Mobile Robot Localization Using Model-Based Maps

Paul R. MacKenzie

B.A.Sc., (University of Waterloo), 1991

Department of Electrical Engineering

McGill University

Montréal

May, 1994

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Engineering

Abstract

This thesis describes the coupled tasks of using a mobile robot to construct a map with noisy range sensors (sonar) of an initially unfamiliar environment, and using such a map to determine the robot's position and orientation. The map need not necessarily represent the actual spatial structure of the environment so much as it is meant to represent the major structural components of what the robot perceives. These "features" of the environment are modeled as straight line segments and are assembled together to form a map. One problem with this approach is that maintaining an absolute coordinate system for the map is difficult without periodically calibrating the position and orientation of the robot, due to the unbounded accumulation of positional errors as the robot moves and rotates.

An approach to calibrating the robot's position, known as *localization*, is presented. In suitable environments, it is possible to use sonar data to perform this operation given coarse estimates of position and orientation, which are iteratively refined to high accuracy using the map and a set of sonar measurements from a single position. Provisions are made for verification of the results of localization using *quality indicators*, which give a measure of the confidence in the accuracy of a refined position estimate.

The approach is then generalized to allow global localization, where position and orientation estimates are not available.

Using a number of sample environments, experimental results show that in spite of the inherent noisiness of sonar sensors, accurate localization of a mobile robot is achievable. In addition, the constructed maps are general enough to be used for purposes other than localization, such as path planning and collision avoidance.

Résumé

Cette thèse aborde deux problèmes reliées à la localisation d'un robot mobile. Le premier concerne la construction de cartes d'environnement inconnu à l'aide de détecteurs ultrasoniques, et le second concerne la détermination de la position et de l'orientation du robot à partir de ces cartes. La représentation cartographique ne correspond pas nécessairement à la structure spatialle exacte de l'environnement, mais plûtot à une représentation des composantes majeures de structure telles que percues par le robot. La modalisation de ces composantes se fait à l'aide de segments de droite, et la conjonction de ces modèles forme la carte de l'environment. Dû à l'accumulation d'erreurs de position et d'orientation du robot, un des problèmes rencontré avec cette approche est le maintien d'un système de coordonnées cartographiques absolu sans calibration périodique de la position et de l'orientation.

Comme méthode de calibration, le procédé de *localisation* est utilisé. Dans des environnements convenables, des données ultrasoniques servent à établir la localisation à partir d'une estimation de la position et l'orientation du robot. Cette estimation est rafinée de façon itérative jusqu'à haute précision à l'aide de données et d'une carte créée à priori. Pour vérifier le résultat de localisation, un *indicateur de qualité*, qui donne une mesure du niveau de confiance en la précision de l'estimation, est défini.

Cette approche de localisation se généralise pour permettre une détermination de position et d'orientation lorsqu'aucune estimation est disponible.

A partir d'un certain nombre d'environnements d'essai, les expériences démontrnt la possibilité d'obtenir une localisation précise malgré la présence inhérente de bruits dans les données ultrasoniques. Aussi, la généralité des cartes construites pour le procédé de localisation permet leurs utilisation pour d'autres fins tels que l'appréhension de collision et la planification de trajectoire.

Acknowledgements

I wish to wholeheartedly thank my supervisor Gregory Dudek for his assistance, guidance, unscheduled discussions and brainstorming sessions throughout the realization of my Master's degree. Dr. Dudek has played an active role in my research from day one, and I truly appreciate his time, effort and advice not only in the preparation of this thesis, but in all aspects of academic life. I am also indebted for the use of his mobile robot simulation and control program, "Daemon", and his uncompromising willingness to correct the occasional bug therein...

I also wish to thank Marc Bolduc and Claudia Pateras for their help with the proofreading of this thesis. Between the two of them I think I received excellent advice on some of the finer points of thesis writing.

This research has been made possible by a scholarship from the Natural Sciences and Engineering Research Council, to whom I wish to express my gratitude. I am also grateful to Canadian astronaut Stephen Maclean, who advised me that McGill University offers one of the best research atmospheres in Canada, and he was right.

Finally, I would like to thank my parents, Margaret and Albert Eugene, for their love and support during my Master's degree and throughout my entire university career – thanks folks!

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

One of the common themes of many mobile robot systems is the use of some form of map for navigating within an environment. This is especially true of mobile robots demonstrating at least some minimal amount of autonomy in their tasks. Applications such as path planning and collision avoidance are two examples of tasks that require that the robot explicitly model its environment in some way. In fact, the major difficulty in autonomous navigation is not so much the tasks (such as the two mentioned above), but rather the extraction of useful information from sensor data and its relationship with a navigation map [23].

An important subproblem embedded in navigation is that of self-localization. Simply put, localization answers the robot's question "where am I?" and usually must be solved before proceeding with higher level operations such as path planning. In cases where a range sensing device is mounted on board a mobile robot, measurements taken with this sensor will usually be relative to the robot. In order to maintain any kind of global coordinate system within a map of the environment, the location of the robot must be known before any of the objects sensed can be correctly placed on the map. In most cases, the location of the robot consists of the position of the robot in space and the robot's orientation. This information, referred to in this thesis as the robot's *pose* [14, 31] (also known as the robot's *configuration* [6]) is what must be correct in order for accurate placement of sensed objects within a global coordinate system[1]. For clarification, the term *pose* will refer to the position together with the orientation of a robot in a global coordinate system, i.e. one would say *pose* $= (x, y, \theta)$, while *position* $= (x, y)$ and *orientation* $= \theta$.

A preliminary question is: why are sensors needed to perform localization? Given an initial pose of a mobile robot, its current pose should be calculable by integrat-

[1]Since the location of a sensed object is relative to the assumed pose of the robot, any error in the estimate of robot pose is also present in the position of the sensed object

ing the robot's motion history, assuming this is known. This process is commonly known as *dead reckoning.* The problem with dead reckoning is that small errors in the integration of orientation, distance and velocity will accumulate progressively as a mobile robot moves through its environment (errors such as those brought about by the problems of wheel slippage, uneven ground, or unexpected collisions) unless corrected. Uncorrected errors are integrated over time along with the velocity history and therefore errors in absolute pose accumulate disastrously with successive motions of the robot. Eventually, the robot's estimate of its own pose will be drastically different from its true pose, and therefore it would need to be periodically or continuously *recalibrated* in order to minimize this error.

1.1 The Approach

Given the necessity of performing localization, we need to know what kind of map is required to perform it. At a first glance, a complete and accurate *a priori* map of the environment would appear appropriate in many domains. However, such accurate metric maps are rarely available, nor is it always easy to fabricate them. More importantly, even when metric maps (for example, architects' floorplans) are available in a usable form, they tend to not portray the environment in a fashion consistent with typical robotic sensing devices. For instance, commonly-used sonar devices fail to detect many existing structures (like overhangs or thin cylinders) and may "detect" many structures that are not physically present (such as illusionary walls in corners and other structures due to multiple reflections of the sonar beam). For these reasons, it would be most valuable if a mobile robot could construct and maintain a map of an environment in terms of its own perceptual mechanisms. In simpler terms, the robot should be able to build the map of the environment *as it "sees" it*, not as human beings may see it. Considering that the robot must be guided by its own sensors, this is not an unreasonable supposition.

Before localization can be accomplished, at least a *partial* map of the environment is required to which new range measurements are compared[2]. Therefore, the use of localization as described here assumes that the robot has previously entered an unfamiliar environment and has constructed a map using the map construction techniques described in this thesis. The measurements taken by the robot's range sensor show the assumed positions of objects in the environment based on the sensor's present pose estimate and are compared to the map. At this point the discrepencies between the sensor data and the map indicate the error in the estimated pose. The term *range data point* refers to a point in space where the range sensor has observed an object (based on the range of the object from the sensor and the pose of the sensor), and this term is used throughout this thesis.

The localization techniques in this thesis are based on refining a coarse estimate of pose. This *local* localization assumes that such an estimate is available but contains errors, such as an estimate obtained from dead reckoning or another coarse localization technique. The error in the pose estimate for which local localization functions correctly is bounded, with the actual upper bound being determined by the environment and the robot's place within it – in general, the local localization technique discussed in this thesis cannot correct a coarse pose estimate whose error is too large.

Local localization is in contrast to *global* localization, where no a priori estimate of pose is provided by the user, and the robot must rely only on its sensors and its map. As is shown in this thesis, global methods can be based on local ones. Locally convergent pose estimation (*local* localization) is further decomposed into position correction and orientation correction. All the components of localization are fully explained in their appropriate sections.

[2]The word partial is used because a complete map showing all the details of an area is not necessarily required. As long as there is enough correspondence between visible features in the environment and what was seen at the last known location, it is possible for localization to be performed. Whether a representation of the environment with this kind of minimum criterion can be called a map is debatable.

1.2 Thesis Outline

This thesis begins by describing the task of map construction, where objects in the world are modeled in a way conducive to localization. Once the models and the methods of obtaining them are described, the methods of localization are discussed, beginning with position correction. The results of position correction then lead to the development of *quality measures*, with which verification of these results can be done. These measures also allow for orientation correction and global localization, and are discussed on this basis.

The outline of this thesis follows:

Chapter 2 discusses some related work that has been done in the area of mobile robot map construction and localization. The three basic approaches to localization are discussed with examples from the literature.

Chapter 3 gives a brief introduction to the robot used in the experiments done for this thesis, and how range data are gathered.

Chapter 4 introduces the approach to map construction. The methods used for the clustering of point data, line fitting, splitting and merging are presented.

Chapter 5 begins the discussion of localization by considering correction of the error only in the position of the robot. An iterative weighted sum of vectors approach is discussed, as well as issues such as the handling of errors and of convergence. Some illustrative results are shown.

Chapter 6 discusses how one may measure the quality or confidence of a particular pose estimate. The two basic types of measures, the least-squares measure and a neighbourhood occupancy measure are presented, as well as third measure that is a combination of the first two. Some examples using real data are given.

Chapter 7 continues the discussion of the measures of quality by demonstrating how these measures allow correction of orientation in addition to position to give full pose localization. The problem of global localization (where no initial pose estimate is given) is also addressed here. Both are formulated in terms of the optimization of non-linear functions of the quality measures.

Chapter 8 presents experimental results of the approaches developed in the previous chapters as applied to one real and two simulated environments. This is in addition to the example real environment in earlier chapters which was used to illustrate the various aspects of the approach. Quality measures, regions of convergence and global localization results are presented for each sample environment.

Chapter 9 concludes the thesis with a discussion of the topics presented regarding map construction and localization, together with a summary of the findings obtained from experiments.

Chapter 2 Related Work

Work in the areas of mobile robot map construction and localization generally began in the early 1980s, and since then there have been several proposed approaches to these problems. This chapter presents a background to the localization issue: map making solely for the purposes of path planning, collision avoidance, etc. will not be the focus.

There are basically three approaches to localization:

1. active localization
2. passive localization
3. integration of past kinetic history

Active localization refers to the approach where beacons are placed in the environment at known locations so that a mobile robot may receive transmissions from them, and thus calculate its global position based on its position relative to the beacons. This has been done using both ultrasonic beacons [19] and infra-red beacons [18]. The Global Positioning System (GPS) of the United States is also an example of active localization, but using satellites as beacons. The main criticism of this approach is that while robot positioning may be achieved, it is necessary to modify the environment in order to do so. For situations where it is desired to introduce a mobile robot to an unknown environment, it may not be possible to use this approach, at least as far as *exploration* is concerned. In situations where the environment is relatively stable (such as a warehouse floor), this approach may suffice.

Passive localization is the approach where the environment is not modified, and the onus is on the robot to scan the environment to determine its position, usually with the aid of a map. The term "passive" does not refer to the sensor: sonar, laser and infra-red range finders are active sensors, but the environment is not "actively" aiding the robot. This is by far the most popular approach, since the use of environment maps is useful for other tasks in addition to localization, and various inexpensive range sensing devices are available on the market. The various facets to this general approach is covered in more detail in their own sections in this chapter.

The third approach refers to using the past movement history of the robot to calculate its present position (assuming of course that its initial position was known). The simplest method is commonly known as *dead reckoning*. This is a relatively simple book-keeping operation where, for example, the shaft-markings found on most wheeled robots are counted as the robot moves and rotates. The problem with using this alone is that errors accumulate without bound as the robot continues to move. Wheel slippage, gear backlash, the use of finite-precision arithmetic are a few of the kinds of errors that affect dead-reckoning. From time to time another localization method must be used to reset the robot to a correct position, which, by some remarkable coincidence, is the subject of this thesis. How often this must be done depends on the degree of error the robot accumulates as it moves.

A step up from the distance/angle monitoring of dead-reckoning are the systems that use inertial data [34]. These systems have the potential to be quite accurate, even in less structured environments than indoor situations. Like dead-reckoning, the error in the position estimate still approaches infinity in the limit, but in this case much more slowly. In general these systems tend to be very expensive, but in the future the costs of accelerometers and gyroscopes and the like may come down to the point of making this approach cost competitive.

2.1 Passive Localization Approaches

2.1.1 Grid-Based Techniques

Grid-based techniques involve the construction of a two-dimensional and in some cases a three-dimensional grid which covers the environment in which the robot may move. Moravec and Elfes did early work on the use of certainty or *occupancy* grids [27, 12], which took a probabilistic approach to whether a given cell in the environment grid contains an obstacle or free space. Lim and Cho did further work on extending this idea to accurate sonar modeling [3].

Any obstacle or part of any obstacle perceived within a cell is assumed to be present within the entire cell; therefore, one concern with this approach is the size of the grid. For maps representing large areas, a small cell size would preserve detail but would increase the memory requirements of the map. Large cell size decreases memory but also decreases usable free space due to the larger neighbourhoods surrounding obstacles. Kuc and Barshan use a cell size based on an estimate of how far the robot can move without a collision [20], but in general there is no rule governing cell size.

Grid methods commonly perform localization by directly comparing a global or known occupancy grid map with the local grid map containing newly acquired data, using a cross correlation data matching technique. The accuracy of this method is limited in that spurious data is weighted equally with good data, since all cells are used in the correlation. Another concern is that the amount of data stored is proportional to the size of the cell array, and it follows that the speed of a matching algorithm would be affected by an environment whose cell size is small.

Grids can also be used in conjunction with other techniques to reduce the search space involved. Gonzalez et al. use a feature-based approach but overlay a grid to reduce the number of feature models compared during matching [14].

2.1.2 Feature-Based Approaches

These approaches all involve the use of a map of the world that contains features usable for localization. There are two primary varieties:

Feature-Matching: features are extracted from sensed data and matched to features in the map

Data-to-Model Error Minimization: the discrepancy between raw sensed data and map features is minimized

Gonzalez et al. refer to these approaches as *feature-based* and *iconic* methods respectively [14].

The features of the world map may be provided a priori by the user or may be "discovered" by the robot as it explores the world. There are numerous examples in the literature of the first instance [5, 13, 4, 22, 23, 6, 17], but on-line robot map construction for the purpose of localization is no stranger to the scene either [24, 14, 10].

The feature-matching techniques, as stated, involve extracting features from newly acquired sensor data and matching them to known features in a map. Drumheller [6] extracts straight line segments from a sonar contour (a 360° sonar sensor sweep) and attempts to eliminate implausible combinations of line segments using a number of geometric constraints. Holenstein *et al* extract features from a sonar contour in a similar fashion, but instead of a best fit approach they compare each pair of extracted line segment models to all pairs of reference objects (for instance, the walls on a map) and calculate all possible geometric transformations that would match the two pairs. The transformations for all pairs of line segments are then clustered in (x,y,θ) space and the coordinates of the largest cluster is taken to be the true robot pose [17]. Fennema *et al* attempt to minimize a quadratic error model, which is based on the distances between models and extracted features [13]. A concern inherent to all three of these approaches is that features must be extracted from a single scan of the environment, and to "ensure" that these features are "well seen", dense data scans must be taken[1]. In addition, if a particular sensor scan of the environment yields data but no discernible features, then the matching will not be reliable.

Localization in vision-guided robots has also been tackled by feature-matching. Roth *et al* match 2D image features to 3D model features and apply a global consistency check for verification [31]. Nashashibi and Devy use a laser range finder to extract 3D planar faces and then match them to a 3D model of the environment, which is then applied to a generalized Kalman filter [28].

The data-to-model error minimization or *iconic* approach involves minimizing some error measure which is a function of the individual distances between the raw

[1]In this context, scan density refers to the angular separation of adjacent measurements, so making one scan every 3° is more dense than one scan every 12°.

sensor data and the stored environment map. Raw sensor observations are commonly paired with models representing the object(s) in the environment from which the readings were obtained. Similar to the approach taken in this thesis, Cox [4] paired single range data points to the line segment model closest to them in the Euclidean sense. He then linearized a non-linear equation about an estimate of the robot's position and used a closed form least-squares linear regression to find a solution. Gonzalez, Stentz and Ollero used the same minimum distance criteria to pair their laser range data to line segment models, but used an iterative algorithm to find a Jacobian matrix from which a least-squares fit of robot position and orientation could be found [14].

Leonard, Durrant-Whyte and Cox have developed a system which formalizes the localization process as a vehicle-tracking problem [23, 22, 24]. Their approach belongs in the *feature-matching* category since their observations take the form of extracted regions of constant depth (RCDs - these are angular sectors in a dense 360° sensor scan around the robot whose distances from the robot are constant over the sector) from single scans rather than the raw sensed data points. The authors preferred these RCD observations over straight lines or raw data because they claim RCDs agree more with the properties of ideal sonar data [23]. While this is most certainly true, these properties manifest themselves only to a significant degree in specular environments. In real-world indoor environments, most objects are not ideally specular, and thus RCDs have a radius of curvature large enough to be well approximated by a straight line. In addition, truly non-specular planar objects do not visibly exhibit the RCD property; rather, we have found that these objects are well modeled by straight line segments having linear depth variations.

Their system included a sensor model to predict observations; focusing on the regions neighboring these observations helps to reduce focus on probable object locations, reducing the search space and helping to avoid spurious data. Once the sensor model predicts an observation, an extended Kalman filter is employed to track the observed object as the robot moves through the environment. The authors considered the cases of planes, corners and cylinders as models (to which they refer to as *geometric beacons* for use with sonar sensors. Map construction was also considered in the light of localization [24] where they used a model of the sensor to predict range measurements, and verified them as a function of time. Observations agreeing with predictions were classified as "expected"; any others were "unexpected". By establishing a confidence measure for each geometric beacon (environment feature), only those of high confidence were used for localization. Confidence would be increased or decreased depending on whether the beacon was expected or unexpected by the predicted measurements made using the sensor model. It is worthwhile to note that this beacon confidence approach is independent of the specific approach to localization, as well as how the beacons were initially extracted. Therefore, as long as an accurate sonar sensor model is available, i.e. one whose measurement predictions match those of the real sensor, it is possible to use this confidence approach without the requirement of their particular localization algorithm.

A more recent approach to localization has been examined by Dudek and Zhang [11]. This approach involves training a neural network with raw sensor data in order to associate a given observation directly with a known position. Once so trained, this method is advantageous in that no formal map is required, and therefore no models of the environment need to be devised. Since the choice of the models of environmental features often limits the environment in which a particular system may be used, the neural network approach is less bound by this constraint. The problems with this approach are that training the neural network may be very time consuming compared to other methods, and that the lack of a map precludes other tasks being performed with the same data, such as path planning, obstacle avoidance, and others that require a map.

Chapter 3 The Robot

The robot used for experimentation in this thesis is *Pollux*, a three-wheeled cylindrical mobile robot with a 12-transducer TOF (Time of Flight) sonar ring evenly spaced around it (figure 3.1). The sonar ring rotates with the wheels so that the same transducer always faces the forward direction. Each transducer acts both as transmitter and receiver. In the conventional TOF system, such as the Polaroid system [2], a

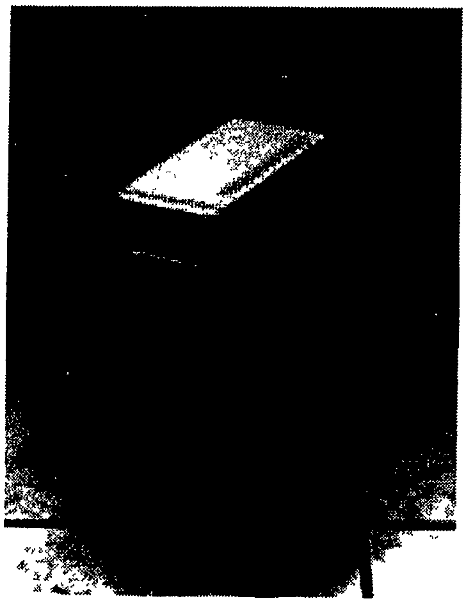


Figure 3.1: Pollux: a mobile robot

transducer sends out a pulse and receives an echo. The time delay t_{delay} between sending the pulse and receiving the first echo is converted to a distance measure d using the simple relation:

$$d = \frac{ct_{delay}}{2} \tag{3.1}$$

where c is the speed of sound in air. However, the beam of the pulse sent out by the transducer spreads out into a cone as it travels [9, 20, 36], and so any returned echo is due to an object somewhere within this cone. Since a single point per scan is desired rather than a range of possible points, the point at the axis of the cone is chosen as the most likely position for the object (figure 3.2) This minimizes the error

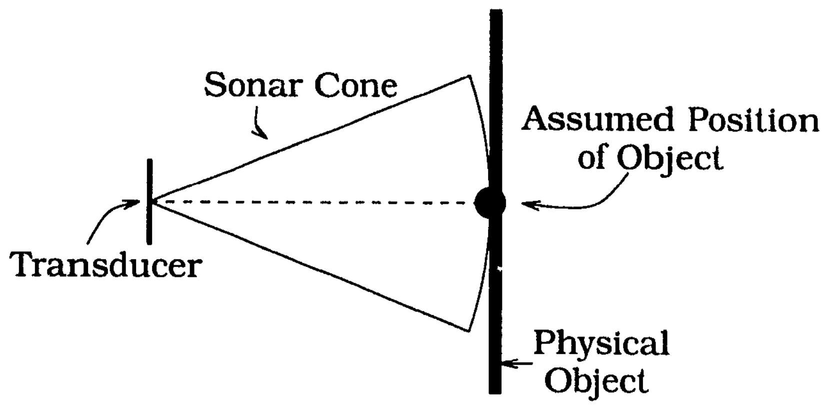


Figure 3.2: Finding the Most Likely Position of an Object within the Sonar Cone

of any guess of position within the cone.

Since the arrangement of transducers around the robot is fixed with respect to the forward direction of the robot, we therefore know the orientation of each transducer. For scans more dense than the physical transducer separation, the robot can rotate slightly to obtain denser measurements. If we consider the robot's local coordinate system in polar coordinates, then all the sonar echos received have coordinates (r, θ), with $\theta = 0°$ being the forward direction (figure 3.3). If we know the robot's correct pose (recall pose $\equiv$ position and orientation) with respect to the global coordinate system, then it is simple geometry to find the global coordinates of all the sonar data points. Once again, we see the importance of knowing where the robot is located. Errors in pose estimates translate directly into errors in the locations of objects in the environment.

Figure 3.4 illustrates a type of "map" based solely on the raw range data from the sonar sensors. As can be seen from figure 3.4, the general shape of objects (in this case, walls and other planar surfaces) can be distinguished. However, this task done so naturally by humans is difficult to formulate into a usable algorithm for the robot – if it were easy, some general shape problems in machine vision could be solved. As far as the robot is concerned, it only has a large collection of individual range data

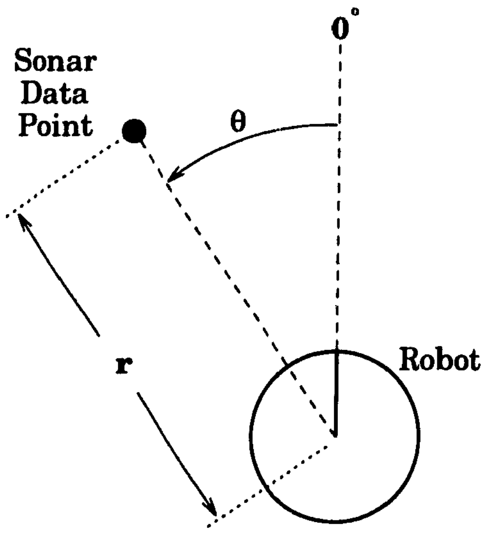


Figure 3.3: Sensed object placement is at (r,θ) relative to the robot

points. Certainly storing every sonar point is not the best way to build a useful map – while human beings can discern the presence of objects, the robot cannot and would be limited in its capabilities if this type of map were its only resource.

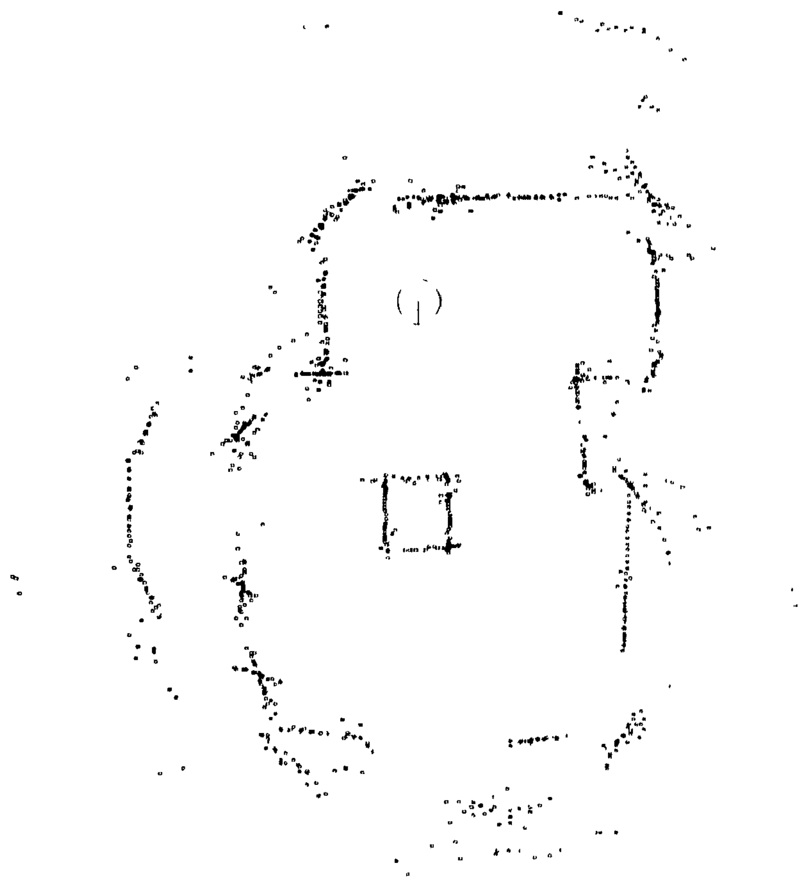


Figure 3.4: An Example of an Accumulation of Sonar Range Data: this is an overhead view of a sample environment, where each dot represents the location of a response within the environment (in global coordinates) from the robot's sonar sensors. The circle within the map represents the robot, and the line within it indicates the robot's orientation.

Chapter 4 Map Construction

4.1 The Approach

The mapping approach described here belongs to that class of approaches where maps are composed of models used to represent discernable features of the environment. Two-dimensional modeling only will be considered. This is based on the assumption of a small robot with fixed height sensors (true for all experiments done in this thesis, since Pollux was the robot used). In many indoor, office environments two-dimensional modeling is valid because the commonly encountered objects such as doors, walls, chairs, desks, etc. with planar surfaces may be considered as two dimensional but extended into the height dimension.

One could use different models for the various office-type obstacles likely to be encountered [23] but it was decided to attempt to build maps using linear models only. Since we are operating in an environment where objects are assumed to be made of planar surfaces perpendicular to the floor, the constant height of the sensor assumption validates this linear model choice. Each model would thus consist of a line segment in space, and could be thought of as representing a section of a wall or other obstacle although, in fact, some linearly-shaped clusters of observations may not correspond directly to existing structures. For instance, some sonar data may form linear clusters when the sensor observes a corner between two walls, due to the effects of multiple reflections of the sound waves.

As illustrated in figure 4.1, modeling with line segments agrees well with the threshold-based sonars, where each measurement corresponds to the first over-threshold response for a brief ultrasonic "chirp". By examining the characteristics of sonar [8, 2, 9] we can roughly describe each outgoing chirp as a 12 steradian measurement cone and the first object of sufficient size within this cone results in a single response at that object's distance. Consequently, a similar orientation that also includes the same object will return the same measurement (unless it hits a closer object). As a result, even a small object will produce a collection of measurements at similar distances that are nearly linear in structure [10] (figure 4.1). In spite of this difficulty

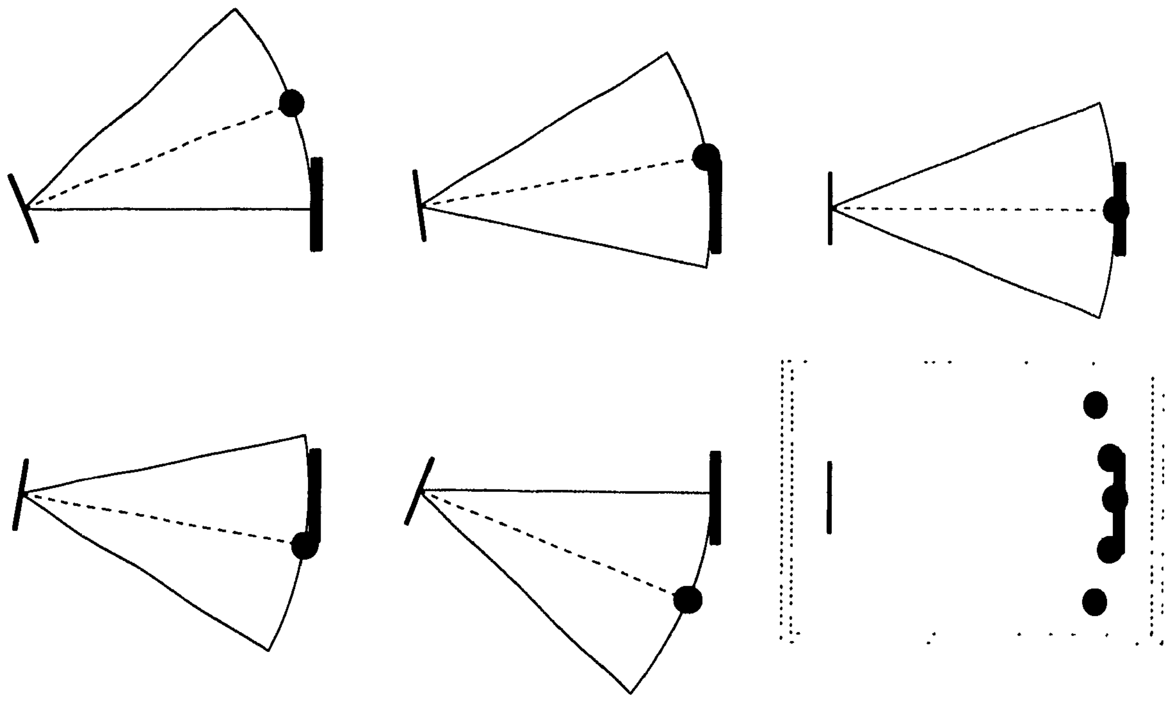


Figure 4.1: A Dense Scan of a Small Object

with sonar, using simple line-segment models can build very useful maps, especially for the localization aspects that will be discussed in later chapters.

Extracting line-segment models from raw sensor data can be accomplished using one of the many line-fitting algorithms available, and one such approach is explained in this chapter.

4.2 Clustering

Clustering is the first step in fitting line-segment models to sensor data. Here the data corresponding to presumably distinct objects in the environment are separated to facilitate line fitting. Since we require separate models for separate objects, we can exploit the fact that disjoint data point sets correspond to separate objects by dividing the data into groups *before* line fitting. This allows fitting to be done one object at a time, and avoids fitting a single line segment to a group of distinct objects.

4.2.1 The Sphere-of-Influence Graph

A *neighbourhood graph* may be defined as a set of connections or *edges* between points in space. The sphere-of-influence graph, as proposed by Toussaint [32] is a neighborhood graph [33] that can be applied to any finite set of unordered points in a plane. It attempts to capture the essence of a *primal sketch* for dot patterns of arbitrary complexity. This graph has an interesting feature in that it may consist of either a connected graph or a collection of disconnected pieces where appropriate, and its algorithm does not require any tuning of parameters or thresholds. It is precisely this characteristic of determining when and where to form disconnected pieces that makes the sphere-of-influence graph so useful for clustering for the line fitting application.

