AUTONOMOUS NAVIGATION ALGORITHMS FOR INDOOR MOBILE ROBOTS

by

Omid Ehtemam-Haghighi

A thesis submitted to the faculty of graduate studies
Lakehead University
in partial fulfillment of the requirements for the degree of
Master's in Control Engineering

Electrical Engineering

Lakehead University

September 2009

To my mother, father, brothers and sister for their love, encouragement, and support.

Contents

List of Tables

List of Figures

Abstract

In many autonomous mobile robotic applications, the capability of finding a good path is of primary importance, especially for the places which are cluttered by random obstacles that may appear on the robot's path.

Numerous approaches for autonomous mobile robot navigation and obstacle avoidance have already been proposed that utilize different types of sensors as the range finders. The most effective methods are usually the ones that take advantage of sensor fusion to address the shortcomings due to each individual sensor.

Our research focuses on using a color monocular vision system for the self-localization of a mobile robot at the startup phase, while in the navigation phase, the proposed algorithm combines the information collected from the monocular vision system and a belt of ultrasonic sensors mounted on the robot to detect the obstacles and avoid them as it approaches the predefined goal. The experimental results confirm the robustness of our method.

Acknowledgements

First and foremost, I wish to express my deep gratitude to my dissertation advisor Dr. Abdelhamid Tayebi, Professor at the Department of Electrical Engineering at Lakehead University, for his valuable guidance, assistance and support throughout my graduate studies. It was Dr. Tayebi who introduced me to the field of Robotics and helped me to understand this technology. I also would like to thank Dr. Xiaoping Liu for his supports to understand the "Fuzzy Logic Systems".

I wish to thank the members of my master's defense committee for their suggestions and support on this research. Their comments on this research are greatly appreciated.

My parents have encouraged me throughout my education and career, and I will always be grateful for their sacrifice, generosity and love.

Thunder Bay, Ontario
September 3, 2009

Omid Ehtemam-Haghighi

Chapter 1

Background

1.1 Introduction

Autonomous mobile robot navigation has been a challenging topic in robotic research for a long time. Despite what we see in science fiction movies, that robots have the ability to maneuver precisely and easily, in reality it appears to be a very tough task to accomplish. For example, to give a mobile robot the ability to perceive its surrounding environment, such as identifying features and detecting patterns, requires a lot of effort and research in various disciplines such as Electrical and Mechanical engineering, Computer science, etc.

1.2 Why Mobile Robotics Research is Significant?

There are varieties of tasks nowadays that heavily demand using mobile robots and their capability to navigate autonomously, especially in hazardous or inaccessible environments to humans such as submarine explorations, radioactive material removal/displacement, and military/antiterrorism missions.

Besides the typical industrial utilization of mobile robots such as transportation and repetitive tasks, they can also be used in more challenging tasks such as surveillance and inspection; where a robot must make decisions based on its sensory perceptions. It should also be capable of distinguishing important objects from unimportant ones and perform the appropriate action upon detection.

1.3 A Quick Review on History of Mobile Robotic Research

The history of mobile robotics dates back to 1865 [1] when John Brainerd created the Steam Man which was used to pull items. In 1885, Frank Reade Jr. built the Electric Man which was more-or-less an electric version of the Steam Man. Nikola Tesla created a remote controlled submersible boat in 1897. Westinghouse created ELEKTRO, a human-like robot that could walk, talk, and smoke in 1938. W. Grey Walter created his first robots, Elmer and Elsie, known as the turtle robots, in 1949. They were capable of finding their charging station when their battery power ran low. Since that time, the role of artificial intelligence became more evident. In the late 1950s Minskey, Greenblood and Gosper attempted to build a Ping-Pong playing robot. Due to technical difficulties with machine hardware, eventually a robot was built that would catch a ball using a basket instead of the robot's gripper. In 1969, Nils Nilsson at Stanford University built a mobile robot, named SHAKEY, that possessed a visual range finder, a camera, and binary tactile sensors

[1] *http : //pages.cpsc.ucalgary.ca/ ~ jaeger/visualMedia/robotHistory.html*

and was connected to a DEC PDP 10 computer via a radio link [2]. Endowed with a limited ability to perceive and model its environment, Skakey could perform tasks that required planning, route-finding, and the rearranging of simple objects. Since that, time artificial intelligence has made a great progress.

In the late 1970s, Hans Moravec [44] developed CART in the Artificial Intelligence laboratory at Stanford University. The robot was capable of following a white line on a road. A television camera mounted on a rail on the top of CART would take pictures from several different angles and relay them to a computer, which performed obstacle avoidance by gauging the distance between CART and obstacles on its path. In addition to works that have already been done on the mechanical parts of the mobile robots, a tremendous amount of progress has also been made in the image processing field which is a vital part for a mobile robot. Many robots can detect objects based on their colors. Color recognition can be fulfilled in a variety of methods such as Color Profile [19], Spatial-Color Joint Probability [15], Similarity of Color Histograms [59], Histogram Threshold [12], and state-of-the art technologies which are based on Neural Networks [56, 37].

1.4 Thesis Objective

The goal of this research is to implement an autonomous navigation algorithm, with obstacle avoidance, for a mobile robot, using sonar sensors and a vision system. Our experimental platform is a four wheel drive B21r mobile robot equipped with odometers, two belts of sonar sensors, and a color monocular vision system.

Initially, the mobile robot finds its position with respect to the room by deploying a vision-based self-localization algorithm which detects a pair of red and green lines on the floor that are defined as the landmarks.

Knowing the initial position and orientation, the mobile robot moves toward a predefined target using an autonomous navigation algorithm which is constructed based on the concepts of the Vector Field Histogram (VFH)[10] method, which permits the detection of unknown objects and avoiding collisions while simultaneously steering the mobile robot towards the target. The algorithm uses sonar sensors, odometers and a monocular vision system.

[2] *http : //www.ai.sri.com/shakey/*

Chapter 2

A Survey on Vision for Mobile Robot Navigation

2.1 Vision Systems, Image Processing and Visual Servoing

In an intelligent robot, vision is one of the most important senses that helps it to locate or track objects. Machine vision is an interdisciplinary area that includes geometry, physics, computer science, numerical methods as well as probability and statistics [63]. In a typical image processing application for robot vision, five steps are usually considered. First, the image must be grabbed and digitized in a two-dimensional array $I(x, y)$ where x is the column and y the row of the array. Then, by using appropriate algorithms, the disturbance effects such as noise, inhomogeneous illumination, and motion blur are removed from the image. In the third step, so-called segmentation phase, pixels related to the objects are discriminated. Feature values such as height, width, compactness, and circularity of these objects are determined in the fourth step and then are used for object classification in the fifth step.

In high level software tools, image processing libraries generally support steps one to four with operators [20].

2.1.1 Color Models

The color attribute of an object such as hue, saturation, intensity, and spectrum can be used in computer vision to solve a task [34, 21]. Light reflection from other objects, however, can affect the color attributes [20] which sometimes makes it very difficult to determine the right object. This problem can be addressed by choosing an appropriate color space. The *RGB* color space in which each color is a function of the intensity of red, green, and blue spectrums it contains, is an example which suffers from variations in the illumination. This shortcoming is almost remedied in the *YUV* color space in which the color part and illumination part are distinct. Practical experiences confirm that in applications which perform image analysis on color images, the *YUV* color space seems to be a better choice.

The *RGB* color space can be linearly transformed to *YUV* color space by the following transformation matrix [20]:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.514 & 0.101 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.1}$$

where the domain of R, G, B, Y, U and V is $[0; 255]$.

It can easily be shown that if any constant, c, which represents the effect of illumination, is added to the *RGB* color space elements, only the Y component of the *YUV* color space will change and others will stay unaffected [20]:

$$Y(R+c, G+c, B+c) = Y(R, G, B) + c \tag{2.2}$$

$$U(R+c, G+c, B+c) = U(R, G, B) \tag{2.3}$$

$$V(R+c, G+c, B+c) = V(R, G, B) \tag{2.4}$$

Moreover, if the color parts are normalized and the weights are varied, the robustness due to the changes in the illumination can be improved [20]:

$$\begin{pmatrix} Y' \\ U' \\ V' \end{pmatrix} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{2\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{2\sqrt{3}} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.5}$$

2.1.2 Filtering

Image filtering can be used for enhancing the image quality acquired by the robot's camera. As well, filters can be used for extracting special features from an image. This can be helpful for navigating the robot to its target. Mean and Median filters are two types of filters which are widely used in image processing. Both of these filters are used for removing the noise from an image.

In the Mean Filter, the average value of the RGB components of the current pixel and its neighbors that are specified in the filtering window are evaluated and assigned to the corresponding pixel in the output filtered image. The filtering window can have any arbitrary size and can also consist of any combination of the neighboring pixels. It is important to notice that as the size of the filtering window increases (the quantity of the pixels in averaging operation increases), the quality of the resulting image decreases.

The function of the Median Filter is similar to Mean Filter, but instead of mean value, the median value of the pixels in the filtering window is determined and assigned as the new value of the corresponding pixel in the output image. The median is evaluated by ascending order sorting all the RGB values in the filtering window. If the length of the averaging window is an odd number, the value in the center of the sorted arrays for each R, G, and B components is taken as the median value for the corresponding component; otherwise, the average value of the two values in the center of the array for each of the R, G, and B components is calculated as the output. As well as the Mean Filter, the Median Filter reduces the quality of the image.

A Median Filter returns a better result for removing the "salt and pepper noise", because a Median Filter completely eliminates the noise, while the RGB components of the noisy pixels contribute in the averaging procedure in the Mean Filter.

There are also some other filters such as the Highpass Filter, the Band Filter, the Sobel Filter, and the Gabor Filter which are widely used for image segmentation and object detection [13].

2.1.3 Morphological Image Processing

Morphological image processing comprises a set of techniques for digital image processing which are based on mathematical morphology. These techniques are especially suitable for the processing of binary and gray-scale images. In this technique, regions of the image are processed with a *structuring element* which is a small grid and is represented by pixels. The structuring elements can vary in form and size, but all of them constructed from the squares which contain pixels. They also have a pixel in the middle which is considered their reference point.

Erosion and ***Dilation*** operators are two fundamental operations in morphological image processing. In erosion operation, first, it is checked to see if the structuring element completely fits in the pixel set that is supposed to be eroded. If true, the erosion operator constructs an eroded set in such a way that all of the pixels belonging to the original set, that are placed in the structuring element, are transformed to the reference point [20]. The produced image, after applying the erosion operator, is smaller in size compared to the original image size. In contrast to erosion, dilation is a transformation for producing an image with the same shape as the original one, but expanded in size. The size of the produced image, after applying the dilation operator, is directly related to the structuring element. The dilation operator can also be used for merging the edges, or some parts of the image which are close to each other (provided that the distance between these edges or parts comparing to the size of structuring element are short enough). The formal representations of erosion and dilation which are denoted respectively as $E_S(A)$ and $\delta_S(A)$ on a pixel set A with the structuring element S are as follows [20]:

$$E_S(A) = \{p^i | S \subseteq A \quad 1 \leq i \leq n\} \tag{2.6}$$

$$\delta_S(A) = \{p^i | S \cap A \neq 0 \quad 1 \leq i \leq n\} \tag{2.7}$$

where $p^i, i = 1, 2, ..., n$ is the representation for the pixels that are contained in the $E_S(A)$ or $\delta_S(A)$ and the center of S is positioned on p^i.

Edge detection is another technique which is used in morphological image processing. It examines the neighboring pixels around a specified pixel to find the jumps in gray-values. These jumps are considered as the edges in an image. Edge detection comprises the convolution of two matrices. One is the original image I and the other one is a filter mask K. The edges in matrix I can be detected by searching for the jumps in gray-scale values in the resulted matrix J. This task can be accomplished by considering the neighboring pixels of each pixel in x and y directions and calculating the derivative of matrix J in those directions. Since the x and y parameters are discrete in nature, the derivative of matrix J in x and y directions can be denoted as follows:

$$\Delta_x J(x,y) = J(x,y) - J(x-1,y) \tag{2.8}$$

$$\Delta_y J(x,y) = J(x,y) - J(x,y-1) \tag{2.9}$$

The gradient of the image intensity can be determined by using the Sobel operator, which is a two dimensional convolution of two matrices, as follows:

$$G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} * I \qquad G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * I \tag{2.10}$$

The gradient magnitude at each pixel $I(x,y)$ is:

$$G(i,j) = \sqrt{G_x^2(i,j) + G_y^2(i,j)} \tag{2.11}$$

and the direction of the gradient at that pixel is determined by:

$$\Theta(i,j) = \arctan(\frac{G_y(i,j)}{G_x(i,j)}) \tag{2.12}$$

Skeletonization (***thinning***), is a morphological operator which is widely used in applications that incorporate image processing. These applications can vary from character recognition to military route finding. The produced skeleton, after applying the thinning operator, is a compact representation of the image object and preserves the topological properties of the objects [5]. Since thinning procedure reduces the number of pixels of the objects that can be involved in computations, it can improve the computation time.

Skeletonization has been a popular topic in image processing research in the past three decades and many methods have been proposed in the literature which can be classified into two categories [17]: Distance Transform methods and Parallel thinning methods.

The idea of distance transformation, which is proposed for binary pictures in the Euclidean plane by Blum [6], is based on a "fire line"that starts from the circumference of an object and propagates with constant speed towards its inside. The skeleton is formed by the quench points which are defined as the points where two fire lines meet. Many discrete approximations of this fire propagation technique have been reported so far [17]. In most of these algorithms, to avoid the area that has already "burned", the pixels located on the object's contour are traced sequentially. Xia [61], whose algorithm will be discussed in detail in the next chapter, has further refined the sequential tracing by making the next fire front during the tracing of the current fire contour.

Thinning algorithms which are based on the distance transformation method have two advantages: First, they are reversible, therefore the original object can be recovered from the skeleton. Second, they can be utilized efficiently on sequential machines. The disadvantage of these types of algorithms is that it is very difficult to implement them on parallel machines. In Figure (2.1) the result of applying Xia's skeletonizing algorithm to a binary image is shown.

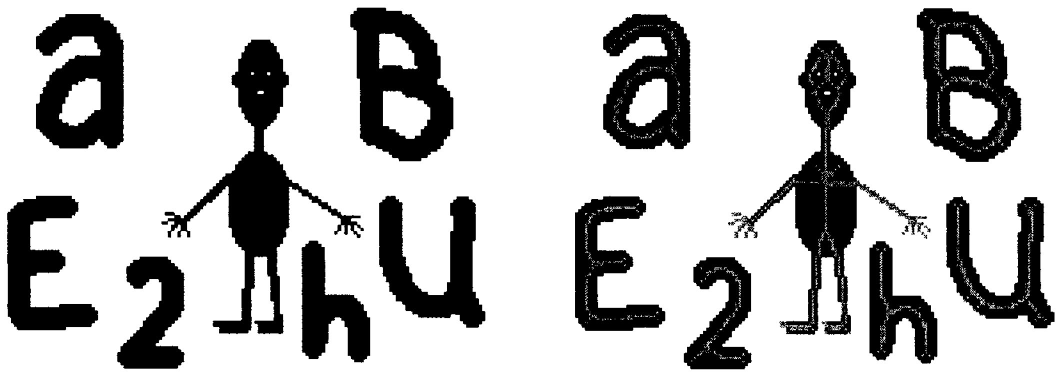


Figure 2.1: Original Image (left), Result after applying a skeletonizing algorithm on the original picture (right).

