NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SONAR-BASED LOCALIZATION OF MOBILE ROBOTS USING THE HOUGH TRANSFORM

by

Khine Latt

March 1997

Thesis Advisor Xiaoping Yun

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
SONAR-BASED LOCALIZATION OF MOBILE ROBOTS USING THE HOUGH TRANSFORM

5. FUNDING NUMBERS

6. AUTHOR(S)
Latt, Khine

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

For an autonomous mobile robot to navigate in an unknown environment, it is essential to know the location of the robot on a real-time basis. Finding position and orientation of a mobile robot in a world coordinate system is a problem in localization. Dead-reckoning is commonly used for localization, but position and orientation errors from dead-reckoning tend to accumulate over time. The objective of this thesis is to develop a feature-based localization method that allows a mobile robot to re-calibrate its position and orientation by automatically selecting wall-like features in the environment. In this thesis, the selection of features is accomplished by applying the Hough transform to sonar data. The Hough transform makes it possible to select the optimal feature (the longest wall, in this case) without finding all possible line segments from the sonar data. A least-square line fitting method is then employed to construct a model of the line segment that represents the feature selected by the Hough transform. The algorithm developed was tested using synthetic and real sonar data. Experimental results demonstrated the effectiveness of the proposed localization methods.

14. SUBJECT TERMS. Autonomous mobile robots; Hough transform; localization; Nomad 200 mobile robot			15. NUMBER OF PAGES 78 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

SONAR-BASED LOCALIZATION OF MOBILE ROBOTS USING THE HOUGH TRANSFORM

Khine Latt
B.S.Ch.E., Pennsylvania State University, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE
with a Major in
ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

March 1997

Author: Khine Latt
Khine Latt

Approved by:
Xiaoping Yun, Thesis Advisor

Roberto Cristi, Second Reader

Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

ABSTRACT

For an autonomous mobile robot to navigate in an unknown environment, it is essential to know the location of the robot on a real-time basis. Finding position and orientation of a mobile robot in a world coordinate system is a problem in localization. Dead-reckoning is commonly used for localization, but position and orientation errors from dead-reckoning tend to accumulate over time. The objective of this thesis is to develop a feature-based localization method that allows a mobile robot to re-calibrate its position and orientation by automatically selecting wall-like features in the environment.

In this thesis, the selection of features is accomplished by applying the Hough transform to sonar data. The Hough transform makes it possible to select the optimal feature (the longest wall, in this case) without finding all possible line segments from the sonar data. A least-square line fitting method is then employed to construct a model of the line segment that represents the feature selected by the Hough transform. The algorithm developed was tested using synthetic and real sonar data. Experimental results demonstrated the effectiveness of the proposed localization methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

During the 1970s and early 1980s, research in robotics was mainly focused on robotic manipulators and their applications in manufacturing automation. In the last ten years, there has been a growing interest in autonomous mobile robots, owing to their potential application in a wide range of operations. HelpMate, an autonomous mobile robot developed by Transition Research Corporation in Danbury, CT, is being utilized in a number of hospitals for delivering food and medicine to patients' rooms. With constant improvement in their intelligence, coupled with a decline in their costs, autonomous mobile robots are expected to find their way into schools for delivering books from a library, and into homes for vacuuming floors.

Autonomous mobile robots have many military applications as well. Equipped with proper detection sensors, mobile robots can be deployed to search for land mines (in foreign battlefields) and unexploded ordnance (in domestic lands previously utilized for ammunition storage and firing ranges, e.g., Fort Ord). Clearly, the use of autonomous mobile robots greatly reduces the danger to the personnel involved in such operations.

In addition to safety, mobile robots offer other advantages as well. Humans have a limited attention span. Searching a large field such as one at Fort Ord is a repetitive and tiresome task. The accuracy of searching by humans will inevitably depend on the degree of concentration. Robotic systems are ideally suited for such repetitive and tedious tasks (aside from safety concerns) because they are not subject to fatigue and degradation in attention and thus provide uniform accuracy in searching tasks. As well as being deployed in battlefields, another application of mobile robots is in maintaining security by patrolling areas such as offices, warehouses, airports, and other facilities. Here again, robots are well suited since they are not adversely influenced by the repetitiveness and monotony.

An important issue associated with the autonomous operation of a mobile robot is localization, i.e., finding real-time position and orientation of the robot in a certain coordinate system. When operating in outdoor environments, the availability of GPS makes the localization problem significantly simpler. But GPS is not a universal solution

for all applications. Obviously, a mobile robot that operates in an indoor environment is not able to receive GPS signals. Likewise, a mobile robot that operates near buildings and trees may not be able to establish connections with four satellites simultaneously. Even if GPS were available, its accuracy may not be sufficient for some applications.

Dead-reckoning is a widely used method for self-localization for mobile robots. This method computes position and orientation by integrating wheel displacements that are typically measured using optical encoders. For example, an automobile odometer uses dead-reckoning to keep track of distance traveled. Due to the nature of integration, position and orientation errors accumulate over time. The longer a mobile robot moves, the poorer its localization accuracy. Dead-reckoning errors are classified into systematic and non-systematic types (Feng et al., 1996). Systematic errors are due to such things as unequal wheel diameters, differences in actual and measured wheel diameters and wheelbase, wheel misalignment, and encoder errors. Non-systematic errors are due to external factors such as uneven floors and wheel slippage on the floor. In most applications, wheel slippage is the main cause of dead-reckoning errors.

To enable an autonomous mobile robot to operate over an extended period of time, there is a clear need to correct or reduce dead-reckoning errors. One general approach to correcting dead-reckoning errors is to provide a mobile robot with the ability to recognize physical features or landmarks in its environment as it moves along and to calibrate itself with respect to these features when it arrives at the previously traveled area again. In this thesis, the calibration of mobile robot coordinates in a laboratory environment will be investigated. The objective is to develop algorithms that allow a mobile robot to automatically select existing laboratory features such walls or benches to calibrate its position and orientation.

II. APPROACH

A. LITERATURE SURVEY

As stated in the previous chapter, this thesis investigates localization of mobile robots. The problem of localization has been actively studied by many researchers in the literature. One approach assumes the availability of a world map. As a mobile robot moves in a previously mapped environment, it attempts to recognize features in the environment and to find a recognized feature in the map. If a feature is found in the map, the robot's position and orientation relative to this feature is calibrated using the map. If a feature is not found in the map, the map is updated by adding this feature. Representative work of this approach includes Drumheller (1987), Crowley (1989), and Gonzalez et al. (1994). When using this approach, one must solve two main problems: feature recognition and correspondence (or matching). The first problem is concerned with how to recognize a feature (wall, corner) using vision, sonar, laser range finder, or other sensor. The second one is concerned with how to match a recognized feature with the corresponding one, if it exists, in the map. If a world map is not initially available, it may be constructed using a dynamic map-building method (Vandorpe et al., 1996) and by exploration of unknown environments (Bauer and Rencken, 1995).

Since the recognition of environmental features is a difficult problem, some researchers proposed to use beacons (Leonard Durrant-Whyte, 1991). Beacons are a special type of targets that can reliably be recognized by appropriate sensors. The use of beacons significantly simplifies the recognition problem, but it requires that beacons be installed in the environment where a mobile robot is to be deployed. This approach may not be applicable in certain applications where the installation of beacons is not feasible or not permitted.

Various sensors have been used for the purpose of localization. Sonar is widely used for localization due to its speed and simplicity (Drumheller, 1987; Crowley, 1989; Kuc and Viard, 1991; Adams et al., 1994). Other sensors used for localization include laser range finders (Vandorpe et al., 1996), vision (Yeh and Kriegman, 1995; Yang, 1995), optical gyroscopes (Komoriya and Oyama, 1994), and a combination of sonar and

infrared (Curran and Kyriakopoulos, 1993).

B. PROBLEM STATEMENT AND PROPOSED APPROACH

This thesis studies the localization problem of a Nomad 200 mobile robot (Nomadic Technologies, 1993). The mobile robot is equipped with 16 sonar sensors that are equally spaced on a ring-like structure. Two adjacent sensors are separated by 22.5°. The robot uses two coordinate systems for navigation: a local robot coordinate system and a global world coordinate system. The world coordinate system is an inertial system in which the position and orientation of the robot is represented. The robot coordinate system is attached to the robot and moves with it. When the robot is first turned on, the two coordinate systems are identical. The position and orientation of the robot is at (0,0,0). The orientation of the robot (and of the world coordinate system) is very much arbitrary. Therefore, it is highly desirable to line up the axes of the world coordinate system with a wall or other major feature so that it will be easier to specify the goal position and orientation of the robot. A current practice is to lift up the robot and carefully position its x-axis or y-axis along a wall, which is highly impractical. One objective of this thesis is to enable a mobile robot to automatically establish a world coordinate system whose axes are parallel to major features in the environment. To this end, a supplementary objective is to develop a localization algorithm which is based on a natural feature in the laboratory environment using the sonar sensors. To achieve these two objectives, it is necessary to solve the following problems:

(1) How to select a feature from the environment that is the best for localization purposes, and

(2) How to recognize the selected feature using the sonar data.

Walls and wall-like features such as a long bench are good features for calibration and localization. Walls can be represented by line segments. Due to the noise nature and low resolution of sonar data, short line segments are more error-prone for recognition. Therefore, it is desirable to select the longest wall in the robot's environment as a feature.

The previous work, e.g., Vandorpe et al. (1996), attempts to first solve the recognition problem and then solve the selection problem. Linear regression is used to recognize all possible line segments. This requires evaluation of the relative positions of sonar data points to determine if they belong to a line segment. Furthermore, data points are assumed to be gathered in a geometrically sequential order. A newly acquired data point is only tested against the present line. If it does not fit into the present line, a new line is started.

In this thesis, the selection problem and the recognition problem are solved simultaneously. A line segment corresponding to the longest wall is extracted. This is made possible by using the Hough transform (Hough, 1962; Duda and Hart, 1972). While the Hough transform is widely used in computer vision (Ballard and Brown, 1982; Haralick and Shapiro, 1993), its use in localization on sonar sensor data is new.

The mathematical basis for the Hough transform lies in point-line and point-curve transformations, or *plane transformations*. It converts the problem of finding collinear points into a problem of finding concurrent lines (i.e., lines passing through a single point). Each point in the original data space, or coordinate space, is transformed into a straight line or a sinusoidal curve in the parameter space. By using this method, there is no need to recognize all line segments to select the longest one. The method is able to find all the points that form the longest line segment. Furthermore, this method does not require that data points be in any order.

In this work, a 'picture' or image is obtained by collecting sonar data over a finite interval using the Nomad 200 mobile robot. The sonar returns plotted on the coordinate plane represent the environment of obstacles that the robot detects. A wall, or other wall-like surface, appears as a line in this picture, and the problem is to identify collinear points within the data collected. This can be done by processing the sonar 'picture' via implementation of the Hough transform to determine the existence of prominent lines. The longest line is extracted and least-square line-fitting is applied to determine the line segment that will be used as a reference for localization.

The next chapter, Chapter III, discusses parametric representations of a line, the Hough technique and its implementation as a computer algorithm, and the line estimation algorithm. In Chapter IV, the results of testing the algorithms developed with both synthetic and real sonar data are presented. The findings and issues of the experimental data are discussed in Chapter V, and concluding remarks and recommendations for future work are given in Chapter VI.

III. THEORY AND IMPLEMENTATION

A. PARAMETRIC REPRESENTATION OF A LINE – PLANE TRANSFORMATIONS

A straight line in the x-y coordinate plane can be described completely by just two parameters, the slope and the y-intercept. The familiar parameterization is:

$$y = m_0 x + b_0 \tag{1}$$

where m_0 is the slope and b_0 is the y-intercept. This line, when plotted in the m-b parameter plane, becomes a single point. It can be seen that, given a point (x_0,y_0), the set of all lines through (x_0,y_0), each of which is specified by some (m,b) pair, will be transformed into a straight line in the parameter (m-b) plane (see Figure 1).

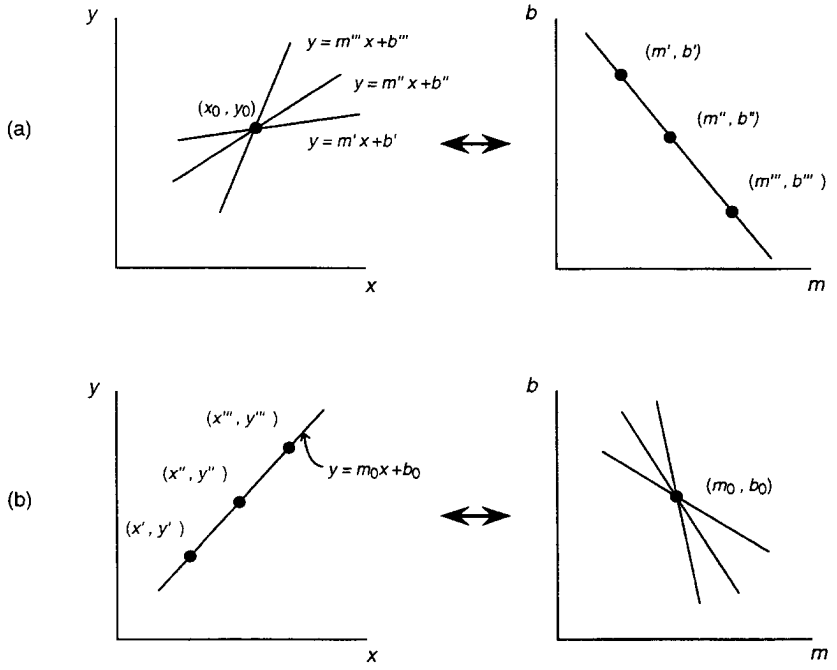

Figure 1. Point-line transformations. (a) A point with lines of various (m ,b) parameters through it transforms into a straight line in the parameter plane. (b) Collinear points transform into straight lines intersecting at a single point in the parameter plane.

This is evident from the transformed equation which indicates the linear relationship between m and b:

$$b = -x_0 m + y_0 \ . \tag{2}$$

It can further be seen that, if points lying in line with each other in the coordinate (x-y) plane are transformed into their respective lines in the parameter plane, those lines in the parameter plane will have a single intersection point. The (m,b) coordinates of this intersection point are exactly the slope and intercept of the line through the points in the coordinate plane. Note that in using this parameterization, there exists a singularity in that the slope, the independent variable in this case, is unbounded (i.e., when a line in the coordinate plane is parallel to the y-axis, the slope goes to infinity). Figure 1 illustrates the point-line relationships.