The definition of the sphere-of-influence graph is as follows [32] (see Figure 4.2):

Definition 1 *Let $S = \{p_1, p_2, \ldots, p_n\}$ be a finite set of points in a plane. For each point $p_i \epsilon S$, let R_i be the distance to the nearest neighbour of p_i, (i.e. the closest point to p_i) and let C_i be the circle of radius R_i centred at p_i. The sphere-of-influence graph is a graph on S with an edge between points p_i and p_j $(i \neq j)$ if and only if the circles C_i and C_j intersect in at least two places.*

To relate this to clustering, we can say that if any two points share an edge of the graph then they belong in the same cluster, and so, each disconnected piece of the graph will form a cluster. Figure 4.3 illustrates this relationship between the graph and clustering.

The sphere-of-influence graph can be computed with complexity $\mathcal{O}(n \log n)$, where n is the number of points [32].

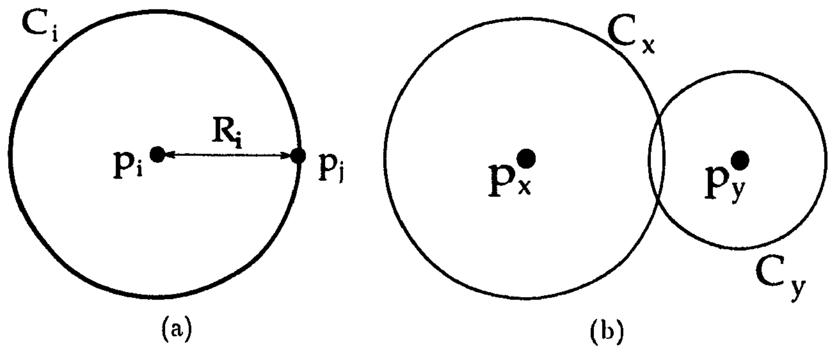


Figure 4.2: Sphere of Influence Clustering: *(a)* Around each point p_i, define the circle C_i, whose radius is the distance to p_i's nearest neighbour p_j. *(b)* Any p_x and p_y are in the same cluster if and only if C_x and C_y intersect in at least two places.

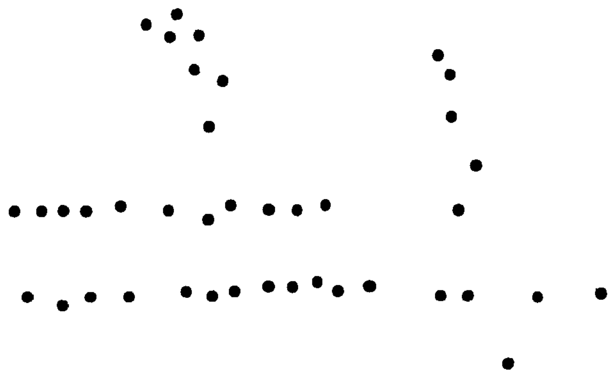
4.2.2 Modification for Noisy Sonar Data

With the sphere-of-influence graph it is conceivable that a small number of points very close together could form their own cluster, even among a dense group of points. This problem could easily arise if a range sensor scanned the same object twice: the two data points representing the object's position would likely be very close together. A cluster consisting of just these two points would be of little use in determining the overall shape of the object, and would in fact suggest the existence of a very small distinguishable object (see figure 4.4). However, given the limited sensor resolution of range sensing devices such as sonar, such a suggestion would not be plausible.

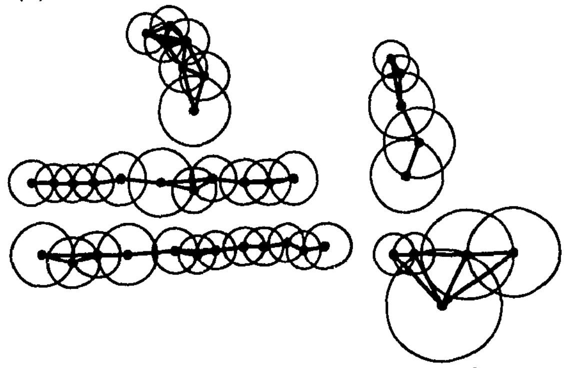
A solution to this problem is to fix a lower limit on R_i, guaranteeing a minimum distance to associate neighbours, which effectively places a lower bound on cluster size and excludes the possibility of tiny clusters. The value of this lower limit depends directly on the accuracy of the particular range sensor.

4.3 The Line-Segment Modeling Strategy

Assigning line-segment models to the individual data clusters is done with a *fit-split-merge* strategy (Figure 4.5). Its basic operation is as follows: given a cluster of data points, a single line segment is fit to the entire set of points. If the fit is good (the



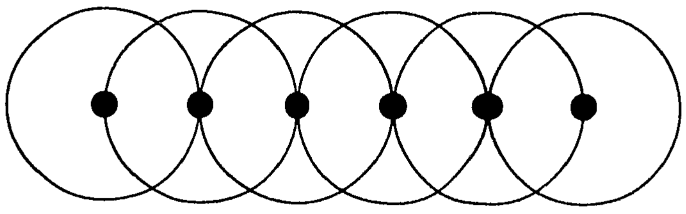
(a) A set of unordered points before clustering.



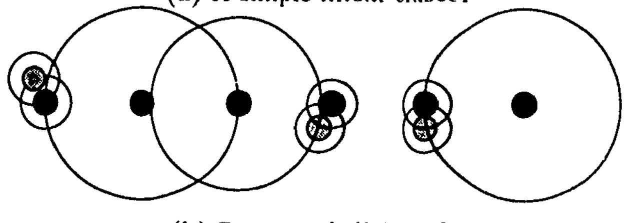
(b) Application of the Sphere-of-Influence graph.

Figure 4.3: Clustering and the Sphere-of-Influence Graph. Given a set of points in a plane, clusters in the data are obtained. Connecting lines denote edges of the graph. The disconnectedness of the clusters is clearly visible by the limits of the circles.

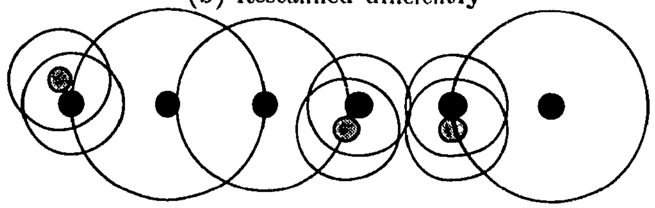
line segment fits the data well) then this model is retained as a description for this cluster. If the fit is poor, then the data is not well modeled by a single line, and the cluster is divided into two sub-clusters. Fitting is then attempted on each of the new sub-clusters individually. For instance, a set of points that forms a corner (i.e. a junction of two walls) is not well modeled by a single line segment - it would be better to use two perpendicular line segments. Fitting and splitting are performed recursively until each cluster has a line segment that fits well, or until a stopping condition is reached. An *a priori* stopping criterion is needed to prevent splitting of the clusters into too many tiny groups (in the worst case two points each, since a line fit to two points is always a good fit). After all lines have been chosen, any



(a) A simple linear cluster



(b) Rescanned differently



(c) Using a minimum radius rule

Figure 4.4: Effects of Too-Small Clusters: (a) shows a simple linear cluster, assumed taken from a flat object. (b) shows the object rescanned, but this time three of the points differed by a small amount (due to sensor error), so that now two clusters are formed instead of one. With the application of a minimun radius rule, as in (c), we get the cluster as before.

line segments that are close together and co-linear are merged into a larger single compound line.

4.3.1 Line Fitting Algorithm

The line fitting step of the algorithm is based on a least-mean-squares method independent of the choice of coordinate axes, known as *eigenvector line fitting*. Basically, the fit line minimizes the sum of squares of the ***perpendicular*** distances from the points to the line [7, 30, 15]. In two dimensions, given n data points in the form

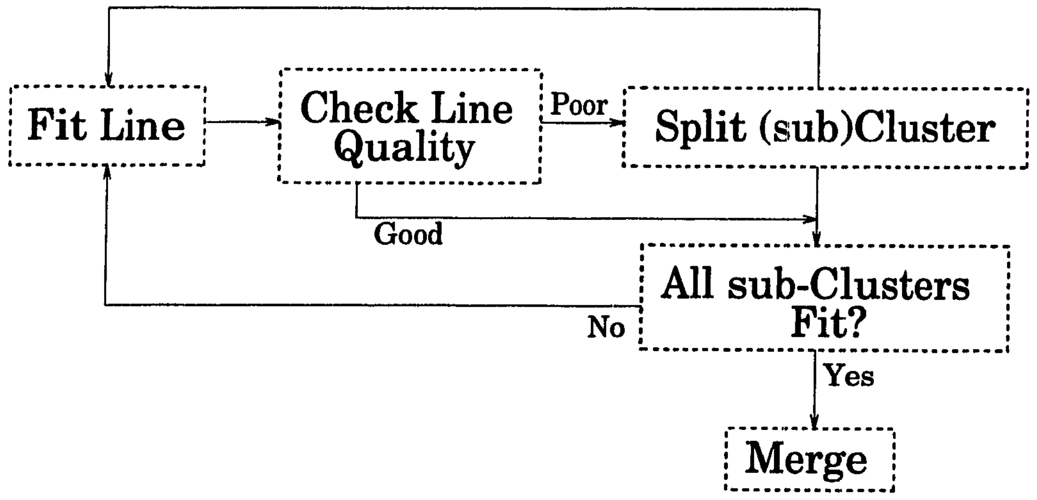


Figure 4.5: The Recursive Fit-Split-Merge Strategy for fitting straight line segments to a single cluster of range data points: If a single line does not fit a cluster well, split the cluster in half and try to fit a line to each sub-cluster. Continue recursively until all sub-clusters are either fit acceptably or are judged too unsuitable to be fit with a line segment.

(x_i, y_i), the covariance or scatter matrix S is calculated as:

$$S = \frac{1}{n}\begin{bmatrix} \sum_{i=1}^{n}(x_i - \bar{x})^2 & \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{n}(y_i - \bar{y})^2 \end{bmatrix} \tag{4.1}$$

Computing the principal and secondary eigenvalues λ_{prin} and λ_{sec} of S and their respective eigenvectors $\vec{e}_{prin}$ and $\vec{e}_{sec}$ finds the direction of maximum and minimum variance in the cluster. This gives information about the shape of the cluster. Since the goal is to fit straight lines, the direction of maximum variance is the orientation at which such a line would be best placed, since the cluster is most "spread out" in this direction. Therefore, the best straight line $\hat{\ell}$ given the n data points is a line parallel to the principal eigenvector $\vec{e}_{prin}$. Since the line must be fit onto the cluster, this best fit line also must pass through the mean of all n points. Note that this line is of infinite length, as no end points are defined in the above procedure. These may be easily calculated by finding the perpendicular projections of each data point (x_i, y_i) onto $\hat{\ell}$, and then assigning the end points to the two projection points furthest from the mean (one projection point on each side of the mean along $\hat{\ell}$), yielding the line segment $\hat{\ell}_{seg}$.

In order to make this computation a little more robust (in addition to its robustness with respect to the choice of coordinate axis) an additional step may be taken. Whenever a line segment is fit to a set of points, a fraction of the points is discarded. By sorting all the points by the distance to their nearest neighbour (which we know already since they have been clustered) and discarding those few whose neighbours are most distant, then we are eliminating the most likely outliers. The fraction to be discarded is arbitrary but must balance keeping the true shape with eliminating outliers. By experimentation, discarding the most outlying 5% of points has shown to be more effective than keeping all points, while not losing the essential shape of the cluster.

4.3.2 Splitting Non-Linear-Shaped Clusters

Now that a line segment has been fit to the cluster, a measure of the quality of its fit is also needed before we can decide whether or not splitting the cluster is necessary. To start, we need to visualize what eigenvector line fitting tells us about the shape of the cluster. $\vec{e}_{prin}$ and $\vec{e}_{sec}$ are perpendicular vectors, thus they allow us to visualize the cluster as a shape which may be defined by two vectors, such as a rectangle or an ellipse. If we arbitrarily consider the shape to be an ellipse, then we are effectively fitting an ellipse to the cluster. The eigenvectors $\vec{e}_{prin}$ and $\vec{e}_{sec}$ govern the orientation of the ellipse (they define the orientation of the major and minor axes) and the eigenvalues λ_{prin} and λ_{sec} define the shape (they are the lengths of the ellipse's major and minor axes). Note that if we had chosen the rectangle as the fitted shape, λ_{prin} and λ_{sec} would be its length and width respectively. Since the eigenvectors and eigenvalues are based on variances, these ellipses will be referred to as variance ellipses (figure 4.6).

Before deciding how to use the variance ellipse to estimate the "fit quality", we need to clarify some properties that $\hat{\ell}_{seg}$ should possess in order to fit a cluster well.

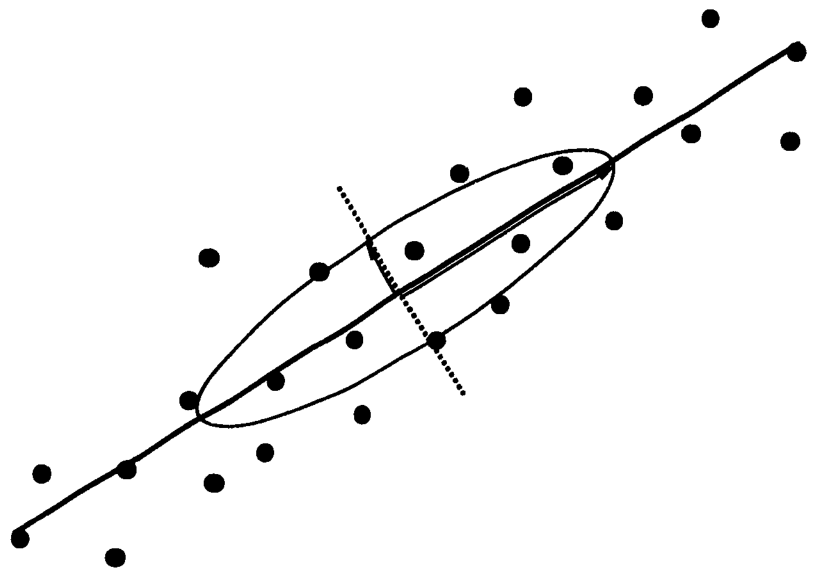


Figure 4.6: Interpreting Clusters as Variance Ellipses: The eigenvalues and eigenvectors of the data points' covariance matrix govern the shape and orientation of the ellipse that best fits into the cluster. The shape of this ellipse provides information as to how well the cluster may be modeled as a single straight line segment.

Two main factors considered here are the *elongation* of the variance ellipse,

$$\kappa = \frac{\lambda_{prin}}{\lambda_{sec}} \tag{4.2}$$

and the *length* $\|\hat{\ell}_{seg}\|$. Elongation is defined as the condition number of the cluster's covariance matrix, which is the ratio of the principal to the secondary eigenvalue. An elongated ellipse has the property that when $\lambda_{prin} \gg \lambda_{sec}$, the maximum variance dominates the secondary and so the cluster tends to be well fit by a single line - in this case we do not wish to split the cluster and re-fit. A variance ellipse where $\lambda_{prin} \sim \lambda_{sec}$ is more circular in shape than elliptical, meaning the cluster is less likely to be in the shape of a single line, and should be split. There is the case, however, where a cluster should be split despite a large κ: if λ_{sec} is still significant (i.e. more than just a few centimetres for our sensors). There may be some structure perpendicular to $\hat{\ell}_{seg}$ that should not be missed (figure 4.7). Only if $\hat{\ell}_{seg}$ is very long could we have an elongated ellipse *and* a significant minimum variance. Therefore it is desirable to split long lines just in case this "masked" perpendicular structure exists. If there is no such structure and the data is indeed shaped like a long line segment, then the merging of line segments after the fitting process (section 4.3.3) will make the correction, so nothing except time is lost in doing the extra splitting. We now define the *quality* of

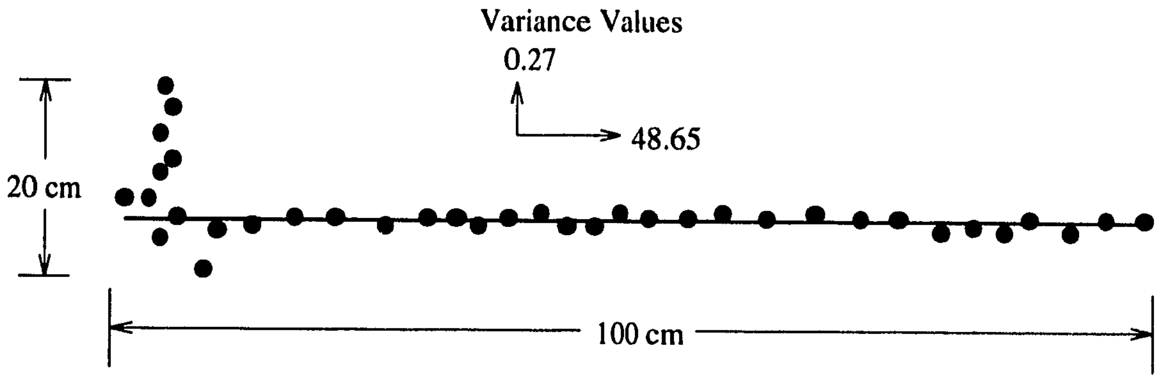


Figure 4.7: Splitting Despite Large κ: Given the dimensions of the cluster and the eigenvalues as shown, one can see that despite the large κ of 180, this cluster should still be split due to the presence of the 20 cm corner.

a fit line as:

$$\varepsilon = \frac{||\hat{\ell}_{seg}||}{\kappa} \tag{4.3}$$

In order to start with a simple frame of reference, we have defined ε so that a perfectly fit line has a "quality" of zero. While it may seem more intuitive to use high values for high quality, we would have a value of infinity for perfectly fit lines, which is too cumbersome to work with. If a straight line fits a cluster perfectly (each point is on the line), then the variance perpendicular to the line is zero, ie $\lambda_{sec} = 0$, the elongation κ is infinite, and the line quality ε is zero. For example, consider the simple case where we have n points that all lie on the same line, such as $p_i = (i, i)$, for $i = 1, \ldots, n$. Clearly $\sum_{i=1}^{n}(x - \hat{x})^2 = \sum_{i=1}^{n}(y - \hat{y})^2 = \sum_{i=1}^{n}(x - \hat{x})(y - \hat{y})$ and so the covariance matrix S is singular and therefore one of its eigenvalues is zero. This means that $\kappa = \infty$ and so $\varepsilon = 0$. In this way we have an easy to use, recognizable value for a perfectly fit line.

In order to decide whether or not a cluster should be split, we apply thresholding to the value of ε. A typical threshold is 1, which means we should split any line whose length is greater than its elongation. Smaller values such as 0.5 would allow more detail in a map yet would require more lines to do so. The converse is true for thresholds greater than unity.

There are additional factors in the line generation process that are not accounted for in the definition of ε. For one, we require a lower bound on the absolute length of $\hat{\ell}$. A tiny line should not be split into even tinier lines if its length is already equal to the precision of the sensor, and so an additional threshold should be applied here. The actual number of data points used to fit the line is also an issue. After splitting a number of times there may only be a small number of points left in a given section of the cluster, which may not be enough to ensure a line segment consistent with the overall cluster.

Once it has been decided that a split needs to be made, the cluster is divided into two parts. The minor axis (the line through the cluster's mean and parallel to $\vec{e}_{sec}$) is used as the *splitting border* to separate the cluster into the two sub-clusters, and each is then treated as a cluster in its own right and fit with a line segment. Figure 4.8 illustrates an example of fitting and splitting a single-corner-shaped cluster, a shape that is not well modeled by a single line.

4.3.3 Merging of Line Segments

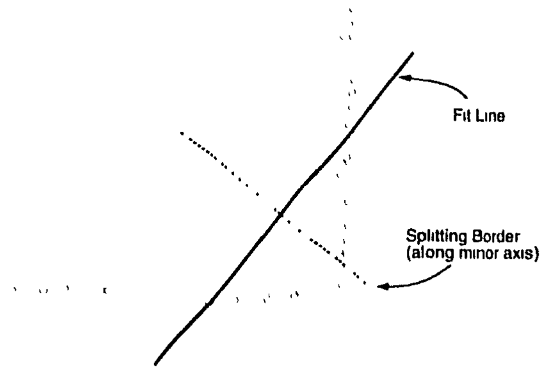
After fitting and splitting, the collection of existing line segments is inspected and evaluated for the merging stage of map construction, which is the combination of pairs into new line segments.

Merging is performed for three reasons:

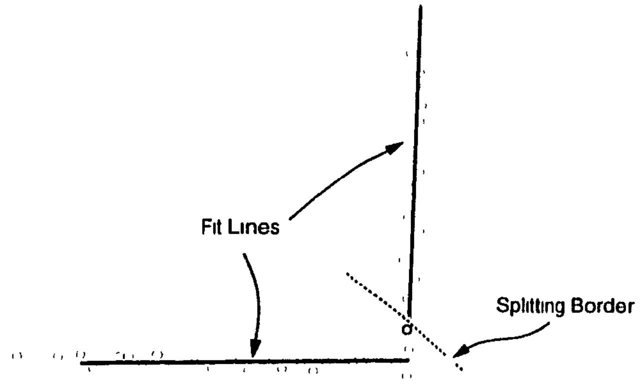
1. To *combine* short parallel line segments from *a single cluster* that may have been over-split due to its size, shape, etc. (figure 4.9a)

2. To *combine* line segments from *different views* of the environment. For instance, consider two adjacent sections of a wall seen by the sensor, each with its own line-segment model of its part of the wall. Since the two sections are part of the same wall, it is better to have a single line segment representing the whole wall rather than two line segments representing two adjacent wall sections (figure 4.9a).

3. To *update* the line-segment models with newer measurements. Often only a



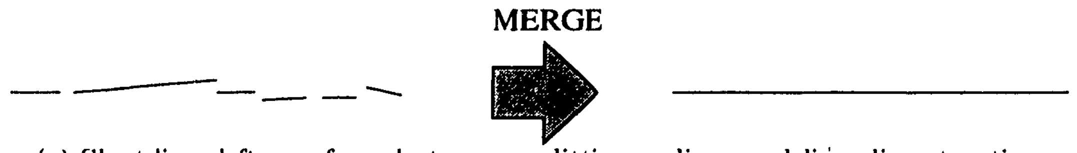
(a) Fitting 1 Line Segment to a Corner-Shaped Cluster.



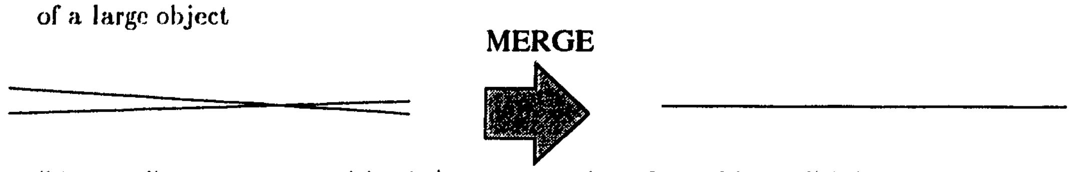
(b) Using Cluster-Splitting to Fit Line Segments

Figure 4.8: An Example of the Cluster Splitting Process

small part of an object is seen from a given position. As the robot moves to a position from which it can view the object more clearly, it is able to obtain more precise data, and this new data should be merged with the old data to update the model of the object. For example, if the front of a small filing cabinet is seen from many positions, the model representing it would be a line segment formed from a combination of lines fit from those positions. Since some of these views may be somewhat inaccurate (due to distance, angle of incidence, etc.), using the data from multiple views makes an accurate description of the same object part more likely (due to the effects of possibly large amounts of data used to construct the model). This is different from (2) in that the line segments to be merged are not adjacent but are models of the same part of the object (figure 4.9b).



(a) Short lines left over from cluster over-splitting, or lines modeling adjacent sections of a large object



(b) Two line-segment models of the same section of an object, differing due to small measurement errors

Figure 4.9: Cases Requiring Merging: scenarios where merging is required

It is assumed that parallel and neighbouring line segments represent either adjacent or the same sections of an object, such as a large wall. Therefore, any two line segments are merged together if they are approximately parallel (their orientations are within some threshold θ°_{merge} of each other) and are neighbours in a Euclidean distance sense (the distance between the two closest points on differing lines is less than some d_{merge}).

Merging is performed in a manner similar to that used for line fitting: eigenvector line fitting. This time we are not dealing with a set of data points, but as far as eigenvector line fitting is concerned, we are. This is due to the fact that when we store a line-segment model in computer memory, we keep the *number* of data points used to construct the model (**not** the points themselves), as well as its *mean* and *covariance matrix*. Given only these statistics of two sets of points, it is a straight forward matter to obtain the same statistics of the combined set, as follows:

Suppose we have two line segments, and we know for $i = 1, 2$ the values for n_i (the number of points used to construct the line), $\bar{x}_i$ (the mean of the x coordinates), $\bar{y}_i$ (the mean of the y coordinates), and the covariance matrix

$$\begin{bmatrix} a_i & b_i \\ b_i & c_i \end{bmatrix}$$

We wish to find the statistics for a single line segment obtained by merging the two, having N data points, means $\bar{X}$ and $\bar{Y}$, and covariance

$$\begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

So we have [16]:

$$N = n_1 + n_2 \tag{4.4}$$
$$\bar{X} = \frac{1}{N}(n_1\bar{x}_1 + n_2\bar{x}_2) \tag{4.5}$$
$$\bar{Y} = \frac{1}{N}(n_1\bar{y}_1 + n_2\bar{y}_2) \tag{4.6}$$
$$A = \frac{(n_1 - 1)a_1 + (n_2 - 1)a_2 + n_1(\bar{x}_1 - \bar{X})^2 + n_2(\bar{x}_2 - \bar{X})^2}{N}$$
$$B = \frac{(n_1 - 1)b_1 + (n_2 - 1)b_2 + n_1(\bar{x}_1 - \bar{X})(\bar{y}_1 - \bar{Y}) + n_2(\bar{x}_2 - \bar{X})(\bar{y}_2 - \bar{Y})}{N}$$
$$C = \frac{(n_1 - 1)c_1 + (n_2 - 1)c_2 + n_1(\bar{y}_1 - \bar{Y})^2 + n_2(\bar{y}_2 - \bar{Y})^2}{N} \tag{4.7}$$

So, given two line segments, each with its own cluster statistics, we can combine their statistics and apply eigenvector line fitting to obtain the same result as if we were given all the data points individually in the first place. This ensures that overall line merging will be independent of the order of the merging of pairs of lines, plus we are not obliged to store all the range data points for each line segment (a veritable benison indeed!).

Figure 4.10 is an example illustrating the combined, step-by-step results of fitting, splitting and merging. Here multiple splits and merges are required to best model the shape of the two-corner-shaped cluster.

To demonstrate these map construction capabilities, consider an "architectural" map of a real environment (a portion of laboratory space) as shown in figure 4.11a, and the map incrementally constructed from it by exploring the environment (figure 4.11b). The range data points were gathered by the robot as it moved along the path marked by the dotted line. Each time new data was acquired, the map was updated. The dark lines in figure 4.11(b) represent the line segment models of the

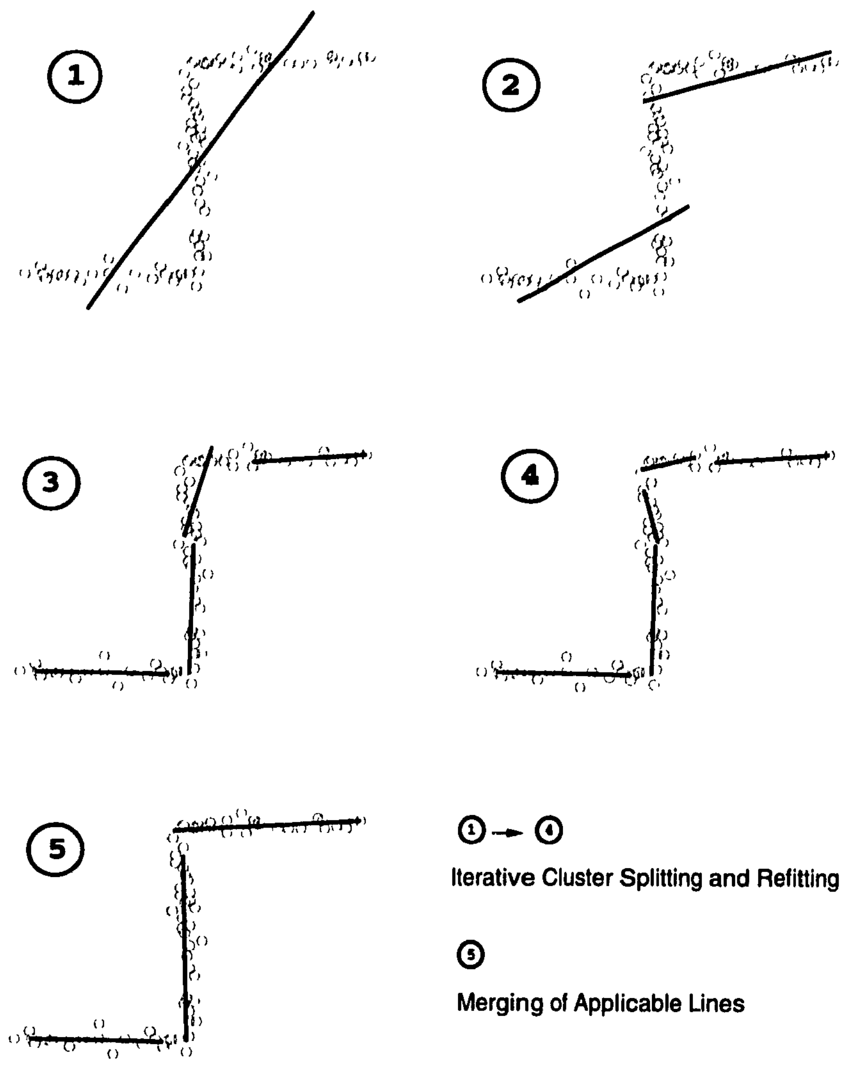

1
2
3
4
5
1 → 4
Iterative Cluster Splitting and Refitting
5
Merging of Applicable Lines

Figure 4.10: The Combined Actions of the Fit-Split-Merge Strategy

map, and their relationship to the actual range data is easily seen. It is important to realize here that this environment was not entirely scanned before building the map - while this is indeed possible, incremental map construction allows the robot to correct the positional and orientational errors that accumulate as the robot moves. This will be explained fully in later chapters.