On the other hand, Parallel Thinning methods use local neighborhood patterns as sufficient evidence to determine whether a pixel is a skeleton pixel or it should be removed [17]. The removable pixels of the object's contour are deleted iteratively by template matching of the neighbors of a pixel with the predefined patterns. Numerous algorithms of this type are proposed and in all of them the selected patterns should guarantee the connectedness of the pixels in the skeleton. The advantage of this type of thinning method is its capability for choosing custom design patterns to delete or preserve certain endpoints which makes the resultant skeleton more interpretable for some specific applications. These algorithms are well designed to take advantage of parallelism [17]. They can also be implemented on sequential machines; however, they suffer from large computation time.

Segmentation, another type of morphological operators, can be applied to an image to partition some distinct regions. Segmentation operation can be carried out top-down or bottom-up. The resulting segments after decomposition have no common region or overlap and the original image can also be recomposed from them. If a fast segmentation algorithm is desired, segmentation can be accomplished locally, in which the pixels that have similar criteria are mixed. However, if segmentation with better resolution is desired, the procedure can be performed globally where it considers larger neighborhoods for calculations [13]. Threshold operation is one of the simplest but effective algorithms that can be used for fast segmentation. In this algorithm, the regions of the image having gray-scale values lower or higher than specified minimum or maximum values are assigned a value of "0"while the rest of the pixels in the image are assigned a value of "1". Furthermore, to cancel the effect of inhomogeneous illumination in the segments, the minimum and maximum values of the pixels can be adjusted automatically by statistics of the smoothed image [60, 33]. The smoothed image can be acquired by applying a mean filter to the original image. The formal notation of threshold operator is as follows:

$$I_T(x,y) = \begin{cases} 1 & if \quad I_O(x,y) \leq I_S(x,y) + \texttt{offset} \\ 0 & else \end{cases} \tag{2.13}$$

where $I_O(x,y)$ and $I_S(x,y)$ are the values of the pixel in the original and smoothed images respectively. I_T is the output in binary format. The parameter "offset"is also necessary to adjust the precision of the output image.

2.2 Vision Based Navigation

2.2.1 Types of Navigation

Based on the research results during the past three decades, it is important for a navigation system to have some knowledge (as database) of what it is supposed to see. In some of the first vision systems, the geometry of the space and metric information were essential issues. Due to this issue, CAD models, which are a type of representation of the interior space were used in early systems. Thereafter, they were replaced by simpler methods like topological maps, occupancy maps, or even sequences of images. In general, all navigation efforts can be categorized in three groups [23]:

1- Map-Based Navigation: Systems that depend on user-created geometric models or topological maps of environment.

2- Map-Building-Based Navigation: Systems that construct their own geometric or topological models of the environment based on the information gathered through sensors.

3- Mapless Navigation: Systems that do not have any knowledge about the space where they are supposed to navigate. In these systems, objects in the environment are detected based on resorting or tracking the objects by generating motions based on visual observations.

2.2.2 Map-Based Approaches

In Map-Based Navigation, the environment surrounding the robot can be represented by variety of models from a simple graph of interconnection between the components to a complete CAD model of the environment. Occupancy map [44], which has been used in the very first vision systems, is an example in which all the objects in the environment are projected into a horizontal plane. In the proposed methods for occupancy map, the VFF (Virtual Force Fields) [8] is an improved representation in which each occupied cell exerts a repulsive force on the robot whereas the goal exerts an attracting force. S-Map [30] is a more sophisticated version of the occupancy map. In this approach the 3D space is squeezed into a 2D map. In some other methods, the errors in the measurement of the position coordinates of the objects in the space, and quantization of those coordinates into "occupied/vacant"cells construct the uncertainty in an occupancy map. Two example of this method are as follows: 1) Histogram grids or Vector Field Histogram [9, 10] in which detection of an object or a vacancy is used to increase or decrease the corresponding cell's value in the grid respectively; 2) Fuzzy operators [48] where a fuzzy value representing the occupancy or vacancy of the corresponding place is held in each cell.

In general, the map models that are used in Map-Based navigation can be coarsely divided in two categories as follows [20]:

- Grid-based Maps, in which it is possible to model the environment as detailed as wished. Egocentric maps [51], which can represent the environment from the robot's point of view, are examples of grid-based maps that can also be developed for recognizing objects with dynamic position. The egocentric maps are usually acquired through sensors such as sonar, a ring bumper or laser range detectors. Moreover by merging several egocentric maps, allocentric maps can be produced.
- Graph-based Maps (Topological maps) are used as graphs in which static objects are represented by nodes of a graph and each node contains the detailed information of the objects to which it is assigned [36]. Topological maps are usually used to solve abstract tasks.

In any map-based navigation, a sequence of landmarks is provided to the robot so that it can find its position by matching between its observation and the data stored in its database. This computation, which is also called self-recognition, can be divided into following four steps [7]:

- **Acquiring the sensory information,** such as capturing digital images.
- **Landmark detection,** which is usually composed of edge detection, filtering, smoothing, segmentation of the image, depth, color or motion detection.
- **Setting up matches between the observation and expectation,** which is usually searching in the database to find some matches with respect to some measured criteria.
- **Calculating the position,** based on the observed landmarks and their position in the database.

For vision-based localization, three different approaches are proposed in the literature which will be described in detail as follows:

Absolute Localization

In absolute localization the initial position of the robot is unknown. In this type of localization, matching between the observation and expectation is established by the navigation system and required information for the operation is derived from the database. In this method the errors with respect to the sensors are represented by tolerances and the effects of uncertainty are analyzed. The same set of observation may match multiple expectations which can lead to ambiguities in localization; however, these ambiguities can be resolved by methods such as Markov localization [58], partially observable Markov processes [55], Monte Carlo localization [26, 16], multiple-hypothesis Kalman filtering based on a mixture of Gaussians [14], and so on. In 1993, a real-time vision-based algorithm for self-recognition was proposed by Atiya and Hager [1]. They set some points as landmarks that are stationary with respect to the position and the orientation of the robot as it moves in its environment. Finding these points based on the received camera images was the main idea of their method. Atiya and Hager used a set-based method to deal with uncertainties with respect to: 1) Ambiguities in constructing the correspondences between the landmark points and observed pixels, 2) Errors with respect to the observed image points, and 3) Errors in determination of the coordinates of the landmark points. In the set-based method, the uncertainty is represented in the form of intervals (sets) and observation of a landmark j in the image plane is expressed by a pair of observation intervals, $o_j = (o_l, o_r)$, with respect to the left and right cameras' image plane. These intervals consist of coordinates of the landmark with respect to the camera coordinates, plus/minus a certain tolerance ϵ which represents the error in sensor measurement. The coordinates of this landmark with respect to the world frame plus/minus a tolerance δ is also denoted by landmark location interval p_j. If the robot's position $\mathbf{P}$ is known, by using a known projection operator $proj(\mathbf{P}, p_j)$, p_j can be projected onto the image plane. The outcome of this projection is an "expected observation interval"h.

By constructing a set of matches, $\Lambda = \{< o_i, p_j >\}$, a set-based model of uncertainty is achieved. Each new observation results in a new interval. Taking the intersection of all of the intervals is the next step. The more the number of observations are, the smaller the intersection will be. Atiya and Hager then divided the problem of finding the position and orientation of the robot $\mathbf{p} = (\mathbf{x}, \mathbf{y}, \theta)$ into two sub problems:

Problem 1. *Given n point location intervals $p_1; p_2; ...; p_n$ in a world coordinate frame that represent the landmarks and m pairs (stereo) of observation intervals $o_1; o_2; ...; o_m$ in the robot camera coordinate frame, determine a consistent labeling, $\Lambda = \{< o_i, p_j >\}$, such that*

$$Proj(\mathbf{p}; pj) \bigcap o_i \neq \emptyset, \qquad \forall < o_i; p_j > \in \Lambda \tag{2.14}$$

(In the case that every o_i participates only once in the labeling, a "consistent"labeling is achieved.)

Problem 2. *Given the above labeling, find the set of robot positions P consistent with that labeling. That is, find the set*

$$P = \{\mathbf{p} \,|\, Proj(\mathbf{p}; pj) \bigcap o_i \neq \emptyset\}, \qquad \forall < o_i; p_j > \in \Lambda \tag{2.15}$$

Practically, the first problem deals with finding the best labeling between $p_j s$ and $o_i s$ from all possible labeling with respect to some criteria. The second problem can also be solved by finding the best labeling Λ, and applying the equations 6 and 7 presented in [1].

Incremental Localization

Using Geometrical Representation of Space: In many practical cases, the initial position of the robot is known at least approximately. Therefore as the robot executes motion commands, the localization algorithm should keep track of uncertainties not to exceed a bound. If it happens, the algorithm fixes the position based on the information it receives from sensors. Moreover, by using some available probabilistic techniques the positional uncertainties of the robot as it moves can be updated. The FINALE system [35] uses one of these approaches based on the following key elements:

- The position is denoted by $\mathbf{p} = (x, y, \phi)$ and the uncertainty in the position is represented by a Gaussian distribution in which $\bar{\mathbf{p}}$ and $\Sigma_{\mathbf{p}}$ are the mean and covariance respectively.
- Based on the type of motion command, translation or rotation, the system will characterize, parameterize and execute it.
- The new uncertainty parameters of the position, $\bar{\mathbf{p}}$ and $\Sigma_{\mathbf{p}}$, according to the type of motion will be determined.
- After derivation of constraint equation based on the camera calibration matrix and linearizing it, the robot's positional uncertainty is projected into the camera image and after finding a set of image features as candidates for each landmark in the environment, $\bar{\mathbf{p}}$ and $\Sigma_{\mathbf{p}}$ matrices are updated by using a Kalman filter.

Propagation of Positional Uncertainty through Command Motions: If the position of the robot after motion command ($\mathbf{p}'$) linearly depends on its position before motion command ($\mathbf{p}$), or $\mathbf{p}' = h(\mathbf{p})$ is linear, the following relation between the uncertainty parameters exists:

$$\bar{\mathbf{p}}' = h(\bar{\mathbf{p}}) \tag{2.16}$$

$$\Sigma'_{\mathbf{p}} = (\frac{\delta h}{\delta p})\Sigma_{\mathbf{p}}(\frac{\delta h}{\delta p})^T \tag{2.17}$$

where $\frac{\delta h}{\delta p}$ is the Jacobean of the transformation function h.

More equations between the initial and final position after a translation or rotation command are presented in [35] which give the nonlinear relation between $\mathbf{p}$ and $\bar{\mathbf{p}}$. This nonlinearity problem, however, can be overcome by using linear approximation. To maintain the approximation valid, the motion commands should be chosen to be small in size.

Projecting Robot's Positional Uncertainty into Camera Image: To find a given landmark in the camera image, the uncertainty in the robot's position should be projected into the camera image. If a single point with the coordinates (x, y, z) in the world is projected into an image plane, its pixel coordinates (X, Y) can be achieved by the following equation:

$$\begin{pmatrix} XW \\ YW \\ W \end{pmatrix} = TH \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.18}$$

where W is the perspective factor, T is the camera calibration matrix and H is the transformation matrix that transforms a point in the world frame into the robot-centered frame. If the position of the world point is given by a position vector $\mathbf{p} = (\mathbf{p}_x, \mathbf{p}_y, \phi)$, after linearization which yields Jacobian for translation and rotation, the robot position covariance can be projected into the image plane and the corresponding Jacobian can be derived from the following equation:

$$\delta I \equiv \begin{pmatrix} \delta X \\ \delta Y \end{pmatrix} = J(\bar{I}, \bar{p}) \begin{pmatrix} \delta p_x \\ \delta p_y \\ \delta \phi \end{pmatrix} \tag{2.19}$$

where (X, Y) is denoted by I and $J(\bar{I}, \bar{p})$ is the Jacobean of I with respect to $\mathbf{p}$ in the vicinity of the mean value. The mean of I and its covariance matrix can be written as:

$$\begin{pmatrix} \bar{I}\bar{W} \\ \bar{W} \end{pmatrix} = T\bar{H} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.20}$$

$$\Sigma_I = J(\bar{I}, \bar{p})\Sigma_p J(\bar{I}, \bar{p})^T. \tag{2.21}$$

The covariance matrix Σ_I can also be used to measure the uncertainty of a single point in the environment.

Kalman Filtering: In fact the Kalman filter is a stochastic method that can basically be used in many application areas. The Kalman filter supports the estimation of a model's state by the use of appropriate model parameters. Machine-vision applications can use the Kalman filter for the three-dimensional reconstruction by analyzing an image sequence that shows the object at different points in time [20]. In FINALE, the statistical parameters of the robot's position after matching a landmark (in the form of a straight line feature) with an image feature, can be updated by using a model-based Kalman filter. The image sequence can be acquired with a moving camera that effects the state transitions.This filter is derived from the linearization of the constraint equations that must be satisfied by the parameters of a straight line in the environment and the Hough space parameters of the corresponding line in the camera image.

Using Topological Representation of the Space: In this method, which is used for incremental localization, a graph topologically represents a layout of the environment surrounding the robot. These graphs are used for the vision process and usually consist of three different types of nodes to represent different locations such as corridors, junctions, and dead ends. Information about the physical distance between the landmarks which are represented by the nodes at each end is also stored in the links connecting them. This method is used in the NEURO-NAV system [40], [39] in which two groups of neural network modules, Hallway Follower and Landmark Detector, construct its heart. Executing the motion commands and self-localization are two different tasks that are carried out by these two groups of neural network modules simultaneously.

The neural networks are stimulated by the cells of the Hough space resulted from the Hough transform of the detected edges in the camera image. Different regions of the Hough space are inputs to different networks and to make the computation faster, the image is down-sampled. To find the edges, a Sobel operator is applied as well.

In NEURO-NAV, the outputs of the different neural networks are fed to a Supervisory Rule-Based Controller and the proper outputs for steering are generated [40]. Later, FUZZY-NAV, which is a more improved version of NEURO-NAV, was developed by Pan et al [49]. This approach is based on the fuzzy nature of the outputs of the neural networks that are typically between 0 and 1. This characteristic can be used as the degree of confidence in the direction of motion with respect to each node. In the new approach, the commands to the Robot Control Module are generated from the *Fuzzy Supervisory Controller* which is a replacement for the *Rule-Based Supervisory Controller* of the NEURO-NAV. In the new fuzzy system, the outputs of all of the neural networks are fed to a real-time fuzzy expert system and then appropriate commands are generated.

Localization Derived from Landmark Tracking

In this method, the mobile robot can be localized if its approximate location and the landmarks seen in the camera are both known and trackable. Landmarks can be artificial such as circles with unique bar-codes, reflecting tapes stretched along the robot path, etc. Landmarks can also be natural ones, like doors and walls. Landmark tracking, in many cases, is accomplished by using a simple template matching. In the case that there is only one camera mounted on the robot, landmarks should be chosen according to some specific characteristics so that the localization of the robot can be carried out by simple algorithms. Tracking can also be made easier when sideways-mounted cameras are used. In the latter case the effects related to scaling and perspective are eliminated.

Another landmark tracking system in the Map-Based approach was also developed by Hashima et al [24]. In this system, by applying correlation techniques, the natural landmarks are tracked and obstacles, if any, are detected at the same time. Computations in this process are carried out by a specially designed hardware TRV [53] capable of calculating local correlation of 250 image regions, while the capturing frequency is 30fps. Localization is carried out by multiple landmark detection algorithms which search for landmarks and compare them with the templates in the database. As the robot moves, the algorithm tracks the landmarks and selects new landmarks to be tracked from the map. By using a stereo vision, the 3D pose of the matched landmark is calculated and after comparing it with the land mark in the map, the position of the robot is estimated.

