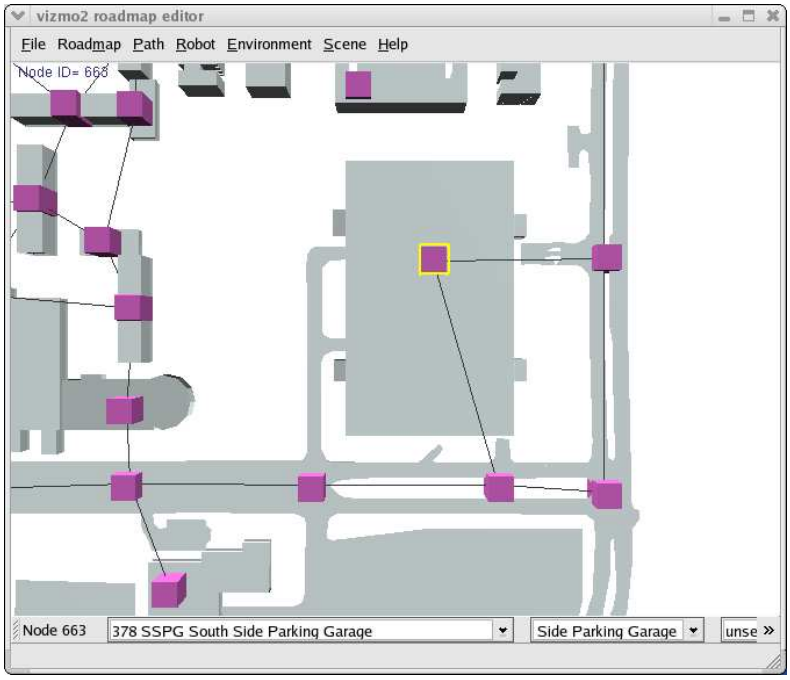
field	type
id	int(11)
start_time	timestamp(14)
campus_dir	char(100)

APPENDIX B

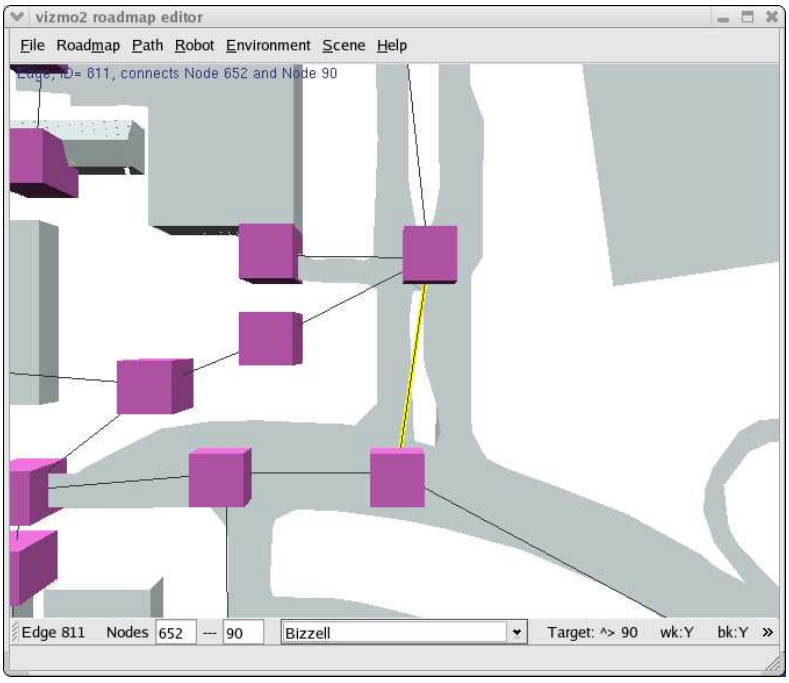
DIALOGS IN CAMPUS ROADMAP EDITOR

The roadmap editor for campus navigator has been designed for intuitive and easy use. When a vertex or an edge in the roadmap is clicked, the corresponding user interface items appear at the bottom of the screen. In Figure 106(a), the selected vertex is the South Side Parking Garage building. Other user interface items not shown in the figure are bus stops and departments/services. In Figure 106(b), the selected edge is Bizzel street and walk (marked as wk: in the figure) and bike (bk:) are enabled (marked as Y). Other items not shown in the figure are car, motorcycle, bus, and wheelchair.

Figure 107(a) shows a list box which pops up if the Bus Routes button is pressed. In the figure, the selected vertex has three bus stops (Cotton Bowl, Replant, Yell Practice). Similarly, Figure 107(b) shows that the selected edge (marked by yellow lines and not seen in the gray scale image) has Replant bus route in it. Note that the mark $< \wedge$ indicates the direction of the bus route at the edge, which is northwest.