Figure 4.11 also reveals an interesting aspect of this method when used with sonar sensors. The upper right shows what is referred to as a *spurious wall.* This is a wall that does not exist in the environment, yet is consistently seen by the sensor from

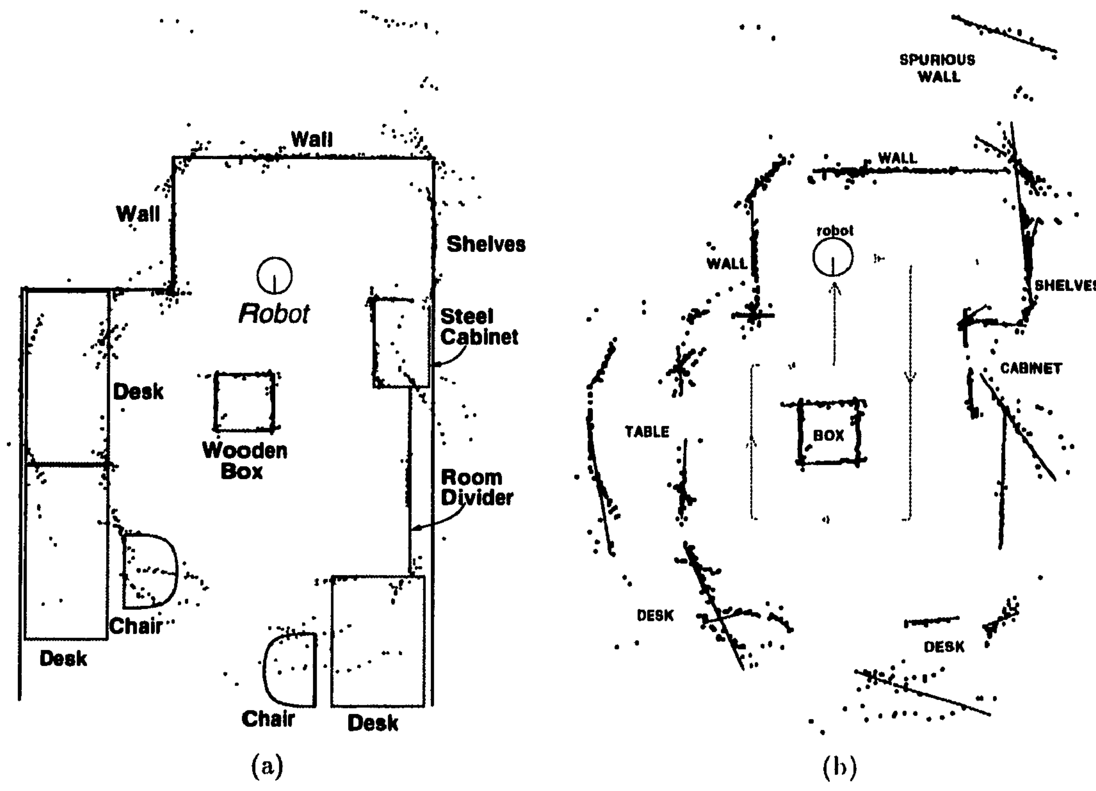


Figure 4.11: An Example of Map Construction: The robot is in the environment shown in (a). In (b), as the robot moves in a simple path (dotted line) around a wooden box in the centre of the area, it scans and updates its map, which consists of the line segments shown overlaying the range data. The range data is shown here for comparative purposes, but this map was not constructed from this entire set as a whole; rather, it was made incrementally from the various individual scans.

particular locations due to the effects of multiple reflections of the sonar pulses. These may seem to be a liability to the map, but they need not be. As long as that part of the environment where these illusionary walls were formed does not change, every time the robot is at or near the location where it first saw the spurious walls, it will see them again - in other words, spurious walls are consistent. In chapter 1 it was stated that the map need only be in terms of the robot's own perceptual mechanisms, not necessarily conforming exactly with what human beings would see. Consistently seeing these ghost walls makes them valuable as environment features in their own right. Of course, they can only be seen from a limited area in the environment, but they still have enough use that they do not need to be discarded (which removes the necessity for deciding which walls are real and which are illusionary).

4.3.4 Sonar-Specific Considerations

Although the aim of this thesis is to develop a localization method independent of a specific type of range sensor involved (for example, it should work for laser range-finders as well as sonar), for any given sensor such as sonar there are some considerations particular to it. Sonar is much more than sending out a thin ray of sound and waiting for its echo. While the thin line model may be applicable for laser range finders, the "chirp" of a sonar transducer spreads out in a roughly conic fashion (of course in reality it is much more complicated than this [20, 1, 21, 9]). Kuc and Barshan have discussed a physical model where the sonar beam consists of two distinct regions: a near zone and a far zone [1]. In the near zone, the sonar beam can be modeled as a cylinder of the same radius as the transducer. The far zone is modeled as a 3 dimensional cone diverging from the transducer (up to a certain distance from the transducer), and the half-angle θ of this cone is calculated as:

$$\theta = \sin^{-1} \frac{0.61\lambda}{a} \tag{4.8}$$

where λ is the acoustic wavelength and a is the radius of the transducer. The system of Kuc and Barshan had $\theta = 10°$, while our system is closer to $\theta = 12°$ (a=20mm, f=49.4 kHz, $c \simeq 343$ m/s and $\lambda = \frac{c}{f}$). The near zone model is valid for ranges up to $\frac{a^2}{\lambda}$, which for our system is approximately 55 mm. Since we are almost always dealing with distances further than this (the robot tends to avoid being so close to obstacles for safety reasons), we ignore the near zone and consider a simplified model where sonar diverges as a cone of fixed half-angle θ from the transducer in 2 dimensions. We ignore the 3rd dimension for this simple model as we are using 2 dimensional maps, although in some cases the 3rd dimension does play a role, as is described below.

Since we are not modeling the surface specularity of objects in the environment, a worst case approximation is to assume that the sensor responds to the first object within the sonar cone, regardless of where it is within that cone[1]. We assume any object seen by the sensor is at the centre of the cone (as we want to minimize the error in the orientation of the object from the transducer).

To see a case where 3-D effects affect the modeling, refer to figures 4.11 and 4.12: when the robot is away from the desk (the top left desk in figure 4.11(a)), the sonar cone intersects the desktop even though the robot could fit underneath (figure 4.12(a)), resulting in lines fit at the position of the right edge of the desktop. When closer to the desk (figure 4.12(b)), the cone had not diverged enough to see the desktop, so the distance measurements were of the wall behind the desk. In this case it may be that the sonar pulses will reflect off the floor or the underside of the desktop. However, since these surfaces are at a very large incident angle to the cone, the sonar pulses are reflected toward the rear wall, and since the half-angle of the cone is small (10° to 12°) the error in the distance measured is small. Having these two different objects that are very close together (i.e. the front of the desk and the wall behind it) modeled differently is not necessarily a problem, since the robot will not see both the desktop and the wall behind the desk from the same position.

A particular aspect of threshold-based sonar sensing is, as figures 4.1 and 4.13 illustrate, that the true size of objects may be smaller than sets of sonar measurements may indicate. Given that we know the location of the sensor for each range measurement, we can consider the worst case where the two end points of the line segment model were obtained when the centre of the sonar cone was directed past the edge of the object (as in figure 4.13(c)). By approximating the arc of the sector bordered by the sonar cone as a straight line, then we can assume that for a given end point of a line-segment model, the length has been overestimated by

$$d_{overest} = r_p \cos \theta \tag{4.9}$$

where r_p is the (assumed) range returned by the sensor. Finding $d_{overest}$ for both

[1]Even planar objects common to office environments come in a plethora of surface types, so accounting for individual specularities is very difficult.

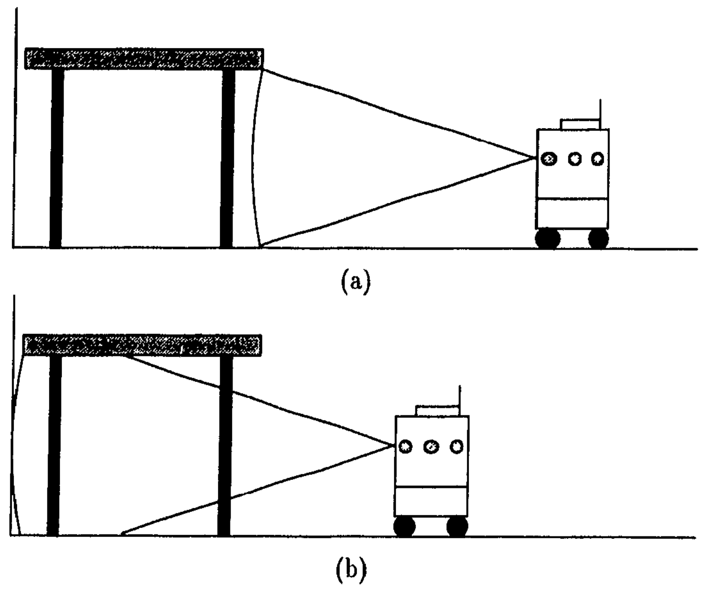


Figure 4.12: 3 Dimensional Effects of the Sonar Cone: (a) The robot is far away from the desk and so it "sees" the desktop. (b) As the robot nears the desk, its cone does not intersect the desktop and proceeds to the wall behind the desk. Note that the half angle of the sonar cone has been enlarged for illustrative purposes.

ends of the line segment allows us to *shorten* it to account for this extra length aspect. While it is true that the object may actually be as large as or larger than the measurements indicate, in a worst case situation we cannot be certain of the true sizes of objects until the robot moves closer. It is also possible that for far models the shortening could reduce the length to zero - in this case, instead of eliminating these models altogether, we may wish to retain a minimum length model just so the robot can remember that an object was seen.

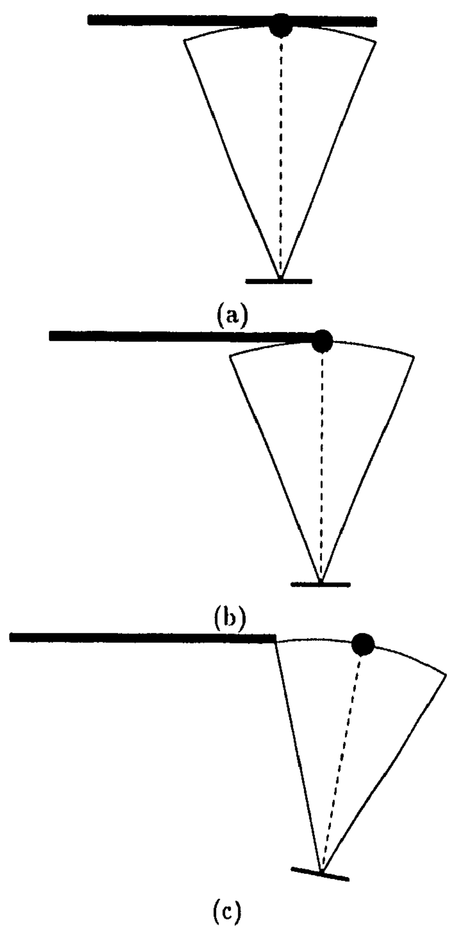


Figure 4.13: Line Lengthening Effects of the Sonar Cone: (a) Scanning the Middle of a Planar Object, (b) Scanning the Edge of a Planar Object, (c) Scanning Past the Edge of a Planar Object

4.4 Summary

In this chapter the issue of constructing, maintaining and updating a map from range data was addressed. There were two main steps in map construction: clustering and line fitting. Clustering was done to separate distinct areas of the environment so that they may be modeled by line segments, under the assumption that one cluster usually represents one object in the world. The clustering algorithm was based on the sphere-of-influence graph, perfectly suited for linear shaped clusters.

To each cluster was then applied the line fitting process, which attempted to assign one or more line-segment models to the cluster. This was done using a *fit-split-merge* strategy, which began with fitting a single line to a cluster using an eigenvector line fitting algorithm. If the fit was not good, the cluster was split into two parts and a line segment was fit to each half cluster. This process of fitting and splitting continued until the entire cluster was modeled adequately. Finally, the merging of neighbouring parallel line segments was done to account for any over splitting of long linear clusters, to combine models representing parts of an object into one model representing the whole, and to update models with new data. The existence of spurious or illusionary walls when using sonar sensors was mentioned and ignoring the fact that they do not correspond directly to physical objects was justified.

To illustrate the combined process, an example using real sonar data was presented, showing the results of an incrementally built map in a partially enclosed section of a laboratory.

Finally, improving the modeling process for sonar range sensors was discussed. A simplified model of sonar and how objects can be seen as larger than they are was given, and then the method of using it to adjust the size of the line-segment models was described.

The problem of how to navigate the robot in order to best collect the range data was not addressed in this chapter. This problem lies within the realms of path planning and exploration, and is beyond the scope of this thesis.

Chapter 5 Position Correction

This chapter presents the heart of the localization process, which is referred to here as *position correction*. This process takes a coarse estimate of the robot's position in its environment and uses it to find its true position (or at least the position as close as possible to it), thereby minimizing the error in the position estimate. Although in reality both position *and* orientation need to be corrected, at this stage only positional errors are corrected. Orientation estimates are assumed for this chapter to contain neglible error. Correcting orientation errors is presented in chapter 7, and is done independently of position. The assumption of an error-free orientation estimate is made at this point in order to construct the basic building block of localization.

Three things are assumed present for position correction:

1. a fair[1] estimate of the robot's pose (position and orientation)
2. an *a priori* known map of the robot's immediate environment (constructed via the methods discussed in chapter 4)
3. a set of range data

Figure 5.1 outlines the major stages in position correction. In the first step, *classification*, each point of the range data is matched with a *target* line segment, which is the model closest to it in a Euclidean sense, and is assumed to represent the real world object from which the range point was obtained. The next step, *calibration*, calculates a weighted sum of all the individual vector differences between range points and their targets, and applies this vector to the current position estimate, generating a new, more accurate position estimate. These two stages are performed until the position estimate converges to a stable value, or until some stopping criteria is met

[1]exactly what constitutes a "fair" estimate is related to the region of convergence discussed in section 5.2.3 – for this chapter one may assume that the position error in the estimate is small

(which would be the case if the true position could not be found). Finally, the refined position estimate is verified using a unity-scale quality measure (to be discussed in detail in chapter 6), which guarantees that convergence occurred to the correct position. Each of these stages is discussed below.

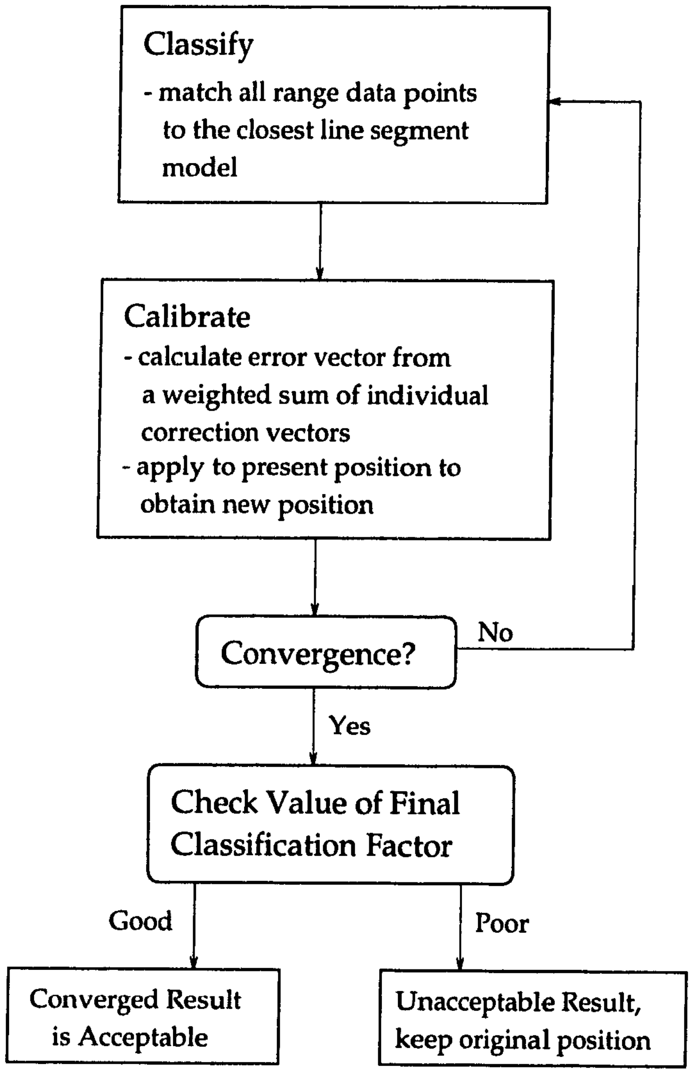

Classify
- match all range data points to the closest line segment model
Calibrate
- calculate error vector from a weighted sum of individual correction vectors
- apply to present position to obtain new position
Convergence?
No
Yes
Check Value of Final Classification Factor
Good
Poor
Converged Result is Acceptable
Unacceptable Result, keep original position

Figure 5.1: Stages of Position Correction

5.1 Classification

The first stage of position correction is called *classification*. In this stage, each range data point is *classified*, meaning each is paired with a *target* line segment model. This target model, constructed from prior data, is assumed to represent the object in the environment (whether physical or illusionary) from which the range point was obtained. Since we are given an estimate of the robot's position, one can assume for the most part that if the error in this estimate is small enough, then the line segment **closest** (in a Euclidean distance sense) to a given range data point will correspond to the same real world object. The effects of when this is not the case are discussed in section 5.2.3.

In classification, we are assuming that for small errors in initial position estimate, the range data points will be nearest to the same line segment models as would be the case if the estimate was perfect. In section 5.2.3, the limitations of this assumption are examined, and upper bounds discussed.

Extending this target pairing idea a bit further, we can project each range data point p_{rd} to a position p_{proj} onto the (imagined) infinite line pasing through p_{rd}'s target line segment. This in effect takes a range data point p_{rd} and finds a position p_{proj} which is the *projection* of p_{rd} onto the target line. p_{proj} can be considered to be the location of some obstacle that we saw at p_{rd} **if** there was no error in the position estimate of the robot. Therefore, we now have p_{rd}, where the robot saw some obstacle, and p_{proj}, where the model of the obstacle is located. Now, if the error of the model is small (i.e. the object is physically at the same location as the model indicates) then p_{proj} is the true position of the obstacle seen at p_{rd} (figure 5.2). It follows then that the vector $p_{proj} - p_{rd}$ shows how far and in what direction the position estimate has to be corrected in order for p_{rd} and p_{proj} to coincide. This vector will be henceforth referred to as a *correction vector*, because it provides a direction and a distance that would *correct* the error in the position of p_{rd}. Section 5.2 discusses how the total set of correction vectors (one vector from each range data point) contributes to the final correction of the robot's position estimate.

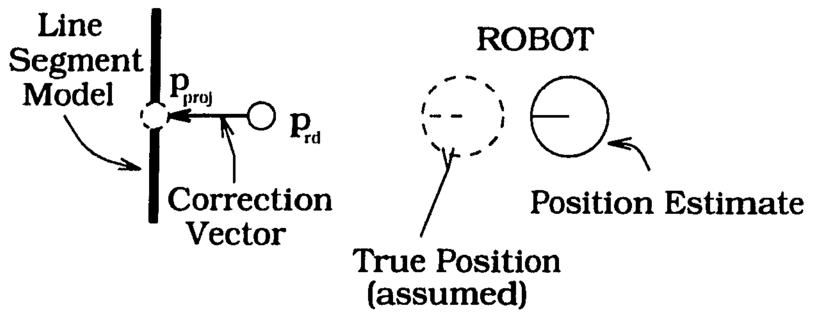


Figure 5.2: Projecting Onto Target Line Segments: by projecting a range data point p_{rd} onto p_{proj} on its target line segment model, we are assuming that the true robot position is as far away from the position estimate as p_{rd} is from p_{proj}. This *assumed* position may not in reality be so, but for the single range data point in question it is the position that reduces the error between the range data point and the line segment to zero.

It is important to note that while the targets are considered as line segments when deciding which point is paired with which target, the targets are treated as infinite lines when the projections are performed.

5.2 Calibration

This section examines the stage called *calibration*, which uses the correction vectors from the Classification stage to calculate an error vector with which the position estimate of the robot may be corrected.

5.2.1 Theory

Figure 5.3 shows an ideal environment of three walls in which an ideal sensor is used. Here the estimated position of the robot is different from its true position as shown; therefore the range data points differ from their true position by the same amount. The arrows show the results of the classification stage: each is a correction vector from a point to its target indicating the direction and distance the point would have to move to put it on the target. Consider the leftmost model: alone, it cannot correct the robot's position completely, since all the correction vectors are parallel to the x-axis and the robot's position error requires calibration in both axes. The

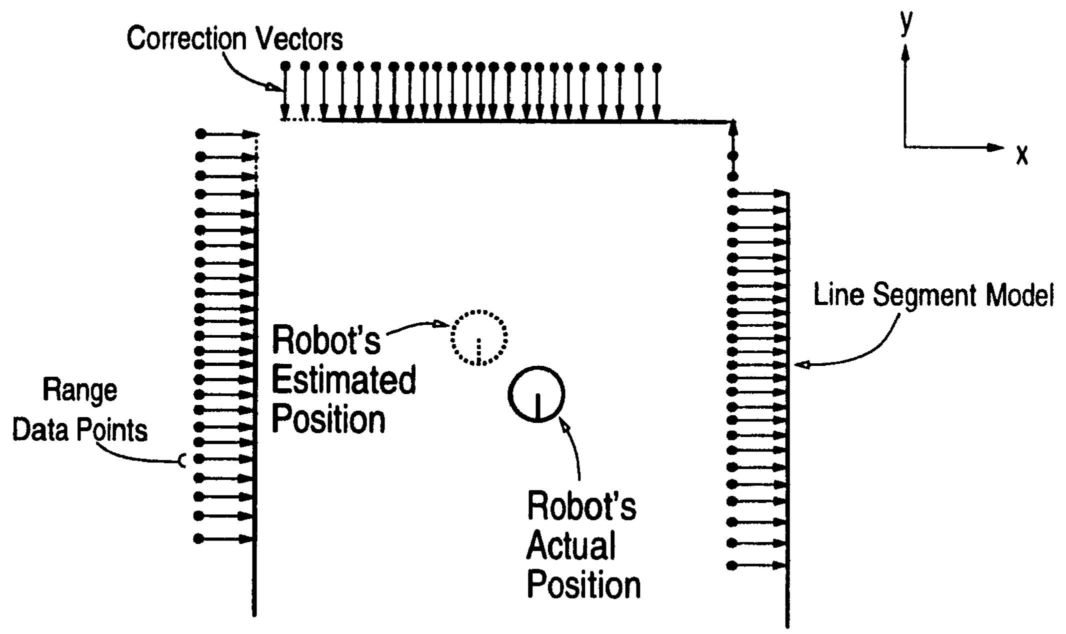


Figure 5.3: The Ideal Case of Position Correction: with ideal models and an ideal sensor, the error in the Robot's Estimate Position is a weighted sum of Correction Vectors

correction vectors for the top line segment are parallel to the y-axis, and so only corrects for errors in the y-axis. It is therefore required that all correction vectors be combined in some way to account for the fact that a single vector can only correct in one dimension.

This one-dimensional limit on each line segment comes from the geometric constraint (in fact, the lack of constraint) that derives from matching to a one dimensional (a line) model. We refer to this problem in this context as the *long hallway effect*, precisely analogous to the *aperture problem* in motion estimation [26]. Observation of position (or motion) of a section of a straight line provides constraint information only in the direction of the normal to the line. In practice, a robot in the middle of a long hallway can only correct its position in the direction perpendicular to the hallway. Movement parallel to the axis of the hallway gives no displacement information since the use of line segment models infers that all parts of the walls look identical, and therefore cannot be distinguished in order to calculate a displacement. This problem may be avoided (in principle) if two or more non-parallel lines are visible from the position of the robot. In the strictest sense this pure one dimensional constraint is valid for infinite lines only. Using finite line segments in the long hallway example, a robot could theoretically correct its position completely if it could see the end of the hall. However, using the end points of line segments as features is dangerous if the map used for correction does not represent the whole environment. For example, if the robot is following a long wall and has only mapped a fraction of it, we do not want the robot to be fooled into thinking it can see the end of the wall just because an end point of a line segment model is nearby.

5.2.2 The Use of Weighted Sums

We now wish to combine the effects of the individual correction vectors. A robust estimator can be constructed by weighting each correction vector differently and then combining them into a weighted sum to find the error vector. This is done instead of a simple average of x and y vector components because an average would be too sensitive to outliers: since all range data points would be given equal weight, any distant, noisy or unreliable data would undesirably affect the resulting vector.

Before discussing the nature of the weighting functions, it is useful to see how they should be used. If each individual correction vector $\vec{v}_i$ is considered to be a simple vector displacement $\vec{v}_i = (\Delta x_i, \Delta y_i)$, we can find the overall displacement $\vec{V} = (\Delta X, \Delta Y)$ using:

$$\Delta X = \frac{\sum_{i=1}^{n} \omega_i \Delta x_i}{\sum_{i=1}^{n} \omega_i} \tag{5.1}$$

$$\Delta Y = \frac{\sum_{i=1}^{n} \omega_i \Delta y_i}{\sum_{i=1}^{n} \omega_i} \tag{5.2}$$

where ω_i is a weighting function for $\vec{v}_i$ (defined below).

In general, given a fair position estimate, range data points close to their target line segments are more likely to be correct than their further-distance counterparts. We can expand on the assumption made during the classification stage: if we assume that a fairly good position estimate is available (where range points are close to their targets models), points far away from any line segment are most likely either noise (erroneous echoes, simple noise, etc.) or are the result of unforeseen objects. The latter have no place in calibration since they are not represented by current models. Therefore, it is preferable to weight close points higher to reduce the likely erroneous contributions of these distant points. On the other hand, if *all* points are far from the line segment models, it is likely that the position estimate is poor, and so we would like all the data points to have a roughly equal voting strength. This distance dependency can be applied to the weighting functions in equations 5.1 and 5.2, as follows:

$$\omega_i = \omega(d_i), d_i = \|\vec{v}_i\| \tag{5.3}$$

There are hence two main considerations for the design of an appropriate weighting function:

- The function should have unity value for short distances and should approach but not reach zero for long distances in order to eliminate outliers.
- The function should be less sensitive to differences in distance at short distances (i.e. a small first derivative). This allows close data points in a distance range around the line segments to be weighted relatively equally.

Table 5.1 shows several functions, only one of which satisfies both constraints (the others are present as a comparison). While the step and linear functions nicely allow for constant weighting for small d, for large d the weighting is zero. Another undesirable effect is that data points may jump from non-zero weighting to zero weighting with just a small change in distance (due to the discontinuity in the first derivative), making calibration more sensitive to the sudden inclusion of outliers. This makes the choice of threshold a problem as well. Conversely, an inverse function allows an asymptotic approach to zero for large d, but the weights are far too large and non-uniform as $d \to 0$. The exponential function does not suffer from extrememly large weights near $d = 0$, but it is not constant for small d. Only the sigmoid function offers a low-d range of relatively constant weighting with the asymptotic approach to zero for high-d. One may think of the sigmoid as a smoothed step function, in order to create a so-called "soft non-linearity". This type of weighting is a scalar multiplication, and so it does fit well with the type of weighting scheme in equations 5.1 and 5.2. The parameter c in the expression for the sigmoid indicates at what distance d the soft step occurs. Control of this parameter allows selection of coarse or fine position correction, since changing c changes what range data points are considered outliers.

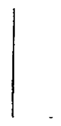
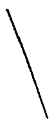
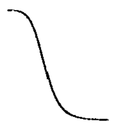
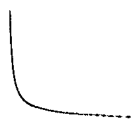
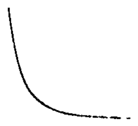
Function Type	Sample Expression	Features	Disadvantages
Step 	$step$(-d-c)	constant weight for small d	zero weight for large d, too abrupt
Linear 	$1 - \frac{1}{d_{run}}(d - d_{lin})$	constant weight for small d	zero weight for large d
Sigmoid 	$1 - \frac{d^m}{d^m+c^m}$	nearly constant weight for small d, small but non-zero weight for large d, smooth transition	
Inverse 	$\frac{1}{d}$	small but non-zero weight for large d	too large and non-uniform weight for small d
Exponential 	e^{-d}	small but non-zero weight for large d	non-uniform weight for small d

Table 5.1: Function Type considerations for Distance Weighting

5.2.3 Error and Convergence Issues

Given ideal classification, the weighted sums of correction vectors will provide a vector that can be added to the current position estimate to find the actual position of the robot. However, for substantial errors in the initial position estimate, errors in classification are usually unavoidable.

These errors, which appear in the individual correction vectors as errors in the distance between range data points and the real world object from which they were measured, hinder the position offset functions from delivering the true robot position in a single step. There are three main sources of error responsible:

A. Sensor Error

With any range sensing device, there are unavoidable errors in its measurements. With a range sensor such as a laser range finder, this error is an accuracy limitation in the range along its beam, which is itself variable for different surface reflectivities. For sonar, the situation is even more complicated. Sonar sensor modeling is an art unto itself, and is much more complicated than tracing straight rays. Real world structures such as corners, edges and cylinders give quite different measurements than their shapes would indicate. Much work has been done in this area [9, 21, 1, 23, 25], but accurate modeling both presupposes knowledge of real world reflectance properties and is computationally costly.

B. Modeling Error

Since the models of the objects in the environment may be based on erroneous measurements, there will almost certainly be some error in the line segment models of the environment themselves. This error would be minimal in an environment composed of objects well modeled by straight, flat line segments - structures such as walls and doors. However, in most indoor office-type environments all objects are not flat and wall-like (such as chairs, desks, people, garbage bins, etc.) and can only be approximated by line segment models.

C. Classification Error

In section 5.1 we assumed that the line segment model closest to a given range data point represents the object in the real world from which that data point was measured.

Obviously this will not be the case except for very small position estimation errors or very simple environments. Even in the ideal case of figure 5.3 a few *misclassifications* are visible (top right corner). Misclassifications are range data points that associate with the wrong model. These may be few in number, but they still have an effect on the overall weighted sum. This effect is usually slight for good initial position estimates.

So far it has been illustrated that correspondence errors prevent accurate position correction with a single weighted sum. However, there is nothing to stop us repeating this process, with the results of the first position correction used as the new position estimate for the second iteration. The process can be performed repeatedly until the position converges, assuming it does converge. There is no guarantee of convergence, but as will be discussed in the next chapter, it is possible to verify that correct convergence has taken place.

Recall that we use a sigmoid function to give higher weights to range data points close to their target line segment. The iteration process can be further refined to exploit the non-uniform sigmoid weighting so that in early iterations, the drop to nearly zero in the sigmoid function $\omega(d)$ (table 5.1) takes place only at very large d, which in effect weights all but the most distant range data points equally. This is done by using a large value of c in the equation of the sigmoid in table 5.1. Since this is effectively ignoring distance weighting, we obtain a coarse position estimate correction based on both good data and outliers together. Therefore, initial iterations use a large value of c and apply distance weighting only to extremely distant data points, if any. Classification and calibration are iteratively applied until the estimate converges, or until some maximum number of iterations has been reached (if this happens, the estimate may not be able to converge).

Since the sigmoid $\omega(d) \approx 1$ except only for very large d, any range data point outliers (those that do not rightfully correspond to a line segment model) are also being included. So, *after* the first convergence, the soft step decrease of the sigmoid is brought closer to $d = 0$, i.e. the sigmoid parameter c is decreased. Position correction is then re-applied until convergence with this new weighting function. Assuming we move closer to the actual position each convergence, this change in $\omega(d)$ is a *shrinking neighbourhood* around each line segment model, with only those points inside each neighbourhood weighted highly. In the final few iterations the virtual neighbourhoods have been decreased enough to ensure that only the very closest (and presumably best) data points will be considered. Iteration until convergence is performed repeatedly until further changes in the weighting function do not affect the resultant position, allowing progression from a coarse position correction to a precise one where only the best (most confident) data is used.

There are some cases when convergence will not succeed. If *enough* range data points are correctly classified initially, the corrected position estimate will be nearer to the actual position than the first (the estimate will not necessarily move *directly* toward the actual position, but the distance between them will certainly decrease) and successive iterations will move the position estimate as close as possible to the actual robot position (ie convergence will occur). If there are too many initial misclassifications, convergence will fail.

Those data points correctly classified all tend to work together, meaning their respective correction vectors are components of the true vector difference between the actual and estimated positions. Incorrectly classified points give rise to correction vectors that in general do not tend to work together, and the weighted sum of these vectors tends to have a smaller effect than the others.

One way to perceive this situation is to consider an error vector $\bar{e}$ associated with a correction vector $\bar{v}$ (refer to figure 5.4, a simple case of the robot in a corner, with outside edges nearby). Given that $\bar{v}$ projects a range data point to a position on its target line, we define $\bar{e}$ as the vector difference between that projection point and the position of the *true point*. The true point may be considered as the position of the range data point if there was no error in the position estimate. Clearly for any such range point $\bar{v}+\bar{e} = \bar{E}$, where $\bar{E}$ is the error vector representing the error in the robot's position estimate, neglecting modeling and sensor error. We do not however know

$\bar{e}$, so using $\bar{v}$ only provides a component of $\bar{E}$ orthogonal to the point's target line segment. For incorrect classifications, $\bar{e}$ may be very large to make up for the poor $\bar{v}$, and so ignoring $\bar{e}$ leaves us with a potentially large error for that range point. However, the resulting $\bar{e}$ of incorrect classifications is more a function of the distribution of line segments in the map than the true position of the robot. If we consider a map as a set of randomly oriented line segments, the $\bar{e}$ vectors from incorrect classifications will also be randomly distributed, and together will not contribute as much as the correctly classified points whose correction vectors are components of $\bar{E}$.

It is clear that $\bar{e}$ is parallel to the target line segment for a perfect object (where many or all range data points will be on or very close to their representative line segment model) and a correct classification. This is to be expected due to the long-hallway effect previously mentioned.

So, to summarize, if a range data point is correctly classified, then its projection is a point on the infinite line through the target line segment model closest to it (section 5.1). This may not be the point in the model representing the same real world position from which the range data point originated, but it differs only by a vector parallel to the model. The key point here is that as long as the range data point is classified correctly, i.e. it is matched with *any* point on its target line segment, its correction vector will be a component of the vector $\bar{E}$ which represents the error in the position estimate of the robot. If the point is misclassified, the error vector can be much larger than the error in the position estimate, as illustriated in figure 5.4. Yet, this type of error is not as critical as it may at first seem. As mentioned previously (page 47) the correction vectors of correctly classified range data points tend to work together, while those of incorrectly classified points do not since they are randomly distributed.