2.2.3 Map-Building

Generating a model description of the environment is not an easy task specially when it is combined with metric information. Because of that, automated or semiautomated robots capable of exploring their environment have been suggested by some of the researchers. These robots can build an internal representation of the environment surrounding them. One of the useful methods for modeling the environment was proposed by Moravec and Elfes in which the environment is modeled by a data structure which is known as the occupancy grid [44]. The data structure contains the information gathered from ultrasonic sensors. The plane of the floor is gridded as squares and each square is modeled by a cell. A value with respect to the probability of occupancy is assigned to each cell of this grid. Histogram grids or the Vector Field Histogram [9, 10] are improved versions of this method which is used in today's robots.

The maps which are generated by occupancy-grid-based approaches are rich in geometrical details, but they are very sensitive to the accuracy of the sensors and robot's odometer. These shortcomings made the method less efficient for large-scale and complex spaces.

In a new method for map learning, Thrun [57] suggested a combination of occupancy-grid-based and topology based approaches. By using neural networks and Bayesian integration, the system learns how to represent the environment by a grid-based model. The grid-based-representation is then transformed into a topological representation.

There are also some other methods for map building. Panoramic views [62], trinocular vision and Kalman filtering [3, 2], 3D reconstruction from image sequence [64], and sensor fusion techniques [65] are some examples that are used for map building.

2.2.4 Mapless Navigation

In this type of navigation, the environment surrounding the robot is totally unknown. In fact a map is usually built automatically before any navigation can be carried out. However, in the methods surveyed in this section no maps are created and the robot navigates based on the observation and extraction of the information about the elements in the environment. In these methods, navigation can be carried out without the knowledge of the exact position of the elements. Optical-flow-based, appearance-based, and behavior-based approaches are some of the outstanding techniques which are used in mapless navigation. The first two methods are described here and for the latter one the reader is referred to [45], [46], and [25].

Navigation Using Optical Flow

It is believed that the lateral position of the eyes of an insect is used in a navigation mechanism that is based on motion-derived features rather than depth information. In insects, the binocular field is extremely narrow and very little information of depth can be extracted. The depth information, however, can be extracted by motion parallax. By changing the relative speed, an insect can alter the accuracy and the range of operation. A divergent (not aligned) stereo approach imitating the *centering reflex* of a bee, was used in a robot called *robee* [52]. If robee yaws from the middle of the corridor, the velocity of the image seen in the left and right cameras are different and the robot must move toward the side whose images change with a slower rate. In general if u and v are the horizontal and the vertical flow components, the following equation according to the fundamental optical flow constraint can be derived:

$$\frac{\delta I}{\delta x}u + \frac{\delta I}{\delta y}v + \frac{\delta I}{\delta t} = 0 \tag{2.22}$$

For a robot that moves on a flat ground plane, the optical flow along the vertical direction is zero and the above equation can be written as:

$$u = -\frac{I_t}{I_x} \tag{2.23}$$

where I_x and I_t are the x-spatial and time derivatives of the image. It is important to note that before any derivative computation, the images must be smoothed with respect to time and space.

Considering that the image flow of the right and the left camera are in opposite directions, their difference can be expressed as:

$$e = u_L + u_R = T_M(\frac{1}{Z_R} - \frac{1}{Z_L}) \tag{2.24}$$

where T_M is the robot forward motion. Z_R and Z_L are the projection of T_M into the right and left images. By applying a PID controller and the above equation, as described in [52], it is possible to keep the robot centered in a hallway. The controller, however, does not let the direction of the motion be changed (which happens in rotational motion).

Nevertheless, the above mentioned equation is valid provided that: 1) The cameras are mounted as close as possible to the rotation center of the robot, 2) The rotational speed is not larger than a certain factor times the translation speed, and 3) The independent term that appears in the equation of motion is compensated for in a calibration procedure.

The system will also face problems if it cannot detect sufficient texture on any of the walls which makes the robot halt. If the texture on one of the walls is insufficient, the robot will maintain its velocity and direction with respect to the other wall and follows the texture on it. The latter case is called *sustained behavior* [52].

Navigation Using Appearance-Based Matching

In this method, the navigation system can direct the robot to its goal by comparing the images with "memorized"templates of the environment and generating proper commands or controls. Gaussier et al. [22] and Joulian et al. [28] used neural networks for developing an appearance-based navigation system. By merging a series of images taken with 70 degrees of field of view, a panoramic 270-degree image is constructed. To obtain a Vector Field representation of the environment, each vertical row of the panoramic image is averaged. Then for any local maximum of the vector, a local view or a 32 $\times$ 32 subwindow is extracted. A set of all local views for each panoramic image defines a "place"for the robot in the environment. A direction (azimuth) to the goal is assigned to this "place"subsequently. At the end, a neural network which is trained for the association between the "places"and azimuths can lead the robot to its goal. In another method proposed by Matsumoto et al. [38], the robot is guided to its goal by matching between templates and a sequence of images which are down-sampled version of the camera images. Templates are composed of down-sampled images which have been recorded in the training procedure and play the role of "memory". A dedicated processor is responsible for finding the correlation between images. After selecting an image as the "place "of the robot, the displacement between the observed and template images are computed and the steering command can be extracted from a lookup table. This method was enhanced by Jones et al [27] as the images from the database are retrieved by using a zero-energy normalized cross-correlation operator. Later, in 1966 a similar, but faster method that uses the hallway lines as template was introduced by Ohno et al [47]. Their method needs less memory and functions faster.

Navigation Using Object Recognition

Kim and Nevatia [31, 32], suggested a symbolic navigation approach in which the robot takes commands such as *"go to the door"* or *"go to the desk in front"*, etc. Landmark establishment and path selection is based on the symbolic information in the commands. For instance, the robot can realize from a command such as *"go to the desk in front "* that the desk is a landmark and it should follow the straight path ahead of it. Then a "squeezed 3D space into 2D space map"or a "s-map"is constructed and the robot uses a GPS-like path planner to plan its path toward the goal.

Chapter 3

Image Processing Operators for Vision Based Navigation

3.1 Coordinate System

The first step in vision-based robot navigation is the transformation of coordinates between the robot and the vision system. A robot with the coordinate system $\mathbf{X}_M$ which has a camera with the coordinate system $\mathbf{X}_C$, is an example of the case that requires the transformation of coordinate systems. This transformation is composed of translation and rotation. Translation is defined as the distance between the origin of the coordinate system $\mathbf{X}_C$ and the origin of the coordinate system $\mathbf{X}_M$. Orientation of the coordinate system $\mathbf{X}_C$ with respect to the coordinate system $\mathbf{X}_M$ is represented by a vector of angles such as $(\alpha_C, \beta_C, \gamma_C)$ which are the rotations that must be applied to $x_M - axis$, $y_M - axis$, $z_M - axis$ of X_M coordinate system respectively to become parallel with $x_C - axis$, $y_C - axis$, $z_C - axis$ of X_C. Moreover, the location of a coordinate system $\mathbf{X}_C$ in $\mathbf{X}_M$ system can be determined with vector l_C as follow:

$$l_C = (x_M, y_M, z_M, \alpha_C, \beta_C, \gamma_C) \tag{3.1}$$

where position in the $\mathbf{X}_M$ coordinate system is represented by x_M, y_M, z_M, and orientation is denoted by angles α_C, β_C, and γ_C.

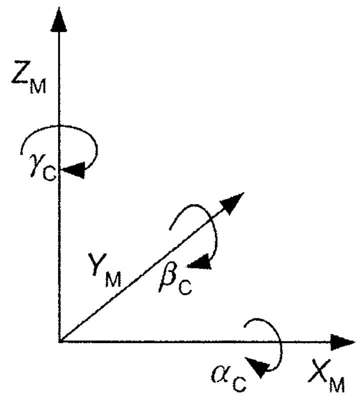


Figure 3.1: The six degrees of freedom.

Orientation of the coordinate system $\mathbf{X}_C$ with respect to the coordinate system $\mathbf{X}_M$ can also be expressed by a 3×3 rotation matrix, M, which is composed of orthogonal unit vectors. In this case, matrix M has the following property:

$$M^{-1} = M^T \tag{3.2}$$

As depicted in Figure (3.1), for transforming the reference coordinate system $\mathbf{X}_M$ to the camera coordinate system $\mathbf{X}_C$, the reference coordinate system should move in the direction of X_M, Y_M, and Z_M as well as rotate around the three corresponding axes by the values of α_C, β_C, and γ_C.

Furthermore, according to the descriptions presented in [20], [50], and [42], if the coordinates of an object in $\mathbf{X}_C$ and $\mathbf{X}_M$ is represented as $(x, y, z)_C$ and $(x, y, z)_M$ respectively, the following relation between the coordinate systems exist:

$$(x, y, z)_C = H_{M,C}.(x, y, z)_M \tag{3.3}$$

where

$$H_{M,C} = (H_{C,M})^{-1} \tag{3.4}$$

and $H_{M,C}$ is a homogeneous matrix.

It is a common case in practical applications to have several coordinate systems. For example, as depicted in Figure (3.2), if the coordinate system of the environment surrounding the robot is represented by $\mathbf{X}_R$ and the coordinate systems of the robot and the camera are denoted by $\mathbf{X}_M$, and $\mathbf{X}_C$ respectively, the coordinates of the objects in the image plane of the camera can be determined in the robot's environment coordinate system, $\mathbf{X}_R$, by the following equations:

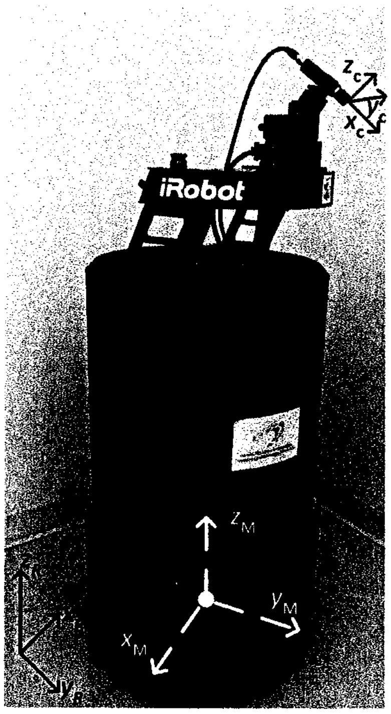


Figure 3.2: View of different coordinate systems.

$$(x, y, z)_R = H_{C,R}.(x, y, z)_C \tag{3.5}$$

and

$$H_{C,R} = H_{M,R}.H_{C,M} \tag{3.6}$$

where $H_{C,R}$, $H_{M,R}$, and $H_{C,M}$ are homogeneous matrices.

3.2 Skeletonization

3.2.1 Yun Xia's Skeletonization Method

Preliminaries

A binary image is defined as a matrix P in which each element (pixel) can have the value of either one or zero. In Xia's method, "0" is a representation for a white color while black color is represented with "1". Objects in the image are represented with black color ("1").

Neighbors: As depicted in Figure (3.3), the eight adjacent pixels n_0,....., n_7 of a pixel P are called "the 8-neighbors of pixel P"and are denoted by $N_8(P)$, while "the 4-neighbors of pixel P", denoted by $N_4(P)$, are defined as n_0, n_2, n_4, and n_6. The n_i of P_k is also denoted by $n_i(P_k)$.

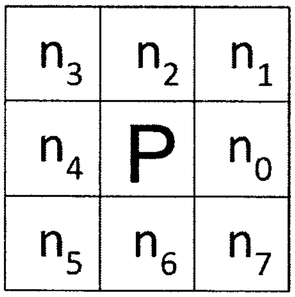


Figure 3.3: Pixel P and its adjacent neighbors

Disk: A pixel together with its j-neighbors ($j = 4$ *or* 8) is called a j-Disk.

Distance: The 4-distance or city block distance between two pixels $I(x_i, y_i)$ and $J(x_j, y_j)$, where (x_i, y_i) and (x_j, y_j) are the coordinates of I and J respectively, is defined as

$$d_4(I, J) = |x_i - x_j| + |y_i - y_j| \tag{3.7}$$

while the 8-distance or the chessboard distance is

$$d_8(I, J) = \max(|x_i - x_j|, |y_i - y_j|) \tag{3.8}$$

Connectedness: A set of pixels is considered to be 4(8)-connected iff there exists a sequence of pixels starting at p_0 and ending at p_n such that any pixel p_i in the sequence of pixels (p_0p_1 .....$p_{i-1}p_ip_{i+1}$.....$p_{n-1}p_n$) is 4(8)-neighbors of the next pixel where $0 \leq i < n$. A curve C from p_0 to p_n is called 4(8)-connected if $\forall p_i \in C$ and $0 \leq i < n$, $p_{i+1} \in N_{4(8)}(p_i)$.

Simple curve: A curve C is called a simple curve if the removal of any pixel from the curve, except the end pixels, violates the 4(8)-connectedness of the curve.

Edge pixel(border pixel):An edge pixel is a black pixel such that at least one of its 4-neighbors is a white pixel.

Contour (border, fire front or wavefront): The set of edge pixels.

Interior pixel: A black pixel which does not belong to any contour.

Interior: The set of interior pixels of an object.

Thin object: An object that has no interior pixel. A thin object is not necessarily a simple curve. For example, two pixel width lines is a thin object but not a simple curve.

As mentioned in the previous chapter, some of the thinning algorithms are based on the Blum's model. In Blum's model an object in the image is modeled by a real object which is composed of dry grass. If a fire starts on all parts of the circumference of the object simultaneously and it burns toward the inside with a constant propagation speed in all directions, the locus of the quench points forms the skeleton of the object. Quench points are defined as the points where two fire fronts meet. This definition works well in the continuous space; however, because of the transformation from the Euclidean metric space to a non-Euclidean one, Blum's idea is not efficient enough for discrete space.

To address this issue, other researchers improved Blum's technique and proposed the grassfire growth and extinction for determining the skeleton in an intermediate stage between continuous and discrete space using a polygonal approximation of the shape [43].

Moreover, Xia [61] used a contour following method to simulate the incineration of the circumference of the grass. This technique will be discussed in detail as follows:

Initial Idea

In Xia's method, instead of conventional scanning of the image for black pixels, the objects' border pixels are tested. Therefore, those pixels that are already marked as burned will not be checked again. Furthermore the next fire front (wavefront) will be constructed during the current contour tracing assuming that those pixels of the next contour which are marked twice or more are actually quench points. In such a way, without any need to check for the neighbor pixels, the value of each border pixels can be used to determine whether it is a quench point.

Fire Propagation

According to Huygens' Principle, each point at a wavefront may be considered as the source that emits a secondary wave. The envelope of all of these secondary waves form the wavefront of the next instant[11]. However, since digitized images are composed of discrete cells, using a circle to represent the second wave does not function efficiently. To address this issue, Xia proposed the notion of 4-disk and potential sector to simulate a circle and the second wave respectively.

Potential Sector: If P_{i-1}, P_i, and P_{i+1} are three consecutive edge pixels, the 4-neighbors of P_i are divided into two parts by the oriented curve $P_{i-1}P_iP_{i+1}$. The part which is located on the left side of this oriented curve (excluding the curve) is called the potential sector of P_i and is denoted by PS(P_i). If all of the pixels of the potential sector of P_i are black (belong to the object), the potential sector represents the second wave which is sent out by P_i (Figure (3.4)).

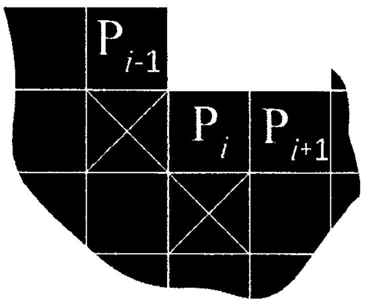


Figure 3.4: P_{i-1}, P_i, and P_{i+1} are three consecutive border pixels. The potential sector of P_i is depicted with crosses.