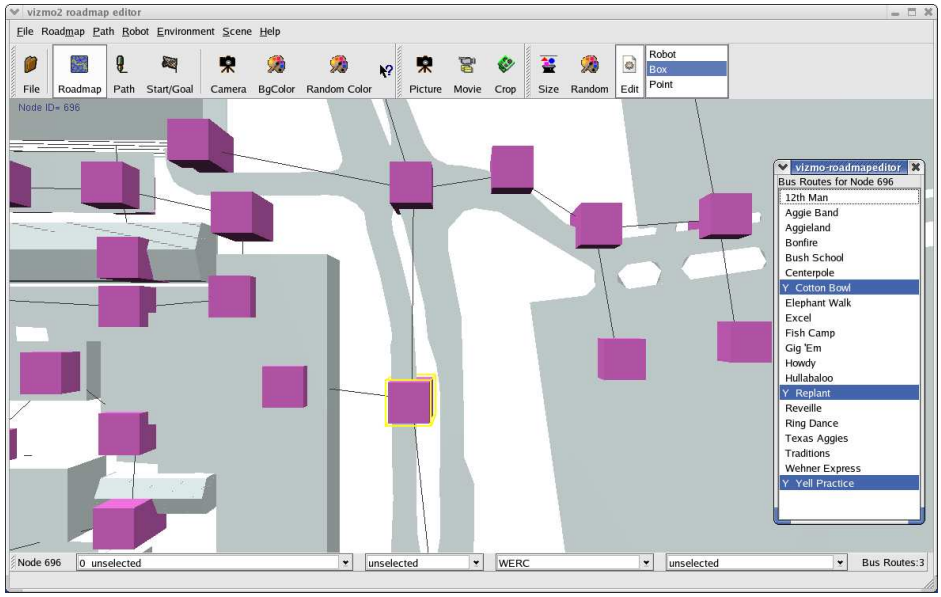


(a)

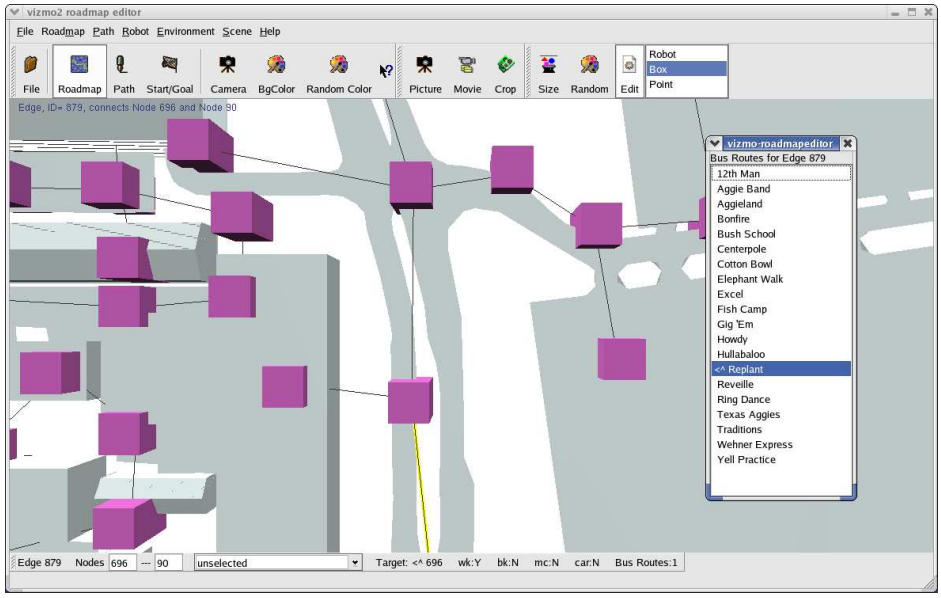


(b)

Fig. 106. Roadmap editor: (a) editing building and parking lot, and (b) editing street.



(a)



(b)

Fig. 107. Roadmap editor: (a) a vertex is selected, and (b) an edge is selected

VITA

Jinsuck Kim was born on October 14th, 1971, in South Korea. Early childhood and public school years were spent in Yoido, Seoul. His current permanent address is Kangnam Sooseo 736 Shindonga Apt. 708-1208 Seoul, Korea. He graduated from Inha University in 1996 and received a B.S. in aerospace engineering. After that, he moved to the United States for graduate study. Under the advice of Dr. Martin Corless at Purdue University, he received his master's degree in the area of dynamics and control in August, 1998. In pursuit of a Ph.D. degree, he transferred to the Department of Aerospace Engineering at Texas A&M University. He spent about a year working with Dr. John Crassidis on a project funded by NASA. Since the year 2000, he has been a student and graduate research assistant in the Department of Computer Science.