As presented by Duda and Hart (1972), a more convenient way to describe a straight line is by using the normal parameterization (Figure 2):

$$r_0 = x \cos\theta_0 + y \sin\theta_0 \ . \tag{3}$$

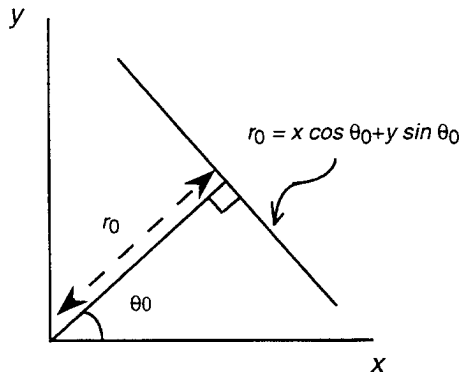

Figure 2. Normal parameterization.

Here, a line in the coordinate plane is described completely by the two parameters, θ_0, the angle of its normal through the origin, and r_0, its distance from the origin.

Following the point-line transformation discussion above, given a point (x_0,y_0) in the coordinate plane, the set of lines through (x_0,y_0), each of which is specified by some (θ, r) pair, will be transformed into a sinusoidal curve in the parameter $(\theta\text{-}r)$ plane (see Figure 3). The curve has the equation:

$$r = x_0 \cos\theta + y_0 \sin\theta \ . \tag{4}$$

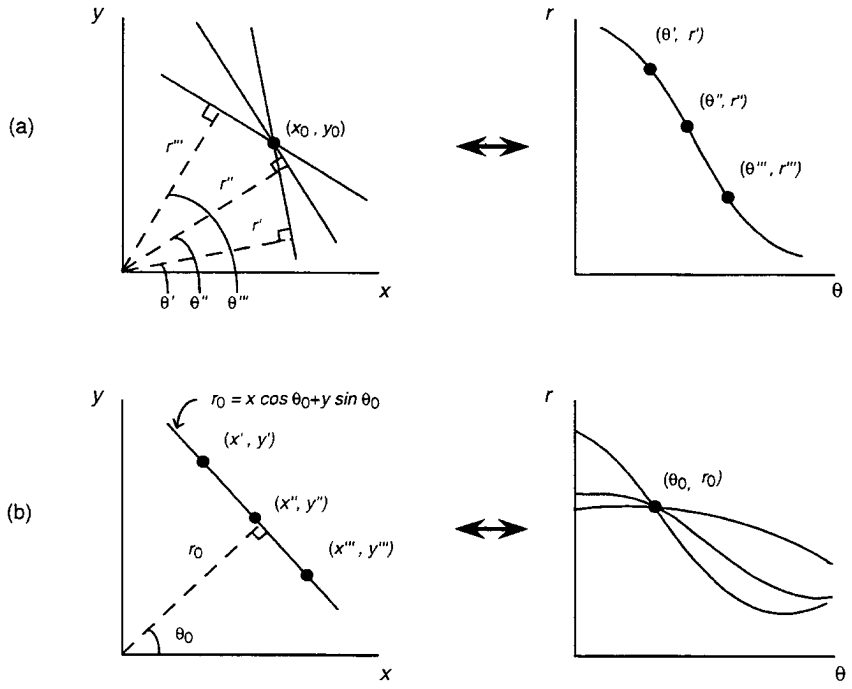

Figure 3. Point-curve transformations. (a) A point with lines of various (θ, r) parameters passing through it transforms into a sinusoidal curve in the parameter plane. (b) Collinear points transform into curves intersecting at a single point in the parameter plane.

In this case, if points lying in line with each other in the coordinate $(x\text{-}y)$ plane are transformed into the respective sinusoidal curves in the parameter plane, those curves in

the parameter plane will have a single intersection point. The (θ , r) coordinates of this intersection point are exactly the angle (or orientation) of the normal and the distance from the origin of the line through the points in the coordinate plane. Figure 3 illustrates the point-curve relationships. Properties of point-curve transformations are described in Duda and Hart (1972).

B. HOUGH TRANSFORM TECHNIQUE

The Hough technique employs the point-curve transformation discussed above. It is based on the premise that collinear points in the coordinate (image) plane will be transformed into curves with a single point of intersection in the parameter plane. To find a line in the image, then, the procedure is to transform the points on the sonar image into the corresponding curves in the parameter plane, and then to locate the intersections of curves to determine the presence of lines.

Analyzing the parameter plane for precise intersection points would clearly be tedious and inefficient. Per Duda and Hart (1972), if the parameter space is discretized into a two-dimensional grid, then each cell in the grid represents a region in which intersecting curves will correspond to a set of *nearly* collinear points since the region encompasses a range of slopes and distances. The resolution of the grid would be based upon the how much 'scatter' is permitted for the points used to estimate the line.

If each cell in the parameter plane grid is examined to determine a set of curves that intersect (pass through) it, then the set of the corresponding coordinate points will provide an approximate line in the image plane. The general procedure is as follows:

1. For a given point, generate a θ-r curve plotted on the parameter plane grid.
2. Note the cells that the curve crosses.
3. Repeat Steps 1 and 2 for every point.
4. Count the number of crossings in each cell.
5. Recover the points whose curves contributed to the total of each cell.
6. Estimate a line for each set of points.

Obviously, some kind of criterion must be imposed for selecting which cells should be considered for further examination since any cell containing two or more curves should give a possible line in the image (a line can be drawn between any two points). It would be neither desirable nor practical to examine every cell, especially as every pair of points in the image would imply the existence of lines, most of which are probably erroneous. Erroneous lines can also result when more than two points in the image coincidentally lie in a line, but represent no physical entity in the environment. While erroneous lines are unavoidable, the likelihood can be somewhat reduced by setting some threshold for the minimum number of points that will be considered to render a line.

In this application, the only lines of interest will be those that most likely represent a significant wall. The assumption can be made that the sonar returns from such a wall will be frequent, i.e., that the greater the number of points contributing the line, the more likely that line represents a wall. The assumption can also be made that the line with the most points is likely to be the longest line since the frequency of the sonar beams intercepting an object of sizeable extent is likely to be greater. Therefore, the line with the most points is selected as a reference wall.

C. HOUGH TRANSFORM CODE

Computer programs were developed to implement the Hough transform technique. The method is executed in three parts: (1) data collection, (2) the Hough algorithm, and (3) the line estimation algorithm. A 'shell program,' ***collect_data.c***, reads in the data from the robot and then invokes the programs ***hough.c*** and ***linest.c*** successively. The program ***hough.c*** is the main algorithm -- it implements the Hough transform and extracts the possible lines present in the image. The program ***linest.c*** performs a line-fitting using the points provided by ***hough.c*** for the extracted line. The present code finds multiple possible lines in a given set of data, and estimates the longest line. For initial testing, the shell program ***synth_data.c*** was used to read an input file of

synthetic data instead of collecting sonar data with the robot. All programs are included as appendices.

1. Data Collection

In ***collect_data.c***, sonar information is collected by the Nomad 200 robot. A real world scenario might be to have the patrolling robot come to a stop, scan its environment and survey the situation. Hence, the robot was programmed to remain stationary in its coordinate position, while its turret was rotated through some nominal angle to collect sonar readings. The coordinates of the sonar reflections from obstacles in the vicinity of the robot were then stored to be processed subsequently by the program ***hough.c***.

Data collection will have some consequence on the spread of the data points. This is an important consideration in finding an appropriate reference wall. Under the assumption that a line containing the most points is also the longest, and likely to be a wall, there is still the possibility of finding a non-contiguous line. For example, a wall with a doorway will look like a line with a segment missing, or objects in the room that are situated in line with each other will look like a line of several segments. But neither of these are appropriate to use as a reference surface for the purpose of calibration.

The Nomad 200 interface software is configured such that the ring of 16 sonar sensors are cycled through consecutively, giving readings which cover all 360° around the robot. The distance between the sonar returns for two adjacent sensors grows larger for obstacles at distances farther away from the robot, and the data points collected grow farther apart. In order to obtain data points closer together, the robot must rotate through some angle less than the angular separation between two adjacent sensors (see Figure 4).

In the case of this work, the sonar data was collected at 7.5° rotations of the robot's turret. Since the separation angle between two adjacent sensors is 22.5°, three cycles of sonar readings can be collected before the data becomes redundant. Thus, a tighter resolution of data can be obtained, as illustrated in Figure 4. The 16 sensors were read at each of the orientations -- initial orientation, initial+7.5°, and initial+15° -- to provide a total of 48 points.

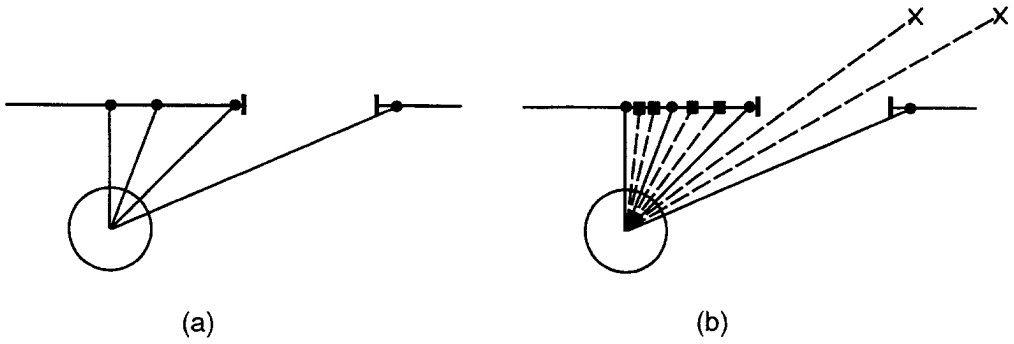

Figure 4. Separation of data points. (a) Sonar readings spread farther apart with distance. (b) Additional points provide information that might otherwise be missed.

2. Hough Algorithm

The mechanics of the Hough algorithm, ***hough.c***, can be viewed as follows: transform the points in the sonar image into curves in the parameter plane; superimpose a two-dimensional grid over the parameter plane; construct a two-dimensional accumulator array whose elements (hereinafter referred to as 'bins') correspond to the cells in the parameter plane grid; for each point, note each cell that its curve crosses; increment the corresponding bin in the accumulator array and keep track of the point; and, finally, for each bin of interest, recover the points associated with it. Figure 5 illustrates the steps.

a. Defining the Grid and the Accumulator Array

The resolution of the grid is set by the size of the increments, *θ-resolution* and *r-resolution*, which are chosen to discretize the θ and r parameters. If the resolution is too coarse, then the possible lines found will have to be estimated from points that are more widely dispersed. If the resolution is too fine, perhaps finer than the precision of the measurement, then the same line will be found by several cells that are grouped together. The difficulty is in allowing enough tolerance in the data so that a line (i.e., a reference wall) can be estimated as accurately as possible.

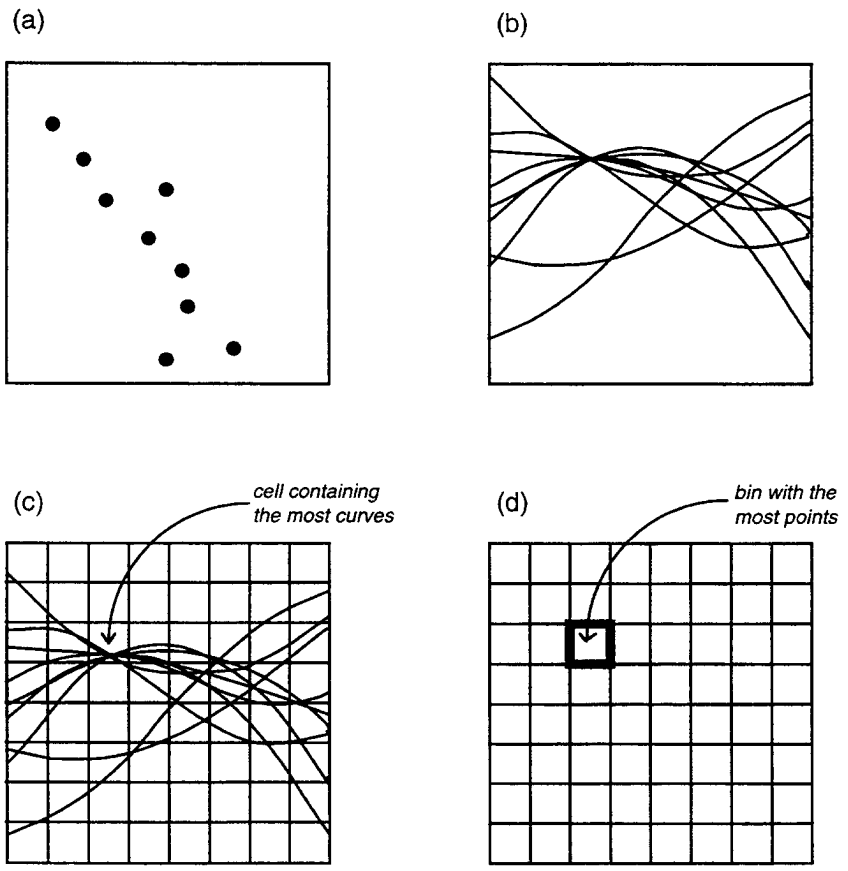

Figure 5. Hough algorithm steps. (a) Sonar picture. (b) Plane transform. (c) Parameter grid (d) Accumulator array.

Since the accumulator array size has to be finite, a consideration must be made regarding the limits of the parameters. The convention that θ is defined in the interval [0,180°] while r can take on both positive and negative values was used. A negative value for r indicates that the line's normal to the origin falls in the third or fourth quadrant of the coordinate plane, and its angle, measured from the x-axis, is that of the normal extended into the first or second quadrant. Figure 6 illustrates this definition.

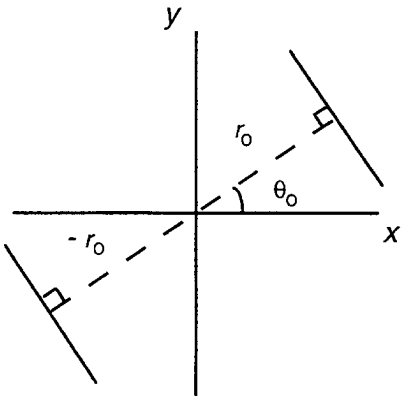

Figure 6. Definition of θ and r.