Therefore to realize the propagation of the grassfire, all the pixels of the potential sector of each border pixel should be marked during the contour tracing.

Fire Extinction

According to Blum's method, those places where two fire fronts meet are considered as quench points. However in a digitized image, if a pixel which is marked twice or more (without considering its neighbors) is considered as a quench point, a faulty skeleton may be achieved. An example of this case is shown in Figure (3.5) where $\underline{C}$ is not a quench point.

In this figure, pixels which are marked as C belong to the 4-neighbors of two successive edge pixels, while pixels that only belong to 4-neighbors of only one edge pixel are marked as N. Quench pixels are also marked as Q where several fire front meet. In fact a quench point belongs to the potential sectors of two border points which are located at two distinct parts of the object's contour and the second wave emitted from them propagate in opposite direction [61]. For instance, Q_1 belongs to both PS(P_{i+1}), PS(P_{i+7}). The same case is applicable to $Q2$ which belongs to PS(P_{i+2}), PS(P_{i+4}), and PS(P_{i+6}). It is important to note that Q_2 does not belong to PS(P_{i+3}), PS(P_{i+5}) because it does not belong to their 4-neighbors.

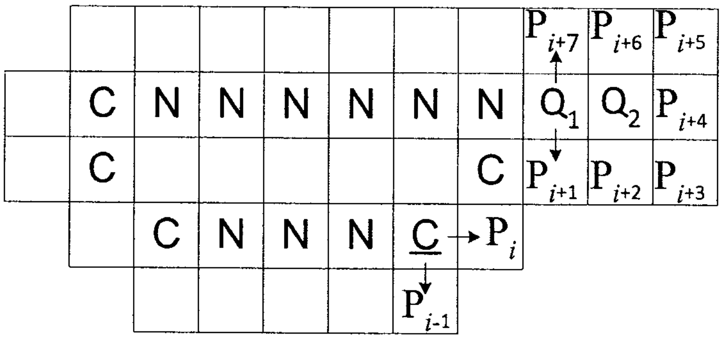


Figure 3.5: $P_{i-1}, P_i, \ldots, P_{i+6}$,and P_{i+7} are consecutive border pixels. Arrows show the association to the pointed edge pixel.

To avoid the artificial intersections, such as the case for $\underline{C}$, and achieve a proper simulation for fire propagation and extinction, Xia proposed the notion of Associated Point.

Associated Point: Let P_i and P_{i+1} be two consecutive edge pixels during contour tracing. The associated pixel(s) of P_i, which is (are) denoted as AP(P_i) is (are) defined as:

$$AP(P_i) = [PS(P_i) - PS(P_{i+1}) \cap PS(P_i)] \cap O \qquad (3.9)$$

where O is the representation for black (object's) pixels and is taken into account to avoid the contribution of white (background) pixels. The term $PS(P_{i+1}) \cap PS(P_i)$ filters those pixels belonging to the potential sectors of both P_i and P_{i+1}. The artificial fire intersection is also prevented with the subtraction sign.

In Figure (3.6), Figure (3.7), and Figure (3.8) different combinations of three consecutive P_{i-1}, P_i, and P_{i+1} and the relative associated pixels for P_i are depicted. In these figures, each row shows different configurations for a pattern that can be acquired by 0°, 90°, 180°, and 270° rotations. It is worthy to mention that to determine the associate pixels of P_i, which are denoted by A, two connected edge pixels located before and after P_i are sufficient.

Note that there is one exception for the equation (3.9) which is depicted in Figure (3.8-c.4)

For the configurations that are shown in Figure (3.8), pixel P_i is marked as an *Extremity* pixel, E, which is preserved as a part of the skeleton.

Algorithm

To implement Xia's algorithm, we assumed that the contour of each object is represented by a *linked list*. Each link of the linked list contains the coordinates of a border pixel as well as two

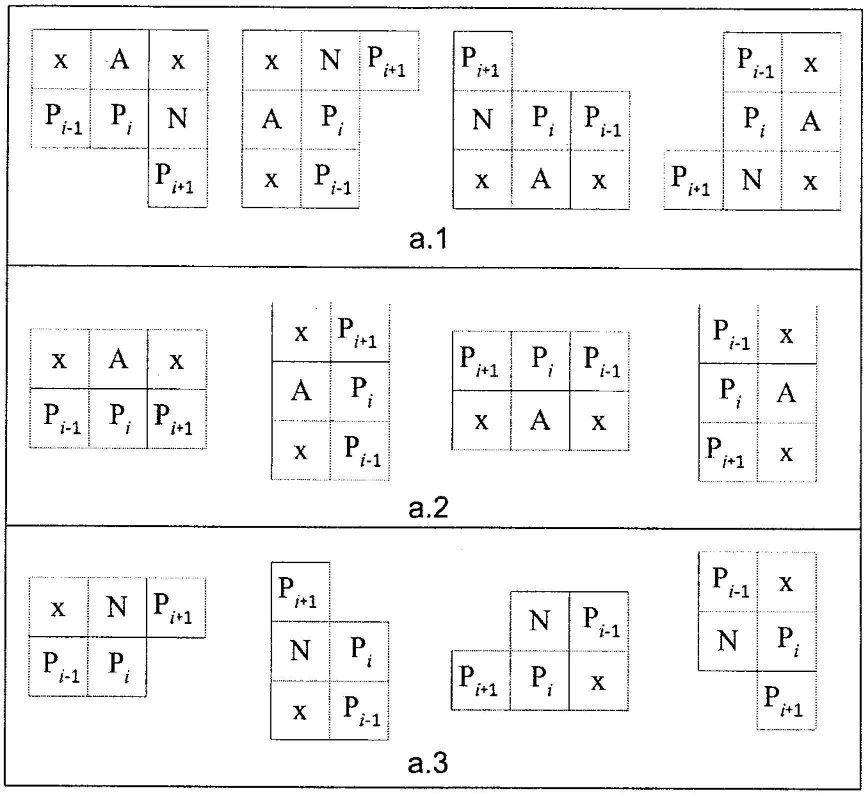


Figure 3.6: A represents associated point of P_i while N is not an associated one. Pixels denoted by x can be either black or white.

other pointers that point to the previous and the next border pixels (links). Furthermore, objects in the binary image form another linked list in which each link contains a pointer to the object's contour linked list as well as two other pointers pointing to the pervious and the next objects (links). The *class* definition in $C++$ for *BorderPoint* and *ContoursStartPoints* are as follows:

```
class BorderPoint
{
  public:
    int x, y;
    int value;
    BorderPoint* NextBorderPoint;
    BorderPoint* PreviousBorderPoint;
};

class ContoursStartPoints
{
  public:
    bool Thinned;
    BorderPoint* firstBorderPoint;
    ContoursStartPoints* NextContour;
    ContoursStartPoints* PreveiousContour;
};
```

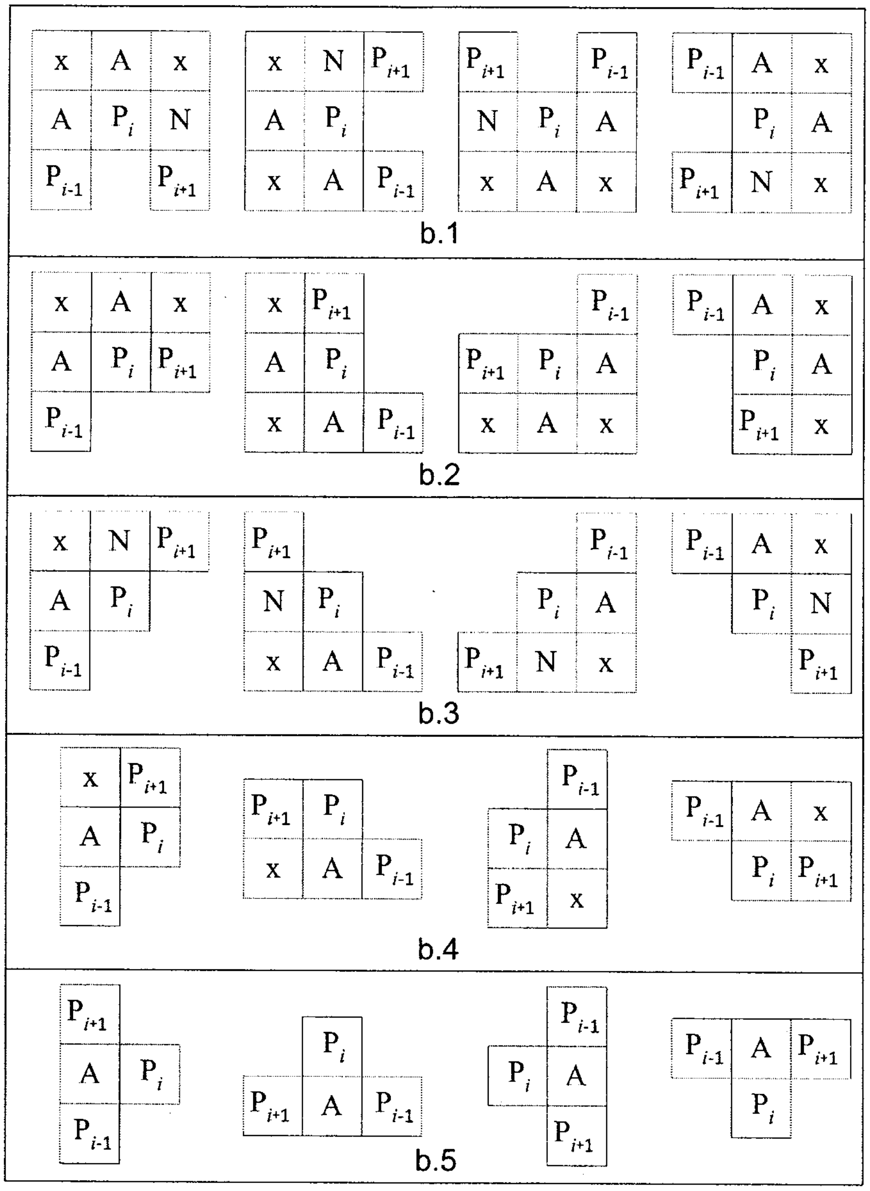


Figure 3.7: *A* represents associated point of P_i while *N* in not as associated one. Pixels denoted by x can be either black or white.

The algorithm is described as follows:

Step 1) The input Image is converted to a binary image and dilated to remove any noise (white pixels in black area) that can disturb the thinning algorithm.

Step 2) The resultant binary image is scanned conventionally (row by row and column by column) until a transition in the color of a pixel from white (0) to black (1) is detected or the end of image matrix is reached. For the latter case the algorithms will be continued in Step 3 while for the former case, scanning freezes and the coordinates of the detected pixel is saved in the first element of the first object's *BorderPoint* linked list. The corresponding cell of the pixel

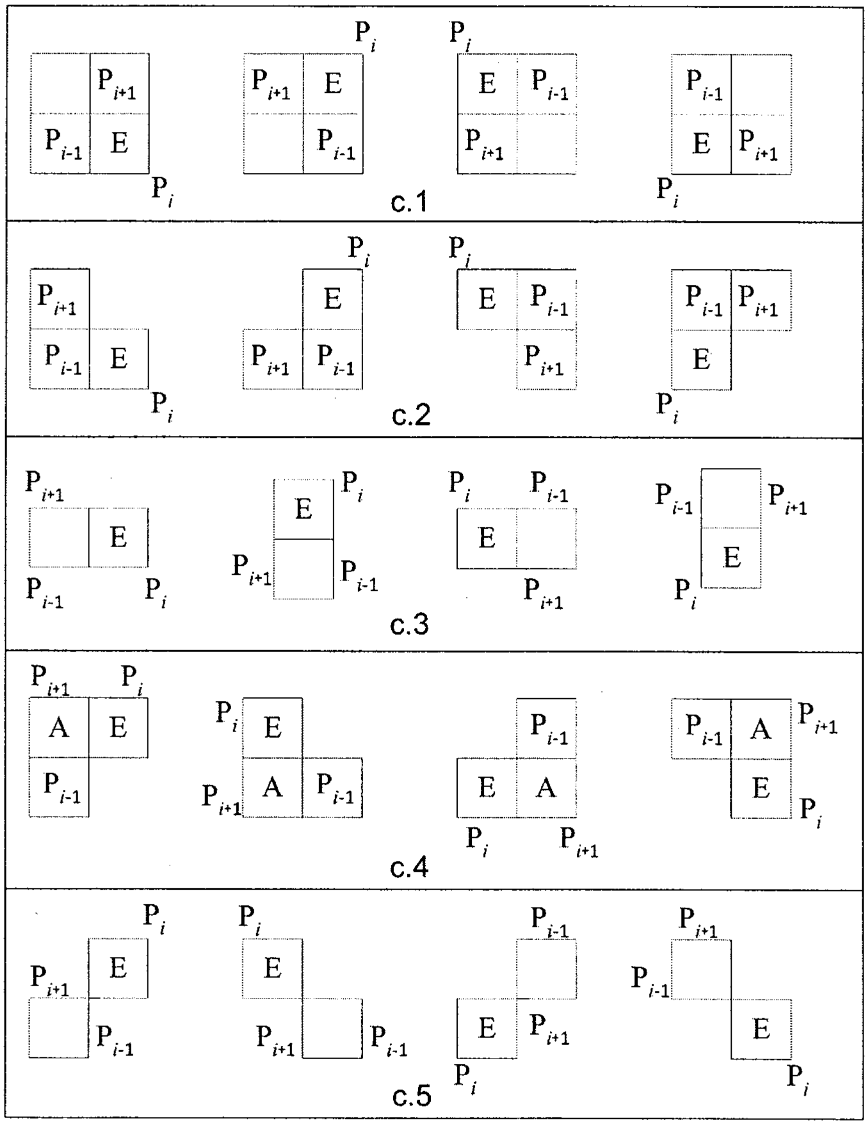


Figure 3.8: A represents associated point of P_i while E represents extremity pixels.

in the image matrix is also incremented by one unit. Thereafter, to follow along the contour, the next border pixel located in the 8-neighbors of the previously detected one is determined (in counterclockwise direction) and its coordinates are assigned to the next element of the current object's *BorderPoint* linked list while its corresponding cell in the image matrix is incremented by one unit as well. The procedure of finding the next border pixel continues until the first border pixel of the current contour is detected as the next neighbor of the current pixel. Thus the contour for the first object is completely determined. Scanning releases and continues until a new border pixel (a new object in the image) is detected or the end of the image matrix is reached (all of the objects in the image are detected). In the later case, algorithm is continued in step 3.

Step 3) Each contour is traced. During the tracing routine, based on matching with patterns in Figure (3.6), Figure (3.7), and Figure (3.8), the values of associated pixels of each border pixel, if any, are incremented and any extremity pixel is determined and is assigned a value of 3.

Step 4) Peeling procedure starts. During this step the contour is traversed and the border pixels in the image matrix that have a value of 2 are assigned a value of zero (white).

Step 5) A linked list of the contours (as described in step 2) is set up again and Step 3 is repeated until all of the objects become thin. The flowchart of this algorithm is depicted in Figure (3.9).

Illustrative Example

In Figure (3.10), an example for better understanding Xia's thinning algorithm and the notion of associated pixel and quench pixel is presented.

First in Figure (3.10-a) the contour of the object is determined and each border pixel is assigned a value of 2.

Then, according to Step 3 of the algorithm, all associated pixels of the border pixels are determined and incremented. The output result is depicted in Figure (3.10-b). The pixels that are associated with two or more pixels and have a value greater than 2 are considered as quench points. As well, those pixels that match the patterns depicted in Figure (3.8) form extremity pixels and are marked as E and assigned a value of 3 so that they can not be eliminated in future iterations.