General predictions of how many correct classifications are required for successful convergence are very difficult – matters of convergence are very dependent on the arrangement of line segment models in the map. Consider Figure 5.5: 6 out of 11 of the range data points are initially misclassified. Here in this particular arrangement of line segments, four of these misclassified points provide a correct component of

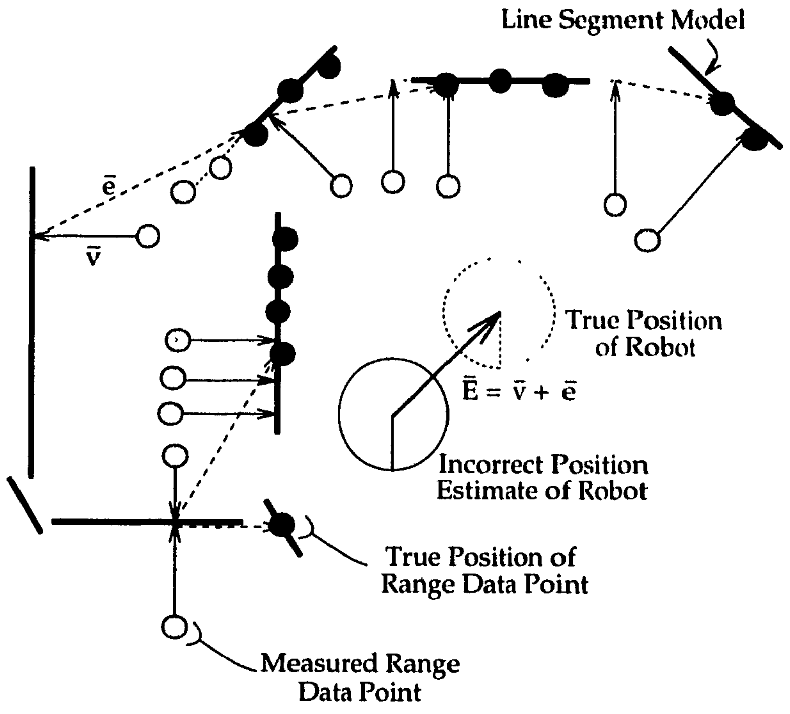
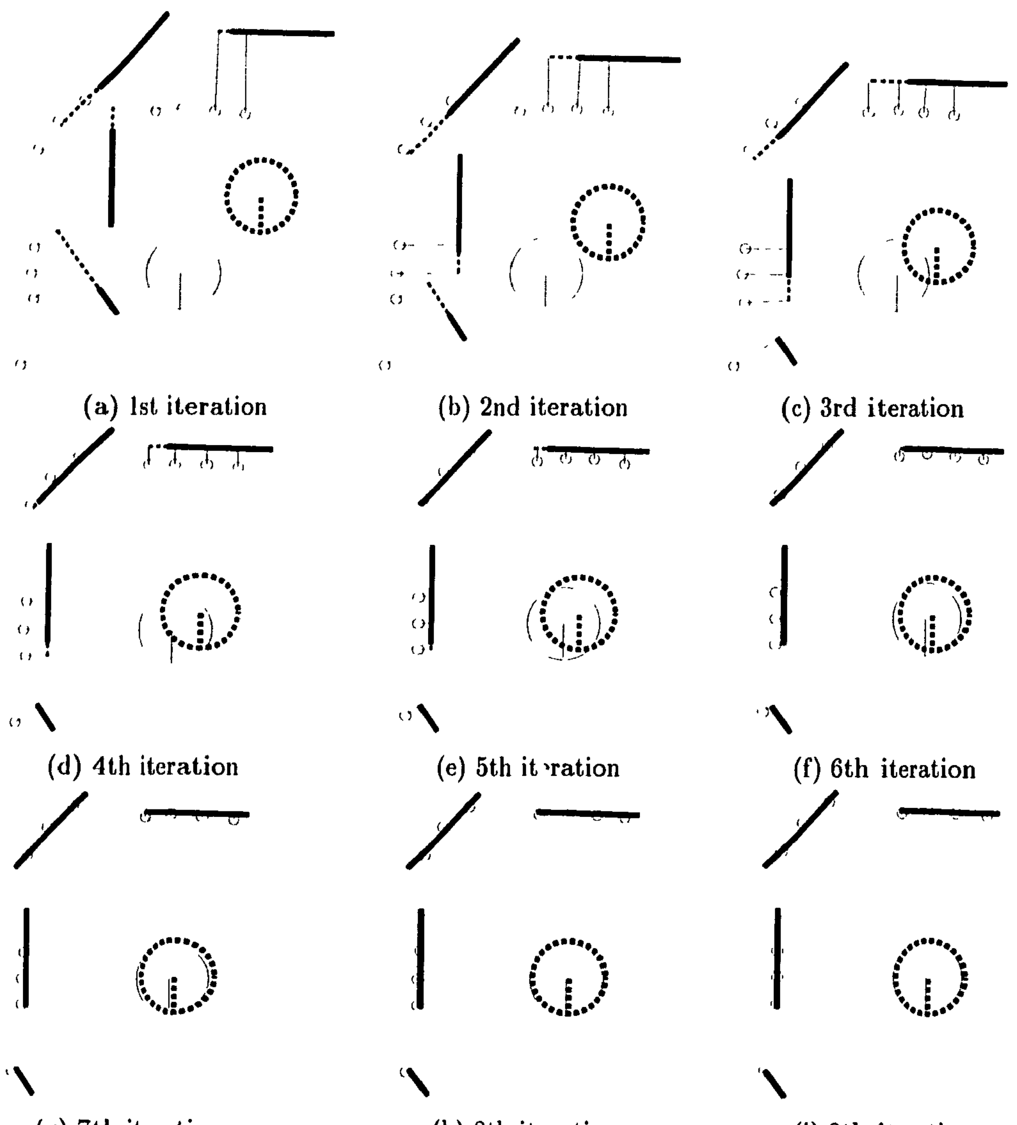

Line Segment Model
$\bar{e}$
$\bar{v}$
True Position of Robot
$\bar{E} = \bar{v} + \bar{e}$
Incorrect Position Estimate of Robot
True Position of Range Data Point
Measured Range Data Point

Figure 5.4: Misclassification of Range Data Points: shown are the true and incorrect estimate of the robot, the true positions of range data points (dark circles), measured range points due to position estimate error (light circles), correction vectors ($\bar{v}$), and the projection error vectors ($\bar{e}$). Note that e is not shown for correctly classified range data points, since they are parallel to their target line segments

the position error vector, while the remaining two do not. The four "accidentally" provide useful information in this case due to the sparseness of the map, where there are few line segment models to which range data points may become misclassified.

The main issue here is that for the first iteration, the algorithm depends on a sufficient number of correctly classified points to *bias* the position correction in the appropriate direction towards the true position, at which point the iterations that follow can use that better estimate for further refinements. Again, this "minimum correctly classified" threshold is dependent on the arrangement of line segments in the map.

This coarse-to-fine approach of the position estimate to the true position can be seen in Figure 5.6 (taken from the map of Figure 4.11 on page 31), which shows one


(a) 1st iteration (b) 2nd iteration (c) 3rd iteration

(d) 4th iteration (e) 5th iteration (f) 6th iteration

(g) 7th iteration (h) 8th iteration (i) 9th iteration

Figure 5.5: Convergence in Spite of Misclassifications: although 6 of 11 range data points are initially misclassified, 4 of these still provide useful vector components since the misclassified line segment model neighbours the correct model and is on the same side of the points. The two misclassified points that provide completely erroneous information do not affect the outcome since they are in the minority of range data points.

such localization that does converge. Initially it only moves toward the true position marked by an * in the x-direction, but the later and more precise iterations show the estimate closing in more directly.

If this type of convergence plot is expanded to a large region around the true loca-

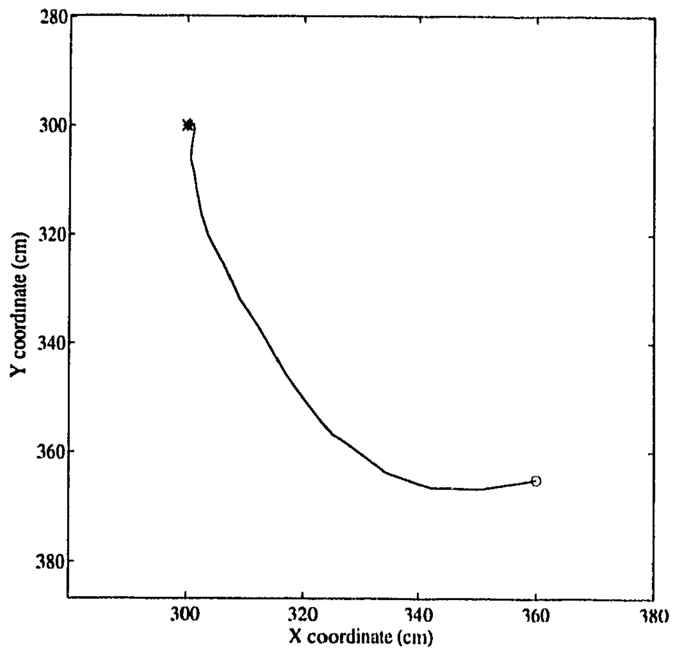


Figure 5.6: The Path of Convergence of the Position Estimate: from the initial position marked "o", to the true position marked "*". Note that the estimate does not move directly toward the true position initially, but does move close enough that the more precise iterations later can do so.

tion, we can find that location's ***region of convergence*** in the given map. In figure 5.7, initial position estimates are spread out all around the actual robot's position (in this case, the home position (x,y)=(300,300)), and their paths of convergence can be observed, as well as any paths that do not converge to home. The initial positions that do not converge are "fooled" by the ***local attractors*** in the environment. These are locations at which the local environment resembles the local environment at the home position, based on the range data scanned there.

It is easier to see the actual region of convergence in Figure 5.8. The entire top left corner of the region (position ($x = 0$, $y = 0$)) converges correctly for this map and robot position because there are no models in this region to confuse the correction. Usually when there are dense collections of line segment models are the chances of significant misclassifications an issue.

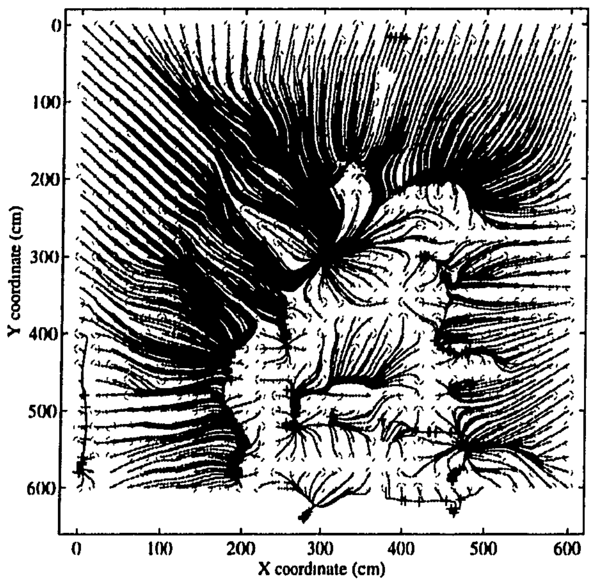


Figure 5.7: Paths of Convergence of Region Surrounding Robot: as in Figure 5.6, for many initial position estimates surrounding the robot.

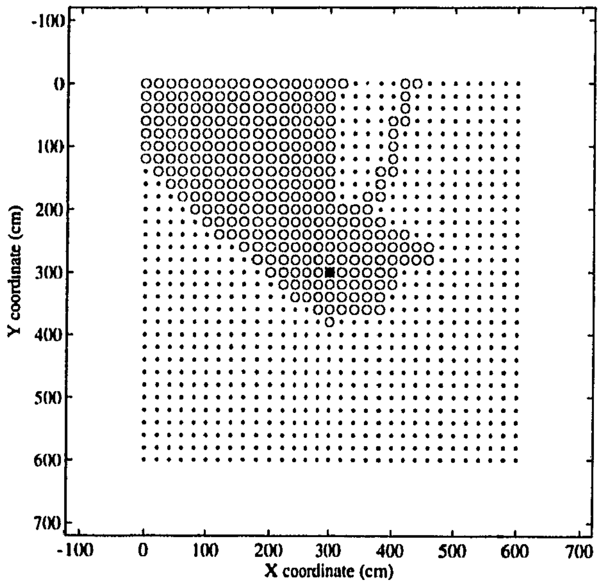


Figure 5.8: Region of Convergence around the True Robot Position at ($x = 300$,$y = 300$): an o represents initial estimate that correctly converges to the true robot position, while a · represents an incorrect correction.

Of course, the convergence plots were made given that we knew the robot's true position, and were simply checking the correction algorithm against it. It is possible to construct these same plots without knowing a priori the true position of the robot, and this will be covered in Chapter 6. It is worthwhile to mention here that at the end of correction, the final solution, converged or not, is checked as to whether the result is feasible as the actual position. If it is judged infeasible, then we know that correction either did not converge or converged to an incorrect position. In other words, if the algorithm did not work, we will know it.

5.3 Summary

This chapter introduced the heart of the localization process, known as position correction. Given an estimate of the robot's position, a map constructed as per Chapter 4, and some new range data, position correction can find the robot's actual position. It is assumed for this procedure that the robot's orientation is known correctly.

This is accomplished in two major steps: classification and calibration. In classification, each range data point is paired with its closest neighboring line segment model, called its *target*, which is assumed to belong to the object in the world from which the point was derived. A *correction vector* is calculated for each point: each is the vector difference between the range data point and its projection onto the infinite line through the target line segment. Once these correction vectors have been calculated, they are combined in a weighted sum that takes into account the distances between the range points and their targets (under the assumption that closer points are likely to be more accurate), doing this using a sigmoid function, which is of the form:

$$1 - \frac{d^m}{d^m + c^m}$$

Section 5.2.3 discussed sources of error that affect correction, the greatest of these being classification error. Due to these inherent errors, correction must be repeatedly performed in order to converge to a solution. After convergence, the parameters of the sigmoid distance weighting function are changed and the corrections are iterated again. This repeats until changing the sigmoid function no longer changes the convergent result. This gives the process a coarse-to-fine precision improvement behaviour, where initially all range points contribute to get a coarse position estimate, and later only perceived *good* data are used for fine tuning.

Sometimes errors prove too great and convergence to the correct solution does not occur. Knowing the true robot position, tests were performed to examine how much error there could be in the initial position estimate before incorrect solutions began to occur. This was tested in a region around the true position to find a *region of convergence*.

Finally, a final check was described that is performed after all iterations have been completed to assure consistency of the solution. This check is discussed in chapter 6.

Chapter 6 Quality/Confidence Measures

6.1 The Need for a Measure of Confidence

As discussed in chapter 5, it is possible that the outcome of correction is not the true position of the robot: either the iterations may not converge or they may converge but to an incorrect location. In cases such as these, it is imperative that it be known that correction did not find the correct robot position; otherwise the results could be disastrous, since an incorrect solution could be even worse than the initial estimate! Once correction is completed, we need some kind of *quality measure* so that we can quantify our confidence in the solution. In addition to this, if we for some reason have multiple solutions (as we will have in chapter 7), we need to compare them and find the best one. Therefore, some criteria for a quality measure are as follows:

1. Incorrect solutions should have a low quality measure, ideally zero.

2. Correct solutions should have a high quality measure.

6.2 Types of Measures

A quality measure is a function of robot pose (pose = position + orientation) calculated by performing some comparison between the range data points and the line segment models.

There are two basic measures used here: the *mean-squared error* measure, and the *classification factor*. Each is discussed below:

6.2.1 The Mean-Squared Error Measure

This quality measure is simply a measure of mean-squared error and has the form:

$$E_{mse} = \frac{1}{n}\sum_{i=1}^{n}(\mathrm{dist}(p_i, \bar{\ell}_i))^2 \in [0, \infty) \tag{6.1}$$

where n is the total number of range data points, p_i is the i^{th} of these, ℓ_i is p_i's target line segment model, $\bar{\ell}_i$ is the infinite line through ℓ_i, and so dist$(p_i, \bar{\ell}_i)$ is the distance from a point p_i to the closest point on the infinite line through the line segment ℓ_i. Distance is taken from the infinite line rather than the line segment to allow for the case where the robot may be following a wall: in this case, range measurements along part of the wall not yet modeled would give rise to a larger E_{mse}, in spite of the fact they belong to the same wall.

This measure is good if its global minimum occurs at the correct location of the robot. At the correct location, it is assumed that all or at least the vast majority of range data points are very close to their classified line segment models, and would then yield a low value for a correct solution. While this may be contrary to the criteria outlined in the previous section, inverting it is a simple way to force it to conform (since there is always at least *some* error with some data points with real data, the chance of inverting zero is neglible).

Figure 6.1 shows E_{mse} as a function of position, where the correct robot position is at (x, y)=(300,300). It is rather difficult to see exactly how different the values are at the correct position, so figure 6.2 shows the same data but with the mean-squared-error measure inverted. There is a problem with this measure however: it is very suceptible to outliers, which will certainly affect the results even if the position estimate is very good. Since it is not known how many outliers are in the range data set, there is no theoretical upper bound on E_{mse}, making decisions of a single good/bad convergence very difficult based on this measure. However, it is useful when used to compare between two solutions.

6.2.2 The Classification Factor

The classification factor (or *classfactor*) E_{cf} is a quantity based on the fraction of all range data points that are *closely classified*. Consider a neighborhood of fixed size x cm around all line segment models. If all the range data points within a neighborhood of a model are counted and then the sum is divided by the total number of points, then we obtain the fraction of the total number of points within x of some model.

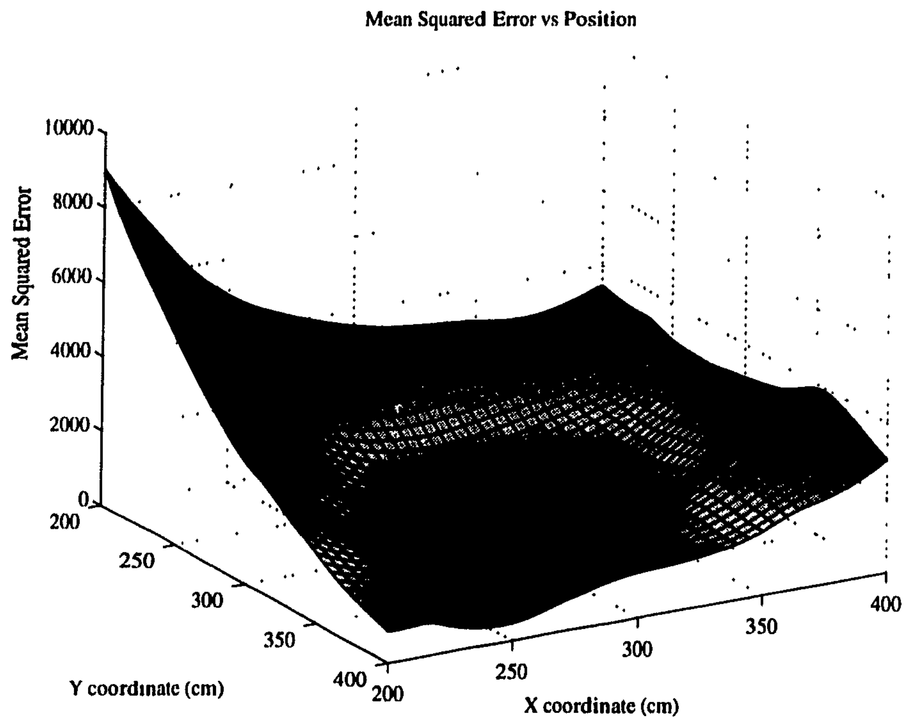


Figure 6.1: The Mean-Squared Error Measure vs Position: the mean-squared error increases dramatically as the position leaves the robot's position at (300,300)

Given that close data points are more likely to be correctly classified, this measure tells us how good our position estimate is. Obviously, incorrect classifications may creep into this and give false readings. However, the only way to get a measure approaching unity is to have a position estimate very close to the actual one, or to be in part of the environment that is **very** similar to the one in which the robot is located. Ignoring the latter, this measure should then give a value close to unity when the position estimate is very close to the correct position, and near zero when the error of the estimate is large.

Using a fixed neighborhood size suffers from the same problem as distance weighting did in section 5.2.2: if a range data point is just a minute distance outside the neighborhood, it will not be counted as being inside that neighborhood. Another range point very close to the first but inside the neighborhood will be counted, and so this abrupt neighborhood boundary increases the sensitivity of the neighborhood

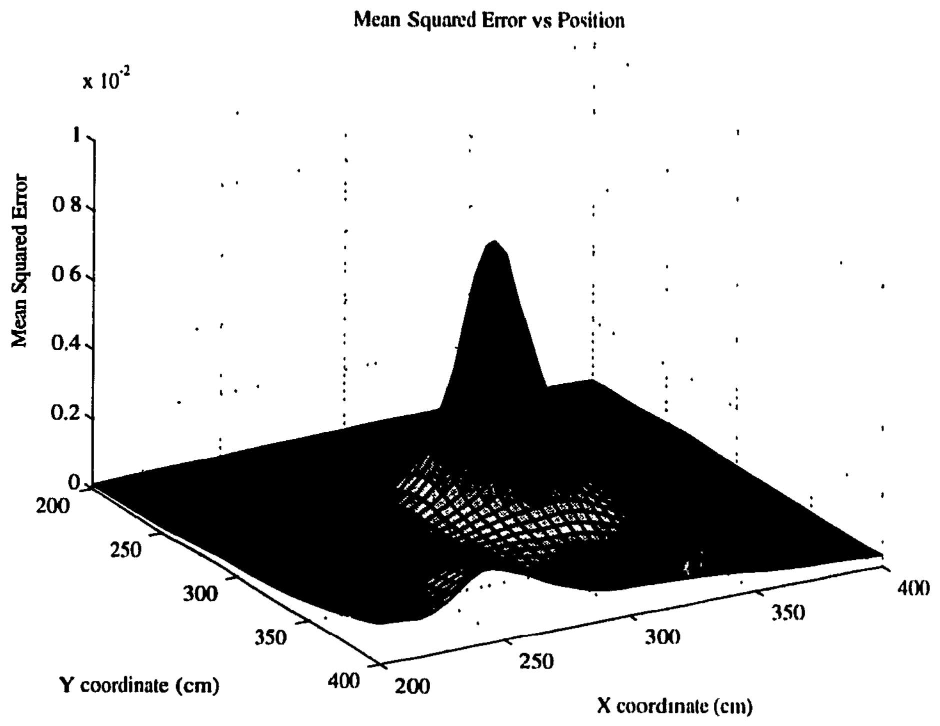


Figure 6.2: The Inverse of the mean-squared-error measure Measure vs Position: with the robot again at (300,300), we can see the global maximum at this point.

to range points just outside its boundary. In order to *smooth* the boundary of the neighborhood, a sigmoid function is applied as a distance threshold, forming "softly" non-linear neighborhoods. Therefore, the *classification factor* is thus defined:

$$E_{cf} = \frac{1}{n}\sum_{i=1}^{n}\left(1 - \frac{d^m}{d^m + c^m}\right) \in [0, 1] \tag{6.2}$$

where: $d = \text{dist}(p_i, \ell_i)$ The cho-

$c =$ a user defined parameter indicating neighborhood size

$m =$ exponent governing steepness of sigmoid, commonly $m = 8$

sen value of c in equation 6.2 depends on the noise estimate of the sensor and the error in the models. If a pose estimate is correct, then all correctly classified range data points should be very close to their target line segments (right on top of the line segments if not for the modeling and sensor error). Ignoring the sigmoid for now, we have a simple step function for the neighborhoods (imagine that the sigmoid just smooths out the step). Experiments with objects such as walls, wooden desks and burlap office dividers have shown that a noise estimate between 5 and 10 cm works well, which is a rather liberal estimate of the uncertainty in the range sensor. The neighborhood should not be too small in case a model does not exactly fit the shape of an object. On the other hand, too large a size defeats the purpose of the measure.

Unlike E_{mse}, the neighbourhood of E_{cf} works with the line *segment* ℓ_i rather than the infinite line $\bar{\ell}_i$ through it. This causes E_{cf} to favour the limited known walls instead of the potential extensions of these walls, and so the two measures complement each other.

Figure 6.3 illustrates E_{cf} in the same environment as the previous figures. At the

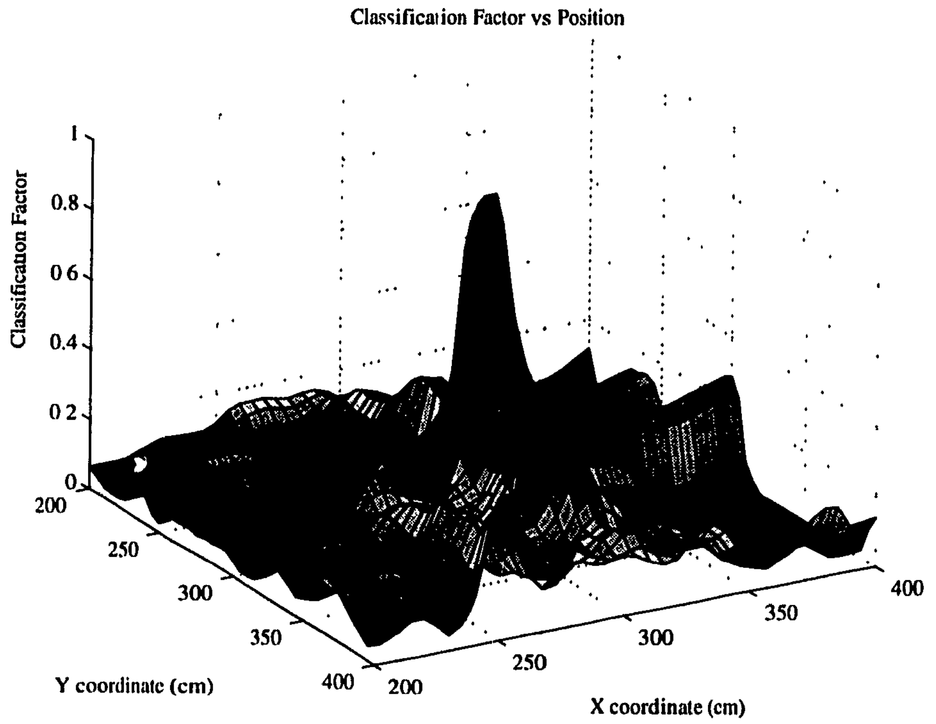


Figure 6.3: The Classification Factor: for this robot position in this environment, E_{cf} reaches a maximum of 0.947.

actual robot position (again, (x,y)=(300,300)), we see the global maximum approaching very closely to unity, while the function is much less everywhere else. The very useful property of E_{cf} is that $E_{cf} \in [0, 1]$, and this allows for a simple threshold test to be made for the likelihood of a convergence being good or bad. Through experimentation in real environments, a good acceptance threshold was discovered to be about 0.6 for typical indoor environments consisting of walls, desks, doors, boxes and suchlike. This value makes sense since we should expect more than half of all points should be within some small neighborhood of their targets, even if there is some small error in the pose estimate.

One problem with E_{cf} is that it is not as useful as E_{mse} when dealing with very fine differences in position. Since it in essence just counts the number of points within a neighborhood, it cannot give precise detail of *where* the points are within it. For instance, if there are two positions on the map (most likely very close together) such that all the range data points are within E_{cf}'s neighborhood, then the number of points counted will be equal, and thus E_{cf} cannot tell which one is closer to the correct position. For this reason, E_{cf} alone is not used as a comparative measure. E_{mse} does not suffer from this, as it deals with actual distances from the line segment models.

6.2.3 The Comparative Quality Measure

While it is quite possible to use the inverse of the mean-squared-error measure as a comparative tool to choose between two solutions, the *comparative quality measure* E_{cqm} adds the effects of both E_{mse} and E_{cf}, as it is a combination of both. It is defined as follows:

$$E_{cqm} = \frac{(E_{cf})^a}{(E_{mse})^b} \in [0, \infty) \qquad (6.3)$$

The exponents a and b respectively weight E_{cf} and E_{mse} relative to each other. Figure 6.4 shows E_{cqm} in the same environment as figures 6.1 and 6.3. The correct robot position at the central peak is now more pronounced with respect to the surrounding positions. This is due to the fact that E_{cf} acts as a non-linear scaling factor to $\frac{1}{E_{mse}}$, and this scale factor approachs unity only when the error in the pose estimate approaches zero. Since E_{mse} is minimized when the pose estimate has minimal error, the area far from the actual pose are de-emphasized.

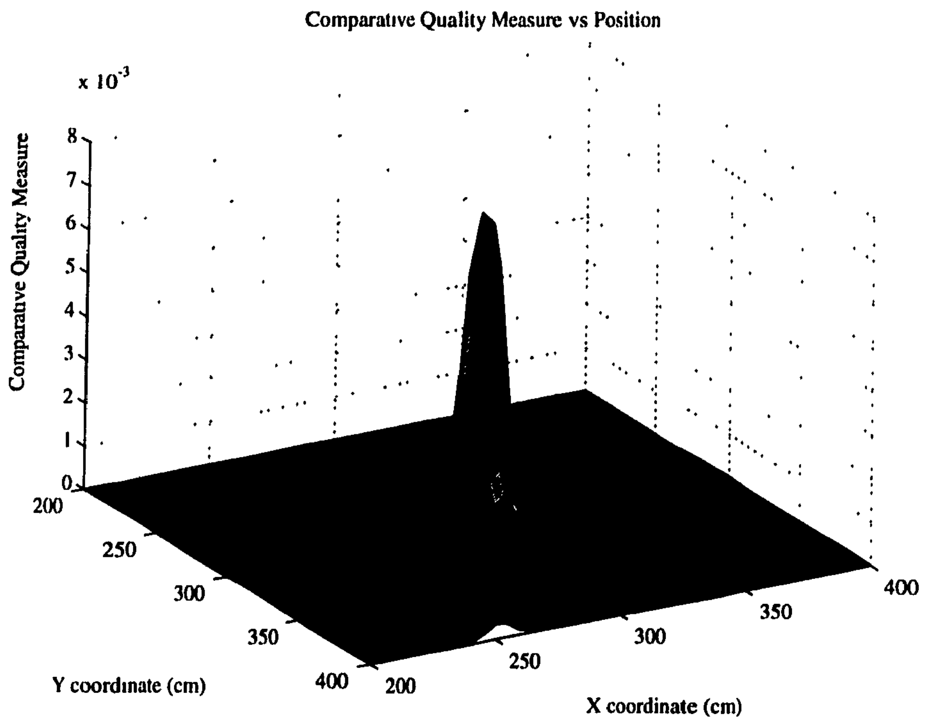


Figure 6.4: The Comparative Quality Measure E_{cqm}; a=2, b=1

In fact, if $1/E_{mse}$ and E_{cqm} are compared on a logarithmic scale (Figure 6.5), then it is clear that E_{cqm} has increased the "importance" of the global maximum by well over an order of magnitude over surrounding positions.

As its name implies, E_{cqm} is a comparative measure only, inheriting this property from E_{mse}. As mentioned above, this measure is used to choose between multiple solutions of position correction, such as is the case in the next chapter.

One unfortunate aspect of E_{cqm} is that the global maximum may be found by gradient ascent methods only close to the global maximum. Away from the global maximum there are many local maxima which make using general gradient ascent impossible.