An estimate of a bound for r is needed to limit the accumulator array size and can be established using the data. Let R_i be the distance from the origin of any point (x_i,y_i):

$$R_i = \sqrt{x_i^2 + y_i^2} \,. \tag{5}$$

Note that this is not the same as r, the distance from the origin to an arbitrary line, defined in Equation (4). It can be shown that the distance R_0 to a point (x_0,y_0) is the largest distance that can be obtained for any line drawn through that point. Recall that the *distance* from a point, such as the origin, to a line is the length of a line segment between the point and the line which is perpendicular, or *normal*, to the line. This *normal line segment* is the shortest line segment that can be drawn from the original point to any other point on the line. For a point (x_0,y_0) in the coordinate plane, there exists only one *line* through the point whose distance r_0 is equal to the distance R_0 to the *point*. This is the line whose normal line segment coincides with the line segment between the origin and (x_0,y_0). Furthermore, every *other* line through the point (x_0,y_0) has its own unique normal

line segment which does not coincide with the line segment between the origin and (x_0,y_0), and is, by definition, shorter than R_0. Therefore, R_0 is the largest distance that can be obtained for any line drawn through the point (x_0,y_0). A logical, and conservative, estimate of the maximum extent of the area covered by the image, then, is the maximum absolute value expected for R_i. Let this estimate be R_{max}, where

$$R_{max} = \max_i \left(R_i \right) . \tag{6}$$

Thus, r is bounded by $[-R_{max},+R_{max}]$.

In order to define a grid with cells of uniform size, R_{max} must be wholly divisible by the *r-resolution* which discretizes r. The range for r can be adjusted to $[-r_{max},+r_{max}]$, where r_{max} is the closest multiple of *r-resolution* larger than R_{max}.

The size of the accumulator array is M columns by N rows, and each bin is indexed by its row and column. The row and column indices must be non-negative integers. The number of columns M is simply:

$$M = \frac{180^\circ}{\theta\_resolution} . \tag{7}$$

Since r ranges from $-r_{max}$ to $+r_{max}$ the number of rows N is

$$N = \frac{\left[+r_{max} - (-r_{max}) \right]}{r\_resolution} = \frac{2r_{max}}{r\_resolution} . \tag{8}$$

The grid plane can be translated directly into the accumulator array. The column index is read directly from the θ-axis marked off in increments of *θ-resolution*. To obtain the row index, the r-axis must be shifted down by an amount r_{max} so that the zero position is at $-r_{max}$. The row index is read from the shifted axis marked off in increments of *r-resolution*. The translation is shown in Figure 7 .

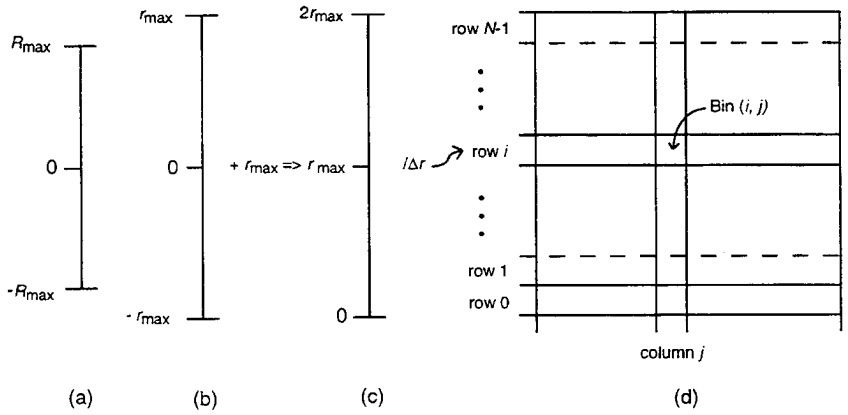

Figure 7. Translation of r into row number. (a) Bounds for r. (b) Bounds for discretized r. (c) Axis shifted to convert into rows. (d) Accumulator array.

b. Discretizing the Plane Transform

Each point in the image is transformed into the corresponding sinusoidal curve, and the curve is mapped onto the parameter grid. The curve is generated by calculating r for each value of the discretized independent variable θ using Equation (4). If the problem was simply to discretize r by truncating any decimal portion, then discretizing the plane transform (i.e., the sinusoidal curve) would be trivial. This does not, however, account for sections where the slope of the curve changes very rapidly such that the curve crosses more than one cell between two consecutive values of θ. It is clear that vital information may be overlooked.

Rather than tabulating the (θ, r) pairs and assigning them to the corresponding bins, a method was devised which takes the approach of examining the *columns* of cells in the grid instead. All of the affected cells in each column are then translated into the corresponding bins in the accumulator array. Figure 8 illustrates this method. As the curve is generated, $r(\theta_j)$ and $r(\theta_{j+1})$ are calculated for j=0,..., M–1 (M is the total number of columns or increments of θ). The pairs $(\theta_j, r(\theta_j))$ and $(\theta_{j+1}, r(\theta_{j+1}))$ represent the endpoints of that part of the curve which falls in column j, and passes through one or

more cells. Next, $r(\theta_j)$ and $r(\theta_{j+1})$ are converted into the corresponding rows, $row(\theta_j)$ and $row(\theta_{j+1})$. Thus, the bins in column j between $row(\theta_j)$ and $row(\theta_{j+1})$ are incremented.

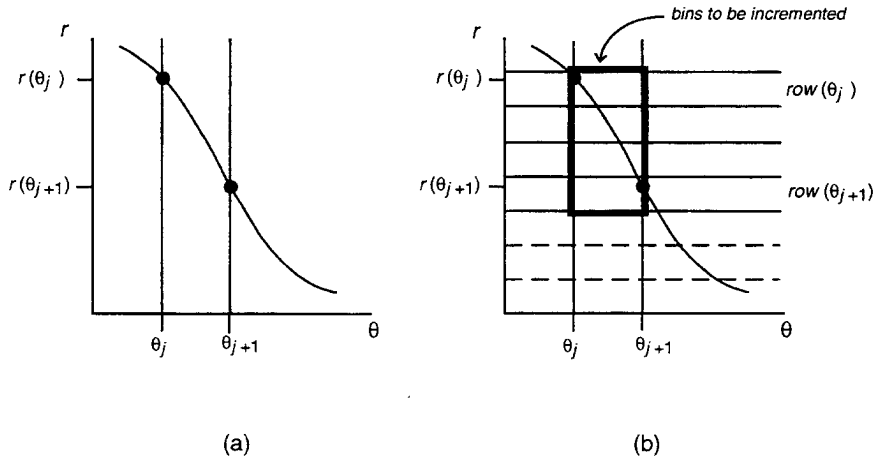

Figure 8. Method of counting which bins to increment for a curve.

c. Finding the Line and Points in the Line

The iterative process of treating each point in the image includes assigning the point to each accumulator array bin which is affected by the corresponding plane transform curve. In other words, the point transforms to a parameterized curve, the curve intersects a series of cells in the parameter plane grid, the cells translate into bins in the accumulator array, and the bins record information which will need to be recovered. The information required is the total number of points in each bin after all the points have been processed, and the points associated with the specific bin(s) of interest. The total number of points will indicate the possible existence of a line and, thus, a wall. The associated points will provide the data to estimate the line.

Counting the points is a simple matter of incrementing each bin whenever it is encountered. Tracking the points, while slightly confusing to implement, is actually

straightforward. The desired product is a list of points that are associated with the accumulator bin containing the most points, but since the specific bin cannot be determined until all the points have been processed, each bin must be somehow linked to all the points that were counted as part of that bin's total. Also, since the same point would have been counted in several bins, what results is a network of links that rapidly becomes quite intricate.

The tracking procedure that was devised builds a chain of structures, named *point structures* (in C programming language), connected to each accumulator bin. A point structure holds information that identifies the point that was counted and the memory address of another point structure holding information about the previous point counted in the same bin. Hence, each point structure leads to the previous point. In order to maintain a connection to the chain of point structures, the accumulator bin retains, in addition to the total point count, the memory address of the point structure containing the last point, i.e., the most recent point, that was counted. When a bin $\boldsymbol{A}$ is found to include the current point being processed, implementation of the tracking procedure is as follows:

1. Increment the point total in bin $\boldsymbol{A}$.
2. Create a point structure, $\boldsymbol{C}$, for the current point.
3. Store the index of the current point in point structure $\boldsymbol{C}$.
4. Copy the address of the last point from bin $\boldsymbol{A}$ into point structure $\boldsymbol{C}$.
5. Store the address of point structure $\boldsymbol{C}$ in bin $\boldsymbol{A}$.

The index of the point is stored instead of the actual point coordinates for programming efficiency. The concept is illustrated in Figure 9: when the current point, $\boldsymbol{C}$, is determined to belong to a bin, $\boldsymbol{A}$, the bin is incremented, the connection from the bin to the chain of points is attached to $\boldsymbol{C}$ (thereby adding to the chain), and the bin is connected instead to $\boldsymbol{C}$, the last point that has been processed. The procedure is repeated when another point is encountered that belongs in bin $\boldsymbol{A}$.

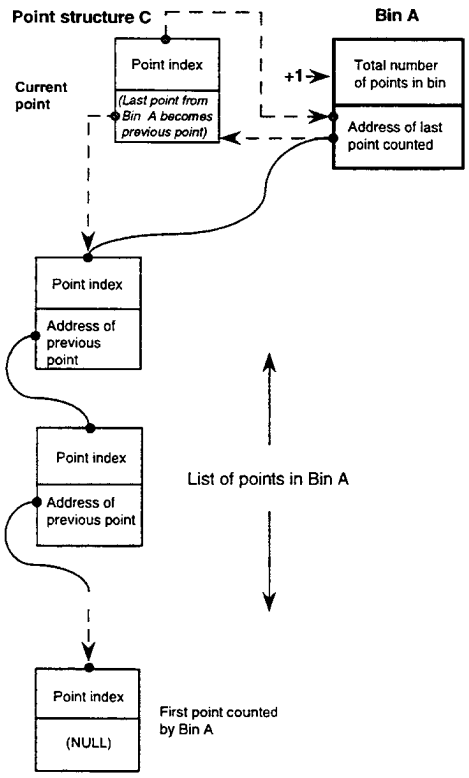

Figure 9. Point-tracking procedure.

3. Line Estimation

A least-square line-fitting method is used to estimate the line from the list of points provided by the Hough algorithm for the longest line. As in the Hough algorithm ***hough.c***, the line estimation algorithm ***linest.c***, based on the algorithm developed in Albayrak (1996), uses the normal parameterization of a line:

$$r = x\cos\theta + y\sin\theta \ . \tag{9}$$

For points (x_i, y_i), $i=1,\ldots,n$ where n is the total number of nearly collinear points, it is desired to find a unique solution for $(\theta\,,\, r\,)$ such that

$$r = x_i \cos\theta + y_i \sin\theta \ . \tag{10}$$

Since the (x_i, y_i) are not perfectly in line, θ and r will have some variation. The best estimate of the line will be that which minimizes the variation in θ and r. Let S be a cost function defined as the sum of the squares of the residuals, a measure of the total variation:

$$S = \sum_{i=1}^{n} (x_i \cos\theta + y_i \sin\theta - r)^2 \ . \tag{11}$$

To minimize the variation, the cost function S is differentiated with respect to each of θ and r. First, to solve for r, differentiate S with respect to r:

$$\frac{\partial S}{\partial r} = 0 = 2nr - \sum_{i=1}^{n} x_i \cos\theta - \sum_{i=1}^{n} y_i \sin\theta \tag{12}$$

so that

$$r = \frac{1}{2n} \sum_{i=1}^{n} (x_i \cos\theta + y_i \sin\theta) \ . \tag{13}$$

Substituting this back into the original function,

$$S = \sum_{i=1}^{n} \left[x_i \cos\theta + y_i \sin\theta - \frac{1}{2n} \sum_{i=1}^{n} (x_i \cos\theta + y_i \sin\theta) \right]^2 \ . \tag{14}$$

Differentiating with respect to θ yields

$$\frac{\partial S}{\partial \theta} = 0 = \mu_1 \sin(2\theta) + 2\mu_2 \cos(2\theta) \tag{15}$$

with

$$\mu_1 = \sum y_i^2 - \sum x_i^2 + \frac{\left(\sum x_i\right)^2 - \left(\sum y_i\right)^2}{n} \tag{16}$$

and

$$\mu_2 = \sum (x_i y_i) - \frac{\sum x_i \sum y_i}{n} \ . \tag{17}$$

Equation (15) above can be satisfied if

$$\sin(2\theta) = -2\mu_2 \tag{18}$$

and

$$\cos(2\theta) = \mu_1 \ . \tag{19}$$

Therefore, θ can be found from:

$$\theta = \frac{1}{2} \text{atan2}\left(-2\mu_2, \mu_1\right) \ . \tag{20}$$

In the ***linest.c*** algorithm, the estimated line, described by its (θ, r) parameters, is found by computing μ_1 and μ_2 from all the points (x_i,y_i).

IV. EXPERIMENTAL RESULTS

A. SYNTHETIC DATA

The algorithms developed were initially tested using synthetic data. As mentioned previously, the shell program ***synth_data.c*** enabled the use of the synthetic input data files and invoked the algorithms for the Hough transform and the line fitting/estimation. Because of the highly iterative nature of the Hough algorithm, a simple exercise using a small data set, ***cross.data***, was run to observe how well the whole code would perform. Then two larger data sets, ***corner.data*** and ***corridor.data***, were tested to simulate possible real world scenarios. For each data set, the influence of the grid resolution was examined.

1. A Simple Exercise: *cross.data*

The data set ***cross.data*** contained 10 points positioned in the form of a skewed cross (see Figure 10). The short arm of the cross contained four points, and the long arm contained seven, with the two arms having one point in common. Figure 10 shows the coordinate plot, the plane transform, the points in the longest line found, and the estimated line.

This was a simple academic configuration which was intended to exercise the code, and which strives only to identify the longer of the two lines. By design, the slopes of the two lines are distinctly different, thus the plane transform clearly shows two distinct intersection regions within the grid, one with four curves and the other with seven. The estimated line was correctly found to be the longer of the two lines. Note that, for this data set, the curve intersections occur at exact points, and the estimated line is, in fact, the exact line which is the long arm of the cross.

An examination of grid resolution offers some limited insight for this set of data. In Table I, results are tabulated using various grid sizes for the number of bins that indicate the possibility of a line.