Next, in Figure (3.10-c), the peeling procedure (Step 4) is applied and the border pixels whose values are less than 3 are eliminated.

Thereafter, by applying Step 5, the next contour is determined and after repeating Step 3, the corresponding associated points and quench points are determined as in Figure (3.10-d).

Finally, the thinned object is achieved in Step 4 whose corresponding result is shown in Figure (3.10-e).

The output of this algorithm when its input is a real binary image is also depicted in Figure (2.1).

3.3 Localizing an Arbitrary Point of the Image Plane on the Floor

Figure (3.11) and Figure (3.12) give the geometrical details of the actual area captured on the CCD image plane of the camera when it is tilted by θ°. The CCD image plane is also depicted in Figure (3.13).

In Figure (3.11), the projection of the camera's focal point, C, on the floor is represented with C_P, and the camera's optical axis is represented by OC which is always orthogonal to the image plane or any virtual image plane parallel with it. The area which is captured by the camera is also denoted by $G_1G_2G_3G_4$ quadrilateral.

If the coordinates of an arbitrary pixel P on the image plane are given, finding the coordinates of its peer on a virtual image plane parallel to the original image plane is a simple task (provided that the distance between the two parallel image planes is given). This resembles the case the camera is mounted on the ceiling and the optical axis of the camera is perpendicular to the floor (Top View Image). In our application, however, the camera mounted on the robot is tilted by a fixed value of θ°, making the $G_1G_2G_3G_4$ quadrilateral as an isosceles trapezoid in which $G_1G_2 || G_3G_4$; therefore the calculations are more complicated (Figure (3.12)). Note that if the camera is both panned and tilted, the $G_1G_2G_3G_4$ quadrilateral will be a trapezium in which none of the opposite sides are parallel.

After further inspection it can be concluded that any row of pixels in the image plane corresponds to a line in the $G_1G_2G_3G_4$ isosceles trapezoid which is parallel with both G_1G_2 and G_3G_4. Under this circumstance and according to the homographic relationship, if all of the pixels located on a same row in the image plane are projected onto the floor, the distance between two

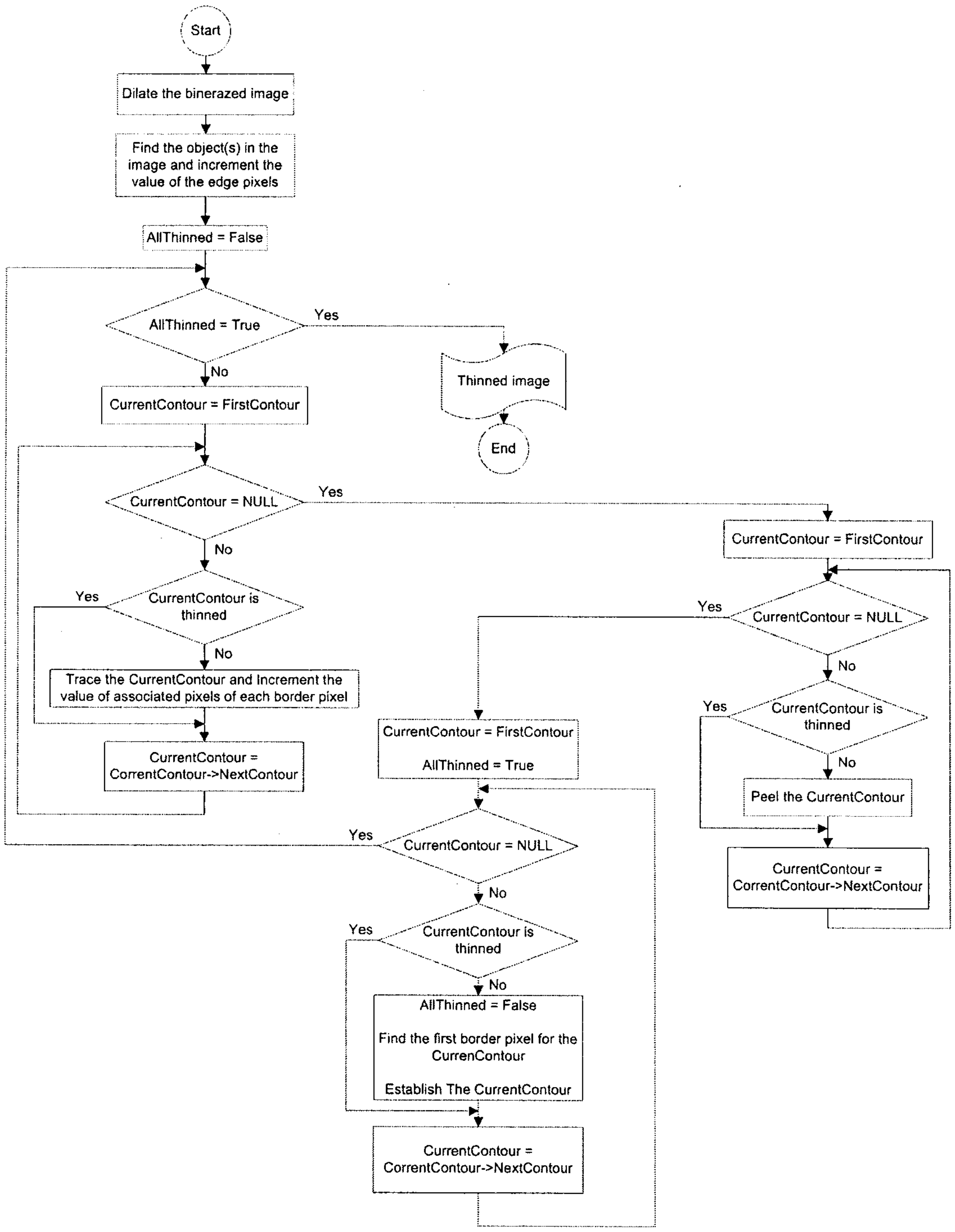

Start
Dilate the binerazed image
Find the object(s) in the image and increment the value of the edge pixels
AllThinned = False
AllThinned = True
Yes
No
Thinned image
End
CurrentContour = FirstContour
CurrentContour = NULL
Yes
No
CurrentContour is thinned
Yes
No
Trace the CurrentContour and Increment the value of associated pixels of each border pixel
CurrentContour = CorrentContour->NextContour
CurrentContour = FirstContour
CurrentContour = NULL
Yes
No
CurrentContour is thinned
Yes
No
Peel the CurrentContour
CurrentContour = CorrentContour->NextContour
CurrentContour = FirstContour
AllThinned = True
CurrentContour = NULL
Yes
No
CurrentContour is thinned
Yes
No
AllThinned = False
Find the first border pixel for the CurrenContour
Establish The CurrentContour
CurrentContour = CorrentContour->NextContour

Figure 3.9: Flowchart of thinning algorithm.

consecutive projected points, which is resulting from the projection of two consecutive pixels on a same row, is always constant. This constant value, however, changes as the location of the row

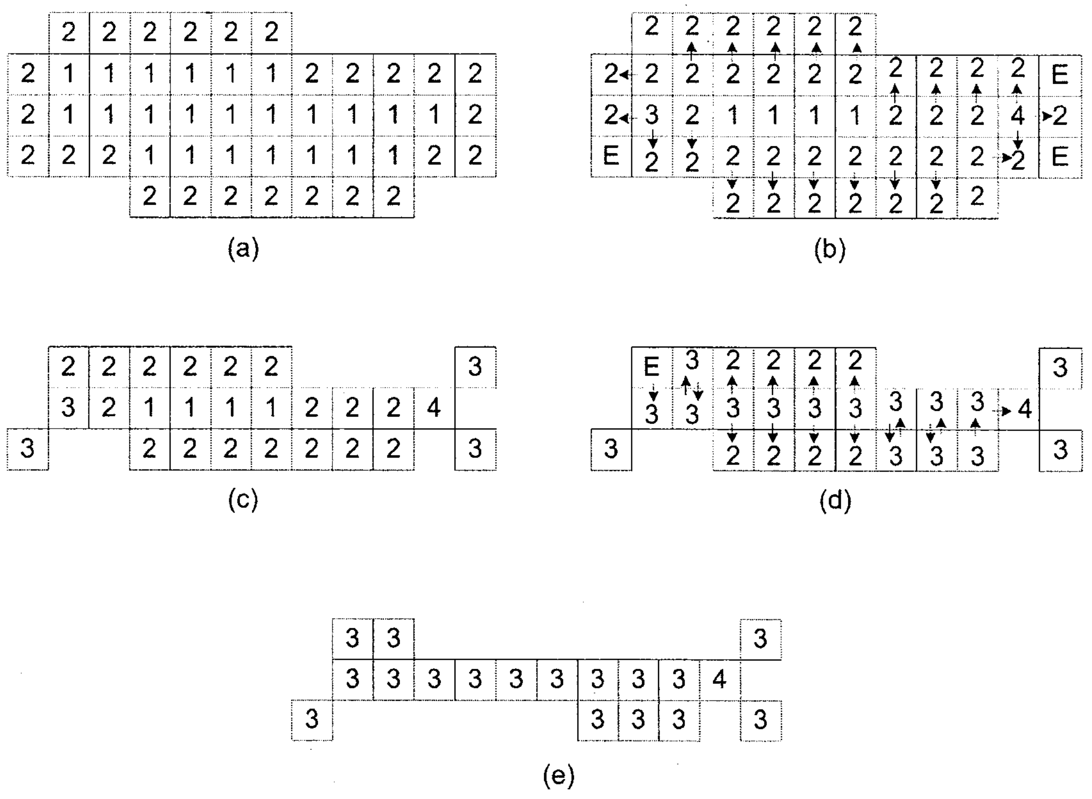


Figure 3.10: The association to each border pixel is shown by arrows.

of pixels changes (Figure (3.12)).

To find the location of a given pixel $P(i,j)$, located on the ith row and jth column of the image plane, the location of the projection of the ith row on the floor should be determined first. For this purpose and to simplify the geometric calculations, the side view of the scene is considered (Figure (3.14), Figure (3.15)).

In these figures, the camera's focal point, the optical axis, and the ith row of the pixels of the image plane are represented with C, OC, and P_i respectively . The optical axis also makes an angle of θ with the normal to the floor. Moreover, the virtual image plane, which is parallel with the original image plane, intersects the floor at a line that passes through its center of gravity (O) and divides it into two equal areas. It is important to note that the derived equations in this section are valid when the floor is flat and θ has a fixed value. Projection of P_i on the virtual image plane and on the floor are also represented with P_{Pi} and P_{Ri} respectively. The image plane of the CCD camera which is used in our project is divided into 120 rows and 160 columns which makes a 160×120 pixel images. Thus, in our calculation, if $i \leq 60$, Figure (3.14); otherwise Figure (3.15) is considered.

In Figure (3.14), since $ON \perp CP_{Pi}$ and $OP_{Pi} \perp OC$, we can conclude

$$\angle OCP_{Pi} = \angle NOP_{Pi} = \alpha$$

In the same way, since $OP_{Ri} \perp CC_P$ and $OP_{Pi} \perp OC$:

$$\angle P_{Pi}OP_{Ri} = \angle OCC_P = \theta$$

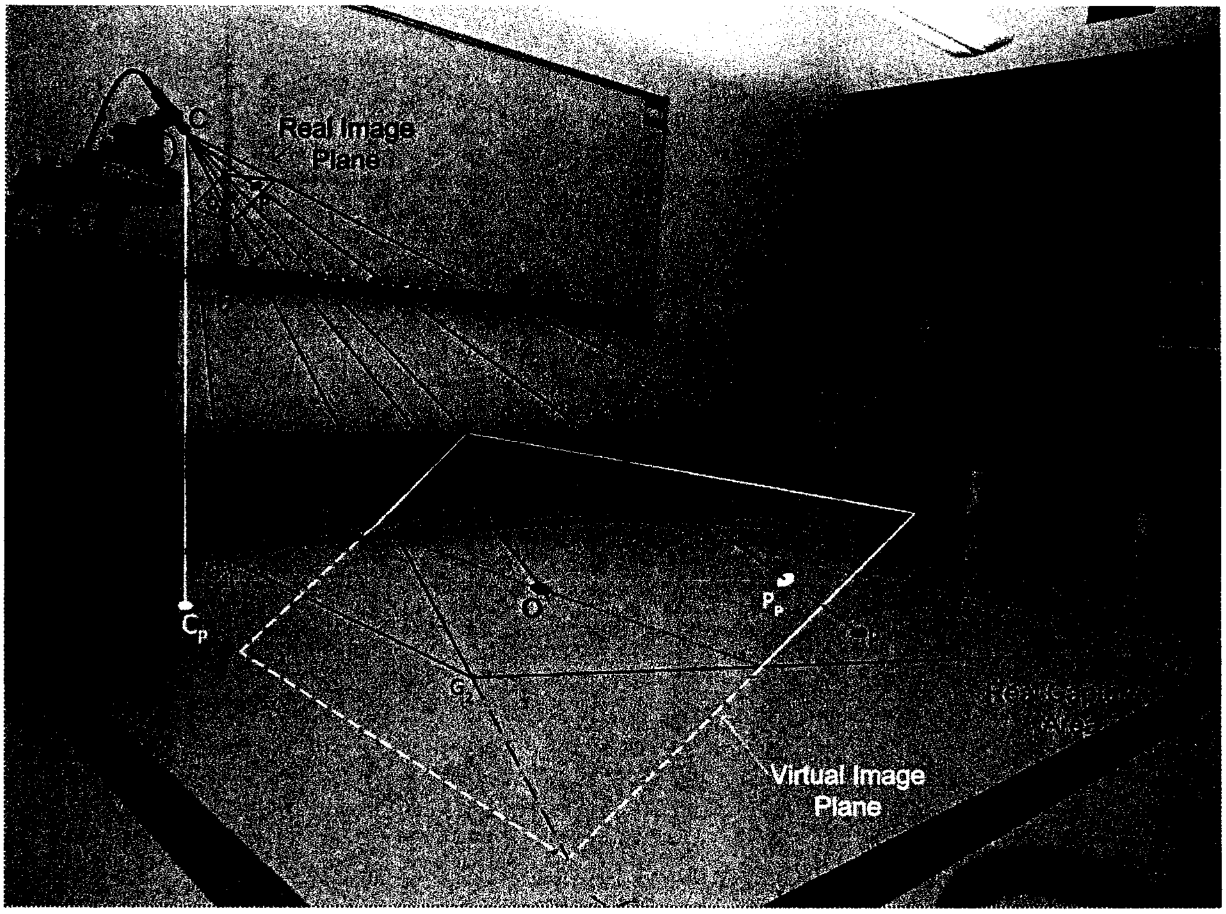


Figure 3.11: The actual area ($G_1G_2G_3G_4$), the Virtual Image Plane and the Real Image Plane

For triangle OCP_{Pi} we can write:

$$\sin\alpha = \frac{OP_{Pi}}{\sqrt{OP_{Pi}^2 + OC^2}} \tag{3.10}$$

and

$$\cos\alpha = \frac{OC}{\sqrt{OP_{Pi}^2 + OC^2}} \tag{3.11}$$

similarly for triangle ONC we can write:

$$\sin\alpha = \frac{ON}{OC} \tag{3.12}$$

On the other hand, for triangle $OP_{Pi}P_{Ri}$:

$$OP_{Ri} = \frac{ON}{\cos(\alpha+\theta)} = \frac{ON}{\cos\alpha\cos\theta - \sin\alpha\sin\theta} \tag{3.13}$$