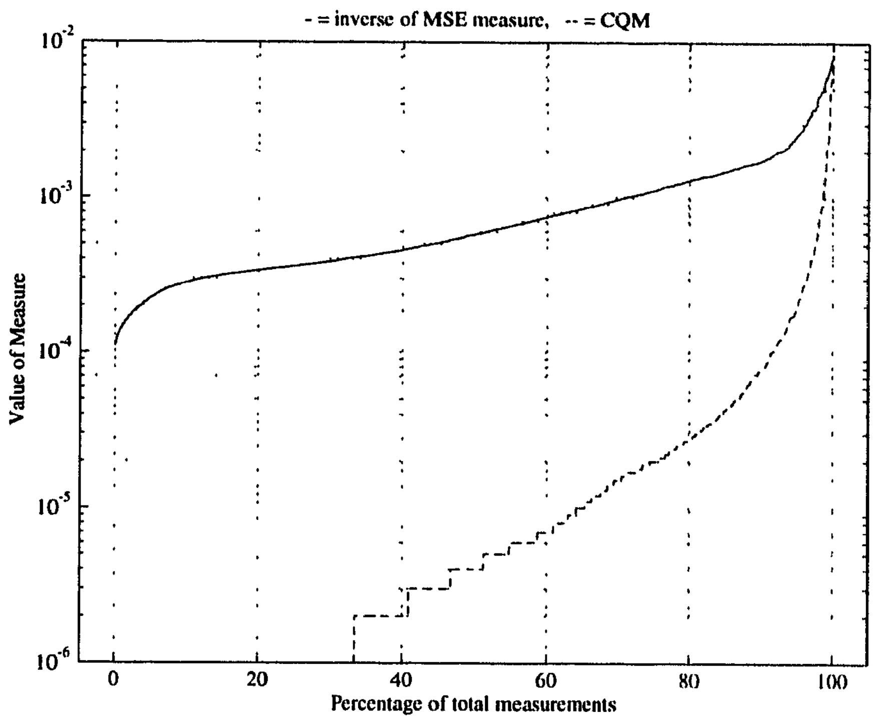
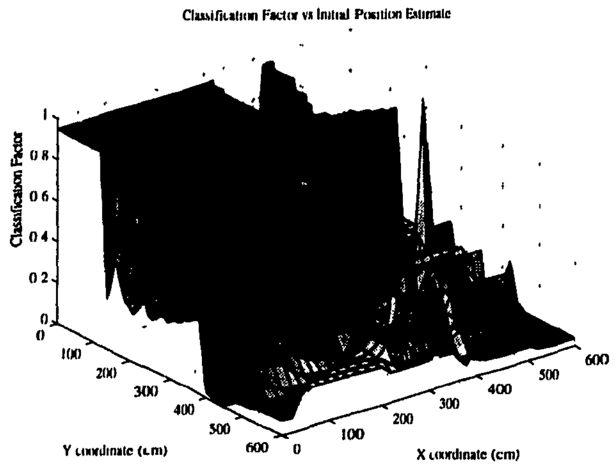


Figure 6.5: A Comparision of the Ranges of E_{mse} and E_{cqm}: a=2, b=1

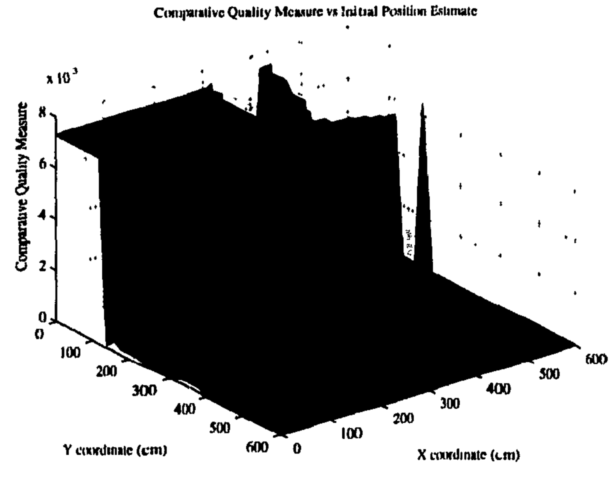
6.2.4 Verification of Quality Measures

One way we can verify the correctness of the quality measures defined thus far is to compare them to the data obtained when the correct robot position was known.

In figures 6.1, 6.3 and 6.4 the domain of the quality measures was simply the area around the actual robot position. In contrast, figure 6.6 shows E_{cf} factor and E_{cqm} vs *initial* position estimate - position correction is performed at each location and the quality of the *converged result* is shown. It is important to notice that the maximum values of these plots for each measure form a region that is almost identical to the region of convergence in figure 5.8 in chapter 5 (it would be identical except for one extra peak in the quality plot). In that figure it was known a priori where the correct position was, but in figure 6.6 this is not known in advance, yet the region of convergence is the same. This simply means that those position estimates around the actual robot position that can be corrected are all of maximal quality.



(a) E_{cf}



(b) E_{cqm}

Figure 6.6: Quality of Converged Solutions vs Initial Position Estimate: Position Correction is performed on each initial position, and the quality measures of the resulting solution are calculated. Therefore, each initial position is mapped to a final position quality measure.

6.3 Summary

This chapter presented an approach to the measurement of the quality of an estimate of robot pose. Two base measurements were presented: the mean-squared error measure (E_{mse}) and the classification factor (E_{cf}). E_{mse} is a measure of the distance between a set of range data and a map (consisting of line segment models), while E_{cf} is a weighted count of the number of range data points that are close to their respective line segments. The advantages and disadvantes of both were presented. A third measure, the comparative quality measure (E_{cqm}), was in fact derived from the first two in an attempt to emphasize the global maximum.

E_{cqm} was presented as the measure to be used for maximizing quality with respect to robot pose. E_{cf} was better suited for absolute tests, where a particular solution of correction could be deemed incorrect or correct based on this unitary scaled measure. When these measures were compared to tests in which the correct position of the robot was known, the results were shown to be the same.

Chapter 7 Extensions of Position Correction

So far we have seen that position correction can correct the position of a robot given a good map and a fair position estimate, and that the results can be confirmed with the E_{cf} quality measure. In this chapter two applications of this will be examined:

1. **Orientation Correction** In addition to correcting positional error, this technique corrects errors in orientation.

2. **Global Localization** When a map of the environment is available but estimates of position and orientation are not, this technique still allows localization of the robot from anywhere in the environment.

7.1 Orientation Correction

Orientation correction is similar to position correction in that it takes an estimate of the pose of a mobile robot, a set of range data measurements and a map of the environment and attempts to minimize the error in the estimate. However, in orientation correction, the orientation of the robot is no longer assumed to be completely accurate, and must be corrected in addition to position. As outlined in chapter 3, errors in robot orientation can have equally disastrous effects on the construction of global coordinate maps as do positional errors, so it is very desirable that these kinds of errors be minimized.

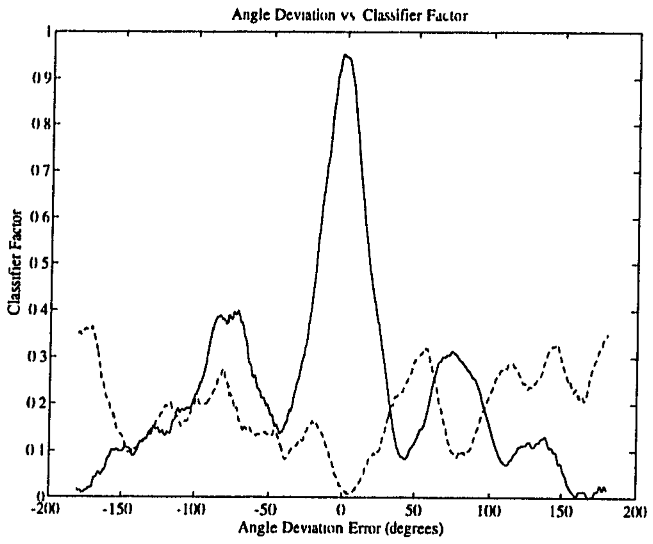
The approach to orientation correction is based on two quality measures, the classification factor E_{cf} and the comparative quality measure E_{cqm}. Consider Figure 7.1: Near the true orientation (an angle deviation error of 0°), (a) E_{cf} and (b) E_{cqm} are well-behaved and appear to be locally convex functions of angle deviation θ_d. Therefore, if the estimate of the robot's position is close enough to the actual position, then the proper orientation of the robot can be found by maximizing E_{cqm}. Since we are assuming that a given pose estimate is good if its E_{cf} value exceeds the acceptance threshold (as discussed in section 6.2.2), we can conjecture that when the estimate is good enough, E_{cqm} will be a locally well-defined, single-peak function. For comparative purposes, the angle deviation test of figure 7.1 was done at two positions: the correct position, and an incorrect position (in this case about 100 cm away). This way we can see that both the position *and* the orientation must be correct (or at least very close) in order for the quality measure functions to be well-behaved enough to be optimized. In position correction, the orientation estimate had to be correct for the actual position to be found. Orientation correction likewise ideally requires that the position estimate is correct, but this is not usually possible. However, if we assume that a small error in the position estimate will not change the optimum orientation by a large amount, then this constraint can be relaxed to allow the small positional error.

We only wish to correct orientation when the error in the position estimate is small, otherwise the quality measures will be too ill-behaved. Therefore, when attempting to correct orientation by maximizing E_{cqm}, it is useful to use a modified E_{cqm}, which we call $\hat{E}_{cqm}$, and defined as:

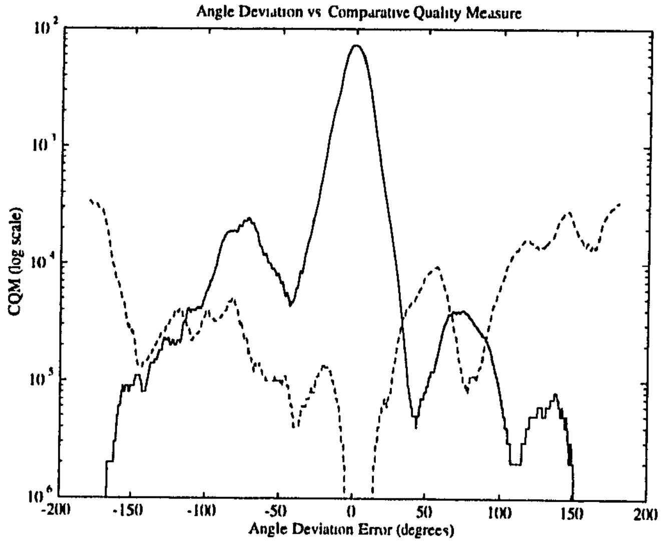
$$\hat{E}_{cqm} = \begin{cases} E_{cqm} & \text{if } E_{cf} \geq \text{Acceptance Threshold} \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

Figure 7.2 shows $\hat{E}_{cqm}$ taken from the same measurements of Figure 7.1, and using a classification factor acceptance threshold of 0.5. $\hat{E}_{cqm}$ is simply the well-behaved region of E_{cqm}, which we found not by looking at a plot such as Figure 7.1 (since the robot would not have access to such a plot), but by applying the classification factor acceptance threshold. Thus, we can ensure that we only attempt orientation correction when E_{cqm} is well-behaved enough to allow gradient ascent to its global maximum.[1] Where the position estimate is such that E_{cf} exceeds the acceptance threshold, we avoid the numerous local maxima of E_{cqm} for angle deviations where

[1]This assumes that gradient methods are used for global maximization - we assumed that the brute force approach of calculating E_{cqm} over the entire domain and comparing would be too time-consuming.



(a) Angle Deviation vs. E_{cf}



(b) Angle Deviation vs. E_{cqm}

Figure 7.1: Variations in Quality Measures as functions of Angle Deviation: with the true robot orientation at an angle deviation of 0°, the quality measures decrease as the error in the angle grows larger. For this environment, the *effective angle range* is about 35°, given an E_{cf} threshold of 0.5. The solid line represents the quality measures at actual robot position and varying orientation, while the broken line represents the quality at an incorrect position.

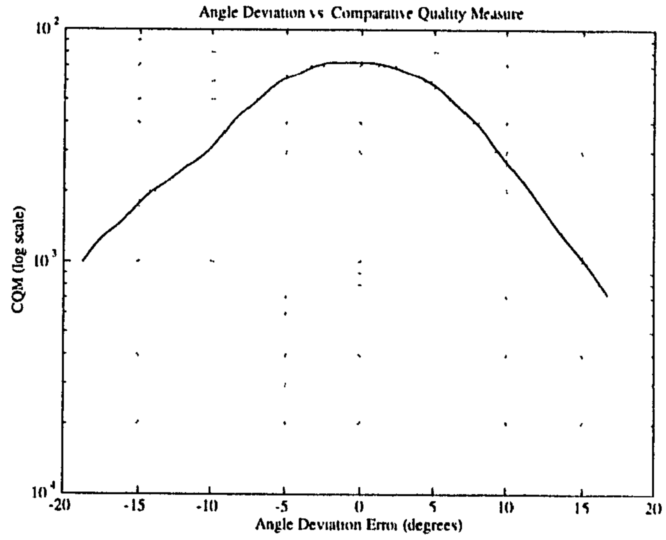


Figure 7.2: $\hat{E}_{cqm}$ is defined over a smaller domain where any angle deviation may be corrected

E_{cf} is too small.

To get some indication of the maximum orientation error that can still be corrected, we can look at the *effective angle range*.

Definition 2 *The* effective angle range *(EAR) is that range of angle deviations about the true robot orientation for which E_{cf} exceeds the acceptance threshold. Any angle deviation in this range is small enough to allow the correct orientation to be found using gradient ascent maximization. This range is not necessarily symmetric about the correct orientation.*

The effective angle range (EAR) for the actual pose Figure 7.1 and Figure 7.2 is about 35°: here angle correction succeeds if the position estimate is error-free and the oriention error is between about $-18°$ and $+16°$. This range is reduced when the position error is increased, but for position estimates close enough to the actual position to exceed the E_{cf} acceptance threshold, experimentation has shown that in general an EAR tends to be on the order of $-10°$ to $+10°$.

Now that we know that it is possible to correct the robot's orientation as well as its position given fair initial estimates, we can construct the following algorithm to perform localization:

1. Initial pose estimate (x_0, y_0, θ_0) available.

2. Perform position correction: given that (x_i, y_i, θ_i) are the results of each iteration, for each iteration find $\hat{E}_{cf}(x_i, y_i, \theta_i)$.

3. If $\hat{E}_{cf}(x_i, y_i, \theta_i) > 0$, then hold x_i and y_i constant and maximize $\hat{E}_{cf}$ as a function of θ by using a non-explicit-derivative-using maximization technique such as Brent's method [29].

4. When the maximum is found, update the orientation estimate θ_i.

5. If θ_i changes very little over the course of a few iterations, ignore the orientation correction steps 3 and 4 in future iterations and concentrate on refining position only. This is done to increase the speed of the algorithm once the orientation has been corrected.

7.2 Global Localization

Global localization refers to the case where a map of the environment is available, but a reliable estimate of the robot's pose is unavailable. To use the localization algorithm developed thus far, a pose estimate is required. Therefore, we can incorporate local localization into global localization in a similar fashion to the way position correction was incorporated into orientation correction: by use of the quality measures E_{cf} and E_{cqm}. In theory, to find the actual pose of the robot we need to perform localization at every pose in the environment and select the result with the highest quality.

We can consider all possible poses (x, y, θ) of the robot within the environment that the map represents as the domain of a *composite* quality function

$$F(x, y, \theta) = \hat{E}_{cqm}(x', y', \theta') \tag{7.2}$$

where x', y' and θ' are the results of the local localization algorithm of section 7.1 applied to x, y and θ. F is called composite because localization is done to (x, y, θ)

before calculating $\hat{E}_{cqm}$, making F akin to a function of a function.

This gives a global quality function that describes the quality of localization when applied to a particular location. In theory we can imagine F calculated for all (x, y, θ) in the environment, and can find the pose of the robot by maximizing F. The actual robot pose would then be (x', y', θ') (from equation 7.2).

The problem with maximizing F is that $F = \hat{E}_{cqm}$, and from the previous section we saw that the global maximum of $\hat{E}_{cqm}$ can only be found by gradient methods if gradient ascent is initiated close to the global maximum. Since we cannot use gradient ascent over the domain of the entire environment, we have no choice but to resort to *sampling* the environment until we find a pose where $\hat{E}_{cjm} > 0$, at which point we can use gradient ascent. Sampling in this context means applying localization to selected poses in (x, y, θ) space and checking the value of $\hat{E}_{cqm}$.

The issue of the sampling size can be resolved with the concept of the region of convergence (section 5.2.3) and the effective angle range (or EAR, in section 7.1). The sizes of these can tell us how far apart we may place the localization samples, since by definition any pose estimate within these regions can be corrected. However, the region of convergence is dependent on the arrangement of line segment models in the map and the robot's location within the map. Similarly, the effective angle range changes with the error in the pose estimate, and neither of these can be explicitly calculated to aid in localization (if they were known, the problem would be solved).

Through experimentation we can find some general values for the sampling spacing by performing localization at known poses and finding their regions of convergence and EARs. Samples could then be spaced accordingly and the poses within the area of the map could be tested. If no solutions are found, then either the map and the environment do not correspond, or the sampling spacing was too large. If the latter is true, the sampling spacing could be reduced and the sampling could be repeated. It is only a matter of calculation time in resampling the poses in the area of the map.

Experiments have shown that in most sufficiently occupied environments[2], posi-

[2]Here the term "sufficient" refers to environments that contain enough objects to uniquely identify individual positions within the environment. As will be shown in chapter 8, similarities between

tion correction convergences successfully within about 50cm, and orientation correction in a $-10°$ to $+10°$ range. Possible exceptions are in maps with a high density of line segment models (rooms that are very cluttered with non-planar surface objects, for example) or maps with a very low total number of line segment models within sensor range of the robot (where the robot cannot see enough of its surroundings to make accurate comparisons).

Once the sample size is known, the straightforward approach would be to sample the entire environment at poses separated by the sample size. However, for large maps this could require very many samples, the majority of which do not find the correct pose. So, in the interests of saving computation time, one may *super-sample* the environment. This strategy is a coarse-to-fine approach: initial samples are placed far apart, and if no satisfactory results are found from these samples, more samples are taken with smaller inter-sample distances. If there are still no satisfactory results, then the process can continue with smaller and smaller sampling sizes, down to the minimum size as dictated by the regions of convergence and EARs as mentioned above (figure 7.3). In the worst case, complete sampling of the environment would be done at the minimum sampling sizes.

Of course, it would be a waste of time to attempt localization at poses already attempted when the sample size was larger. Therefore a list of attempted poses should be kept and compared to any new pose to ensure each pose is tested only once.

Results of global localization are presented in section 8.4 in the next chapter.

different areas in an environment can lead to an incorrect localization

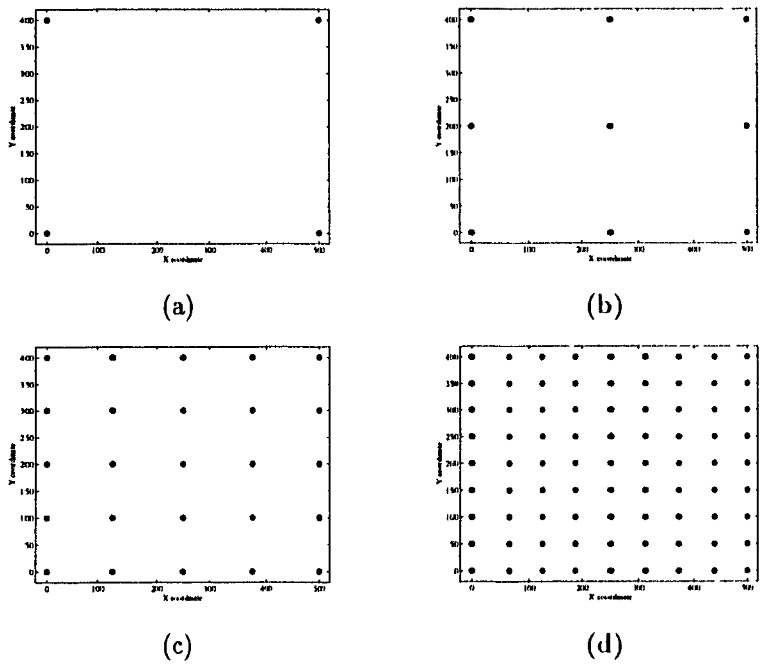


Figure 7.3: Sample Size Reduction: a coarse-to-fine search strategy: in (a), the sample size is large. If no satisfactory localization is found, then the environment is resampled at the smaller and smaller sampling sizes in (b), (c) and (d).

7.3 Summary

This chapter discussed two applications of the quality measures E_{cf} and E_{cqm} over and above verification of a single localization attempt. These applications were orientation correction and global localization.

In orientation correction, the properties of the quality measures were examined for angle deviations around the true orientation. It was found that they were locally convex functions in the neighbourhood of the true orientation, enabling gradient-using optimization methods to find the global maximum that occurs there. The size of this neighbourhood was introduced as the *effective angle range*, or EAR, which bounds the angle error relative to the actual orientation that can still be corrected. A modified comparative quality measure $\hat{E}_{cqm}$ was introduced to aid the optimization process. This measure equals E_{cqm} when E_{cf} is above an acceptance threshold, ensuring that it forms a convex function (although not a strictly convex function).

Given this method to correction orientation, a complete algorithm for local localization (where a pose estimate is given) is given.

The complement to local localization was introduced as global localization, where the robot has no prior estimate of its pose. Following the same reasoning as was done for orientation correction, a function F was constructed using the quality measures whose global maximum occurs at the robot's actual pose. As was the case with orientation correction, the function is locally convex only at the correct pose, but this time we have no estimate of this pose. This discounts gradient methods for optimizing F.

A simple approach to solving this problem was described in terms of sampling the environment represented by the map at different sizes. Initially, samples are sparsely located, and localization is attempted at these poses. If no satisfactory results are found, then the environment is resampled at more densely located samples. This process continues until a satisfactory localization is found, although the sampling sizes should not decrease below the sizes estimated by the regions of convergence and EARs common for that environment.

Chapter 8 Experimental Results

As the theory of localization was developed in previous chapters, the real environment of figure 4.11 was used to illustrate localization's various aspects. In this chapter another real environment and two simulated environments will be assessed in terms of the performance of localization within them.

8.1 Map Construction Results

In this section, map construction in three sample environments is presented. In each of these environments, the robot (virtual, in the first two) wandered about, scanning every few centimetres and incrementally built a map. In addition, localization was performed every few moves to be sure that small positioning errors did not accumulate (on the average, every other move – this was an ad hoc choice, trading off the accuracy of localizing every move with the time required to localize). For the simulated environments, artificial errors were introduced to simulate positioning errors. Uniformly distributed errors in the [0,1] cm range were added to the x and y coordinates every 10cm of movement, as well as [0°,1°] in orientation. For simplicity, the robot used an uncomplicated wandering algorithm for exploration: keep going in a straight line until range measurements indicate an object is too close, at which point randomly pick a new heading and continue. Certainly better exploration algorithms do exist ([35], for example), but are beyond the scope of this thesis.
For the two simulated environments, three maps were constructed:

1. Ideal Sensor Map: for this ideal case, the sensor is modeled as a thin straight line extending from the robot, where the range measured is from the robot to the first object hit, regardless of the angle of incidence.

2. Sonar Sensor Map: this sensor models the properties of sonar, including multiple reflections and incidence angle effects, as per [9].

3. Corrected Sonar Map: these maps use the same sonar sensor as above, but the line segment models of the map are shortened to account for the size of the sonar cone, as described in section 4.3.4.

Figure 8.1 shows a simple room with similarly appearing corners and three obstacles, the largest of which creates a hallway at the bottom of the room (for this map and all such maps in this thesis, the coordinate system is non-standard: the x-axis proceeds from left to right, the y-axis from top to bottom, and orientation is counterclockwise, with 0° along the positive y-direction – the origin would be therefore in the upper right corner of the map.) Figure 8.2 is the second of the two simulated rooms, this time a little more complex, containing a number of "sub-rooms" within it. Both figures show some difficulty with modeling corners with the ideal sensor. This effect comes from the splitting phase of fitting models to objects (section 4.3.2), where a cluster of range data points may not split at a corner, or where a subcluster's elongatedness is acceptable at 45° to the corner walls, yet too small to split. The sonar sensor maps without correction show the effects of misjudging the size of objects. Here we can see the apparent sides of objects extended far past their true borders, as well as a flurry of noisy line segments in the corners of the rooms. The corrected maps reduce this border-extending effect to produce a clearer map.

Figure 8.3 shows the results of a map made from an environment with physical objects. Whiteboards, together with wooden, metal, and cardboard objects were placed in an area enclosed by office dividers and walls to provide a variety of surface reflectivity types. The images of the area in figure 8.4 show details of the area, including the tiled floor surface which adds error to the robot's pose as its wheels pass over the tile edges, in addition to slippage errors common to smooth floors.

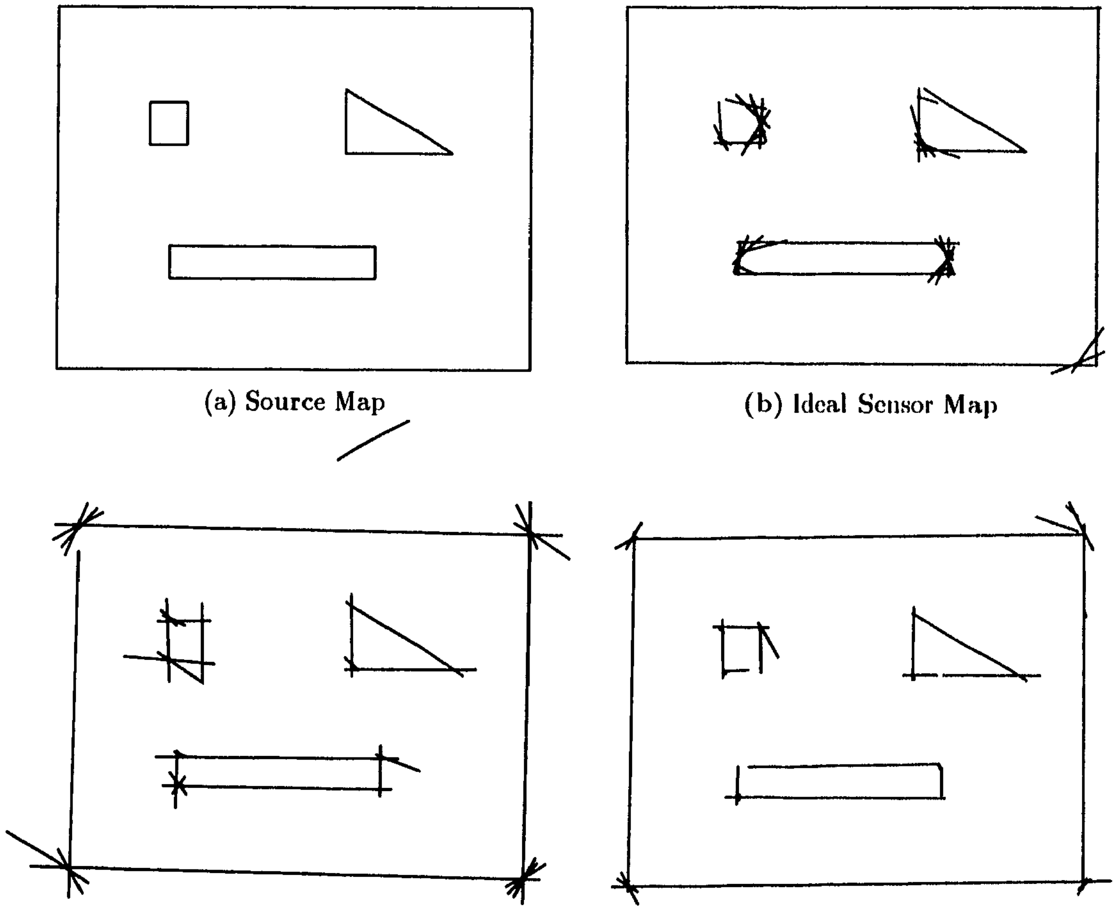
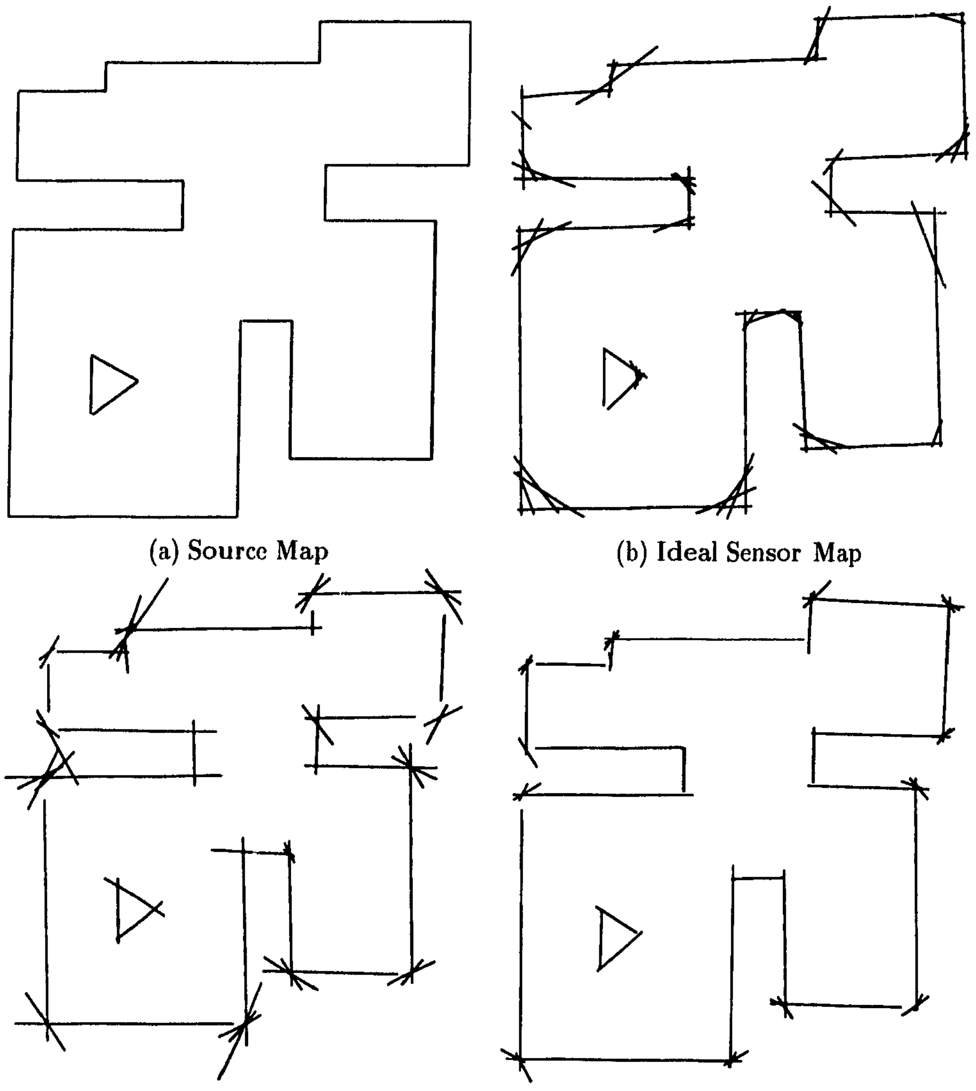


Figure 8.1: Room 1 (simulated): from the source map of (a), 3 maps were constructed: (b), the ideal range sensor map; (c), a map made with a sonar simulator; (d) a map made with the sonar simulator but corrected for the shape of the sonar cone



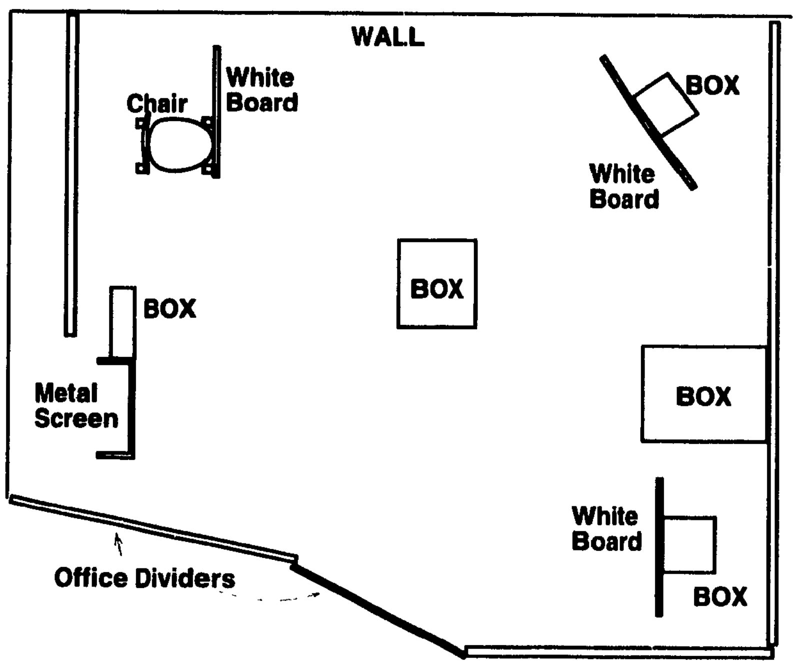
(a) Source Map

(b) Ideal Sensor Map

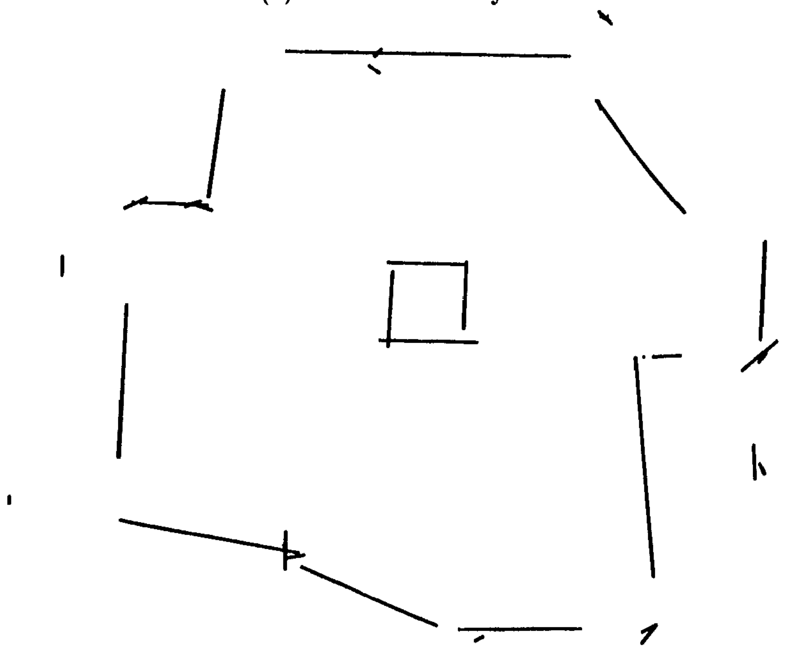
(c) Sonar Sensor Map

(d) Corrected Sonar Map

Figure 8.2: Room 2 (simulated): from source map (a), (b) is the map made with an ideal sensor, (c) shows the sonar simulator map, and (d) shows the sonar-cone-corrected sonar map.