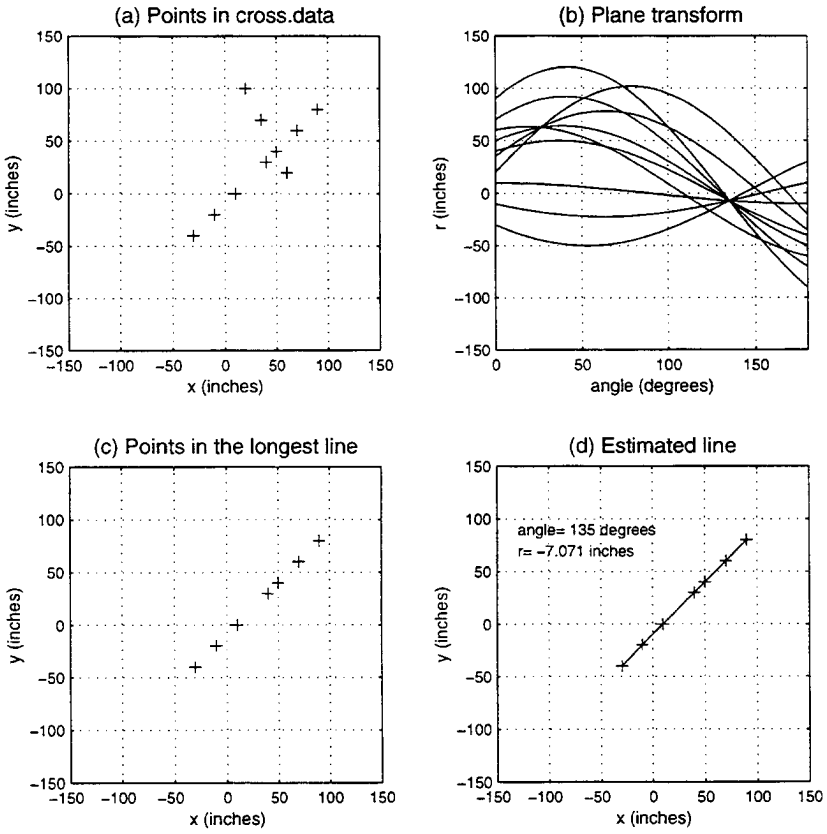

Figure 10. (a) Points in the data set ***cross.data***, (b) plane transform of the points, (c) longest line of points found, (d) the estimated line.

Table I. Number of bins found with a possible line in ***cross.data***.

r-resolution	*θ-resolution*						
	1	*2*	*3*	*5*	*10*	*15*	*20*
1	2	2	2	2	4	2	9
2	2	1	2	2	3	2	5
3	3	2	2	2	2	2	4
5	4	3	2	2	1	2	[3]*
10	6	4	3	2	1	2	[2]
15	10	5	4	2	2	[2]	[2]
20	14	7	6	4	3	[2]	[2]

* [·] denotes erroneous lines.

The threshold for the minimum number of points to be considered for the desired line was set at six, since it was already known that the longest line contains seven points. In theory, if the grid resolution is chosen correctly, only one bin should result in a line, since only one line with more than (or equal to) the threshold value of six points exists. From the table, however, it is apparent that multiple grid cells have captured the same set of curves since multiple bins seem to have found the same line. It can also be seen that more bins are found at the edges of the table, suggesting that cells which are tall and narrow (small *θ-resolution* and large *r-resolution*) or wide and flat (large *θ-resolution* and small *r-resolution*) provide the opportunity for more than one cell to capture the same set of intersecting curves. (Note that this is visually intuitive.) For the more even-sided bins, the algorithm is more discriminating and only one or two bins give a possible line. In this case, the line can be located easily regardless of the resolution, and a larger grid size would more efficient since, with fewer bins, the number of computations would be reduced.

Lastly, for relatively larger grid sizes, erroneous lines may be found (denoted by brackets around the entries in the table). For this data set, the lines are not actually erroneous, but they include an eighth point which does not belong in the line. That is, the eighth point is not perfectly collinear with the rest. It is obvious that the larger grid cell covers a larger area of the plane and will therefore capture curves that pass near the intersection point. For more realistic data sets where points are not perfectly collinear, but *nearly* collinear, this is precisely the idea behind the selection of grid resolution: the size of the cell must be large enough to avoid redundancy and to increase computational efficiency, but not too large that an unacceptable number of stray points are included in the final line estimation.

2. Corner of a Room: *corner.data*

The data set ***corner.data*** contained 50 points and was intended to represent a sonar image of a room corner with other objects in the immediate vicinity. As would be expected of real data, the points in the image do not form clean lines. Figure 11 shows

the coordinate plot, the plane transform, the points in the longest line found, and the estimated line for a given resolution. In this case, *θ-resolution* is 5° and *r-resolution* is 10 inches.

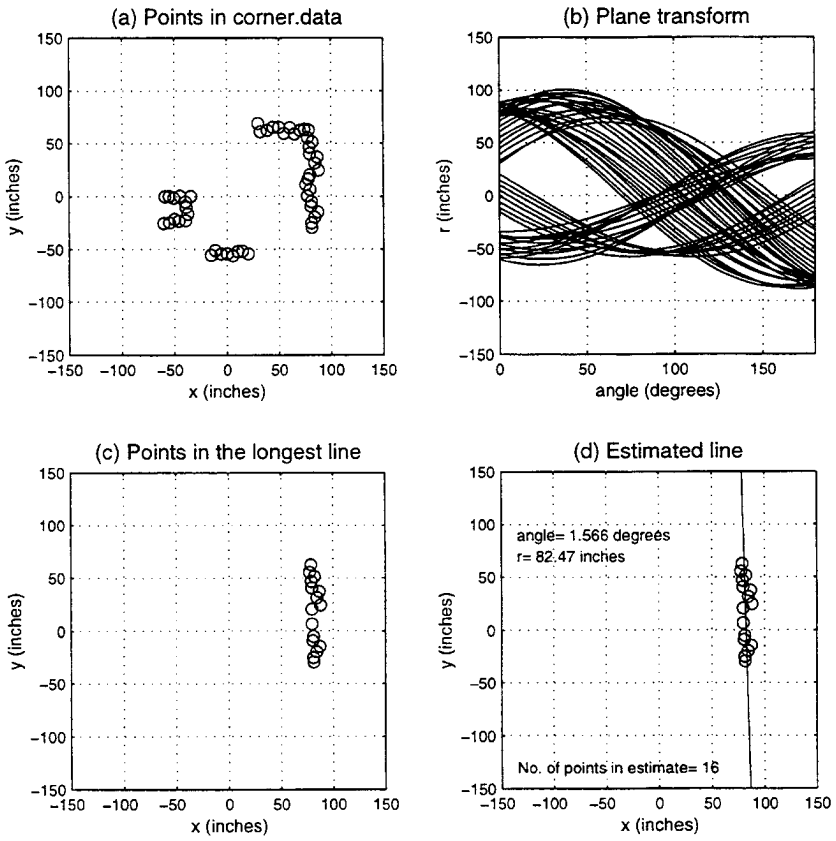

Figure 11. (a) Points in the data set ***corner.data***, (b) plane transform of the points, (c) longest line of points found, (d) the estimated line.

In the coordinate plot, Figure 11(a), the longest line appears to be the vertical line on the far right with $\theta \approx 0°$ and $r \approx 80$ inches. That the points in the image do not form exact lines is exhibited in the plane transform, Figure 11(b), which shows a collection of intersecting regions not especially distinguishable from one another. In Figure 11(c), a

set of points has been found which includes most, but not all, of the points in the line that was expected, and Figure 11(d) shows the line estimated from those points.

In this data set, the grid resolution becomes more significant since the possible lines consist of points which are generally not perfectly collinear, and the problem becomes one of not only finding the desired line, but also estimating it as accurately as possible. The accuracy of the line depends on the points contributing to the line-fitting; that is, each point affects the slope of the final line. Clearly, the grid size directly affects which points are included in a particular bin, i.e., which curves are included in a particular cell. Consequently, grid size will also affect the total number of points ultimately assigned to each bin. For example, a smaller grid size is expected to narrow the spread of points allowed in forming a line, resulting in fewer points being included in the line estimation.

In Table II, the number of points in the longest line found is tabulated for various resolutions. It can be seen that when the resolution window (i.e., grid size) is reduced, the number of points in the longest line found is also reduced, as expected. All this implies is that, with the smaller grid size, the set of points being selected exhibits a tighter degree of collinearity in their relative positions. The estimate resulting from this is only an estimate of the line formed by that specific set of points, but it is not necessarily a better estimate of *the line that is desired.*

From the results of the table, excluding the bracketed entries, it is evident that the line that was initially anticipated is indeed the line found by the Hough algorithm, and it is the line that is desired in this case. However, it should be noted that there is no physical line in existence against which the accuracy of the estimate can be evaluated. Although the image *implies* the values $\theta \approx 0°$ and $r \approx 80$ inches, the data does not represent a real image, and no meaningful computation was incorporated into the data relating to the distribution of points around a line having these specific parameters. In other words, the line did not cause the data. Therefore, while the correct line has been extracted by the algorithm, there is no reason to expect that the line estimate having parameters closest to these values is necessarily the most accurate estimate.

Table II. Number of points in the longest line, the angle θ in radians, and the distance r in inches for ***corner.data***.

r-resolution	*θ-resolution*						
	1	*2*	*3*	*5*	*10*	*15*	*20*
1	10 0.033350 80.4	10 0.033350 80.4	10 0.033350 80.4	11 0.044614 80.1	14 0.073757 80.3	14 0.073757 80.3	[17]* 2.233637 13.4
2	10 0.033350 80.4	10 0.033350 80.4	10 0.033350 80.4	11 0.044614 80.1	14 0.073757 80.3	15 0.097673 79.9	[18] 2.228296 13.8
3	11 0.033371 80.2	12 0.046329 80.2	11 0.033371 80.2	12 0.046329 80.2	14 0.073757 80.3	{16}** 0.108628 80.4	[18] 2.237988 14.9
5	12 0.032550 80.9	12 0.032550 80.9	13 0.027493 81.2	14 0.057245 79.3	16 0.068396 79.8	{18} 0.122093 79.2	{19} 0.132361 79.6
10	16 0.027340 82.5	16 0.027340 82.5	16 0.027340 82.5	16 0.027340 82.5	{18} 0.122093 79.2	{18} 0.122093 79.2	{20} 0.163183 79.1
15	21 0.056670 81.4	21 0.056670 81.4	21 0.056670 81.4	21 0.056670 81.4	21 0.056670 81.4	{23} 0.112749 81.0	{23} 0.112749 81.0
20	16 0.027340 82.5	16 0.027340 82.5	16 0.027340 82.5	[18] 2.145404 10.1	[21] 2.042113 10.4	[23] 2.013184 10.4	[25] 1.986654 8.8

* [·] denotes erroneous lines. ** {·} denotes corner effects.

Two interesting consequences of choosing the grid size inappropriately can be observed. First, the table values denoted with brackets represent lines which are prominently different from the rest, as evidenced by the θ and r values listed. Figure 12(a) shows a representative plot of this line using the points found for *θ-resolution* of 5° and *r-resolution* of 20 inches. It appears that two dense clusters of points can give rise to

an erroneous line. A line can be drawn between any two clusters, and if the two clusters combined contain more total points than any other set of points in a possible line, they will incorrectly be chosen as the 'longest line.' This is an effect of an inappropriate grid size and not simply an effect of too large a grid, although the occurrences of this effect are localized at the edges of the table.

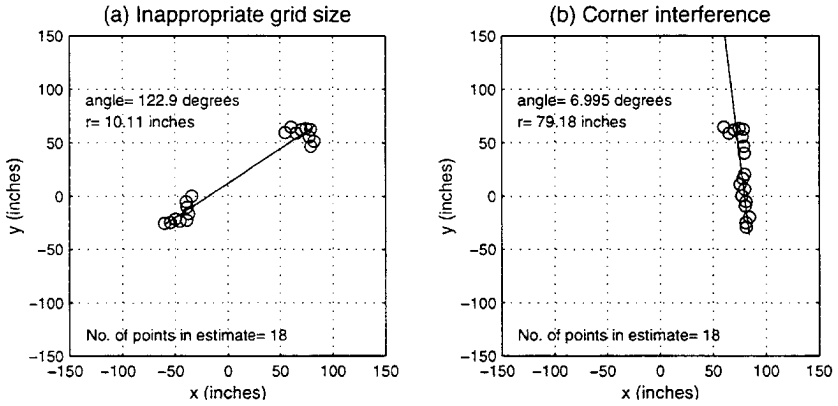

Figure 12. Effect of (a) Inappropriate grid size, and (b) corner interference in ***corner.data***.

Secondly, as Figure 12(b) shows, the corner tends to interfere with the line finding and has the effect of altering the slope of the line by pulling the estimated line away from the vertical. This effect is present in varying degrees throughout the result data, but most pronounced at a few of the larger grid sizes. The table values denoted with braces represent lines containing several points that are included on the corner segment which are not on the line of interest. The plot in the figure is for *θ-resolution* of 10° and *r-resolution* of 10 inches.

The previous paragraphs have attempted to distinguish between correctly finding the desired line and accurately estimating it. The results of Table II show that both problems are influenced by the resolution. Too large of a window can result in a

completely erroneous line or the correct line becoming shifted in its orientation. Too small of a window can impose too stringent a requirement for collinearity which is not warranted and will exclude valid data points which are important to the accuracy of the estimate.

3. A Corridor: *corridor.data*

A possible location to exploit in the selection of an extended surface are the walls of a corridor or hallway. The data set ***corridor.data*** simulates the sonar image in a hallway assuming the robot is positioned at the (0,0) coordinate position (Figure 13(a)).

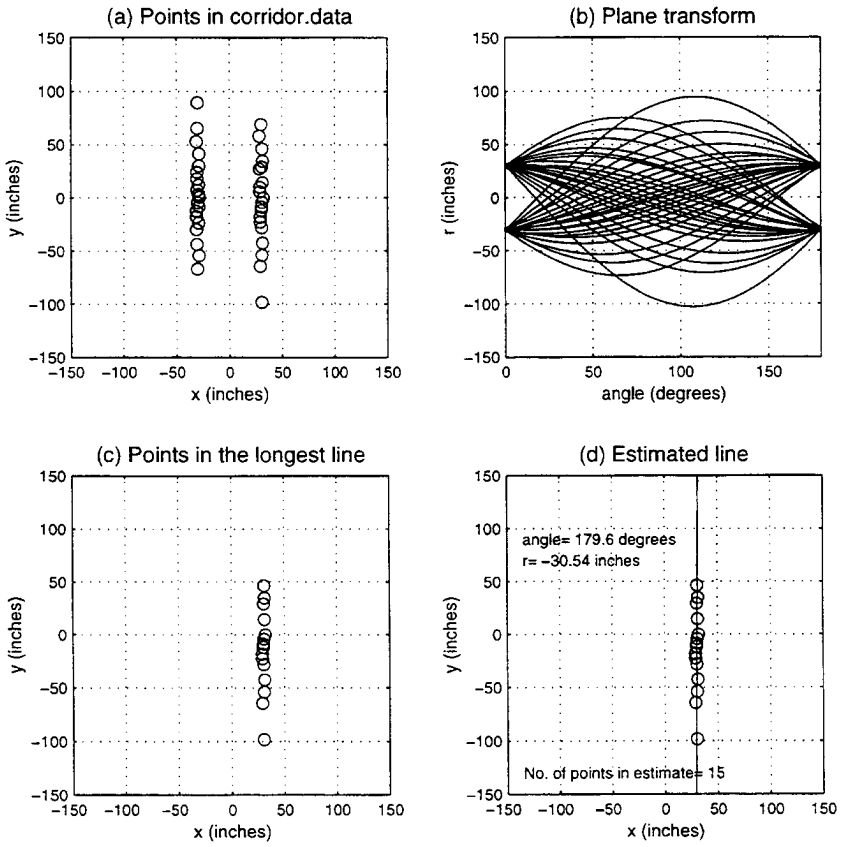

Figure 13. (a) Points in the data ***set corridor.data***, (b) plane transform of the points, (c) longest line of points found, (d) the estimated line.