Combining equations 3.10 and 3.12 we will have:

$$ON = \frac{OC.OP_{Pi}}{\sqrt{OP_{Pi}^2 + OC^2}} \tag{3.14}$$

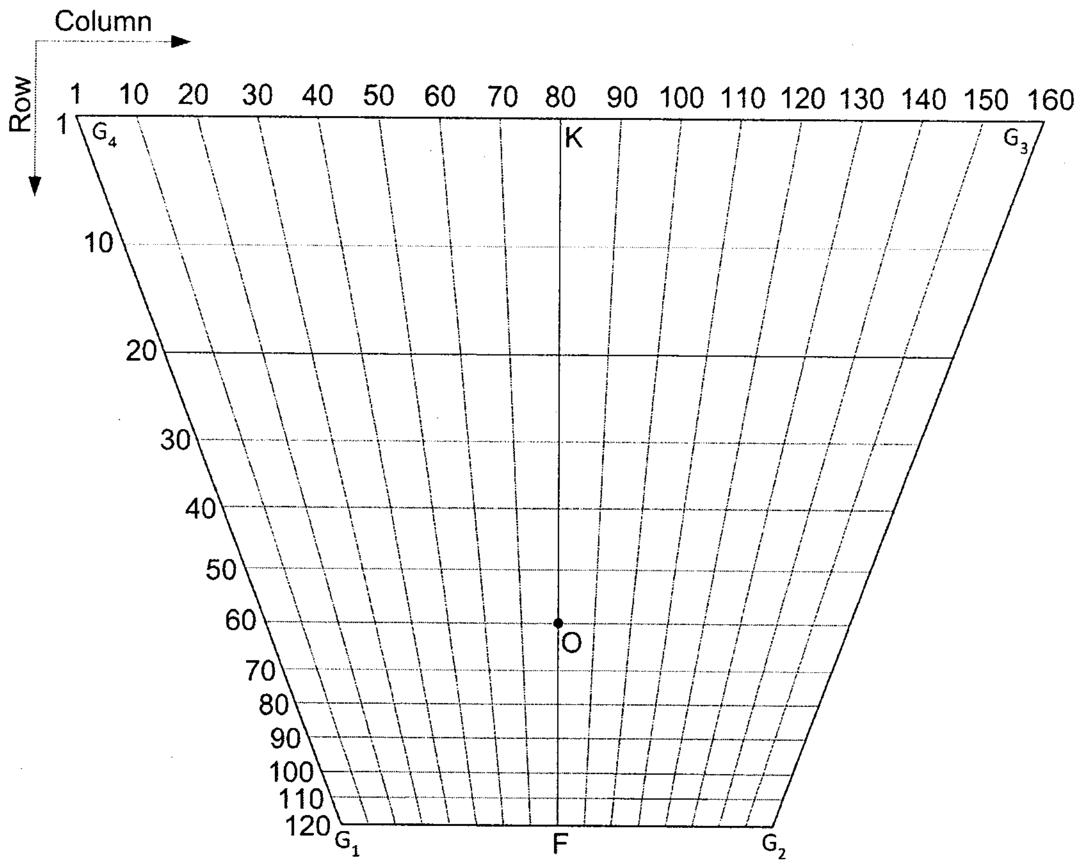


Figure 3.12: Top view of $G_1G_2G_3G_4$.

Substituting equations 3.10, 3.11 and 3.14 in 3.13 yields:

$$OP_{Ri} = \frac{\frac{OC.OP_{Pi}}{\sqrt{OP_{Pi}^2+OC^2}}}{\frac{OC}{\sqrt{OP_{Pi}^2+OC^2}}\cos\theta - \frac{OP_{Pi}}{\sqrt{OP_{Pi}^2+OC^2}}\sin\theta}$$

or

$$OP_{Ri} = \frac{OC.OP_{Pi}}{OC\cos\theta - OP_{Pi}\sin\theta}$$

To generalize this equation for both cases depicted in Figure (3.14) and Figure (3.15), we can rewrite it as:

$$\overline{OP_{Ri}} = \frac{|OC|.\overline{OP_{Pi}}}{|OC|\cos\theta - \overline{OP_{Pi}}\sin\theta} \tag{3.15}$$

where $\overline{OP_{Pi}}$ and $\overline{OP_{Ri}}$ are directional line segments. Furthermore, the value of the following parameters can be evaluated by geometric measurements:

$$|OC| = 160cm$$

$$|OK_v| = |OF_v| = 64cm$$

$$\theta = 40^\circ$$

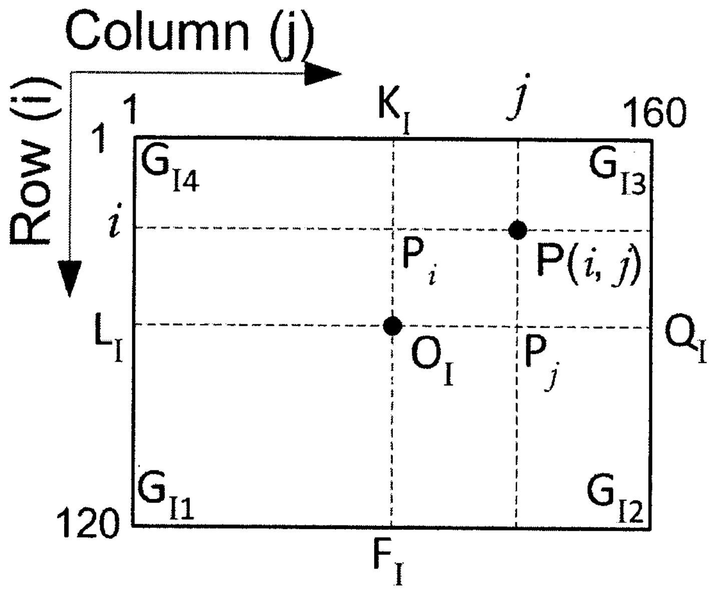


Figure 3.13: CCD camera's image plane.

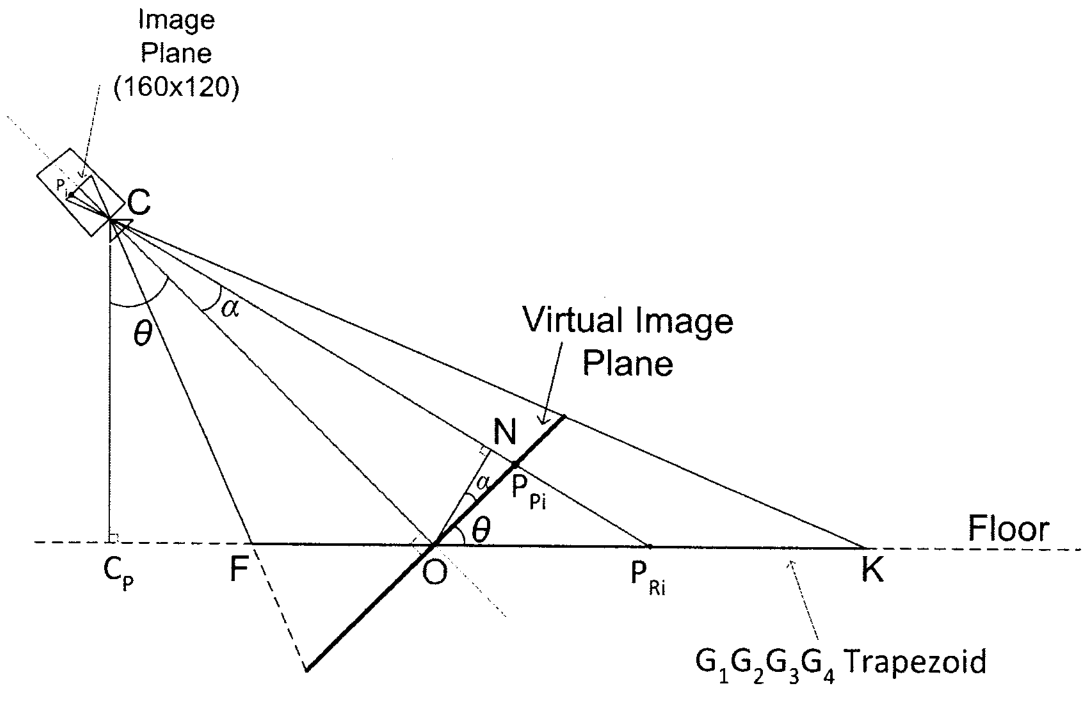


Figure 3.14: Side view ($i \leq 60$).

$$|G_1G_2| = 124cm$$

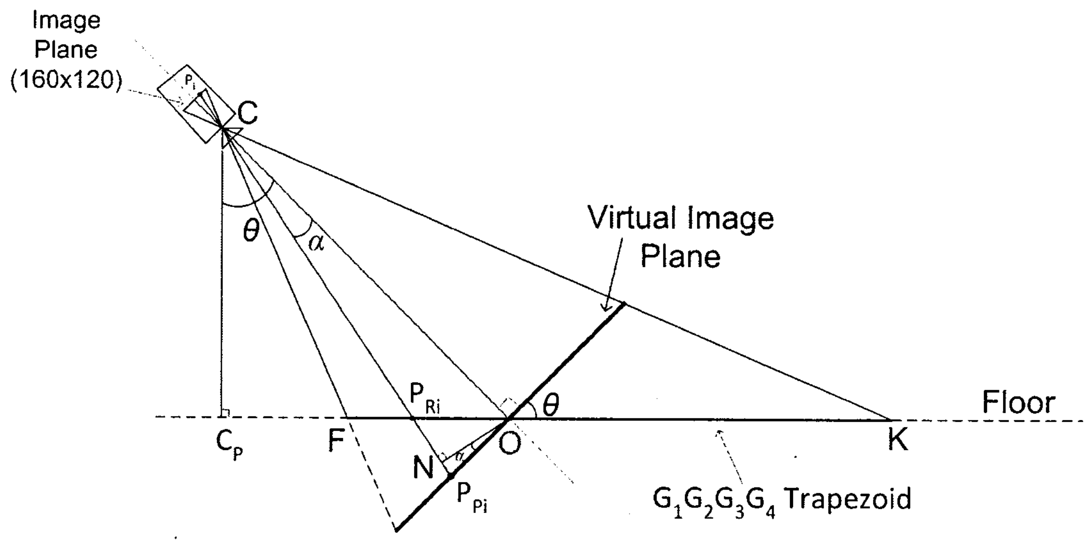


Figure 3.15: Side view ($i \geq 60$).

According to Figure (3.14) we can write

$$\frac{O_iP_i}{OP_{pi}} = \frac{O_iK_i}{OK_v}$$

In Figure (3.13), since P_i is measured from the top left corner, we can conclude

$$O_iP_i = 60 - i$$

and

$$OP_{Pi} = \frac{|OK_v|(60-i)}{60} \qquad i \leq 60$$

A similar result will be achieved if we redo the calculation for Figure (3.15). The general equation for OP_{Pi} can be written as:

$$\overline{OP_{Pi}} = \frac{|OK_v|(60-i)}{60} \qquad 1 \leq i \leq 120 \tag{3.16}$$

Thus, if a row of pixels such as P_i in the CCD image plane is given, its projection on the floor, OP_{Ri}, can be determined by substituting the result of equation 3.16 into equation 3.15.
After finding the location of the projection of the ith row of the image plane on the floor, the location of the jth column of the ith row can be determined as follows:

Considering Figure (3.16), if H_1H_2 is resulted from the projection of the ith row of the image plane on the floor, to localize the point $P_R(i,j)$, the length of H_1H_2 should be calculated first:

$$|H_1H_2| = |G_1G_2| + 2|FP_{Ri}|\tan\varphi$$

$$|FP_{Ri}| = |OF| + \overline{OP_{Ri}}$$

where $\varphi = 24°$ which is found by some geometric measurement and OF can be calculated by substituting $i = 120$ in equations 3.16 and 3.15.

Once $|H_1H_2|$ is determined, according to the homographic relation of each row of the image plane and its projection on the floor (Figure (3.13) and Figure (3.16)) we can write :

$$\frac{\overline{O_IP_j}}{\overline{O_IQ_I}} = \frac{\overline{OP_{Rj}}}{\overline{P_{Ri}H_2}}$$

or

$$\overline{OP_{Rj}} = \frac{|H_1H_2|}{160}(j-80) \qquad 1 \leq j \leq 160 \tag{3.17}$$

Finally, if C_P, which is the projection of the camera focal point on the floor, is considered as

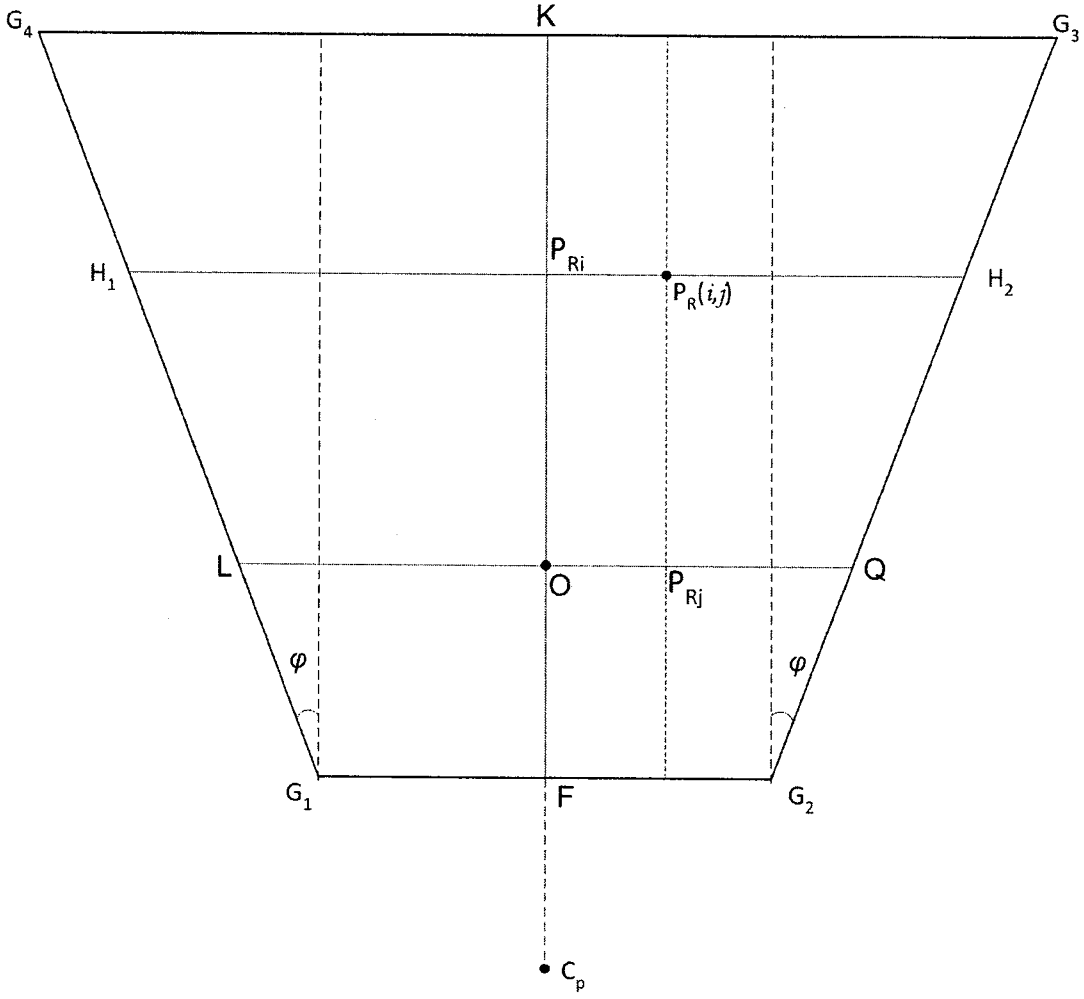


Figure 3.16: Top view of $G_1G_2G_3G_4$. The projection of center of image plane on the floor is depicted with O.

the origin of the coordinate system, the coordinate of the point $P_R(i_C, j_C)$ resulted from the projection of pixel $P(i,j)$ of the image plane on the floor can be determined by the following equations:

$$\overline{OP_{Pi}} = \frac{|OK_v|(60-i)}{60} \tag{3.18}$$

$$i_C = \frac{|OC|.\overline{OP_{Pi}}}{|OC|\cos\theta - \overline{OP_{Pi}}\sin\theta} + |OC_P| \tag{3.19}$$

$$j_C = \frac{|G_1G_2| + 2(|OF| + i_C - |OC_P|)\tan\varphi}{160}(j-80) \tag{3.20}$$

where

$$|OK_v| = 64cm$$
$$|OC| = 160cm$$
$$|G_1G_2| = 124cm$$
$$|OC_P| = 113cm$$
$$\theta = 40^\circ$$
$$\varphi = 24^\circ$$

are determined by geometric measurements while $|OF|$ is calculated by substituting $i = 120$ in equations 3.15 and 3.16 ($|OF| = |OC_P| - (i_C|_{i=120})$).