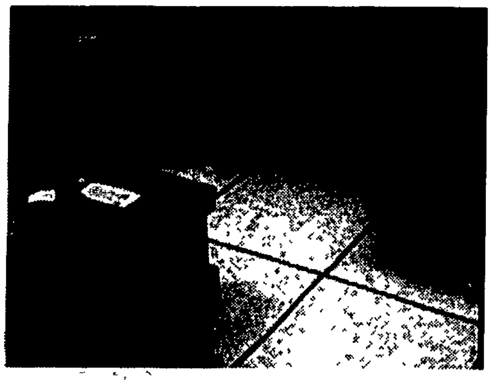
(a) Environment Layout

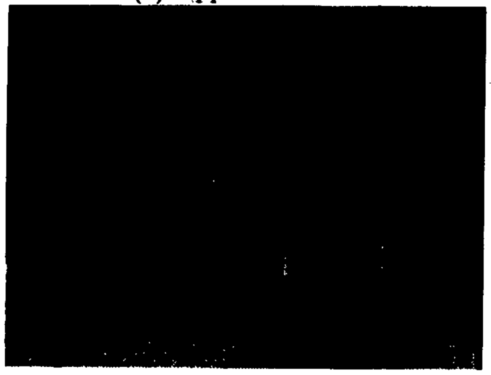


(b) Constructed Map

Figure 8.3: Room 3 - A Real Environment: the layout of the environment is as shown in (a), and (b) shows the map constructed from sonar data with line segment length correction.

(a) Upper Left View


(b) Upper Right View


(c) Lower Left View

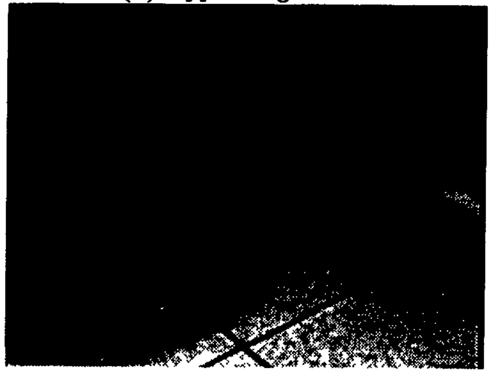

(d) Lower Right View

Figure 8.4: Details of Room 3: these views of the environment show the types of objects present

8.2 Quality Measures

Since the quality measures E_{cf} and E_{cqm} indicate how effective localization is within a particular environment, it is useful to use them to examine the three example environments.

8.2.1 Room 1 (Simulated)

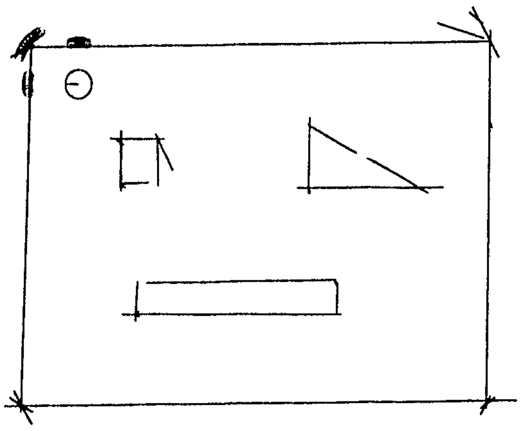
To compare sonar with an ideal range sensor, consider figure 8.5 and figure 8.6 which examine the quality measures of range measurements taken at the example pose shown. As seen in section 6.2.2, each (x,y) point indicates the quality that would be obtained if that point was a localization convergence point, and if there was no error in the orientation[1]. The figures show that all plots have a single dominant peak indicating the true robot position; however, the surrounding positions in the two sonar maps are noisier, especially in the E_{cf} plots.

In order to examine the effects of changing orientation, we can hold one of the x or y coordinates constant and compare the other with changing orientation (figures 8.7 and 8.8). Here we can see convex shape of the measures near the true pose. The peak in figure 8.7(d) is rather noisy and is not convex, and so the local maximum near the global maximum may affect the results. They are however very close together which in this case does not present too much of a problem.

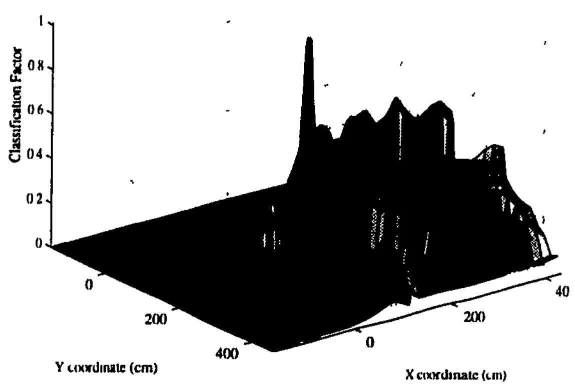
From this point on when the sonar maps are mentioned, they will refer to the *corrected* sonar maps, as the corrected sonar maps are the ones used in real environments.

It is useful to see how a pose within a relatively sparse hallway fares with respect to the quality measures. In figure 8.9, orientation θ is held constant, and x and y are varied. From the locations of the range points in (a) one can see that for the sonar map, the high quality peak forms a line rather than a sharp point. The ideal map has a more rounded peak since the ideal sensor can see more of the hallway, but it is still spread out parallel to the hallway. In (e) one would expect a less rounded peak,

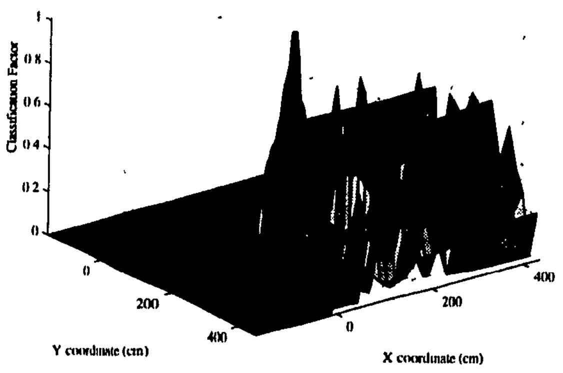
[1]ideally we would like to plot (x, y, θ) vs. quality, but this would require a 4-dimensional plot



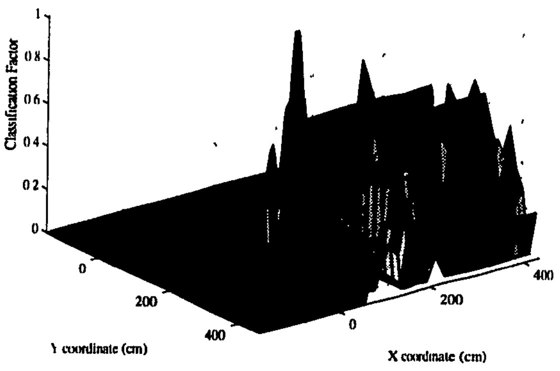
(a) Example Pose 1



(b) E_{cf}, ideal sensor



(c) E_{cf}, sonar sensor

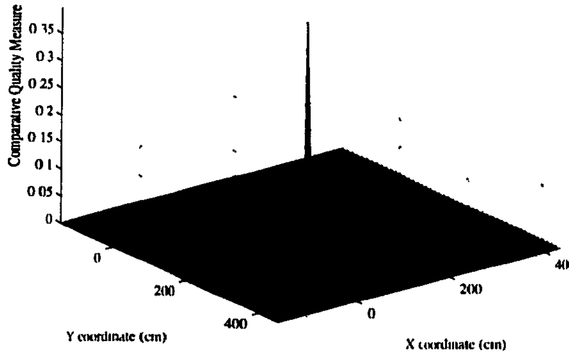


(d) E_{cf}, corrected sonar sensor

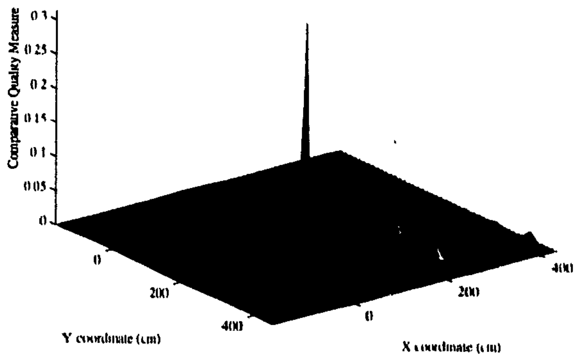
Figure 8.5: E_{cf}, Room 1, Pose 1, x vs. y: with the robot at the position marked in (a), the quality of the range measurements varies over the x-y plane. The sonar map is shown in (a) rather than the source map to show how the range data relates to the map the robot constructs. The small circles indicate the range data. For the ideal sensor map, the range data points follow the walls of the source map exactly. In (b)-(d), the robot's actual position is at the centre of the xy-plane.

but for this particular map, both sides of the hallway were not fit as perfectly parallel lines (since the error in the robot pose is not eliminated completely).

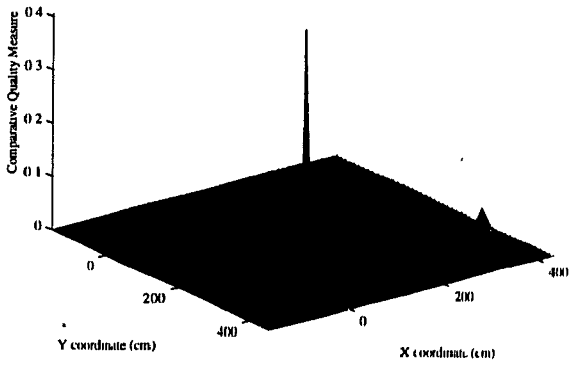
The same effect can be seen if the y-coordinate is held constant and θ is allowed to vary, as in figure 8.10. As the robot and data points turn to face down or up (as seen on the room map), the range points are no longer close to any walls, thus the low quality. One interesting note for this case is that there are two peaks along the θ axis,



(a) E_{cqm}, ideal sensor



(b) E_{cqm}, sonar sensor

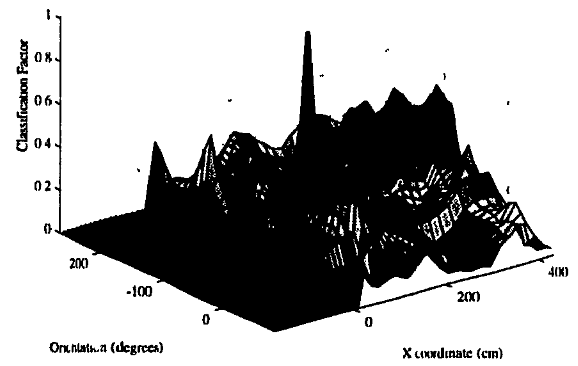


(c) E_{cqm}, corrected sonar sensor

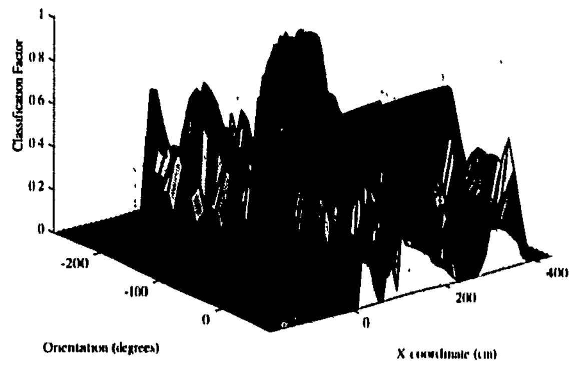
Figure 8.6: E_{cqm}, Room 1, Pose 1, x vs. y: the sharp central peaks indicate that if orientation is held fixed, the highest quality coincides with the actual robot position.

which correspond to the robot facing left or right on the map. Both face parallel to the hallway, and so are very similar. They are less similar in the sonar map because of the imperfectly fit borders of the hallway.

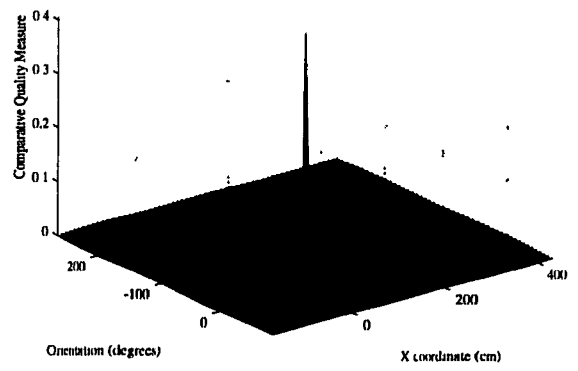
If we compare variations in y-coordinate and orientation θ, the hallway effect is invisible (figure 8.11). Except for some similarity between 90° and −90° in (a), (b) and (d), the quality measures correctly identify the true (y,θ).



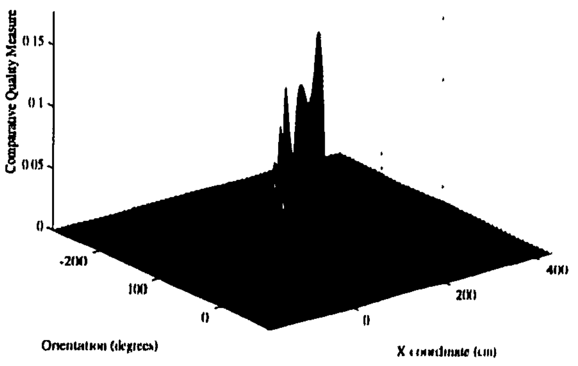
(a) E_{cf}, ideal sensor



(b) E_{cf}, sonar sensor

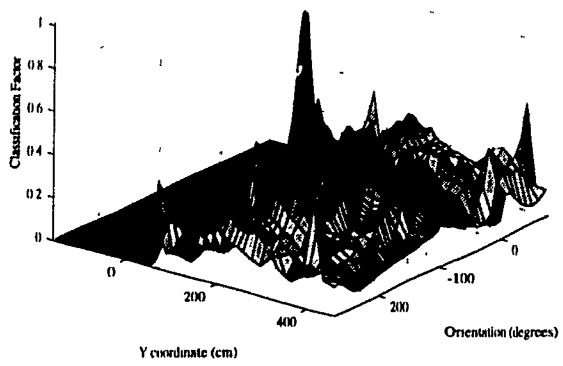


(c) E_{cqm}, ideal sensor

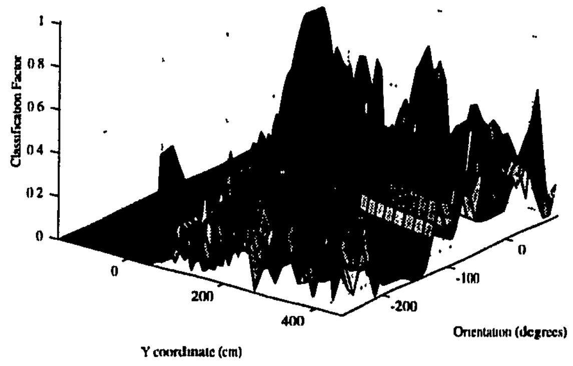


(d) E_{cqm}, sonar sensor

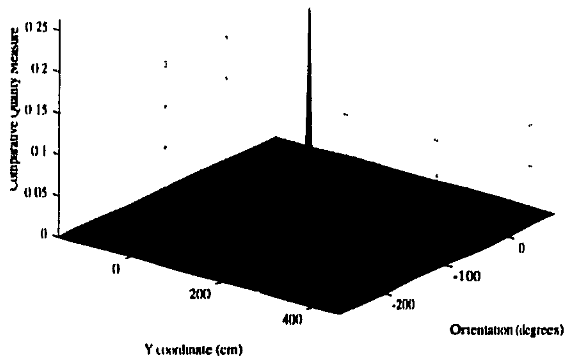
Figure 8.7: Quality Measures, Room 1, Pose 1, x vs. θ: again the central peak indicating the true pose dominates, but in (d) this peak is not convex and is incorrect.



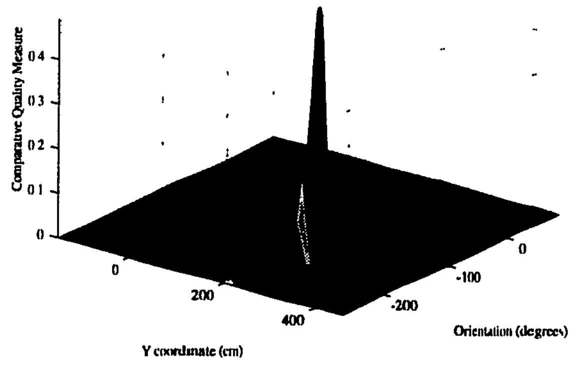
(a) E_{cf}, ideal sensor



(b) E_{cf}, sonar sensor

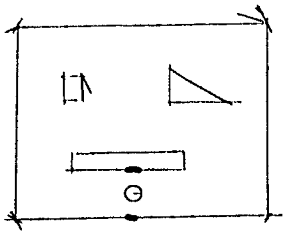


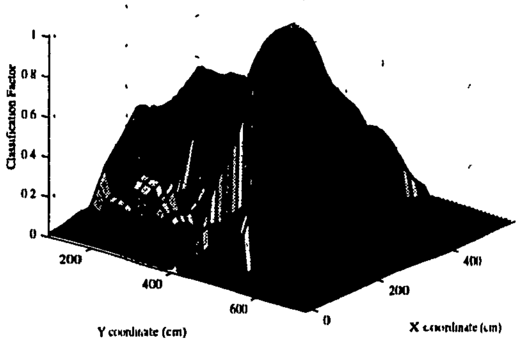
(c) E_{cqm}, ideal sensor



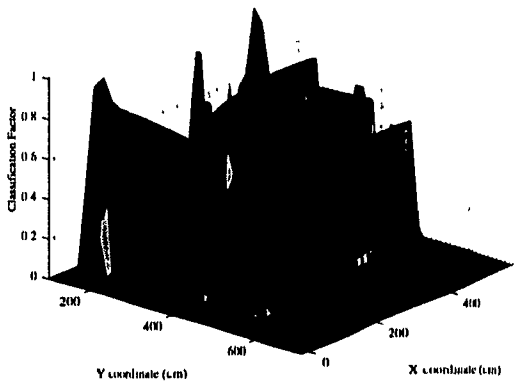
(d) E_{cqm}, sonar sensor

Figure 8.8: Quality Measures, Room 1, Pose 1, y vs. θ: the global maximum is not as sharp in (b), since the small clusters of data points are close to the map's line segments even for larger than usual orientation deviations.

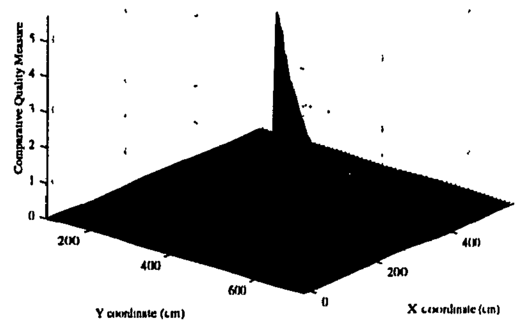

(a) Example Pose 2



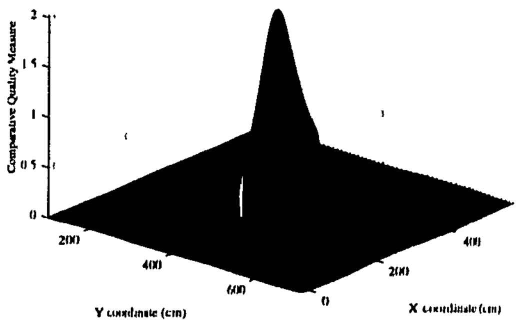
(b) E_{cf}, ideal map



(c) E_{cf}, sonar map

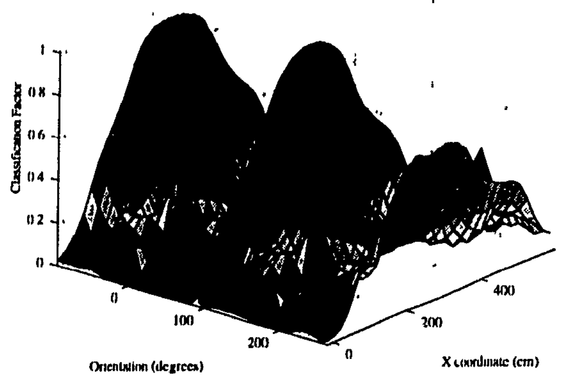


(d) E_{cqm}, ideal map

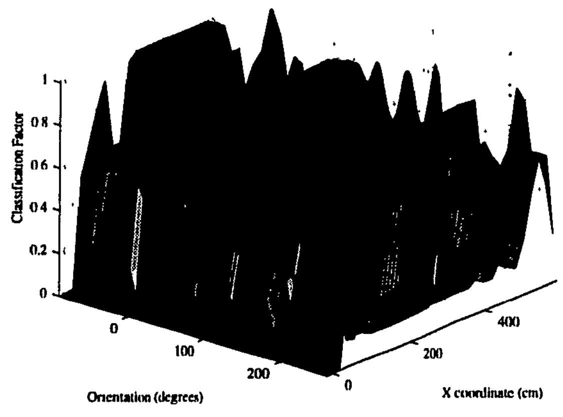


(e) E_{cqm}, ideal map

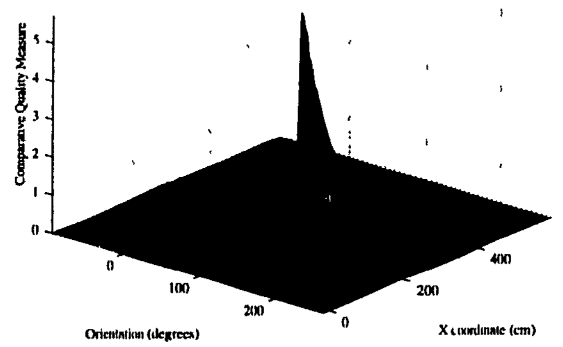
Figure 8.9: Quality Measures, Room 1, Pose 2, x vs. y: the global maximum may not necessarily be unique or completely dominant in long hallways as the robot position is moved parallel to the hallway. This is how one may recognize ***the long hallway effect*** as described in section 5.2.



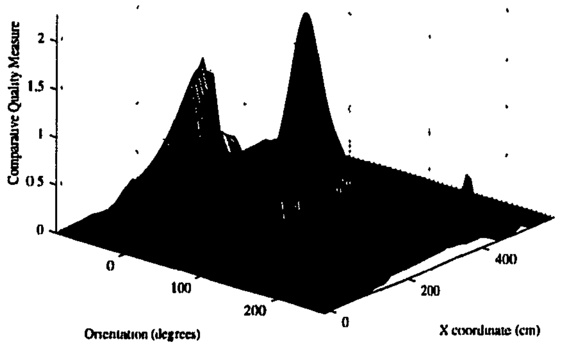
(a) E_{cf}, ideal map



(b) E_{cf}, sonar map

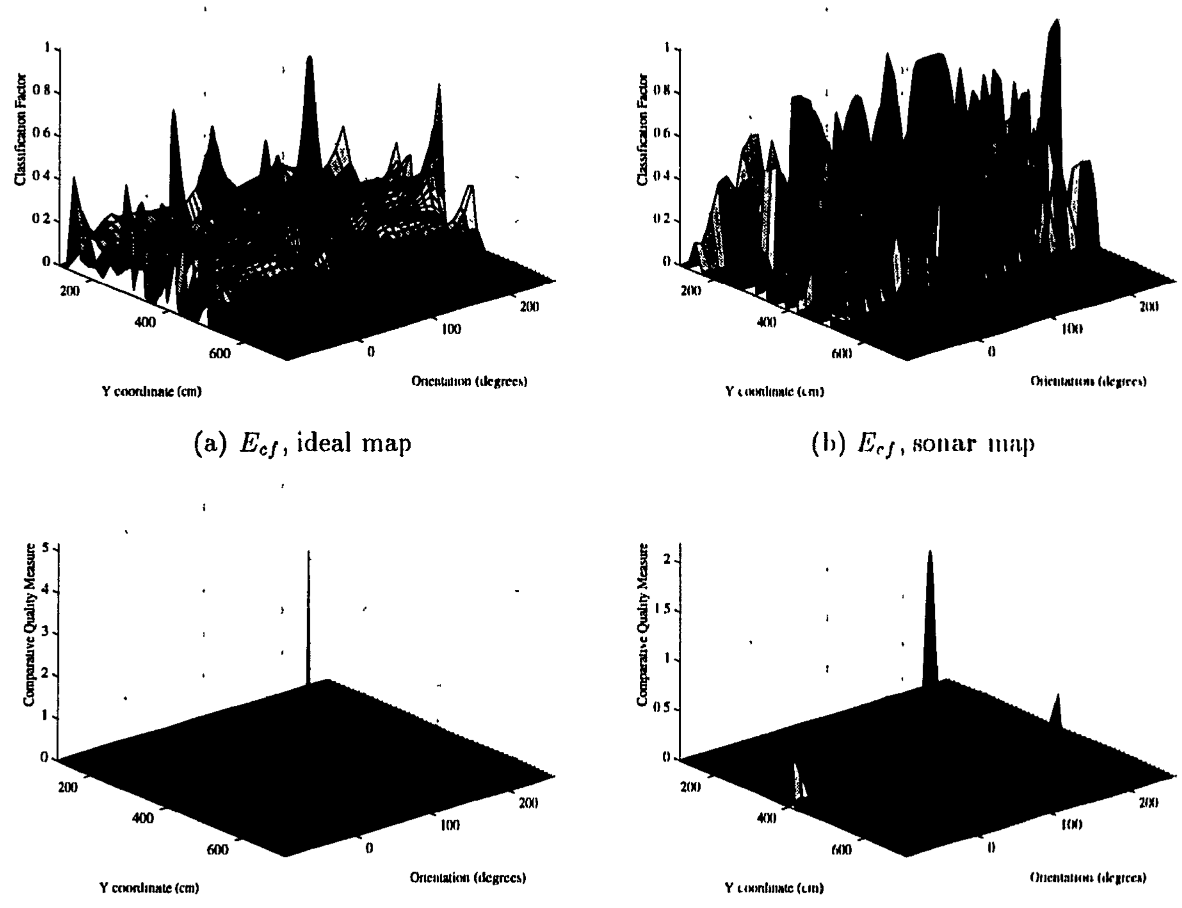


(c) E_{cqm}, ideal map



(d) E_{cqm}, sonar map

Figure 8.10: Quality Measures, Room 1, Pose 2, x vs θ: the long hallway effect is again visible as the x-coordinate is changed. This time there are 2 dominant maxima, one for each direction parallel to the hallway (i.e. down the hallway and back again).



(a) E_{cf}, ideal map

(b) E_{cf}, sonar map

(c) E_{cqm}, ideal map

(d) E_{cqm}, sonar map

Figure 8.11: Quality Measures, Room 1, Pose 2, y vs. θ: the long hallway effect is not as evident here since the y-axis is perpendicular to the hallway. However, the presence of significant local maxima at 180° to the global maximum suggest at least the presence of a short hallway.

8.2.2 Room 2 (Simulated)

If the robot is in a position where visible perpendicular walls surround it, then the robot should be able to localize itself accurately. Consider the pose of the robot in figure 8.12: figures 8.13 and 8.14 show the relationships between x, y and θ for this pose based on the sonar map. As expected, when the test pose is outside of the map area, both quality measures are effectively zero. The true pose is at the centre of each plot, and the global maximum exists at this point for all plots. In addition

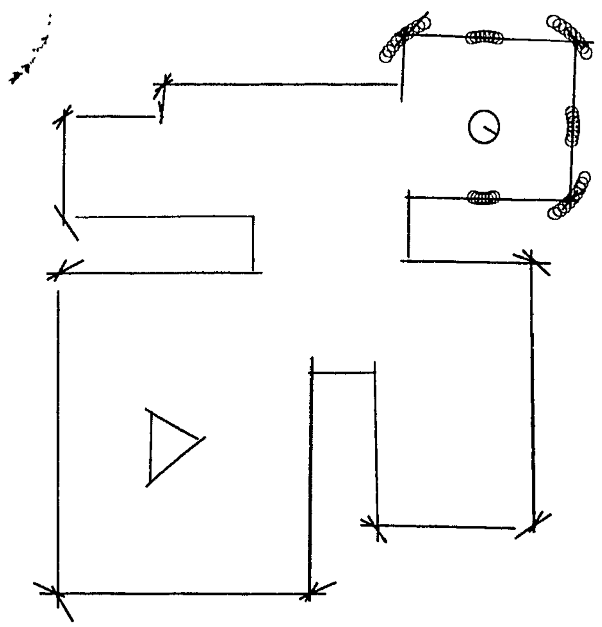
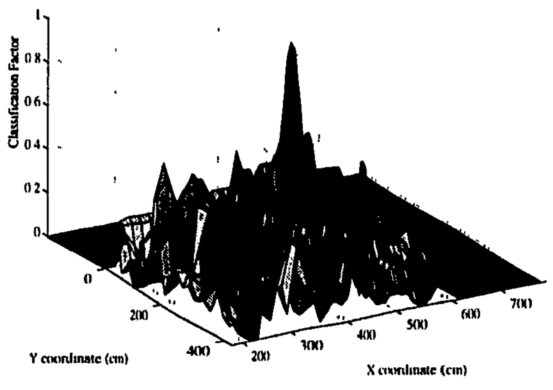
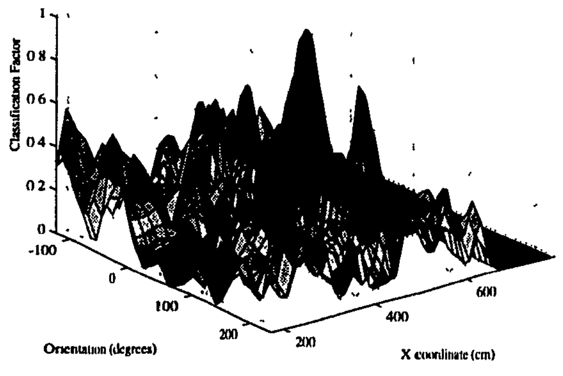


Figure 8.12: Room 2, Pose 1: the robot can detect the walls almost all around it, as well as corners.