Figure 13 shows the coordinate plot, the plane transform, the points in the longest line found, and the estimated line for a given resolution. In this case, *θ-resolution* is 5° and *r-resolution* is 10 inches.

Note that in Figure 13(a), the coordinate plot, the data points are spread further apart at the ends of the segments to depict the characteristic spread of sonar returns at distances farther away from the robot.

The plane transform, Figure 13(b), shows the curves converging on the edges of the plane. These are the intersection regions. Recall that the convention adopted limits θ to the interval [0,180°] while r can take on both positive and negative values. Therefore, a line with $\theta = 0°$ and $r = r_o$ can equivalently be represented with $\theta = 180°$ and $r = -r_o$. Each of the two lines in the image has two (θ , r) representations associated with it, which translates into four intersection regions in the plane transform.

The line found, shown in Figure 13(c), happens to be the wall on the right hand side, although the input data included an equal number of points in both lines. The selection of the longest line in this situation is incidental; when more than one bin are found to contain the same number of points, the algorithm merely selects the first one it encounters. Table III displays the number of points in the longest line estimate, and its θ and r parameter values, for various resolutions. For 49 tests (at the different resolutions) of the algorithm, the right wall was selected 25 times and the left wall was selected 24 times. Each line has an equal chance of being extracted as the longest line.

As discussed above, each of the lines in this data set can be described equivalently by two (θ , r) representations, and, by design (due to the convention adopted), the algorithm will provide the pair with a positive value for θ . This can be seen in the table.

This idealized example of a corridor does not account for irregularities in a real corridor which might cause one of the walls to be less appropriate to use as a reference. For example, support columns may protrude a few inches from the wall surface and affect the sonar reflections. Or, the corridor may have a bend so that one wall will be shorter and the other will end with a corner, thereby introducing corner interference effects.

Also, doors that may be arbitrarily left open or closed could affect the selection of an appropriate reference.

Table III. Number of points in the longest line, the angle θ in radians, and the distance r in inches for *corridor.data*. (Shaded entries denote left wall, unshaded entries denote right wall.)

r-resolution	*θ-resolution*						
	1	*2*	*3*	*5*	*10*	*15*	*20*
1	11 3.134430 -30.5	13 3.133807 30.4	14 0.011145 29.4	14 0.011145 29.4	15 0.010817 29.5	15 0.010817 29.5	14 0.015370 29.3
2	12 3.139746 30.7	14 0.003419 -30.5	14 0.003419 -30.5	15 3.133933 -30.5	15 0.010817 29.5	15 0.010817 29.5	15 0.011884 -29.5
3	13 0.013480 -29.3	14 0.003419 -30.5	15 3.133933 -30.5	15 3.133933 -30.5	15 0.010817 29.5	15 0.010817 29.5	15 0.011884 -29.5
5	13 0.013480 -29.3	14 0.003419 -30.5	15 3.133933 -30.5	15 3.133933 -30.5	15 0.010817 29.5	15 0.010817 29.5	15 0.011884 -29.5
10	13 0.013480 -29.3	14 0.003419 -30.5	15 3.133933 -30.5	15 3.133933 -30.5	15 0.010817 29.5	15 0.010817 29.5	15 0.011884 -29.5
15	13 0.013480 -29.3	14 0.003419 -30.5	15 3.133933 -30.5	15 3.133933 -30.5	15 0.010817 29.5	15 0.010817 29.5	15 0.011884 -29.5
20	20 0.000325 -30.0	20 0.000325 -30.0	20 0.000325 -30.0	20 0.000325 -30.0	20 0.000325 -30.0	20 0.000325 -30.0	20 0.000325 -30.0

B. REAL ENVIRONMENT

Using the Nomad 200 robot, sonar data were collected in the two environments which were simulated with synthetic data: a corner and a corridor. It was desired to observe how the Hough algorithm would the perform with an actual sonar image. The data collected would be representative of typical sonar reflections in a typical environment and may or may not have characteristics which could result in line estimates such as those obtained from applying the Hough technique to the synthetic data.

1. Experimental Considerations

The experiments were set up by selecting locations within the laboratory that provided the necessary surroundings and situating the robot accordingly. For practical reasons, no attempt was made to precisely measure the position and orientation of the robot while it was gathering data. It has already been established that it is virtually impossible to position the robot precisely, even by lifting it into position, hence the motivation for a self-calibration capability (which is the topic of this work). Further, precise measurement of the distances and orientations of walls and other objects in the laboratory, with respect to the robot, was deemed to offer only minor benefit to the analysis. Finally, in the operational implementation of a self-calibration capability, what the robot sees is perhaps more relevant than what physically exists.

Also, no attempt was made to achieve the exact angular rotations for gathering data. The robot's movement was generally within one degree of the rotation commanded and was considered to be adequate for the purpose, i.e., to obtain additional data points at angles smaller than the separation angle between the sensors.

2. Corner Results

A sonar map of a corner is shown in Figure 14; the coordinate plot, plane transform, a "good guess" estimate, and an "average" estimate are shown in Figure 15.

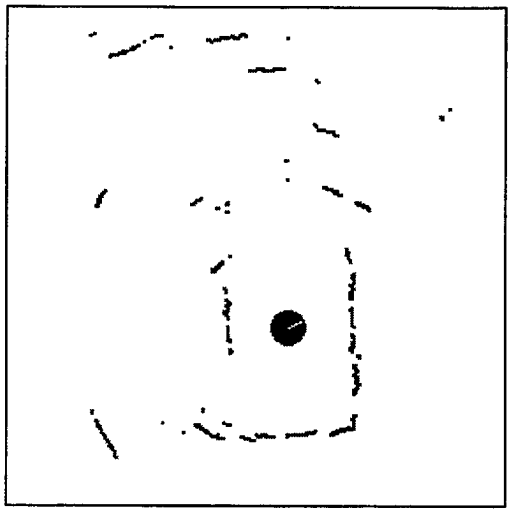

Figure 14. Sonar map of a corner (#1).

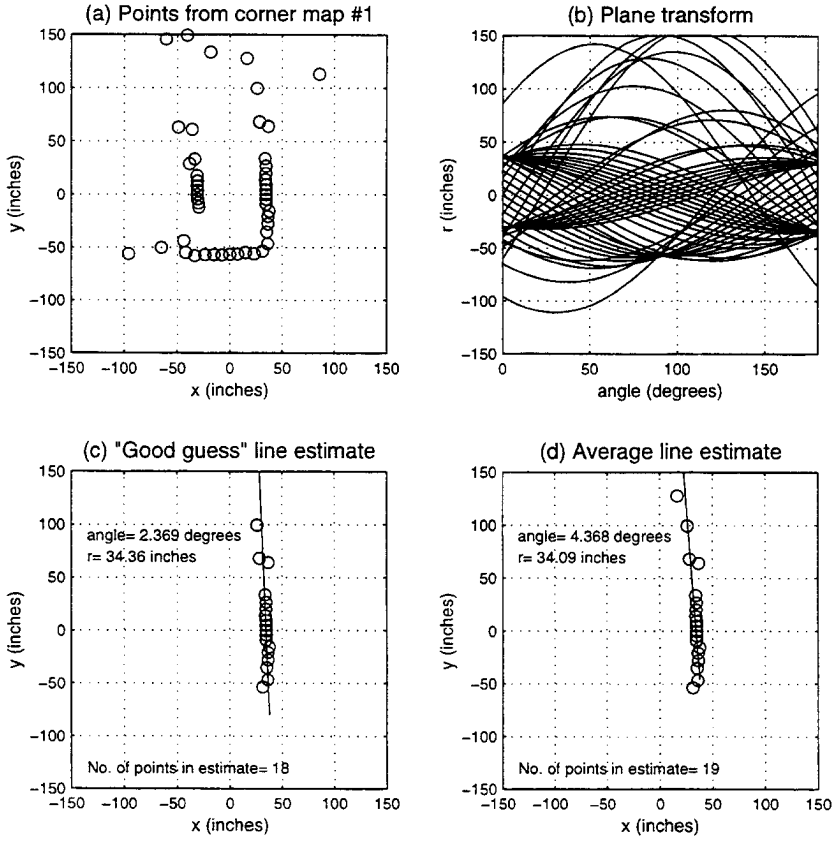

Figure 15. (a) Coordinate plot, (b) plane transform, (c) "good guess" line estimate, and (d) average line estimate for a corner map (#1).

radians (4.36°), which is close to the mean value. (Note: a line representing the exact mean does not actually exist in the experimental results.) The figures suggest that the "good guess" estimate may be a more correct representation of the line; the average estimate is slightly influenced by a few distant points which tend to skew the line.

Table IV. Number of points in the longest line, the angle θ in radians, and the distance r in inches for a corner map (#1).

r-resolution	*θ-resolution*						
	1	*2*	*3*	*5*	*10*	*15*	*20*
1	10 0.015591 34.6	11 0.020558 3407	12 0.025427 34.8	15 0.067529 34.4	17 0.100117 34.3	18 0.147687 34.1	18 0.147687 34.1
2	13 0.027911 34.8	13 0.027911 34.8	13 0.027911 34.8	15 0.067529 34.4	17 0.100117 34.3	18 0.147687 34.1	18 0.147687 34.1
3	13 0.027911 34.8	13 0.027911 34.8	15 0.071760 34.4	15 0.067529 34.4	17 0.100117 34.3	18 0.147687 34.1	18 0.147687 34.1
5	12 0.029225 33.8	13 0.026275 33.9	15 0.049590 3308	16 0.051560 33.9	18 0.086142 33.8	19 0.134223 33.4	19 0.134223 33.4
10	17 0.019783 34.6	17 0.019783 34.6	18 0.041350 34.4	18 0.041350 34.4	19 0.076234 34.1	20 0.126041 33.8	20 0.126041 33.8
15	18 0.041350 34.4	18 0.041350 34.4	18 0.041350 34.4	18 0.041350 34.4	19 0.076234 34.1	20 0.126041 33.8	20 0.126041 33.8
20	20 0.054831 33.2	20 0.054831 33.2	20 0.054831 33.2	20 0.054831 33.2	21 0.106625 32.7	21 0.106625 32.7	22 0.160871 32.0

The line estimates with the minimum and maximum values for θ are shown in Figure 16(a) and (b), respectively. Coincidentally, they occur at the smallest and the largest resolution windows, corresponding to the highest and the lowest degrees of

collinearity. The minimum-θ estimate includes only ten points and represents only a short segment of the line. The maximum-θ estimate is clearly corrupted by the distant points.

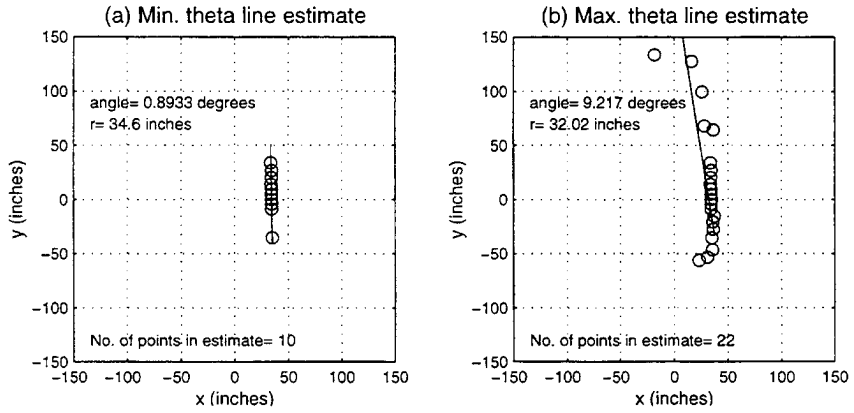

Figure 16. (a) Minimum-θ line estimate and (b) maximum-θ line estimate for the data collected from corner map (#1).

An interesting observation that can be made from Figure 15(c) and (d) and Figure 16(a) and (b) is that neither the corner effects nor the clustering effects were seen in this data.

A second set of sonar data was obtained for the corner configuration with the robot located so that it was at almost equal distance from both walls of the corner. The sonar map is shown in Figure 17. The coordinate plot, plane transform, a "good guess" line estimate and an average line estimate are shown in Figure 18. Results are shown in Table V. The "good guess" estimate is the line with *θ-resolution* = 5° and *r-resolution* = 10 inches. In this data set, θ ranges from a minimum value of 0.1218 radians (6.98°) to a maximum value of 0.2972 radians (17.03°), and has a mean value of 0.2090 radians (11.97°) and a median value of 0.1975 radians (11.32°). The average line estimate of Figure 18(d) is a line obtained with *θ-resolution* = 15° and *r-resolution* = 10 inches; for this line, $\theta = 0.1975$ radians, which is the closest line to the mean in the table and is also the median value. In this case, the difference between the "good guess" and the average estimates is only slightly discernible from the plots.

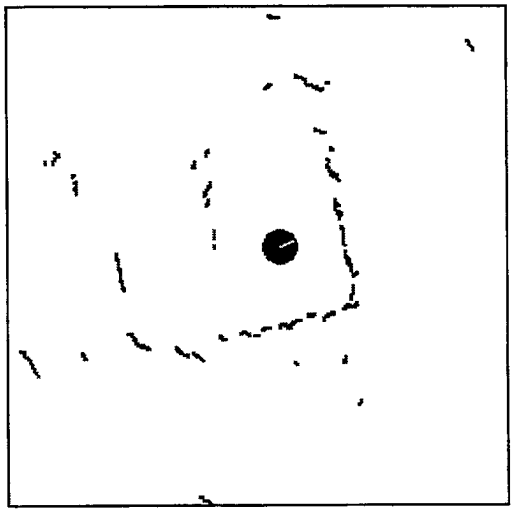

Figure 17. Sonar map of a corner (#2).