3.4 Hough Transform

Hough transform is a technique which is widely used in image processing, computer vision, and vision based navigation to extract the desired features such as lines, circles, etc. from an image. This technique which is based on the voting procedure can be utilized for finding the imperfect instances of the objects within a certain class of shapes. The algorithm constructs an accumulator space, so called Hough space, and applies its voting procedure on the black pixels of the input image. Those cells of the Hough space which are corresponding to some local maxima, can be considered as the candidates for belonging to a certain class of shapes.

The conventional Hough transform is applied for finding the lines in the image; however, it is extended to determine the position of other shapes such as circles or ellipses [1]. In practical automated applications in which digital image processing is involved, identifying simple shapes such as lines or circles is usually a goal. In many cases, finding the pixels in the image that lay on a desired curve can be carried out with an edge detector such as the Sobel edge detector. However, as the result of low quality, imperfection, non-homogeneous illumination and so on, some pixels of the image laying on the desired curve can be missed or not detected by the edge detector. Therefore it is usually non-trivial to categorize the extracted edge features into an appropriate set of lines, circles, etc. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects [54].

Detecting straight lines can be considered as the simplest case of utilizing the Hough transform. Each straight line in the image space has the equation of $y = mx+b$. Therefore each x will have a y as its peer, satisfying this equation. The main idea of the Hough transformation is to parameterize each line based on a pair of identifiers such as (m, b).

Utilizing pure slope m and intercept parameter b of a line as the identifiers for lines will cause the Hough transform to be inefficient especially when dealing with vertical lines ($m \rightarrow \infty$) which makes the Hough space unbounded. This shortcoming, however, can be bypassed if the line is parameterized by another pair of parameters (ρ, θ) as shown in Figure (3.17), where ρ is the shortest distance between the origin and the line, and θ is the angle between the x axis and the normal vector from origin to the line. In such a case, the equation of the line can be rewritten as [18]:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{\rho}{\sin\theta}\right) \qquad or \qquad \rho = x\cos\theta + y\sin\theta \tag{3.21}$$

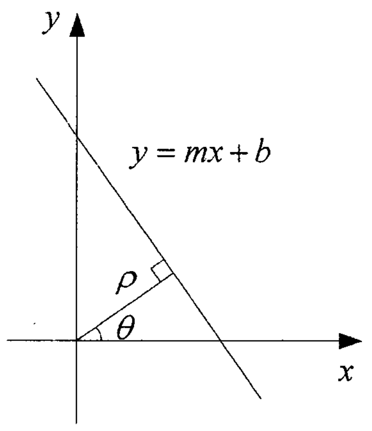


Figure 3.17: The normal parameters for a line

As a result, a pair (ρ, θ) in the Hough space is associated with only one line in the image plane provided that $\theta \in [0, \pi]$ and $\rho \in \mathbf{R}$, or $\theta \in [0, 2\pi]$ and $\rho \geq 0$.

Moreover, for any arbitrary point such as $P(x_0, y_0)$ in the image plane, the number of lines that can pass through it is unlimited and they satisfy equation 3.22. Plotting this equation for each point in the Hough space, (ρ, θ) plane, results in a unique sinusoidal curve.

$$\rho(\theta) = x_0 \cos\theta + y_0 \sin\theta \tag{3.22}$$

By superposing the sinusoidal curves of two different points of the image plane in the Hough space, the parameters of the line that passes through them is determined. This line is parameterized by a pair such as (ρ, θ).

Furthermore, if the Hough transform procedure is applied to a set of points of the image plane which are lying on a straight line parameterized with (ρ, θ), the plots for their sinusoidal curves in the Hough space will intersect at a single point with the coordinates of (ρ, θ). As the number of points in the set increases, the number of sinusoidal curves passing through that point will increase as well. Hence, the number of intersections is a useful indicator in the voting scheme. In other words, with this method, the problem of detecting collinear points can be converted to the problem of finding concurrent curves [18].

To sum up, the properties of the Hough transformation can be summarized as follows:

- Any point in the image plane can only be associated to one and only one sinusoidal curve in the Hough space.
- A point in the Hough space is a representation for a straight line in the image plane.
- Points which are lying on the same straight lines in the image plane are represented by the curves passing through a common point in the Hough space.
- Points in the Hough space which are lying on the same sinusoidal curve are corresponding to the lines passing through the same point in the image plane.

Implementation

The first step of the Hough transformation algorithm is to transform all points in the image plane to the corresponding curves in the parameter space. In general, these n curves will intersect at

[1] *http : //en.wikipedia.org/wiki/Hough_Transform*

$\frac{n.(n-1)}{2}$ points [18] which equals the number of lines passing through all pairs of points in the image plane.

Those subsets of points in the image plane which are exactly collinear are determined by finding the points that intersect coincidentally in parameter space. Unfortunately, this approach is very costly in terms of computation time which grows quadratically as the number of points (n) in the image plane increases. This problem can be addressed if some acceptable error in ρ and θ in the output result is tolerable. In such a case, by following the Hough's basic assumptions, the computation time can be reduced considerably.

Hough proposes the quantization of the $\rho - \theta$ plane with an acceptable error in ρ and θ into a quad ruled grid in which $\theta \in [0, \pi]$ and $-\frac{\sqrt{pq}}{2} \leq \rho \leq \frac{\sqrt{pq}}{2}$, or $\theta \in [0, 2\pi]$ and $0 \leq \rho \leq \frac{\sqrt{pq}}{2}$ where p and q are image plane width and height respectively. The quantized region (Hough space) is then treated as a matrix whose elements are so-called accumulators. The corresponding curve to each pixel (x_i, y_i) in the image plane is included in the matrix by incrementing the value of the elements that are along the curve. Therefore each element of the matrix contains the total number of curves that pass through it. Eventually after transforming all of the pixels in the image plane to the corresponding curves, the matrix can be inspected for elements that have high counts. If an element of the matrix, which plays the role of a cell in the Hough space, is denoted by $M_{(\theta_i, \rho_j)}$ and holds a value of k, then exactly k pixels of the image lie (within the quantization error) along the line whose normal is parameterized by (θ_i, ρ_j).

In Figure (3.18) the image of three black points on a white background is depicted on the left side. On the right side, three sinusoidal curves resulted from the Hough transform of the three pixels are shown. The point at which these three sinusoidal curves intersect gives (ρ, θ) of the normal to the line which passes through them. Note that the center of the coordinate system is located at the top-left side of the original image.

In Figure (3.19) the image of two crossing straight lines is shown on the left side. On the right side the result of the Hough transform which is stored in the Hough space matrix is depicted. The value of each cell (element) of the Hough space matrix represents the number of curves that have been passed through it. Higher cell values are rendered brighter. The two distinctly bright spots are the Hough parameters of the two lines. Note that the center of the coordinate system is located at the top-left side of the original image.

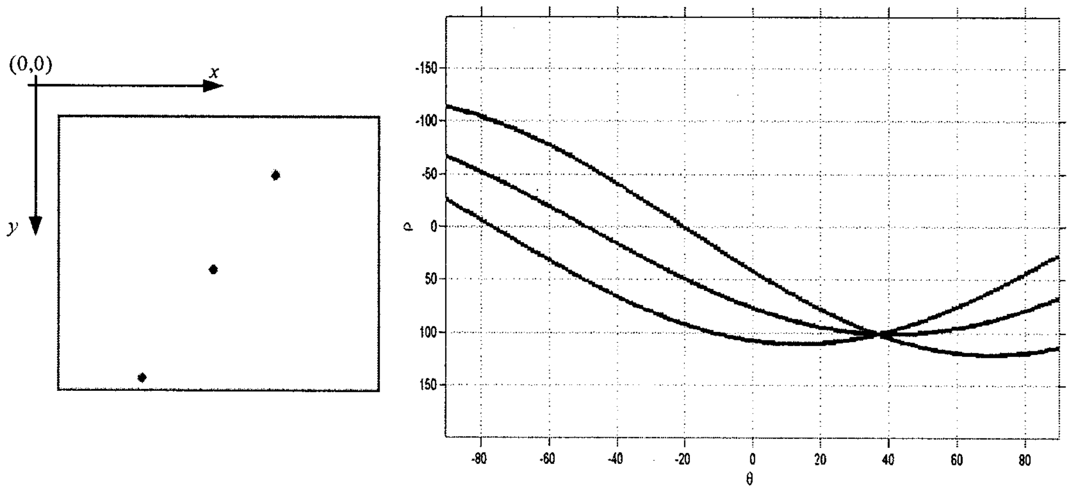


Figure 3.18: Three points lying on a straight line (left) and their corresponding Hough transform (right).

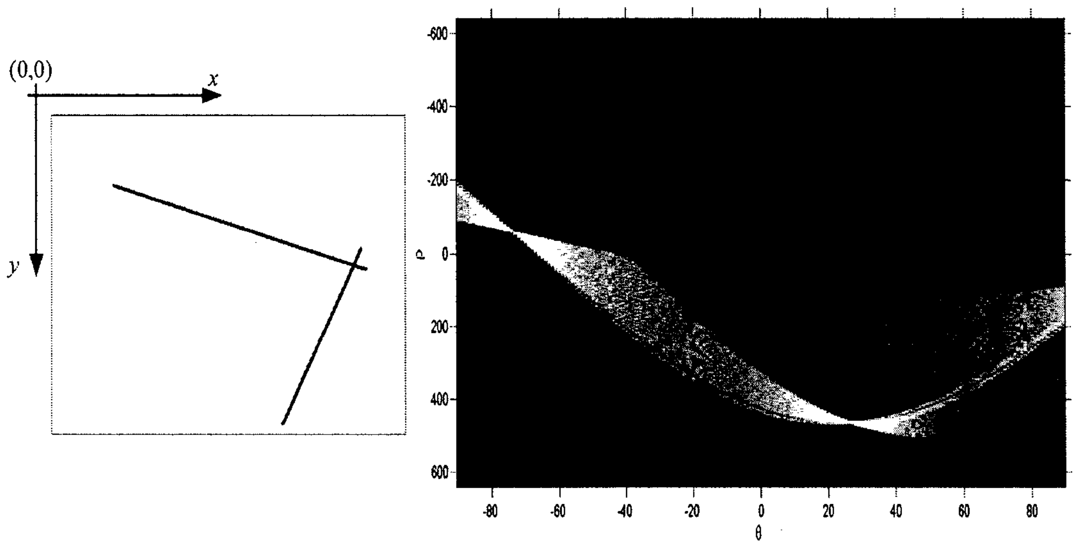


Figure 3.19: Two straight lines (left) and their corresponding Hough transform (right).

Chapter 4

Self Localization

In our application, to perform the self localization procedure, the robot is placed in a square which has two consecutive sides in red and green colors (Figure (4.1)). The location of the two lines with respect to the room is known. Therefore, if the robot can find its relative position and orientation with respect to these lines, its global position and orientation with respect to the environment will be determined. The flowchart for the self-localization procedure is shown in Figure (4.2) and each step will be described in the following chapters.

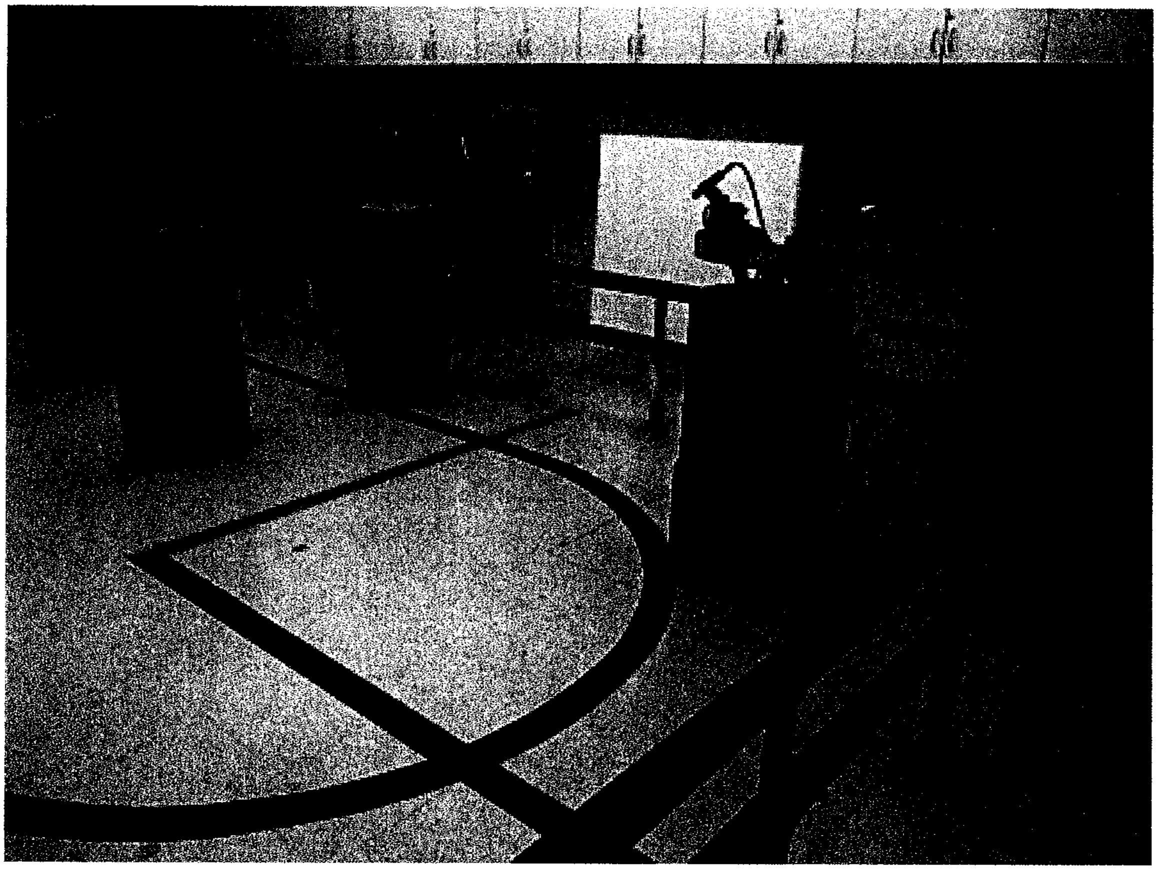


Figure 4.1: View of the Lab and the robot and the red and green tapes on the floor.