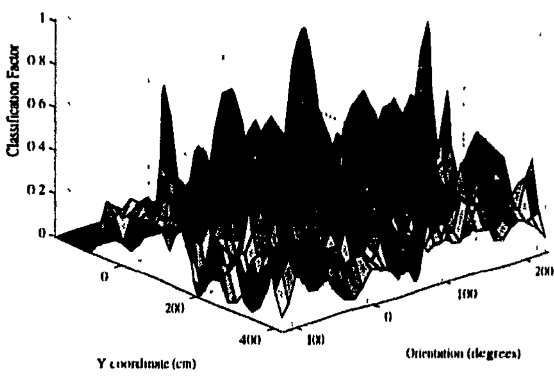
to verifying that the true robot pose implies a global maximum of a quality plot, the plots of E_{cf} can also identify similarities and/or symmetries in the environment. Figure 8.13(a) reveals 2 local maxima where $E_{cf} > 0.6$. The one with the greater y-coordinate refers to a position in figure 8.12 in the semi-enclosed area directly below the robot's present position, where the top right corner is similarly structured to this lower area. The other is not actually a structural similarity in the room, but rather a similar arrangement of line segments. This spot is in the upper left section of the environment, where the top wall and top left corner combine with the top wall of the



(a) E_{cf}, x vs. y



(b) E_{cf}, x vs. θ

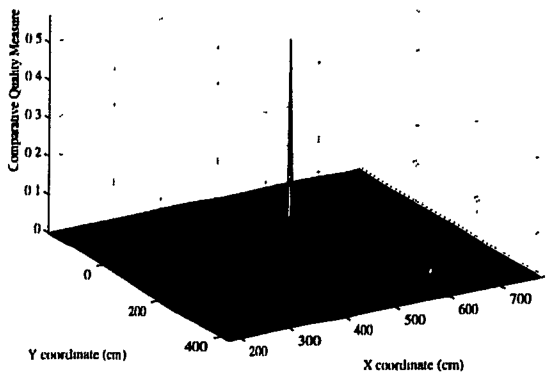


(c) E_{cf}, y vs. θ

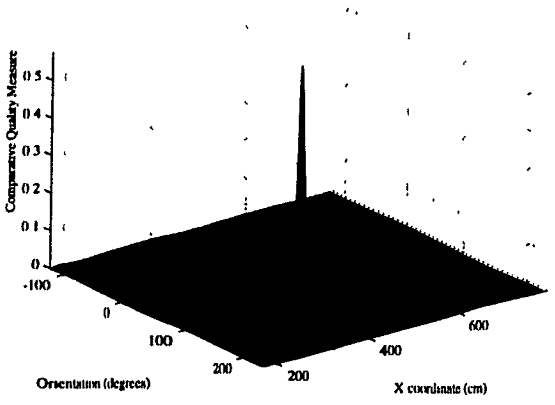
Figure 8.13: Classification Factor E_{cf} for Room 2, Pose 1

open space below (with the triangle). The plots of E_{cqm} do not have this property to the same degree, as they are much more discriminating, dealing with fine pose differences rather than the coarse quality of E_{cf}.

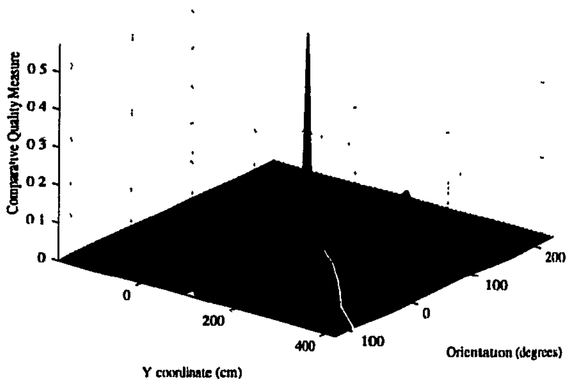
The lower right and upper left areas of Room 2 demonstrate properties very similar to the pose in figure 8.12, primarily the sharp global maximum coinciding with the robot's true pose. The middle section joining these areas exhibits this also, except that no inside corners are visible. However, at the true pose in figure 8.15 (lower left robot position with small circles indicating range measurements), incorrect results begin to appear, as figures 8.16 and 8.17 indicate, in particular 8.17(a) and 8.17(c). One of these high quality but incorrect poses is also shown in figure 8.15 with range



(a) E_{cqm}, x vs. y



(b) E_{cqm}, x vs. θ



(c) E_{cqm}, y vs. θ

Figure 8.14: Comparative Quality Measure E_{cqm} for Room 2, Pose 1

measurements appearing as x's. Here a cluster of the points seem to be in the middle of the room, but their distance from the infinite line through their target line segment (above) is small (recall that the component E_{mse} of E_{cqm} works with infinite lines). This is an example of a case where assuming the range points belong to an extension of their target wall fails, and explains why the E_{cf} plots are correct with respect to the global maximum while the E_{cqm} plots are not.

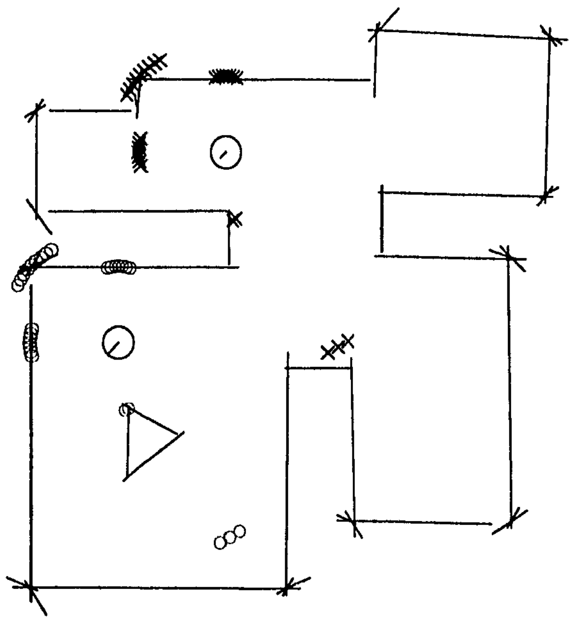
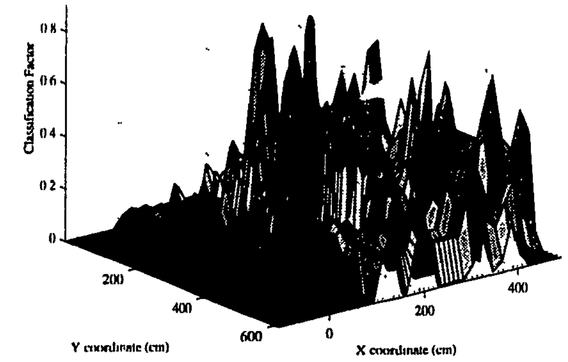
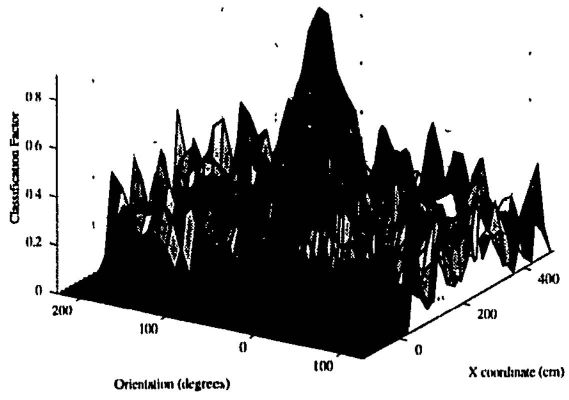


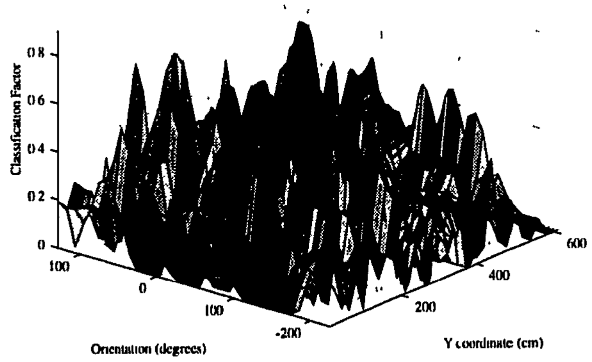
Figure 8.15: Room2, Pose 2: The actual robot pose is the one closer to the triangular object, and its range measurements are indicated by circles. The other robot pose shown has a similar surrounding structure (range points shown by x's).



(a) E_{cf}, x vs. y

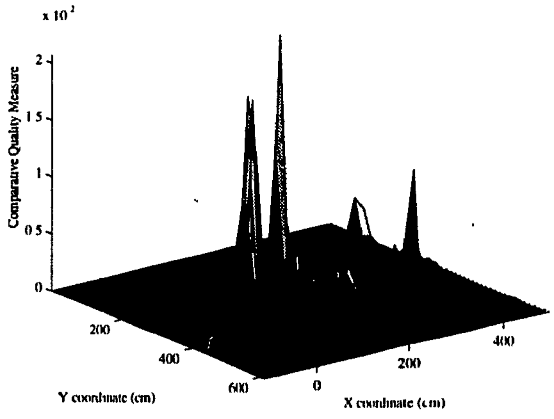


(b) E_{cf}, x vs. θ

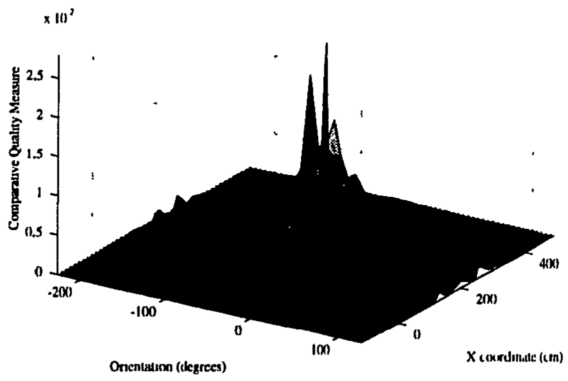


(c) E_{cf}, y vs. θ

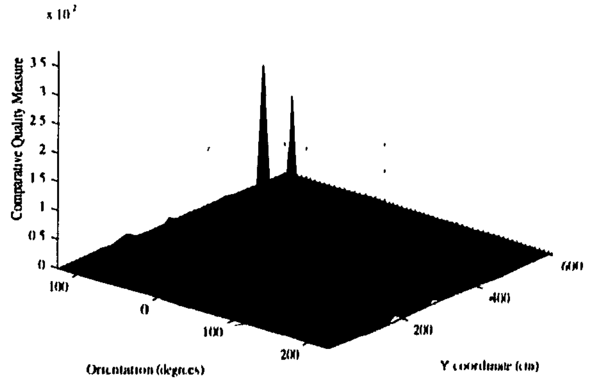
Figure 8.16: E_{cf} for Room 2, Pose 2



(a) E_{cqm}, x vs y



(b) E_{cqm}, x vs. θ



(c) E_{cqm}, y vs. θ

Figure 8.17: E_{cqm} for Room 2, Pose 2

8.2.3 Room 3

The environment shown in figures 8.3 and 8.4 differs from the previous two in that it appears to have few or no areas similar to each other. This would seem to suggest that the quality measures should clearly indicate where the robot's true pose is, given a set of range measurements. We will check this with three example poses. The first example pose, as shown in figure 8.18, is roughly in the centre of an open area. The actual position and orientation for all poses in this environment were measured by hand. Figures 8.19 and 8.20 show that localization can be correctly verified for this

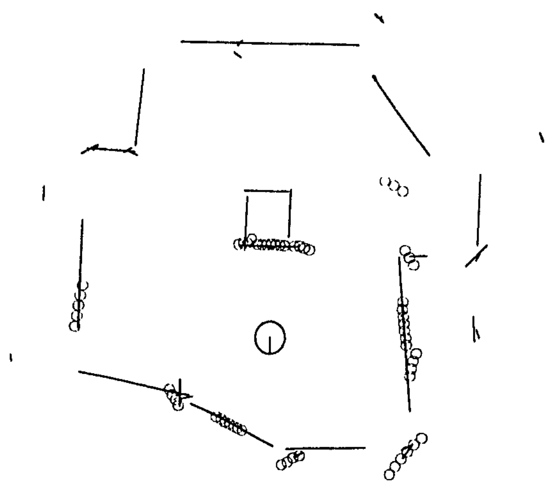
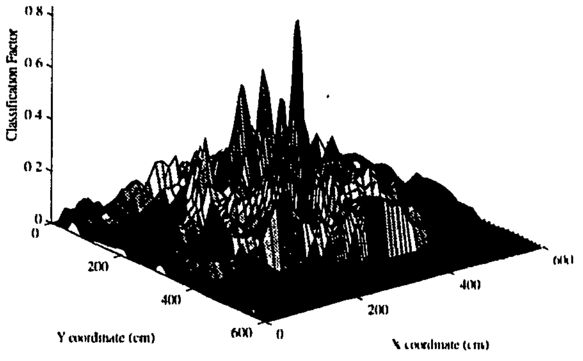


Figure 8.18: Room 3, Pose 1

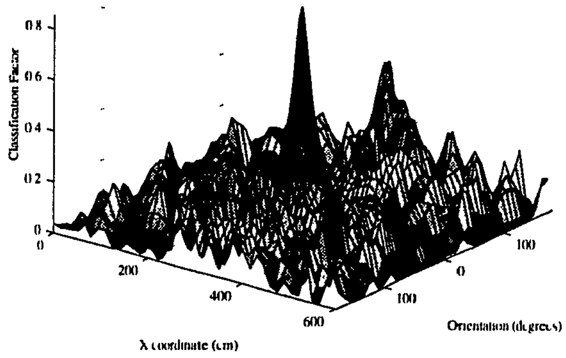
pose since both E_{cf} and E_{cqm} obtain their maximum at the correct robot pose. One observation to make here is that the peak values of both E_{cf} and E_{cqm} are lower than found in the simulated environments. This is likely due to sonar effects not included in the sonar model, such as differing reflectivity in surfaces and 3-dimensional effects.

The pose shown in figure 8.21(a) is another example of a pose that is verified by quality measures. (b)-(d) show E_{cqm} for Pose 2, and again the dominant peaks occur at the correct pose. E_{cf} is not shown since it is also similar to that of Pose 1.

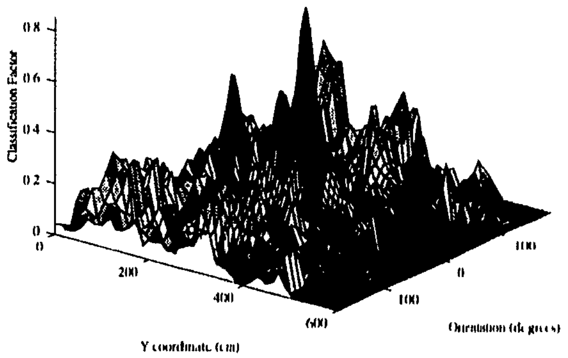
The last pose for which quality measures will be examined is shown in figure 8.22(a),



(a) E_{cf}, x vs. y



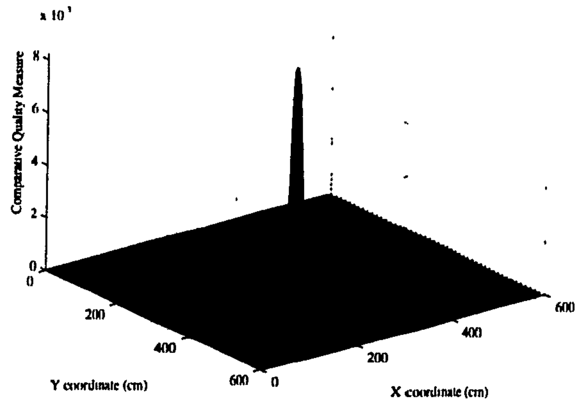
(b) E_{cf}, x vs. θ



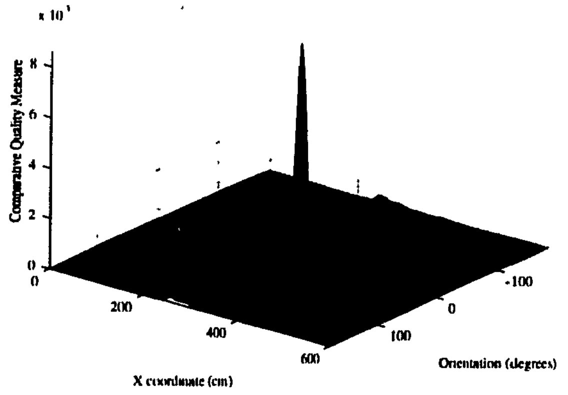
(c) E_{cf}, y vs. θ

Figure 8.19: E_{cf} for Room 3, Pose 1: E_{cf} is not as great at the peak as was seen in the simulations, yet the actual robot position in the centre is still the only position for which $E_{cf} > 0.6$

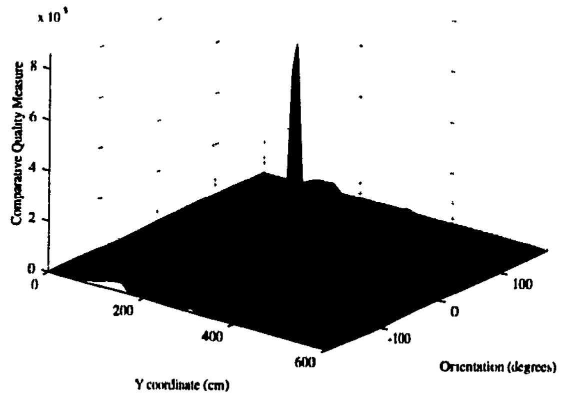
along with E_{cf}. Looking at the plots of E_{cqm} in figure 8.23 we can see that there is indeed a second peak that rivals the global maximum. The pose which gives rise to this peak is just above and to the right of Pose 1 in figure 8.18, where on the map the top border of the box is combining with the right and lower line segments to form a similar area to that which surrounds Pose 3. This is no cause for alarm however, since the modified comparative quality measure $\hat{E}_{cqm}$ as shown in figure 8.23(b) is a single peak maximum, which is used for orientation and global localization.



(a) E_{cqm}, x vs. y

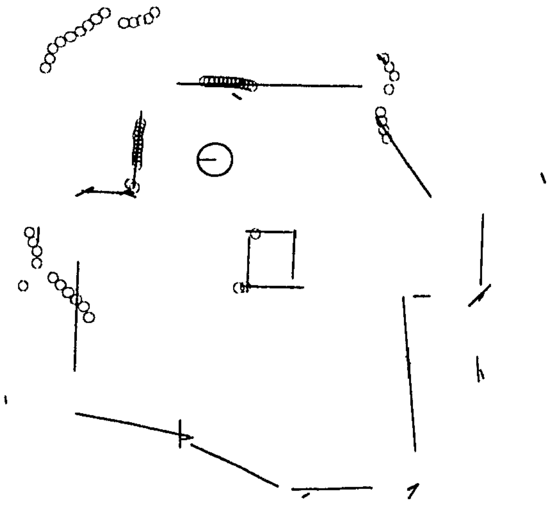


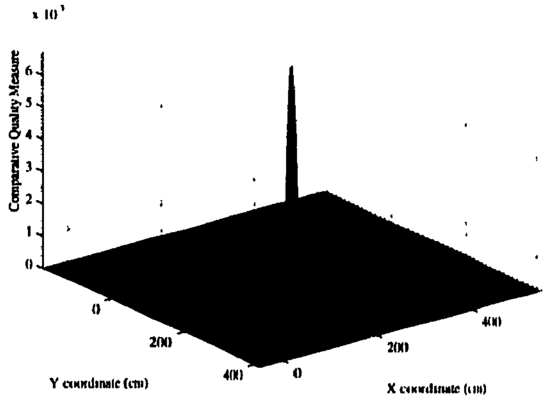
(b) E_{cqm}, x vs. θ



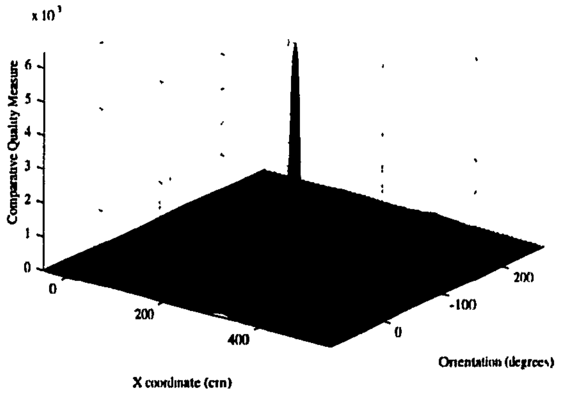
(c) E_{cqm}, y vs. θ

Figure 8.20: E_{cqm} for Room 3, Pose 1: like E_{cf}, this maximum of this quality measure has smaller magnitude than those of the simulations, but the actual robot position is still distinguishable in relative terms

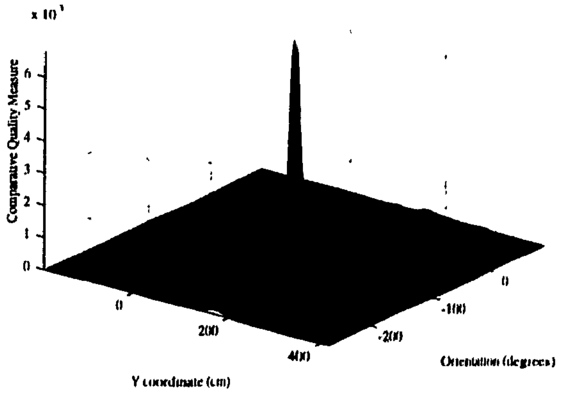

(a) Room 3, Pose 2



(b) E_{cqm}, x vs. y

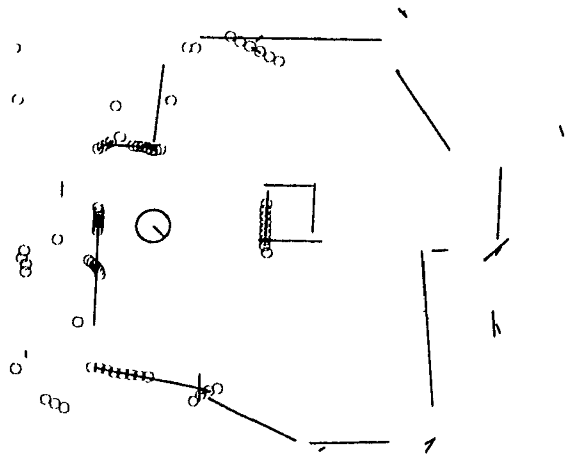


(c) E_{cqm}, x vs. θ

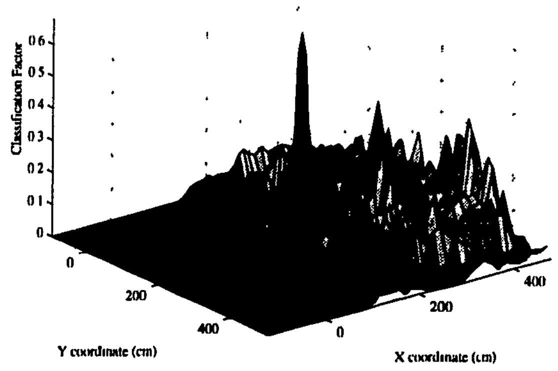


(d) E_{cqm}, y vs. θ

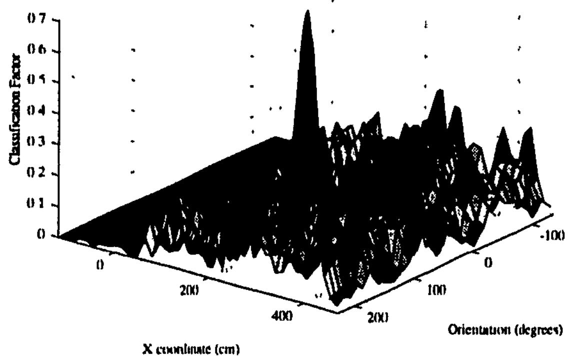
Figure 8.21: E_{cqm} for Room 3, Pose 2



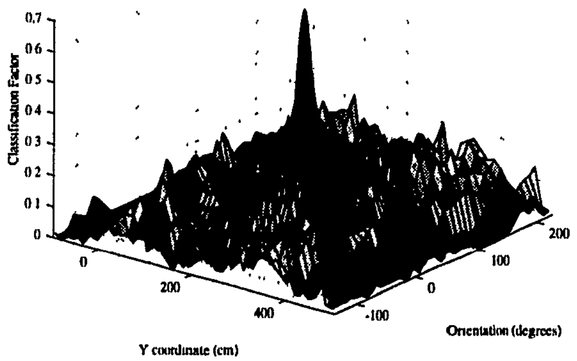
(a) Room 3, Pose 3



(b) E_{cf}, x vs. y

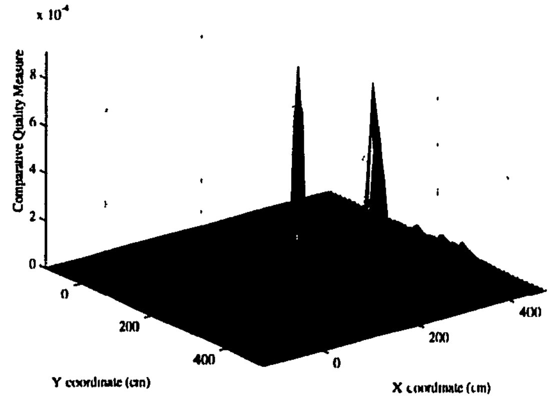


(c) E_{cf}, x vs. θ

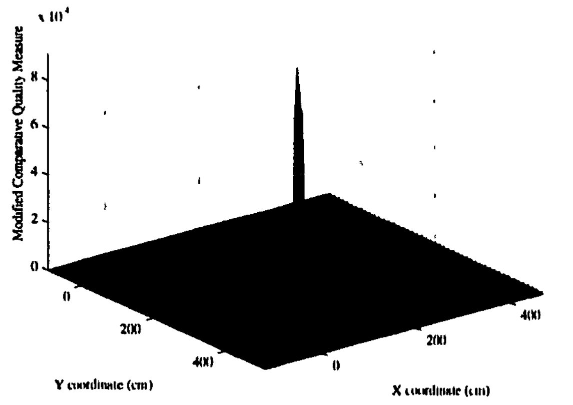


(d) E_{cf}, y vs. θ

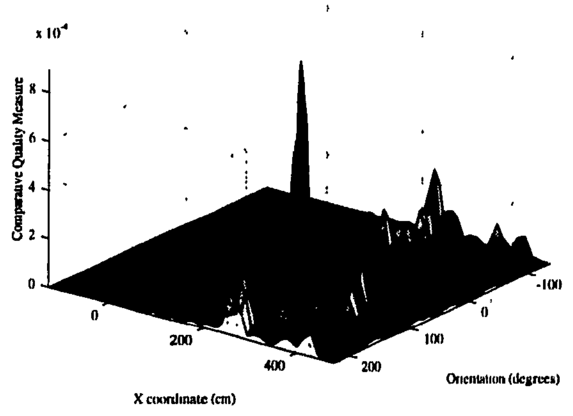
Figure 8.22: Pose 3, E_{cf} for Room 3



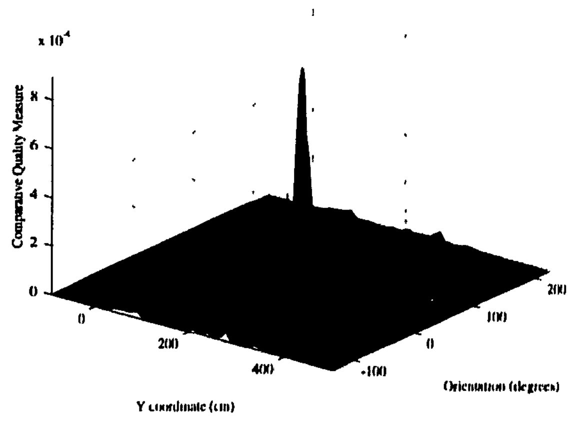
(a) E_{cqm}, x vs. y



(b) $\hat{E}_{cqm}$, x vs. y



(c) E_{cqm}, x vs. θ



(d) E_{cqm}, y vs. θ

Figure 8.23: Pose 3, E_{cqm} for Room 3: (a), (c) and (d) show E_{cqm}, and (b) shows $\hat{E}_{cqm}$ for x vs. y to show that although two maxima may appear in E_{cqm}, for orientation and global localization a low E_{cf} causes $\hat{E}_{cqm}$ to have a single maximum.

8.3 Regions of Convergence

As previously defined, the region of convergence is a set of poses within the environment where each pose acting as an initial estimate results in localization converging to the robot's true pose. This helps to judge how accurate a pose estimate must be in a given area for a given environment (since the size of this region varies with the structure of the environment) for localization to function correctly. In this section we will look at the region primarily in terms of position: all test poses begin at 0° but are free to rotate as localization proceeds. Successful localization is again classified as within 5cm of the actual robot position (hand measured in the case of the real environment). All regions of convergence in this section are from maps constructed with the corrected sonar sensor, both simulated and real.

Figure 8.24 shows four test poses within Room 1. The resulting regions of convergence are shown in figure 8.25. The regions of (b) and (d) are quite small due to the

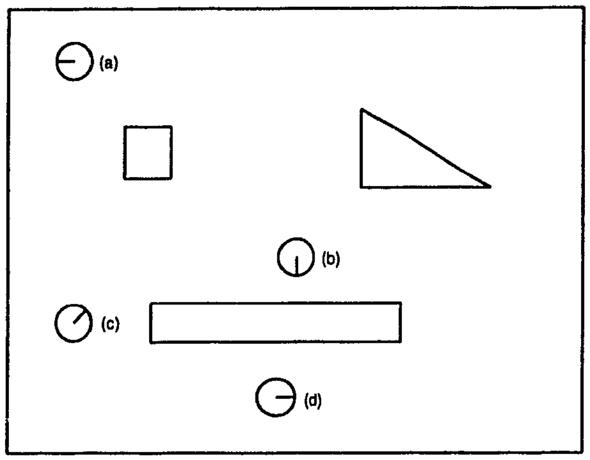


Figure 8.24: Test Poses for Room 1

lack of distinguishable features viewable from their respective poses. In (d), the case is extreme, where the true x-coordinate cannot be distinguished from its neighbours due to the featurelessness of the hallway.

Results are a little better for the test poses of Room 2 (figure 8.26). Figure 8.27 displays the regions of convergence for this environment. Since the test poses have

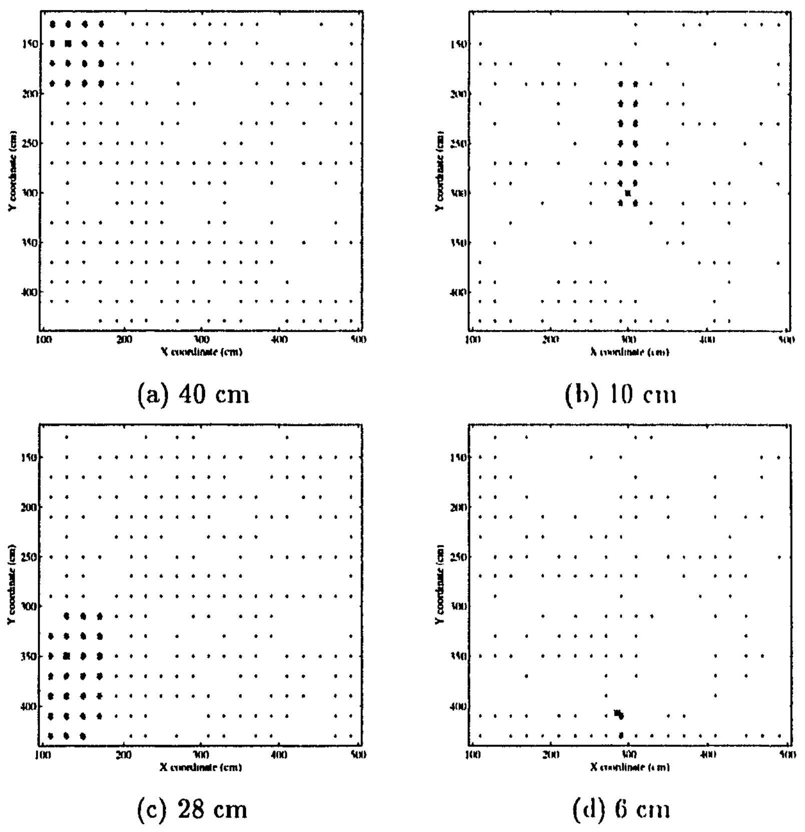


Figure 8.25: Room 1: Regions of Convergence: successful convergences have a final error of less than 5cm, with the true robot positions for each plot being marked with a '*'. The minimum radius of convergence is given in the caption of each position. In this environment with few unique features for localization to discriminate, the region of convergence for each position is not very wide-reaching. While in the long hallway in (d), the region is very small because the proper x-coordinate cannot be distinguished from most others in the hallway.

been placed primarily in the differently sized semi-enclosed areas, the local structure of the environment is distinguishable enough for localization to work over a wider area. However, as seen in the previous section, the quality measures indicate that there is still some similarity between some locations, limiting the size of the regions of convergence.