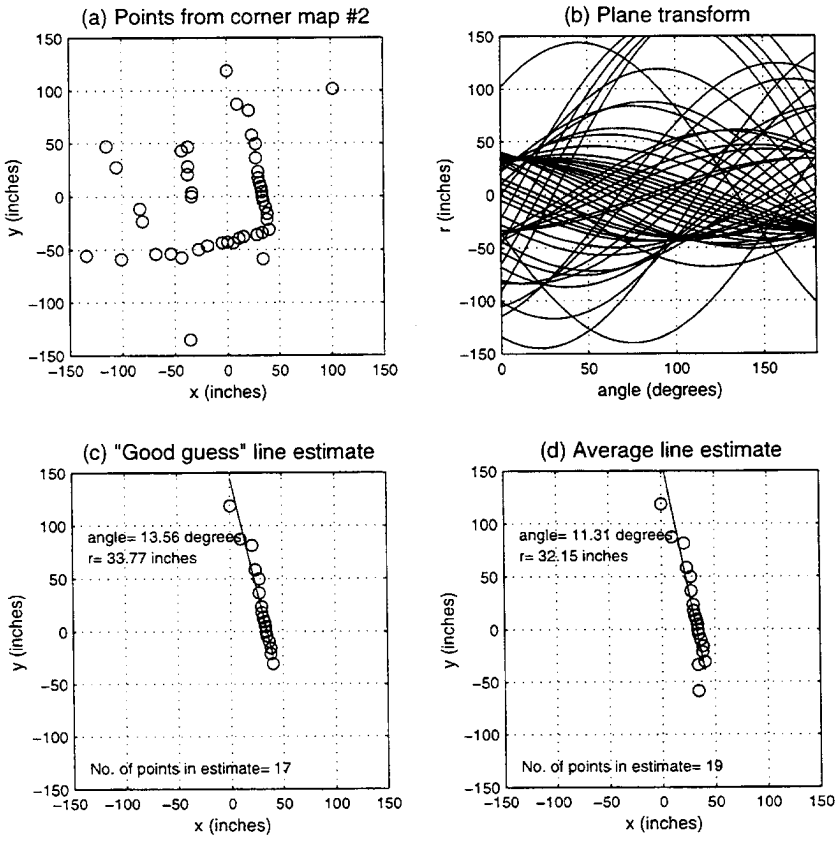

Figure 18. (a) Coordinate plot, (b) plane transform, (c) "good guess" line estimate, and (d) average line estimate for a corner map (#2).

Table V. Number of points in the longest line, the angle θ in radians, and the distance r in inches for corner map (#2).

r-resolution	*θ-resolution*						
	1	*2*	*3*	*5*	*10*	*15*	*20*
1	12 0.184700 33.6	12 0.184700 33.6	13 0.172176 33.7	13 0.189081 33.7	16 0.297184 33.3	17 0.126183 32.9	20 0.250375 32.0
2	11 0.182979 33.6	12 0.184700 33.6	13 0.172176 33.7	14 0.231639 33.3	16 0.297184 33.3	18 0.160909 32.4	20 0.250375 32.0
3	14 0.170384 34.0	15 0.170761 33.9	15 0.170761 33.9	15 0.170761 33.9	18 0.284404 34.0	17 0.194181 31.9	20 0.250375 32.0
5	14 0.167143 33.8	14 0.140667 33.0	15 0.170761 33.9	15 0.148742 33.2	16 0.121845 32.6	19 0.197465 32.1	20 0.250375 32.0
10	16 0.153219 33.4	16 0.153219 33.4	17 0.236717 33.8	17 0.236717 33.8	18 0.284404 34.0	19 0.197465 32.1	20 0.250375 32.0
15	17 0.236717 33.8	17 0.236717 33.8	17 0.236717 33.8	18 0.284404 34.0	18 0.284404 34.0	19 0.197465 32.1	20 0.250375 32.0
20	20 0.183997 31.5	20 0.183997 31.5	21 0.238805 31.2	21 0.238805 31.2	21 0.238805 31.2	21 0.238805 31.2	21 0.238805 31.2

Figure 19(a) and (b) are the minimum-θ and maximum-θ line estimates. Here, these θ values occur at *θ-resolution* = 10° and *r-resolution* = 5 inches for the minimum and *θ-resolution* = 10° and *r-resolution* = 2 inches for the maximum. In Figure 19(a), there is a slight corner effect. In this data set, the corner effect is evidenced by smaller θ values, and since this is the minimum-θ line, this is the line with the strongest corner effect that was found. As in the first corner map results, the maximum-θ line again shows the effect of distant points, and the line is skewed.

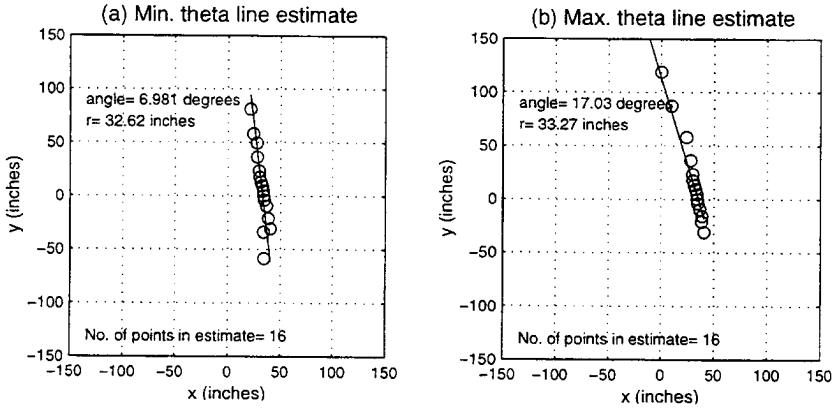

Figure 19. (a) Minimum-θ line estimate and (b) maximum-θ line estimate for the data collected from corner map (#2).

3. Corridor Results

The sonar map for the data collected in a corridor is shown in Figure 20. The coordinate plot, plane transform, and line estimates of each of the two possible lines are shown in Figure 21. In this scenario, the two walls of the corridor were not equivalent in that the wall to the left of the robot (in the top half of the plot) was not a smooth extended surface and was cluttered by various objects, e.g., chalk board, doors, a support column, and a fire extinguisher. The correct wall would then be the wall to the right of the robot, depicted in Figure 21(c). The other wall is shown in Figure 21(d).

The data set was subjected to the same resolution variation exercise as was done for the corner data. The results are tabulated in Table VI. In general, the algorithm successfully found the correct line; the cases where the 'wrong' line was found are shaded in the table. For the correct lines in the data set, θ ranges from a minimum of 1.6123 radians (92.38°) to a maximum of 1.7159 radians (98.31°), and has a mean of 1.6508 radians (94.58°) and a median of 1.6468 radians (94.35°).

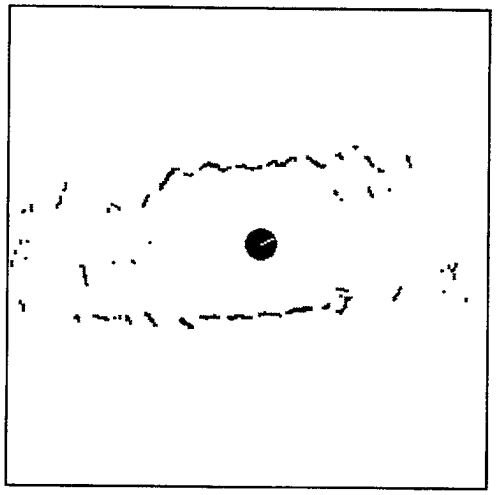

Figure 20. Sonar map of a corridor.

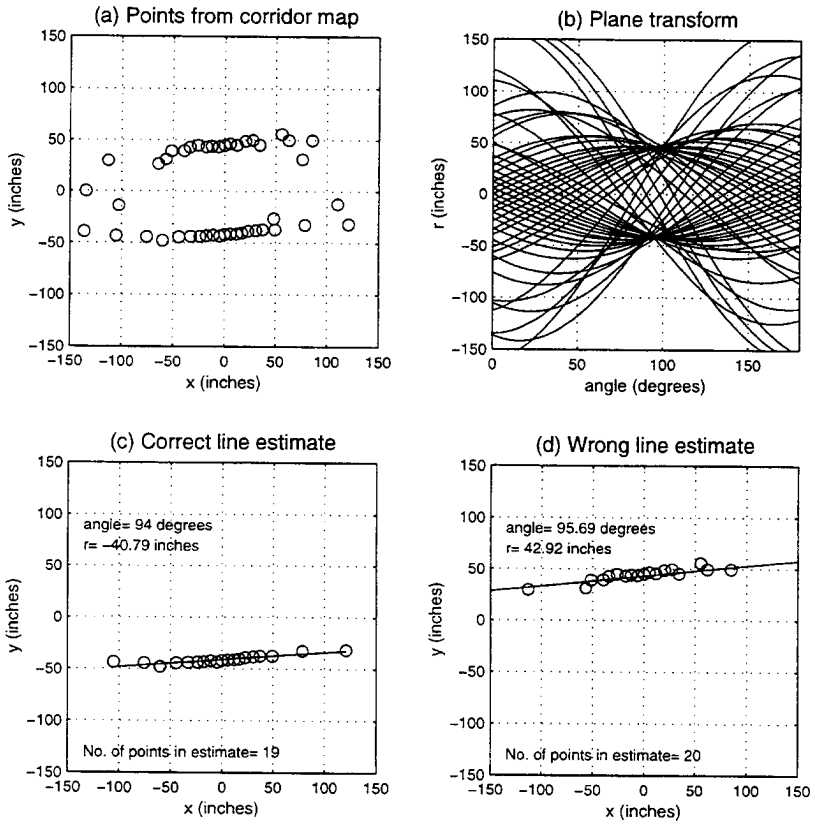

Figure 21. (a) Coordinate plot, (b) plane transform, (c) correct line estimate, and (d) wrong line estimate for corridor map.

Table VI. Number of points in the longest line, the angle θ in radians, and the distance r in inches for the corridor map. (Shading denotes wrong line found.)

r-resolution	*θ-resolution*						
	1	*2*	*3*	*5*	*10*	*15*	*20*
1	11 1.649920 -41.4	14 1.656177 -41.4	14 1.656177 -41.4	13 1.627800 -41.0	18 1.640387 -40.7	20 1.700963 -39.6	19 1.619910 -40.3
2	14 1.642547 -41.0	16 1.661432 -41.2	17 1.653174 -41.0	16 1.630646 -40.7	18 1.640387 -40.7	21 1.715913 -39.0	19 1.619910 -40.3
3	15 1.643732 -41.0	16 1.630646 -40.7	17 1.653174 -41.0	18 1.615943 -40.0	19 1.619910 -40.3	22 1.689885 -38.1	20 1.602665 40.8
5	17 1.662240 -41.3	17 1.662240 -41.3	18 1.653722 -41.1	18 1.653118 -41.0	19 1.640618 -40.8	21 1.715913 -39.0	21 1.606840 40.4
10	17 1.662240 -41.3	18 1.701811 44.1	18 1.653722 -41.1	19 1.705929 43.9	20 1.670125 42.9	21 1.715913 -39.0	22 1.612268 41.0
15	20 1.619825 -40.4	20 1.619825 -40.4	21 1.628448 -39.8	21 1.628448 -39.8	22 1.656345 -38.8	23 1.692189 -37.9	23 1.621339 -37.5
20	17 1.615939 45.0	18 1.701811 44.1	18 1.653722 -41.1	19 1.705929 43.9	20 1.670125 42.9	21 1.715913 -39.0	23 1.614380 -34.0

Several of the cases of a 'wrong' line occur with *r-resolution* of 10 inches and 20 inches. Consider that a grid which is row-discretized using a 10-inch row increment will have every other row division coinciding with the row divisions of a grid using a 20-inch row increment. Therefore, if θ-resolution is the same for both grids, each cell formed in the 20-inch grid will exactly overlap two cells in the 10-inch grid. It is reasonable to expect that for every cell in the 10-inch grid, there is a corresponding cell in the 20-inch grid that will contain *at least* all the same curves. Therefore, for the 10-inch grid cell that

corresponds to the accumulator bin with the most points, if the corresponding 20-inch grid cell does not include any other curve, then both grids will produce the same 'longest' line. In the table, there are three instances of the same wrong line resulting from both the 10-inch and the 20-inch row discretizations, and five instances of the same longest line (regardless of whether or not it is a correct line).

In this real corridor scenario, where one wall was in fact preferable to the other, the algorithm was able to select the preferred (correct) wall in 40 of the 49 resolution windows tested, while with the synthetic data, where neither wall was considered to be preferred, each wall was selected for about half the resolution windows tested. This is perhaps indicative of the fact that, with real data, noisy surfaces will generate noisy data due not only to the actual surface being uneven, but also because the sonar returns will be reflected unevenly.

V. DISCUSSION

In implementing the Hough technique for finding straight lines in a sonar image, this study has shown that a mobile robot equipped with sonar sensors can automatically find and select a wall in its environment as a reference for calibrating its position and orientation. Some issues regarding the selection and estimation of the extracted line must be discussed.

The feature used to select the reference was the longest line found in the image. A concern in selecting the longest line is that the relative extent of walls and wall-like obstacles is represented reasonably well. The rationale was applied (in Chapter III, Section B) that a line found to have the most points would likely be the longest line and, therefore, would represent an appropriate reference wall. This is a relatively safe assumption as long as the data points are fairly evenly distributed in the image obtained. For the data points to be evenly distributed, the data collection procedure must be configured properly. Also, the obstacles in the local area must be at comparable distances so that there is little disparity in the density of the data points representing potential reference walls. For this study, the data points, i.e., sonar reflection points, were distributed evenly so that a short line segment was not likely to have a large number of densely-spaced points in it. Therefore, the line with the most points was likely to be the longest line.

A second concern is in selecting the correct line. Both the synthetic and real data demonstrated the occurrence of erroneous lines. In the synthetic corner data, it was seen that dense clusters of points could lead to a longest line selection that was entirely unsuitable. That is, a line formed by a pair of dense clusters was selected as the reference wall when, in actuality, a wall did not exist there. It was noted, however, that the real corner data exhibited smoother and more even traces of the image than the synthetic corner data (Figure 11(a) versus Figures 15(a) and 18(a)), and the possibility of clustering was not as great. As a result, the correct (desired) line was likely to be selected with the real data. The more difficult problem was the real corridor scenario where one wall was

preferable to the other. From the robot's point of view, the two sides of a corridor could look equally acceptable while, in reality, one side may be cluttered or non-contiguous.

Given that the selected line is the desired 'correct' line, an attempt was made to relate the resolution window to the accuracy of the selected line. The accuracy depends, of course, on the points used in the line fit which, in turn, are influenced by the resolution window. It seems that if the resolution window could be chosen such that it correlated to the allowed tolerance in collinearity, then the line fit would produce an accurate line. Several factors must be considered in determining the tolerance for collinearity. The robot's measurement precision will affect the variance in the data provided by the robot as points where an obstacle has been encountered. Likewise, the physical nature of the sonar reflections will also affect the variance in the data since the echoes are affected by the type and angle of the surface of reflection. This variance in the data determines the tolerance for collinearity. A proper choice for the resolution window which is based on the variance in the data may produce an accurate estimate of that line.

This study did not address line accuracy. This was an acceptable omission since the main purpose of this work was to demonstrate the viability of the Hough technique in enabling a robot to calibrate itself automatically, and not necessarily to verify the parameters of the selected line. It was stated in Chapter IV, Section B, Sub-section 1 that the Nomad 200's exact position and orientation were not measured precisely, and neither were the exact distances and orientations of the obstacles around it, so there was no way of determining the accuracy of the estimate. Instead, the lines resulting from varying the resolution window for each set of data points were compared against each other to observe the influence of the resolution window. However, in order to establish an optimum resolution window, line accuracy must be studied further and precise measurements in both the world coordinate system and the robot coordinate system must be made. If the accuracy of the estimated line can be controlled, the corner effects or the effects of stray points at either end of the line, both of which change the orientation of the line, may be eliminated.

VI. CONCLUDING REMARKS AND FUTURE WORK

The calibration of mobile robot coordinates in a laboratory environment has been investigated, and algorithms have been developed that allow a mobile robot to automatically select existing wall-like features in the laboratory as references against which to localize itself. The Hough technique for finding straight lines was shown to be a viable method for processing a sonar image obtained by the Nomad 200 robot. The sonar data collected by the Nomad 200 resulted in plane transforms with quite distinct intersection regions which enabled the implementation of the Hough algorithm. Additionally, the capability to automatically and simultaneously select and recognize a feature (the longest wall) for calibration, made possible by using the Hough transform, demonstrated the effectiveness of the proposed localization method.

Selecting the reference based on a feature such as the longest line in the image was a straightforward approach. The longest line assumption provided a practical selection criterion which enabled the method to find the desired line in all cases using the real sonar data.

The influence of the resolution window was examined to observe some generalities in the line extracted from the image. As expected, reducing the window provided a good line fit, but it was noted that a good line fit does not necessarily result in the line which represents the reference wall most accurately. The accuracy of the line estimation can only be determined if the precise starting positions and orientations of the robot and the obstacles (potential reference features) around it are known. In future work, a statistical study of the sonar reflections, along with precise measurements of positions and orientations of obstacles in relation to the robot's position and orientation, is recommended so that further study can be pursued to optimize the resolution window.

Also for future work, the algorithm can be augmented with a capability to find a second reference wall which is perpendicular to the first. Hence, both of the positional coordinates, x and y, of the mobile robot can be calibrated; that is the robot can be calibrated in two dimensions.

APPENDIX A. HOUGH ALGORITHM (*hough.c*)

```
/*****************************************************************************
 *
 * PROGRAM: hough.c
 *
 * PURPOSE: To implement the Hough Transform algorithm given a set of coordinate data
 *          points.
 *
 *****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define THETA_RESOLUTION 10.0
#define R_RESOLUTION 10.0
#define PI 3.1415926


struct point {
  int data_index;
  struct point *prev_point;
};

struct accumulator {
  int pt_counter;
  struct point *last_point;
};


void hough_transform(float *all_data, int *no_of_pts, int *line_data_index, int
*no_of_pts_on_line)
{

  int i,j,k,m;
  int no_of_all_data;
  int M_col, N_row;
  float x, y, max_x, max_y;
  float theta, r, nxt_theta, nxt_r, r_max, R, R_max;
  int row, nxt_row, low_row, high_row;
  struct accumulator *A;
  struct point *current_point, *temp_point;
  int THRESHOLD, No_of_lines, most_points;
  struct accumulator *max_bin;
  int *pt_list;

  no_of_all_data = *no_of_pts;


  printf("\nTHETA RESOLUTION= %f \nR RESOLUTION= %f \n",THETA_RESOLUTION, R_RESOLUTION);


 /* Determine the maximum R possible.  Read the input x and y coordinates, determine
  the distance from the origin of each point, and find the maximum distance so that
  the range of r can be bounded.*/

  R_max= 0.0;                          /*Initialize R_max*/
  for (i=0; i<no_of_all_data; i++)     /*Read in points and determine distance*/
   {
    x= *(all_data+2*i);
    y= *(all_data+2*i+1);
    R= sqrt(x*x + y*y);
    R_max= (R_max > R) ? R_max : R;    /*Find the maximum value for R*/
   }

 /*Find maximum r divisible by R_RESOLUTION so we can discretize later to rows*/
  r_max= R_RESOLUTION*ceil(R_max/R_RESOLUTION);
```

```
  printf("R_max=%f,  r_max=%f \n", R_max, r_max);


 /* Define the accumulator array.  We determine the number of columns, M_col, of
  discrete theta increments and the number of rows, N_row, of discrete r increments
  which make up the accumulator array.  Each bin A is indexed by (row,column).*/

  M_col= (int) (180.0/THETA_RESOLUTION);        /*Total number of columns*/
  N_row= (int) (2.0*(r_max/R_RESOLUTION));      /*Total number of rows*/

  printf("Accumulator array:  %d columns, %d rows \n", M_col, N_row);
  printf("Accumulator %d %d\n",sizeof(struct accumulator),sizeof(struct
accumulator)*M_col*N_row);



 /*Allocate memory for accumulator*/
  A = (struct accumulator *)calloc(M_col*N_row, sizeof(struct accumulator));
  printf("Accumulator memory allocated.\n");


 /*Initialize the accumulator bins and point pointer*/
  for (i=0; i<M_col*N_row; i++)
   {
    (A+i)->pt_counter= 0;                   /*Counter starts at zero*/
    (A+i)->last_point= NULL;                /*Point to nothing (NULL)*/
   }
  printf("Accumulator bins and point pointer initialized.\n");


 /* Evaluate each point in the parameter domain.  We generate a theta-r curve
  corresponding to each point and mark the bins that it crosses.  Points that fall
  in the same bin will lie approximately in a line in the x-y plane.  As we keep
  count of the number of curves crossing each bin, we also store the indices of the
  corresponding points and the address of the last point that was stored so that
  we can extract the coordinates later.*/

  for (k=0; k<no_of_all_data; k++)
   {
    x=*(all_data+2*k);                      /*Read the x-coordinate*/
    y=*(all_data+2*k+1);                    /*Read the y-coordinate*/

   /*Look at the accumulator array column by column.  In each column, determine the
    row(s) that the curve crosses.  This gives us the individual bins that the
    curve crosses.*/

    for (j=0; j<M_col; j++)         /*Generate theta-r curve, j is the column number*/
     {
      theta= (float) j * THETA_RESOLUTION;          /*First value of theta to check*/
      r= x*cos(theta*PI/180.0) + y*sin(theta*PI/180.0);
      row= (int)((r+r_max)/R_RESOLUTION);           /*Row below first r*/

      nxt_theta= (float)(j+1)*THETA_RESOLUTION;     /*Next value of theta to check*/
      nxt_r= x*cos(nxt_theta*PI/180.0) + y*sin(nxt_theta*PI/180.0);
      nxt_row= (int)((nxt_r+r_max)/R_RESOLUTION);   /*Row below the second r*/

      low_row = (row < nxt_row) ? row : nxt_row;    /*Lowest row that curve crosses*/
      high_row = (row > nxt_row) ? row : nxt_row;   /*Highest row that curve crosses*/


      /*The following three-step procedure marks the appropriate bins in the column*/

      for (i=low_row; i<=high_row; i++)             /*Row #s of bins to increment*/
       {                                            /*i is the row number of the bins*/

       /*(1) Increment the number of points*/

        (A+i*M_col+j)-> pt_counter +=1;

       /*(2) Create a new point structure that will contain the current point index
        and point to the point structure containing the previous point*/

        if (!(current_point = (struct point *) calloc(1,sizeof(struct point))))
```

```
        {
         printf("No heap memory \n");
         exit(1);
        }

       /*(3) Assign values to store the current point and point to the previous one*/

        current_point->data_index= k;
        current_point->prev_point= (A+i*M_col+j)-> last_point;
        (A+i*M_col+j)->last_point= current_point;       /*Keep new point in the bin*/


       /*printf("Point index=%d col=%d row=%d  counter=%d \n", k, j, i,(A+i*M_col+j)-
>pt_counter); */
      }
     }
   }


 /* Determine the number of possible lines that are present in the data.  This is
  found by examining the total number of points in each accumulator bin and comparing
  it to a pre-selected threshold for the number of points we will consider to be
  adequate to mark a line.  We also now call back the point indices of the curves
  contributing to each bin total. */

  THRESHOLD= 15;          /*Threshold number (integer) of points for a line*/
  No_of_lines= 0;          /*Initialize the line counter*/
  most_points= 0;          /*Initialize the counter to find the most points in a bin*/

  printf("\nThreshold= %d points \n", THRESHOLD);

  for (j=0; j<M_col; j++)
   {
    for (i=0; i<N_row; i++)
     {
      if ((A+i*M_col+j)->pt_counter >= THRESHOLD)
       {
        printf(" Col %d  Row %d, no. of points in bin=%d, points:", j, i,(A+i*M_col+j)-
>pt_counter);

       /*To determine which points are on this line--*/

        current_point= (A+i*M_col+j)-> last_point;
        for (m=0; m<(A+i*M_col+j)-> pt_counter; m++)
         {
          printf(" %d ", current_point->data_index);
          current_point= current_point->prev_point;
         }
        printf("\n");
        No_of_lines +=1 ;            /*Cumulative count of lines in each bin*/

       /*Determine bin with the most points*/

        if (most_points < (A+i*M_col+j)->pt_counter)
         {
          most_points= (A+i*M_col+j)->pt_counter;
          max_bin= (A+i*M_col+j);
         }
       }
     }
   }

  if (most_points == 0){
    printf("THRESHOLD MAY BE TOO HIGH...CHECK AND RESET \n");
    exit(1);
   }

  printf("Total number of lines found= %d \n", No_of_lines);
  printf("Most points found in one bin= %d \n", most_points);

  current_point=max_bin->last_point;
  pt_list= malloc(most_points*sizeof(int));
```

```
    m=0;
    while (current_point){
      pt_list[m++]= current_point->data_index;
      current_point=current_point->prev_point;
    }

 /*  printf("Point list for the bin with the most points: \n");
    for (m=0; m<most_points; m++)
      printf("%d \n", pt_list[m]);
 */
    *no_of_pts_on_line = most_points;

    for (m=0; m<most_points; m++)
      line_data_index[m] = pt_list[m];


  /*Free up the heap memory that was being used for the chains of point structures
    and for the accumulator array.*/

    for (j=0; j<M_col; j++)
      for (i=0; i<N_row; i++){
        current_point=(A+i*M_col+j)->last_point;
        while (current_point){
          temp_point=current_point->prev_point;
          cfree(current_point);
          current_point=temp_point;
        }
    }
    cfree(A);


  /*Should add code to move the point list to the place in memory where the
    accumulator array started, i.e., immediately after all_data.*/

printf("\nEnd of hough.c \n");

} /*THIS IS THE LAST BRACKET*/
```

APPENDIX B. LINE ESTIMATION ALGORITHM (*linest.c*)

```
/*****************************************************************************
 *
 * PROGRAM: linest.c
 *
 * PURPOSE: To estimate a line in the normal parameters given a set of points
 *          extracted by the program hough.c.
 *
 *****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.1415926


void line_estimation(float *all_data, int *line_data_index, int
*no_of_pts_on_line)

{
  int i,j,k,m;
  int no_of_line_data;
  double x, x_2, y, y_2, xy;
  double sum_x, sum_y, sum_x_2, sum_y_2, sum_xy;
  double mu1, mu2, theta, r;
  double x_intercept, y_intercept;


  no_of_line_data = *no_of_pts_on_line;

  printf("\nLine estimation using %d points \n", no_of_line_data);

/*---Determine r and theta-------------------------*/

  sum_x= sum_y= sum_x_2= sum_y_2= sum_xy= r= 0.0;

  for (i=0; i<no_of_line_data; i++){
    x= *(all_data+2*line_data_index[i]);
    x_2= x*x;
    y= *(all_data+2*line_data_index[i]+1);
    y_2= y*y;
    xy= x*y;
    sum_x += x;
    sum_y += y;
    sum_x_2 += x_2;
    sum_y_2 += y_2;
    sum_xy += xy;
   }

  mu1= sum_y_2 - sum_x_2 + ((sum_x*sum_x)-(sum_y*sum_y))/((double)
no_of_line_data);
  mu2= sum_xy - (sum_x*sum_y)/((double) no_of_line_data);

  theta= .5 * atan2((-2*mu2),mu1);
  printf("\n(Computed theta= %f) \n",theta);

  if (theta < 0.0) theta= theta + PI;

  printf("\nCoordinates of points on the line:\n");
  for (i=0; i<no_of_line_data; i++){
```

```
    x= *(all_data+2*line_data_index[i]);
    y= *(all_data+2*line_data_index[i]+1);
    r += ((x*cos(theta) + y*sin(theta))/((double) no_of_line_data));
    printf("%f,    %f; \n",x,y);
   }

  printf("\nmu1= %f, mu2= %f, r= %f, theta= %f \n",mu1,mu2,r,theta);

} /* LAST BRACKET */
```