For the real environment of Room 3 where the features of the environment are more distinguishable, the regions of convergence are much larger. Consider the six example poses of figure 8.28, for which the regions of convergence are shown in figure 8.29.

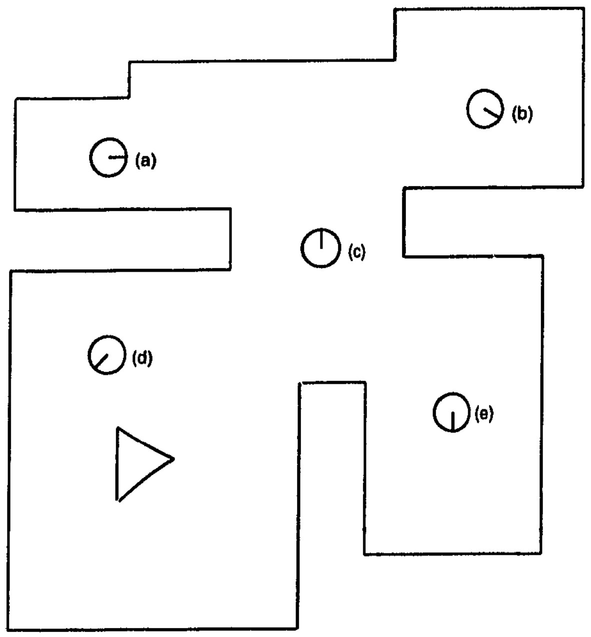
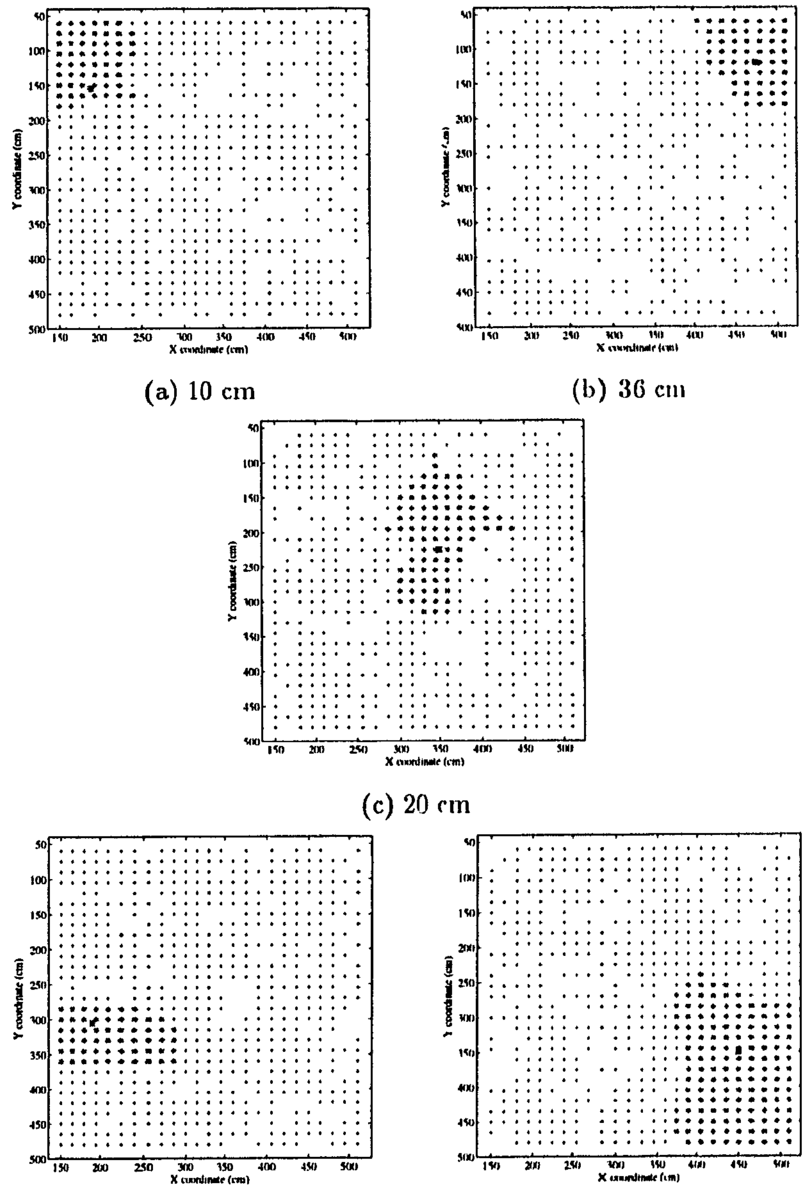


Figure 8.26: Test Poses for Room 2

Thanks to a smaller number of similar features in this environment, the regions of convergence for all the sample poses are quite large, indicating that the initial pose estimates need not be completely accurate for successful localization. We can see how initial estimates *outside* of the region of convergence change by examining the *path of convergence* plot, as was first seen in section 5.2.3. Figure 8.30 shows two *path of convergence* plots using the sample poses (c) and (f). The plots in this figure show that the initial estimates tend to converge together to common positions, and many converge to positions within a small area. This, like the quality measures, is another way to examine similarities of different positions in the room. Since the incorrect convergence positions, or *local attractors* draw pose updates away from the actual robot pose as the algorithm iterates, they must at least fit the range measurements to some extent; in order to do so the arrangement of line segments about the local attractor must be similar to that of the actual pose of the robot.



(a) 10 cm

(b) 36 cm

(c) 20 cm

(d) 20 cm

(e) 60 cm

Figure 8.27: Room 2: Regions of Convergence: successful convergence means the final position error is less than 5cm, and the actual position of each is indicated by a '*'. The minimum radius of convergence is shown for each region. These regions are larger than in Room 1, because the features of the environment are much more distinguishable (the sizes of the semi-enclosed areas are different).

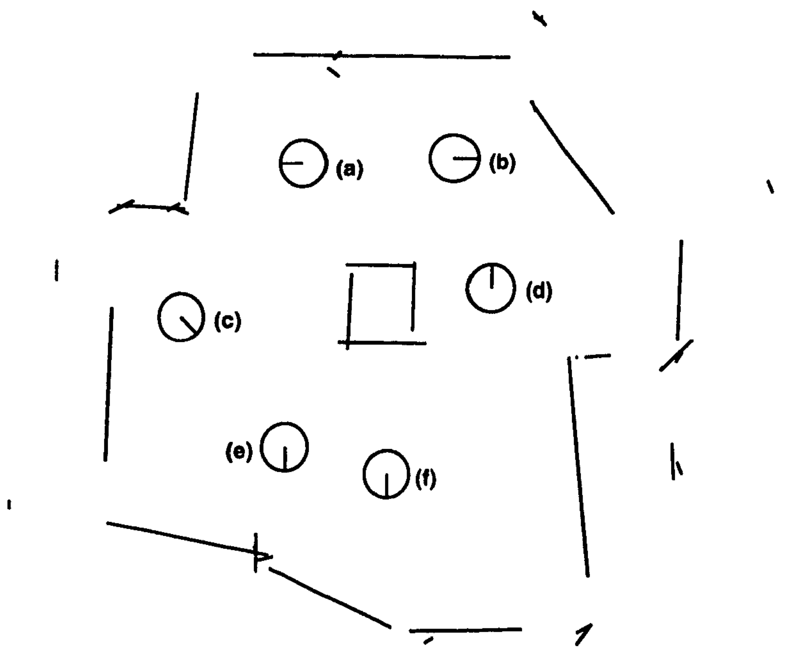
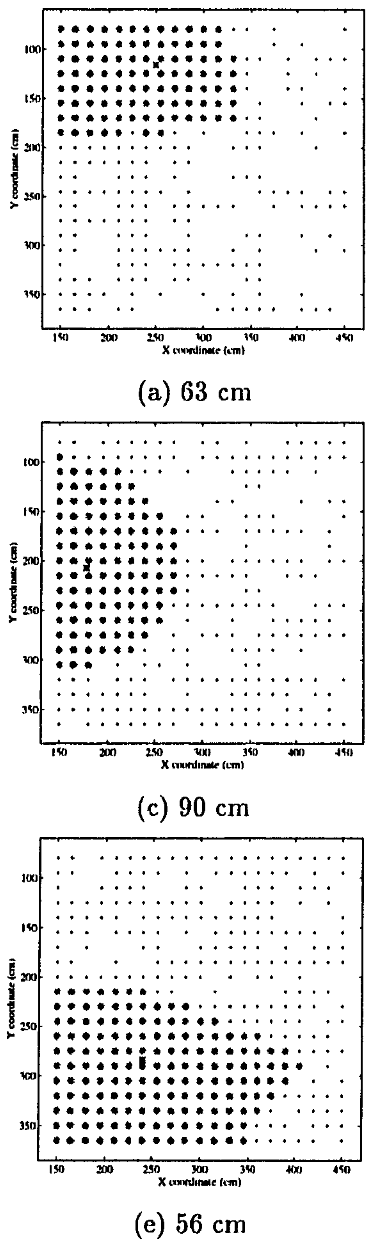


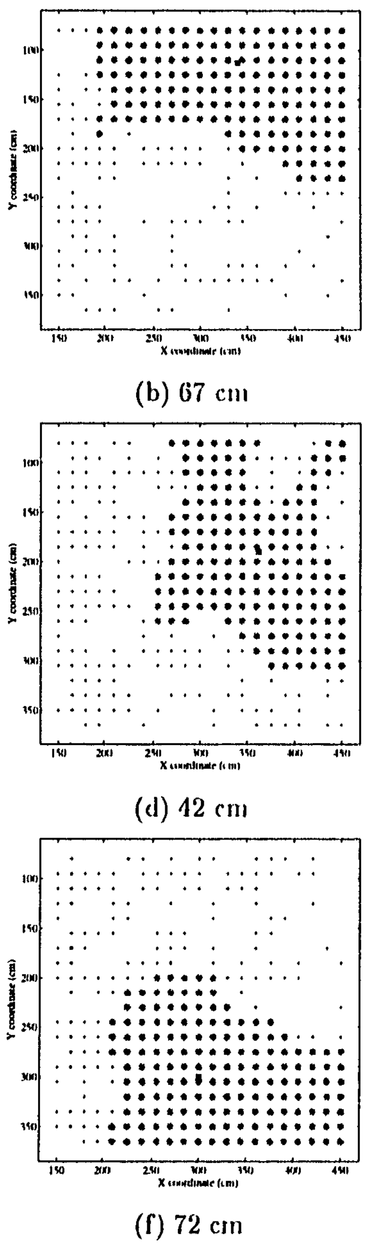
Figure 8.28: Example Poses for Room 3



(a) 63 cm

(c) 90 cm

(e) 56 cm

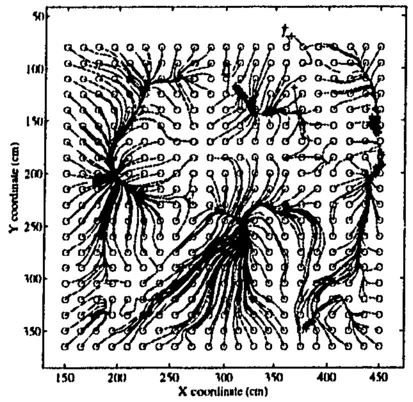


(b) 67 cm

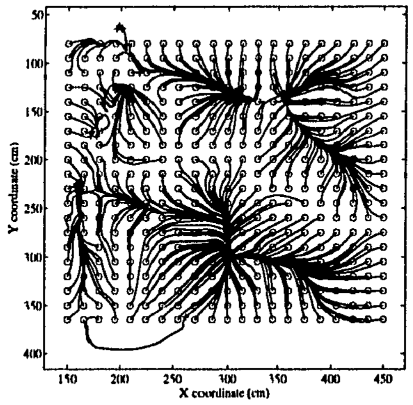
(d) 42 cm

(f) 72 cm

Figure 8.29: Room 3: Regions of Convergence (final error less than 5cm, actual position shown by a '*', minimum radius of convergence shown for each): these regions are all much larger than those in the previous two environments, since here there are fewer similar features to confuse the localization algorithm.



(a) Sample Pose (c), true position at (x,y)=(178,207)



(b) Sample Pose (f), true position at (x,y)=(300,300)

Figure 8.30: Paths of Convergence for Room 3: the small circles indicate initial estimates of position, the lines track the updated estimates as the localization algorithm tries to find the true position, and the small + symbols indicate convergence points. This type of analysis can show where the *local attractors* are, i.e. positions whose view of the environment is similar enough in structure to that of the actual position to draw convergence away from the true position.

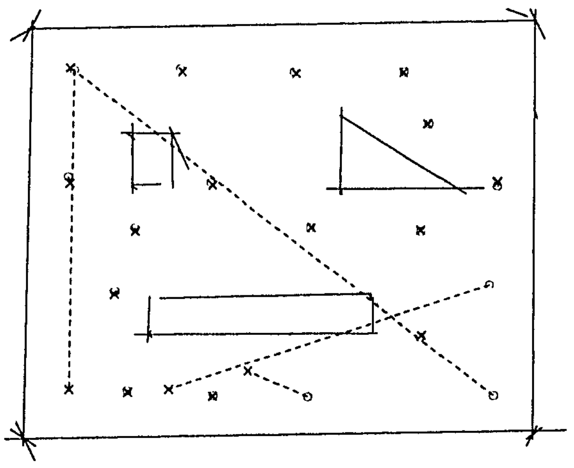
8.4 Global Localization

As explained in section 7.2, global localization refers to the case where an estimate of the robot's pose is not available. In this case the robot has only a map of the environment, a set of range data, and knowledge of the robot's position and orientation *relative* to the data (but not to the map). This section examines how global localization functions in the three sample environments discussed thus far in this chapter. In each case, the results are presented in two ways: a 2-D overlay of the room comparing true robot position and corrected robot position, and a 3-D plot comparing the full initial pose (x,y,θ) with the pose after global localization. The map used for each case was constructed with the *corrected* sonar sensor, and a description of each is contained within its caption. Table 8.1 summarizes numerically the results of the 3 global localization trials.

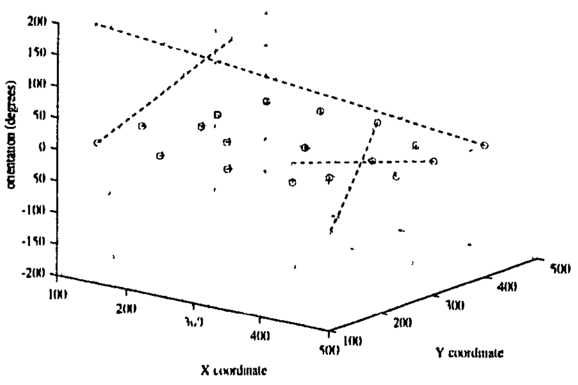
		Localization Errors				
		Min	Max	Mean	S.Deviation.	Median
Room 1	Position (cm)	0.7	456.5	60.9	132.44	2.6
figure 8.31	Orientation (°)	0.1	189.4	32.2	62.03	2.6
Room 2	Position (cm)	0.2	402.3	26.8	94.85	0.9
figure 8.32	Orientation (°)	0.0	174.8	17.0	44.94	2.8
Room 3	Position (cm)	0.1	3.9	2.5	1.56	2.8
figure 8.33	Orientation (°)	0.0	1.5	0.6	0.64	0.3

Table 8.1: Numerical Summary of Global Localization Trials: the results show the high accuracy of localization when correcting Room 3, where most possible robot poses have unique surroundings. Rooms 1 and 2 show the limitations of localization when used in environments with varying degrees of symmetry.

In other experiments with other experimental environments (both real and simulated) we observed that the likelihood of correct global localization tends to increase with the size of the region of convergence for a given pose.

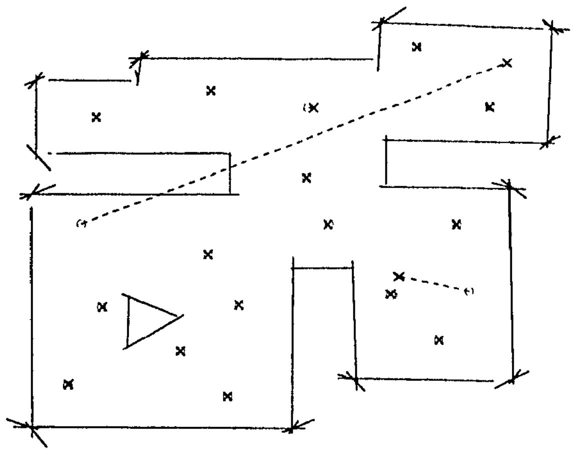


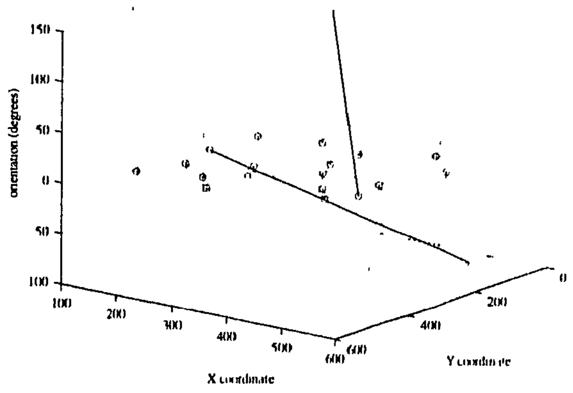
(a) In Comparison to Room Structure



(b) Actual Pose vs. Calculated Pose

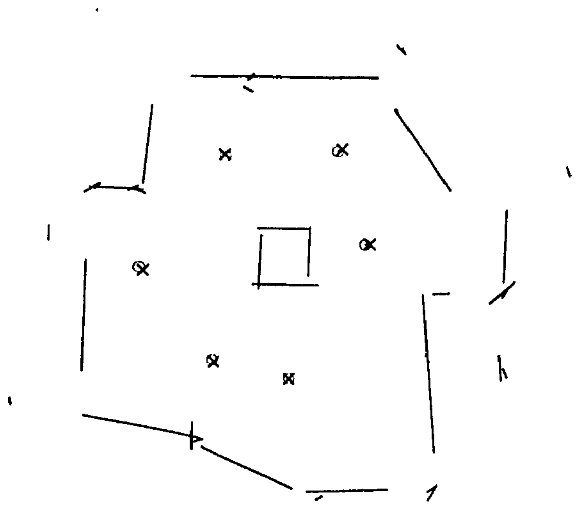
Figure 8.31: Global Localization for Room 1: the actual robot poses are marked by o's, and the poses found through global localization are marked by ×'s, with a dotted line between them. For this example all poses were set to an orientation of 0°, and the 4 incorrect localizations show that the mistaken poses all deviate from the true orientation. In two of the incorrect cases, one corner was confused with another.


(a) In Comparison to Room Structure

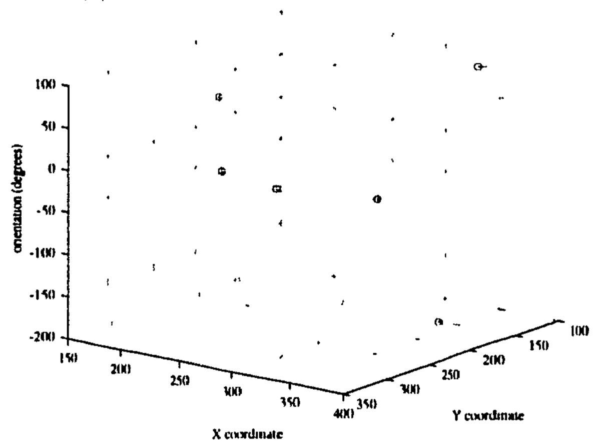


(b) Actual Pose vs. Calculated Pose

Figure 8.32: Global Localization for Room 2: the actual robot poses are marked by o's, and the poses found through global localization are marked by $\times$'s, with a line between them. Again all actual robot poses have an orientation of 0°. This time there are 2 incorrect localizations out of the 18 poses tested.



(a) In Comparison to Room Structure



(b) Actual Pose vs. Calculated Pose

Figure 8.33: Global Localization for Room 3: actual robot poses and calculated poses are marked by o's and ×'s respectively, with a line between them. For this example, the 6 poses of figure 8.28 were used without initial estimates, and in each case the true pose was correctly found.

Chapter 9 Conclusions

A method for the construction of a 2-dimensional map based on range measurements from a mobile robot's on-board sensor has been presented and used as a basis for the development of a system to successfully localize the robot within the environment represented by the map. The map, consisting of a set of straight line segments (modeling the world as a set of planar surfaces), may be incrementally built from measurements of range data from different positions in the environment. These positions may be chosen heuristically by a human user or may come from an external exploration algorithm. This allows a mobile robot to perform map construction and map updating as it explores its environment, while at the same time to use localization to minimize the errors in pose (position and orientation) that accumulate as the robot moves.

The map construction technique as presented is based on a *fit-split-merge* strategy applied to clustered range data. This technique allows distinct features in the environment to be separated and modeled individually, while allowing for single objects (such as a chair or a corner) to be modeled by more than one line segment. The clustering technique needs no user-provided parameters, while the line fitting algorithm allows the user to control the degree of splitting and merging to suit a particular environment or sensor.

The problem of localization was first approached by solving the simpler position correction problem, where a coarse estimate of position was available and no orientation error was present. The problem was solved with an iterated weighted sum of vectors technique, which involved incrementally updating an initial position estimate by reducing the overall difference between range data points and the line segment models of a map. Range points were *classified* (paired) with target line segments assumed to represent the same objects as the measured range points. The distance norm between a range point and a line segment was defined as the length of the vector *perpendicular* to the line segment, with its tail at the range point and its head touching the infinite line passing through the line segment. This perpendicularity constraint was necessary due to the undesirable *long hallway effect*, analogous to the aperture problem in computer vision [26]. The use of weighted sums added robustness by weighting range points closer to their targets more heavily than those far away, since far range points were more likely to be outliers than close ones. However, since it is not known at the outset of localization how best to define "close" for this purpose (since good data may be far away from their true target line segments if the initial position estimate is poor), all points were initially weighted equally to get a coarse position update, and after several iterations of classification and sum calculations only those points very close to a line segment model were weighted heavily, resulting in a fine, precise position update. It was shown that convergence to the correct solution is possible even if some of the range points are classified with incorrect target line segments. A study of the errors involved in the position correction problem led to the concept of the *region of convergence*, inside which any initial position estimate will successfully converge to the true position. Related to this is the *radius of convergence*, which is the distance in any direction from the actual robot's position under which correct convergence will occur. This extra measure was useful due to the non-symmetric shape of these regions. Regions of convergence change from position to position and between environments, but can help to roughly predict a lower bound on the accuracy required for initial position estimates in a particular environment – a useful value for the more general global localization algorithm.

Since position correction fails under some conditions, a method of verification was developed to check its results. These *quality measures* act as functions in the domain of possible poses the robot can take, and are designed to achieve their global maxima at the pose estimate whose error is minimized (which is assumed to be the robot's true pose). Two measures were presented and used to test data: the classification factor, which is a coarse indicator of confidence in a converged solution and takes values between 0 and 1; and the comparative quality measure, which is more sensitive to small errors in a pose estimate but can only be used for comparative purposes since the values of its global maxima are not consistent from environment to environment.

In addition to verification of a converged solution of position correction, quality measures were shown to be useful in correcting a robot's orientation as well. Orientation correction was approached as an optimization problem in the quality measures, and allowed full pose correction given a sufficiently adequate initial (x, y, θ) estimate. The introduction of the *effective angle range* extended the region of convergence into the orientation dimension, allowing the lower bound of accuracy of the estimate to include the robot's angle as well.

Finally, the problem of *global localization* was considered, where an estimate of robot pose is not available. This problem was approached as another optimization problem, this time globally optimizing the quality measures in the 3-dimensional pose domain, with each (x, y, θ) point therein acting as an estimate for a complete local localization. The difficulty with this method is that only the region near the global maximum is convex enough to allow gradient ascent maximization methods. Therefore, a sparse-to-dense sampling approach was devised to find the global maximum without the brute force and time consuming requirements of dense global sampling.

Time-of-flight sonar sensors were the range sensors used for all experiments in this thesis (both real and simulated, except for the ideal simulations). The maps constructed by the robot matched the environments quite closely. Special considerations were given to the mapping process that took into account sonar sensor properties, and these resulted in better maps. Since these considerations were not part of the cluster-fit-split-merge modeling process, we can see that the mapping process is open to accept special considerations that exploit or account for the particular range sensor being used with the mobile robot.

Quality measure experiments were performed to illustrate how the classification factor and the comparative quality measure can be used to both verify localization results (since correct convergence is not guaranteed) and to choose between multiple solutions. In tests with the three rooms, all but one of the cases showed that the global maximum occurred at the true robot pose. The case where this did not occur was due to similarity in the surrounding of the true robot pose and the pose corresponding to the global maximum, and this was illustrated. Of all the trials, this only occurred for one position, showing that only in the most symmetric of environments will the quality measures not indicate a correct solution, making them robust to the many possible environments a mobile robot may encounter.

Regions of convergence were calculated for a number of randomly selected positions within each sample environment. The radii of convergence for these regions were found to be small (6 cm to 40 cm) in a plain environment with few unique geometric features and similar-looking areas. For an environment with a lesser degree of self-similarity and more unique features, the radii increased (10 cm to 60 cm) as expected. For the final trial, the environment with many geometric features yielded the highest radii of convergence, ranging between 42 cm and 90 cm for the six poses tested. These experiments demonstrated that the more unique geometric features an environment has, the less accurate an initial pose estimate needs to be. Knowing the bounds in which localization may work correctly allows a user or an exploration system to judge how far a mobile robot may move before its pose should be recalibrated.

Finally, global localization was tested with the robot in the same randomly selected sample poses as the previous tests. Errors were larger and more frequent for the low-feature environment (up to 4.5 metres and 190° in error for a few poses, with an average of 60 cm and 30°), were reduced with the introduction of more geometric features in the second environment (up to 400 cm and 175° error, averaging 27 cm and 17°), and in the real-world environment containing many geometric features very accurate results were obtained (maximum error of 3.9 cm and 1.5°, averaging 2.5 cm and under 1°). One important and very interesting observation was that all global localization trials demonstrated a median pose error of under 3 cm in position and under 3° in orientation.

These encouraging results show that although sonar has complex geometrical effects and is a relatively imprecise range sensor, it still can be used for accurate mobile robot localization using maps it builds by itself; therefore, one may conclude that super-high accuracy sensors and perfect environmental models are not necessarily required to obtain accurate localization.

Several interesting issues remain to be further explored and included with the present system. For instance, map construction could be made more dynamic by allowing line segment models to be deleted when range measurements show the objects are no longer present in the environment. This has already been studied by Leonard, Durrant-Whyte and Cox [24] based on model confidence values, and could be adapted for use with the approaches discussed in this thesis. Other issues include the use of better optimization methods for global localization, such as simulated annealing or other such techniques for optimizing multi-dimensional ill-behaved functions; and exploration techniques which use the present state of a map to decide where to explore next. Exploration work such as that done by Whaite and Ferrie [35] is independent of localization and could be inserted as a parallel module to the system described in this thesis.

Finally, as has been mentioned a number of times in this thesis, range sensors other than sonar are perfectly usable with this system. Some future work could involve stretching the lower error bounds to the limit when range sensors more accurate than sonar are integrated into the system.

References

[1] Billur Barshan and Roman Kuc. Differentiating sonar reflections from corners and planes by employing an intelligent sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):560–569, 1990.

[2] C. Biber, S. Ellin, E. Shenk, and J. Stempeck. The polaroid ultrasonic ranging system. *Proc. of the 67th Convention of the Audio Engineering Society*, 1980.

[3] D. W. Cho. Certainty grid representation for robot navigation by a bayesian method. *Robotica*, 8:159–165, 1990.

[4] Ingemar J. Cox. Blanche - an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, April 1991.

[5] A. Curran and K.J. Kyriakopoulos. Sensor-Based Self-Localization for Wheeled Mobile Robots. In *IEEE International Conference on Robotics and Automation*, pages 8–13, Atlanta, Georgia, May 1993.

[6] Michael Drumheller. Mobile robot localization using sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2):325–332, 1987.

[7] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, Inc., 1973.

[8] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. Sonar sensing and obstacle detection. In *Proceedings of the Conference on Military Robotics*, Medicine Hat, Alberta, September 1991.

[9] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. Reflections on modelling a sonar range sensor. Technical Report CIM-92-9, McGill Centre for Intelligent Machines, 1992.

[10] Gregory Dudek and Paul MacKenzie. Model-based map construction for robot localization. In *Proceedings of Vision Interface 1993*, North York, Ontario, May 1993.

[11] Gregory Dudek and Chi Zhang. Pose estimation from image data without explicit object models. In *Proceedings of Vision Interface 1994*, Banff, Alberta, 1994.

[12] Alberto Elfes. A sonar-based mapping and navigation system. In *IEEE International Conference on Robotics and Automation*, pages 1151–1156, 1986.

[13] Claude Fennema, Allen Hanson, Edward Riseman, Ross J Beveridge, and Rakesh Kumar. Model-directed mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1352–1369, 1990.

[14] Javier Gonzalez, Anthony Stentz, and Anibal Ollero. An iconic position estimator for a 2d laser rangefinder. In *IEEE International Conference on Robotics and Automation*, pages 2646–2651, May 1992.

[15] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing.* Addison-Wesley, 1987.

[16] William C. Guenther. *Analysis of Variance.* Prentice-Hall, Inc., 1964.

[17] Alois A. Holenstein, Markus A. Müller, and Essam Badreddin. Mobile robot localization in a structured environment cluttered with obstacles. In *IEEE International Conference on Robotics and Automation*, pages 2576–2581, Nice, France, May 1992.

[18] R. A. Jarvis and J. C. Byrne. An automated guided vehicle with map building and path finding capabilities. In Bolles and Roths, editors, *Proceedings of the 4th International Symposium on Robotic Research*, pages 497–504. MIT Press, 1988.

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

[20] Roman Kuc and Billur Barshan. Navigating vehicles through an unstructured environment with sonar. In *IEEE International Conference on Robotics and Automation*, pages 1422–1426, 1989.

[21] Roman Kuc and Omur Bozma. Building a sonar map in a specular environment using a single mobile sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(12):1260–1269, 1991.

[22] John J. Leonard and Hugh F. Durrant-Whyte. Application of multi-target tracking to sonar-based mobile robot navigation. In *Proceedings of the IEEE 29th International Conference on Decision and Control*, pages 3118–3123, Honolulu, Hawaii, December 1990.

[23] John J. Leonard and Hugh F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.

[24] John J. Leonard, Hugh F. Durrant-Whyte, and Ingemar J. Cox. Dynamic map building for an autonomous mobile robot. *The International Journal of Robotics Research*, 11(4):286–298, August 1992.

[25] Jong Hwan Lim and Dong Woo Cho. Physically based sensor modeling for a sonar map in a specular environment. In *IEEE International Conference on Robotics and Automation*, pages 1711–1719, Nice, France, May 1992.

[26] David Marr. *Vision*. W. H. Freeman and Co., New York, 1982.

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

[28] F. Nashashibi and M. Devy. 3D incremental modeling and robot localization in a structured environment using a laser range finger. In *IEEE International Conference on Robotics and Automation*, pages 20–27, Atlanta, Georgia, May 1993.

[29] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1991.

[30] Azriel Rosenfeld and Avinash C. Kak. *Digital Picture Processing.* Academic Press, New York, 1976.

[31] Yuval Roth, Annie S. Wu, Remzi H. Arpaci, Terry Weymouth, and Ramesh Jain. Model-driven pose correction. In *IEEE International Conference on Robotics and Automation*, pages 2625–2630, Nice, France, May 1992.

[32] Godfried Toussaint. A graph-theoretical primal sketch. *Computational Morphology*, 1988.

[33] Godfried Toussaint and Jerzy W. Jaromczyk. Relative Neighborhood Graphs and their Relatives. *Proceedings of the IEEE*, 80(9):1502–1517, September 1992.

[34] J. Vaganay, M.J. Aldon, and A. Fournier. Mobile robot attitude estimation by fusion of inertial data. In *IEEE International Conference on Robotics and Automation*, pages 277–282, Atlanta, Georgia, May 1993.

[35] Peter Whaite and Frank P. Ferrie. Uncertain views. *CVPR*, 1992.

[36] A. Zelinsky. Mobile robot map making using sonar. *Journal of Robotic Systems*, 8(5):557–577, 1991.