APPENDIX C. SONAR DATA COLLECTION PROGRAM (*collect_data.c*)

```
/*****************************************************************************
 *
 * PROGRAM: collect_data.c
 *
 * PURPOSE: To collect 16*3=48 data by rotating the robot 7.5 degrees twice.
 *          The 48 x-y data in the robot's start-up coordinates are in all_data
 *          format, that can be passed to the hough_transform() directly.
 *
 *          Add-on: move the robot in newly calibrated x coordinate.
 *
 *****************************************************************************/


/*** Include Files ***/

#include "Nclient.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define ROBOT_RADIUS 8.81
#define PI 3.1415927

/*** Function Prototypes ***/

void GetSensorData(void);
void collect_xy();
void hough_transform();
void line_estimation();

/*** Globals ***/

long SonarRange[16];  /* array of sonar readings (inches) */
long IRRange[16];  /* array of infrared readings (no units) */
long robot_config[4];

float *all_data;
int *line_data_index;
int no_of_line_data;


/*** Main Program ***/

main (unsigned int argc, char** argv)
{
  int i, j, index;
  int order[16];
  FILE *fp, *fp2;

  int max_no_of_all_data = 48;   /*   16 * 3  */
  int no_of_all_data;


  /* Connect to Nserver. The parameter passed must always be 1. */
  connect_robot(1);


  /* Initialize Smask and send to robot. Smask is a large array that
  controls which data the robot returns back to the server. This
  function tells the robot to give us everything. */
  init_mask();
```

```
  /* Configure timeout (given in seconds). This is how long the robot
  will keep moving if you become disconnected. Set this low if there
  are walls nearby. */
  conf_tm(10);


  /* Sonar setup  */
  for (i = 0; i < 16; i++)
    order[i] = i;
  conf_sn(1,order);

  /* Zero the robot. This aligns the turret and steering angles. The
     repositioning junk is neccessary to allow the user to position
     the robot, though don't ask me why. */
    zr();  /* tell robot to zero itself */



  fp = fopen("all.dat", "w");
  fp2 = fopen("line.dat", "w");

  /* Main  */


  /* allocate memory space for all_data */
  all_data = (float *) malloc( sizeof(float) * 2 * max_no_of_all_data);

  /* get all_data */
  collect_xy(all_data, &no_of_all_data);



  /* call hough_transform() here */
  line_data_index =  malloc( sizeof(int) *  no_of_all_data);
  for (i=0; i<no_of_all_data; i++)
    *(line_data_index+i) = 0;


  hough_transform(all_data, &no_of_all_data, line_data_index, &no_of_line_data);


  /* call line() here */
  line_estimation(all_data, line_data_index, &no_of_line_data);



  /* for testing purpose only.  write all_data to a file. */
  for (i=0; i<no_of_all_data; i++)
    {
      fprintf(fp, " %10.2f",    *(all_data+2*i));    /* x coordinate */
      fprintf(fp, " %10.2f \n", *(all_data+2*i+1));  /* y coordinate */
    }



  for (i=0; i<no_of_line_data; i++)
    {
      fprintf(fp2," %f   ", *(all_data+2*line_data_index[i]));     /* x-coord */
      fprintf(fp2, "%f \n", *(all_data+2*line_data_index[i]+1) );  /* y-coord */
}


  fclose(fp);
```

```
  fclose(fp2);

  cfree(all_data);

  /* Disconnect. */
  disconnect_robot(1);
}


void collect_xy(float *all_data, int *pt_to_no_of_all_data)
{
  int i, j;
  int no_of_all_data;
  float x[16], y[16];

  int count = 0;
  int count2, count3;
  int old_steer,old_turret;

  /* get sonar readings and robot configuration */
  GetSensorData();


  for (j=0; j<16; j++)
    {
      /* eliminate sonar reading if it is out of range */
      if (SonarRange[j] != 255)
       {
       /* robot_config[0-1] are int type and must be cast into float.
          Also, they are in 10th of inch, and are converted into
          inches by dividing by 10.0.  SonarRange[] is in inches.

          robot_config[2-3] are int type and in 10th of degree.
          They must be converted to float before passing to sin()
          and cos() and into radians.
          */

        i = count;    /* to simplify the following eqs */

        x[i] = (float) robot_config[0] /10.0 +
          ( (float) SonarRange[j] + ROBOT_RADIUS) *
          (float) cos((double) j * PI/8.0 + (double) robot_config[3] * PI/(10.0
* 180.0));

        y[i] = (float) robot_config[1] /10.0 +
          ( (float) SonarRange[j] + ROBOT_RADIUS) *
          (float) sin((double) j * PI/8.0 + (double) robot_config[3] * PI/(10.0
* 180.0));

        count++;
       }  /* if */
    }   /* for */


    for (j=0; j<count; j++)
      {
       *(all_data+2*j)   = x[j];
       *(all_data+2*j+1) = y[j];
      }



    /* rotate the robot by 7.5 degrees */
```

```
    GetSensorData();
    printf(" steer= %10.2f  turret= %10.2f  \n", (float) robot_config[2]/10.0,
          (float) robot_config[3]/10.0);
    old_steer = robot_config[2];
    old_turret = robot_config[3];

    while ((robot_config[3]-old_steer)<72)
      {
       vm(0, 20, 20);

       GetSensorData();
      }
       vm(0, 0, 0);
       st();


    printf(" steer= %10.2f  turret= %10.2f  \n", (float) robot_config[2]/10.0,
          (float) robot_config[3]/10.0);


    count2 = count;   /* remember where we left off */

    for (j=0; j<16; j++)
    {
      /* eliminate sonar reading if it is out of range */
      if (SonarRange[j] != 255)
       {
       /* robot_config[0-1] are int type and must be cast into float.
          Also, they are in 10th of inch, and are converted into
          inches by dividing by 10.0.

          robot_config[2-3] are int type and in 10th of degree.
          They must be converted to float before passing to sin()
          and cos() and into radians.
          */

        i = count - count2;    /* to simplify the following eqs */

        x[i] = (float) robot_config[0] /10.0 +
          ( (float) SonarRange[j]  + ROBOT_RADIUS) *
          (float) cos((double) j * PI/8.0 + (double) robot_config[3] * PI/(10.0
* 180.0));

        y[i] = (float) robot_config[1] /10.0 +
          ( (float) SonarRange[j]  + ROBOT_RADIUS) *
          (float) sin((double) j * PI/8.0 + (double) robot_config[3] * PI/(10.0
* 180.0));

        count++;
       }  /* if */
    }   /* for */


    for (j=count2; j<count; j++)
      {
       *(all_data+2*j)   = x[j-count2];
       *(all_data+2*j+1) = y[j-count2];
      }


    /* rotate the robot by another 7.5 degrees */
```

```
    GetSensorData();
    old_steer = robot_config[2];
    old_turret = robot_config[3];
    while ((robot_config[3]-old_steer)<72)
      {
       vm(0, 20, 20);
       GetSensorData();
      }
       vm(0, 0, 0);
       st();


    printf(" steer= %10.2f  turret= %10.2f  \n", (float) robot_config[2]/10.0,
          (float) robot_config[3]/10.0);


    count3 = count;   /* remember the counter */

    for (j=0; j<16; j++)
    {
      /* eliminate sonar reading if it is out of range */
      if (SonarRange[j] != 255)
       {
       /* robot_config[0-1] are int type and must be cast into float.
          Also, they are in 10th of inch, and are converted into
          inches by dividing by 10.0.

          robot_config[2-3] are int type and in 10th of degree.
          They must be converted to float before passing to sin()
          and cos() and into radians.
          */

        i = count - count3;    /* to simplify the following eqs */

        x[i] = (float) robot_config[0] /10.0 +
          ( (float) SonarRange[j]  + ROBOT_RADIUS) *
          (float) cos((double) j * PI/8.0 + (double) robot_config[3] * PI/(10.0
* 180.0));

        y[i] = (float) robot_config[1] /10.0 +
          ( (float) SonarRange[j]  + ROBOT_RADIUS) *
          (float) sin((double) j * PI/8.0 + (double) robot_config[3] * PI/(10.0
* 180.0));

        count++;
      }  /* if */
    }   /* for */


    for (j=count3; j<count; j++)
      {
       *(all_data+2*j)   = x[j-count3];
       *(all_data+2*j+1) = y[j-count3];
      }


    /* return the number of data points collected */
    *pt_to_no_of_all_data = count;

    printf("Number of data collected = %d \n", count);
}
```

```
/* GetSensorData(). Read in sensor data and load into arrays. */
void GetSensorData (void)
{
  int i;


  /* Read all sensors and load data into State array. */
  gs();


  /* Read State array data and put readings into individual arrays. */
  for (i = 0; i < 16; i++)
    {
      /* Sonar ranges are given in inches, and can be between 6 and
      255, inclusive. */
      SonarRange[i] = State[17+i];

      /* IR readings are between 0 and 15, inclusive. This value is
      inversely proportional to the light reflected by the detected
      object, and is thus proportional to the distance of the
      object. Due to the many environmental variables effecting the
      reflectance of infrared light, distances cannot be accurately
      ascribed to the IR readings. */
      IRRange[i] = State[1+i];
    }


  for (i = 0; i < 4; i++)
    robot_config[i] = State[34+i];
}
```

APPENDIX D. SYNTHETIC DATA INPUT PROGRAM (*synth_data.c*)

```
/*****************************************************************************
 *
 * PROGRAM: synth_data.c
 *
 * PURPOSE: To read an input file containing synthetic coordinate data
 *          simulating a sonar image for subsequent processing by the Hough
 *          algorithm hough.c.  The first line of the file must contain the
 *          total number of points to be processed.
 *
 *****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


void hough_transform();

float *all_data;
int *line_data_index;
int no_of_line_data;

void main(int argc, char *argv[])
{

  FILE *fp1, *fp2;

  int i,j,m;
  int no_of_all_data;  /* no. of input data points */

  float *line_data;

  /*-------------------------------------------------------------
    This program takes two file name arguments.  The first file
    contains the coordinates of input points. The second file
    contains the coordinates of points that fit into a line.
    -------------------------------------------------------------*/
  if (argc < 3)
    {
      printf("Please enter two file names. \n");
      exit(1);
    }


  /* open the input file. */
  if ((fp1 = fopen(*(argv+1),"rw")) == NULL)
    {
      printf("Can't open %s  \n", *(argv+1));
      exit(1);
    }


  /* open the output file. */
  if ((fp2 = fopen(*(argv+2),"w")) == NULL)
    {
      printf("Can't open %s  \n", *(argv+2));
      exit(1);
    }
```

```
  /* read the input file, which is a 2-dim array of
     the (x,Y) coordinates of the points.
     First, get the number of data points which is
     given by the first integer in the input file.
     Next, allocate memory for storing the data points. */

  fscanf(fp1, "%d", &no_of_all_data);

  all_data = malloc( sizeof(float) * 2 * no_of_all_data);

  for (i=0; i<no_of_all_data; i++)
    {
      fscanf(fp1, "%f", all_data+2*i);     /* x coordinate */
      fscanf(fp1, "%f", all_data+2*i+1);   /* y coordinate */
    }



  /* call hough_transform()   */

  line_data_index =  malloc( sizeof(int) *  no_of_all_data);
  for (i=0; i<no_of_all_data; i++)
    *(line_data_index+i) = 0;


  hough_transform(all_data, &no_of_all_data, line_data_index, &no_of_line_data);


/*  printf("Point list for the bin with the most points from main: \n");
  printf("no of line pt: %d \n", no_of_line_data);

  for (m=0; m<no_of_line_data; m++)
    printf("%d \n", *(line_data_index+m));
*/
  printf("Return from hough.c \n");


  /* call line_estimation()   */

  line_estimation(all_data, line_data_index, &no_of_line_data);



  for (i=0; i<no_of_line_data; i++)
    {
      fprintf(fp2," %f   ", *(all_data+2*line_data_index[i]));      /* x-coord */
      fprintf(fp2, "%f \n", *(all_data+2*line_data_index[i]+1) );  /* y-coord */
    }


  fclose(fp1);
  fclose(fp2);

}
```

LIST OF REFERENCES

Adams, M., N. Tschichold-Gurman, R. Muller, S. Neogy, L. Ruf, S. Vestli, and D. von Flue, "Control and Localisation of a Post Distributing Mobile Robot," *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, Munich, Germany, September 1994, pp. 150-156.

Albayrak, O., *Line and Circle Formation of Distributed Autonomous Mobile Robots with Limited Sensor Range*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1996.

Ballard, D. H., and C. M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

Bauer, R., and W. D. Rencken, "Sonar Feature-based Exploration," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. I, Pittsburgh, PA, August 1995, pp. 148-153.

Crowley, J., "World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging," *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, May 1989, pp. 674-680.

Curran, A., and K. Kyriakopoulos, "Sensor-based Self-localization for Wheeled Mobile Robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993, pp. 8-13.

Drumheller, M., "Mobile Robot Localization Using Sonar," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 9, No. 2, 1987, pp. 325-332.

Duda, R. O., and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, Vol. 15, No. 1, 1972, pp. 11-15.

Feng, L., J. Borenstein, and H. R. Everett, *Where Am I?* University of Michigan Technical Report UM-MEAM-94-21, December 1994.

Gonzalez, J., A. Ollero, and A. Reina, "Map Building for a Mobile Robot Equipped with a 2D Laser Rangefinder," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994, pp. 1904-1909.

Haralick, R. M., and L. G. Shapiro, *Computer and Robot Vision*, Vol. 1, Addison-Wesley, New York, 1992.

Hough, P. V. C., "A Method and Means for Recognizing Complex Patterns," U.S. Patent No. 3,069,654, 1962.

Komoriya, K., and E. Oyama, "Position Estimation of a Mobile Robot Using Optical Fiber Gyroscope (OFG)," *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, Munich, Germany, September 1994, pp. 143-149.

Kuc, R., and V. B. Viard, "A Physically Based Navigation Strategy for Sonar-guided Vehicles," *The International Journal of Robotics Research*, Vol. 10, No. 2, 1991, pp. 75-87.

Leonard, J., and H. F. Durrante-Whyte, "Mobile Robot Localization by Tracking Geometric Beacons," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 3, 1991, pp. 376-382.

Nomadic Technologies, Inc., *Nomad 200 User's Guide*, Rev. 12/93, Nomadic Technologies, Inc., Mountain View, CA, 1993.

Vandorpe, J., H. Van Brussel and H. Xu, "Exact Dynamic Map Building for a Mobile Robot Using Geometrical Primitives produced by a 2D Laser Range Finder," *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, MN, April 1996, pp. 901-908.

Yang, J. A., "An Algorithm for Localization and Positioning Using Linear Combinations of Model Views," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. II, Pittsburgh, PA, August 1995, pp. 549-554.

Yeh, E., and D. J. Kriegman, "Toward Selecting and Recognizing Natural Landmarks," *Proceedings of 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. I, Pittsburgh, PA, August 1995, pp. 47-53.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center 8725 John J. Kingman Road, STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Road Monterey, CA 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Dr. Xiaoping Yun, Code EC/Yx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	4
5.	Dr. Roberto Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
6.	Dr. Spiro Lekoudis, Code 333 Office of Naval Research 800 North Quincy Street, BCT 1 Arlington, VA 22217-5660	1
7.	Mr. James Fein, Code 333 Office of Naval Research 800 North Quincy Street, BCT 1 Arlington, VA 22217-5660	1
8.	Ms. Khine Latt, Code 333 Office of Naval Research 800 North Quincy Street, BCT 1 Arlington, VA 22217-5660	3