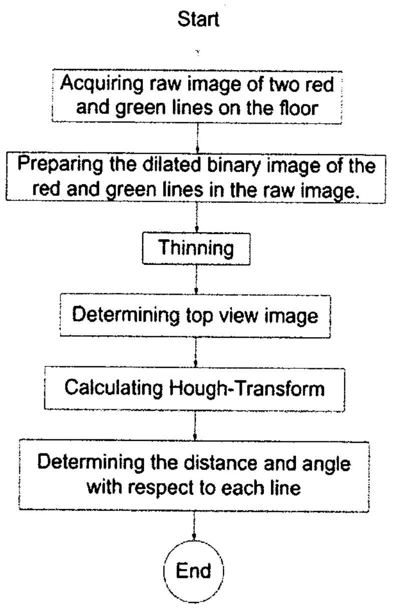

Start
Acquiring raw image of two red and green lines on the floor
Preparing the dilated binary image of the red and green lines in the raw image.
Thinning
Determining top view image
Calculating Hough-Transform
Determining the distance and angle with respect to each line
End

Figure 4.2: Self localization flowchart.

4.1 Acquiring Raw Image

4.1.1 Vision System

The first step in the self localization flowchart is acquiring a raw image. The vision system used in our application has a monocular color camera SONY XC-555. The resolution of the image plane is 160×120 pixels and each pixel holds three bytes describing the red, green, and blue (RGB) components. The camera is equipped with an AGC (Auto Gain Control) which makes it capable of capturing picture even under poor lighting condition. As well it is equipped with an ATW (auto tracing white balance) which allows the white balance to be adjusted according to the color temperature transition of the subject. The ATW is suitable for shooting with variable lighting. The shutter speed of the camera is also set to 1/60 sec. As depicted in Figure (4.3), the camera is adjusted to focus on the distance of 2 meters and its aperture is set to 6.

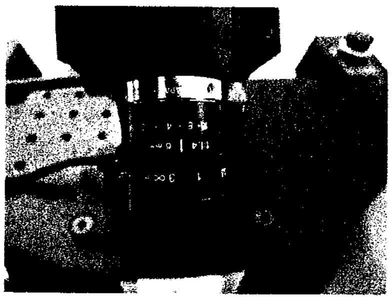


Figure 4.3: Close view of the camera.

The image grabber card, mounted on the robot's PC, captures color images and saves them with a PPM format. Each PPM file has a header which has four expressions length and a body which has three times the number of pixels in the image length.

If the first expression in the header of an image file is "P3", the image is recorded in a plain text format. In this case, the second and third expressions of the header determine the width and height of the image while the forth expression is the maximum value that each of the R, G, and B components can be assigned to (255 in our case). The body of the file contains the information about the RGB components of the pixels of the image, which is recorded row by row and column by column from the top-left to the bottom-right of the image. The body starts right after the header and the RGB components of each pixel are recorded as three consecutive numerical values.

If the PPM identifier is "P6", however, the image is stored in a binary format with the same order described above. In binary format, the size of the file is reduced considerably which has the advantage of reading from and writing to the file with a higher speed.

4.1.2 Raw Image

A typical raw image captured by the robot's camera is shown in Figure (4.4). Experimental measurements reveal that the pixels belonging to the red tape satisfy the following inequalities:

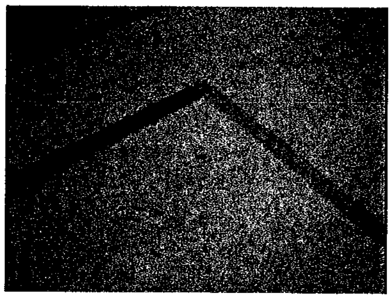


Figure 4.4: Raw color image.

$$\begin{cases} R > 1.8G \\ R > 1.7B \end{cases} \tag{4.1}$$

while for the green pixels:

$$\begin{cases} G > 1.3R \\ G > 1.3B \end{cases} \tag{4.2}$$

As well, sufficient lengths of both red and green tapes fall in the image plane if

$$\begin{cases} \texttt{Number of Red pixels} > R_{th} \\ \texttt{Number of Green pixels} > G_{th} \end{cases} \tag{4.3}$$

$$\frac{1}{RG_{th}} \leq \frac{\texttt{Number of Red pixels}}{\texttt{Number of Green pixels}} \leq RG_{th} \tag{4.4}$$

where $R_{th} = 200$, $G_{th} = 200$, and $RG_{th} = 1.3$ which are found experimentally.

In our work, to capture a proper raw image that contains both red and green colors, the robot is programmed to revolve around its center while capturing color images. After each capture, the number of red and green pixels in the image, which are defined by inequalities (4.1) and (4.2), are counted and if the conditions (4.3) and (4.4) are satisfied, the robot stops revolving immediately otherwise the robot will stop revolving after two rotations.

4.2 Binary Image

Once the robot stops revolving, the algorithm waits for a little while to let the image in the image plane become stable and then it captures another image. Next it scans the image for the red and green pixels and assigns them to black (1) while other pixels not satisfying the conditions for red and green pixels are assigned to white (0). The obtained image is in black and white (1's and 0's). The binary image resulting from Figure (4.4) is shown in Figure (4.5).

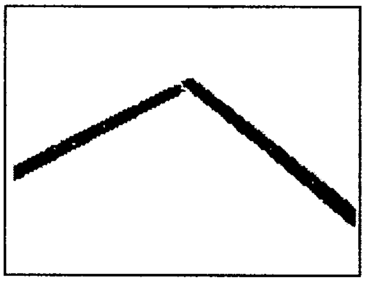


Figure 4.5: Binary image.

4.3 Dilation

To remove the noise effect in the obtained binary image, a dilation operator is applied. The structuring element, S, for dilation in our experiment is chosen as a 3×3 matrix that dilates the objects in the image by one pixel in all directions. The result of applying dilation operator on Figure (4.5) is shown in Figure (4.6).

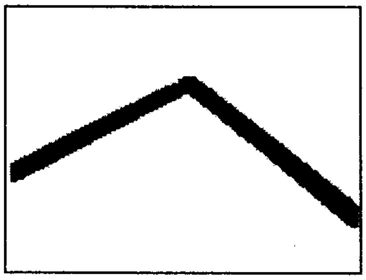


Figure 4.6: Dilated binary image.

4.4 Thinning

To reduce the computation time and to make the results more accurate, especially for the Hough transformation, a compact representation of the dilated image should be prepared. Xia's thinning method, explained previously, is used as the thinning algorithm in our application to produces the skeleton of the dilated binary image of the red and green lines. In Figure (4.7) the result of Xia's thinning algorithm on the dilated binary image of Figure (4.6) is depicted.

4.5 Top view image

The objective of this step is to find the real location of each pixel of the thinned image on the floor. The thinned image is scanned conventionally and upon detection of any black pixel, by applying equations 3.18 to 3.20, its actual location on the floor is calculated and the corresponding pixel of the top view image is marked as a black pixel. The pixels of the top view image are the

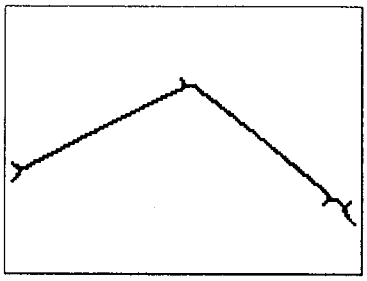


Figure 4.7: Skeleton of red and green lines in the image.

representations for the cells of a gridded floor. The more the number of pixels for a specified length on the floor (higher resolution), the more accurate the calculations are. In our experiment, each pixel of the top view image corresponds to a square with a dimension of 0.5×0.5 cm on the floor. The top view image of Figure (4.7) is shown in Figure (4.8).

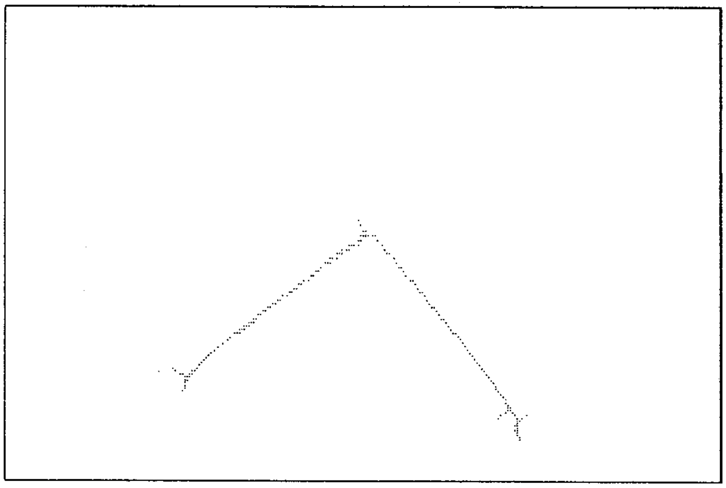


Figure 4.8: Top view image of the skeleton of red and green lines.

4.6 Hough Transform

To determine the location and orientation of the camera (and consequently the mobile robot) with respect to the green and red lines, the Hough transform of the top view image is calculated (Figure (4.9)). Note that the projection of the focal point of the camera on the floor is considered as the origin of the Cartesian coordinate system, $\theta \in [0, 2\pi]$, and $0 \leq \rho \leq \frac{\sqrt{pq}}{2}$ where p and q are top view image's width and height respectively. Since the red and green lines are orthogonal, their corresponding cells in the Hough space are located 90° apart. Knowing this fact, the Hough space is scanned to find the maximum value $H(\theta_{\max}, \rho_{\max})$. Then the regions bounded by:

$$\theta_{\max} - 90^\circ - \Delta \leq \theta \leq \theta_{\max} - 90^\circ + \Delta$$

and

$$\theta_{\max} + 90^\circ - \Delta \leq \theta \leq \theta_{\max} + 90^\circ + \Delta$$

are searched for the local maximum in each region ($\Delta = 5°$ in our case). The two obtained local maximum values are then compared and the largest, $H(\theta_{l-\max}, \rho_{l-\max})$, is selected. The parameters corresponding to each of the lines is determined by the following *if* statement:

```
if($\theta_{\max} - 90° - \Delta \leq \theta_{l-\max} \leq \theta_{\max} - 90° + \Delta$)
{
        $(\theta_G, \rho_G) \leftarrow (\theta_{l-\max}, \rho_{l-\max})$
        $(\theta_R, \rho_R) \leftarrow (\theta_{\max}, \rho_{\max})$
}
otherwise
{
        $(\theta_G, \rho_G) \leftarrow (\theta_{\max}, \rho_{\max})$
        $(\theta_R, \rho_R) \leftarrow (\theta_{l-\max}, \rho_{l-\max})$
}
```

where R and G indices stand for *Green* and *Red* colors respectively.

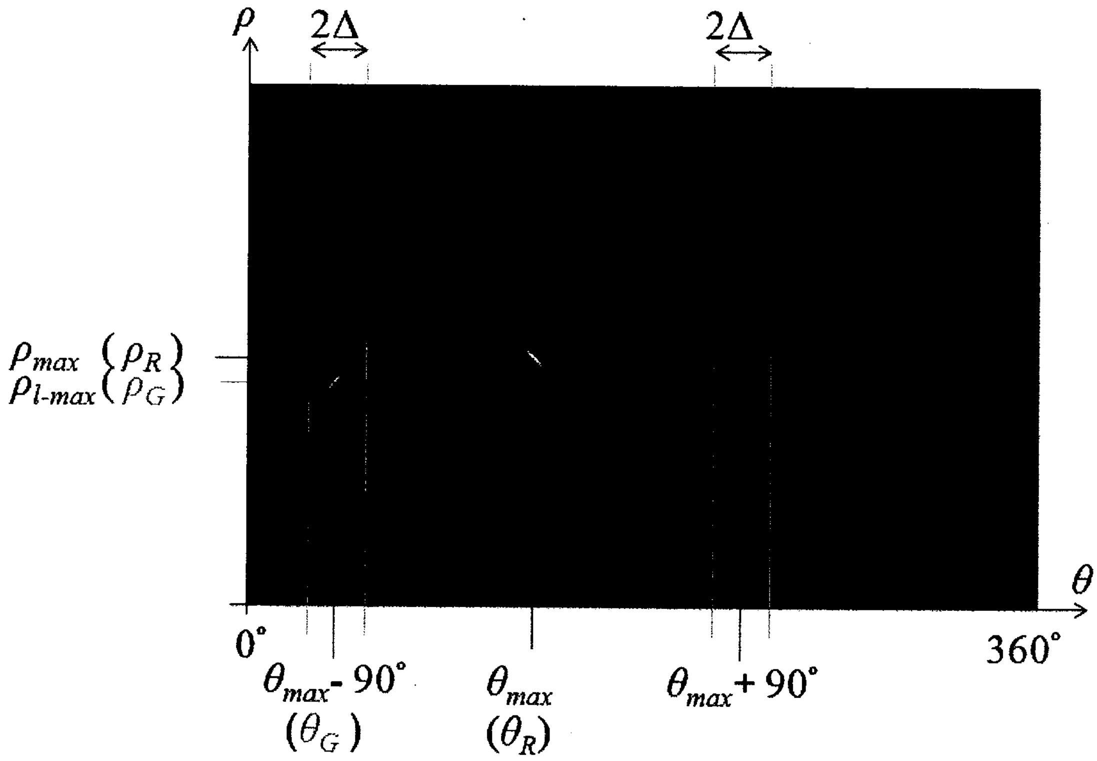
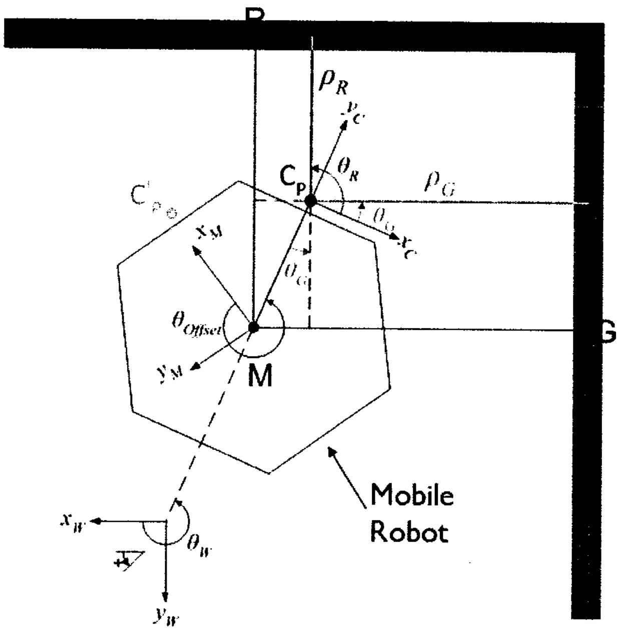


Figure 4.9: The Hough transform of the red and green tapes.

4.7 Determining the Robot's Initial Position and Direction

At the start up, the odometer of the robot is automatically set to zero regardless of the initial position and orientation of robot. The coordinate system of the robot is established at start up. Our algorithm forces the robot to revolve around its center while capturing images of the floor to detect both of the red and green lines as explained earlier. Upon detection, the value of the rotation angle that the odometer system has already measured is saved in a variable such as θ_{Offset} (Figure (4.10)). Thus, if the instantaneous orientation of the mobile robot in its coordinate system, measured by its odometer system, is denoted by $\theta_{Odometer}$, the instantaneous global direction of the robot in the world coordinate system, can be expressed by:

$$\theta_W = \theta_{Odometer} - \theta_{Offset} + (\frac{\pi}{2} - \theta_G) + \pi \tag{4.5}$$



Legend	
C'_P:	Projection of the camera's focal point on the floor at the start up.
C_P:	Projection of the camera's focal point on the floor upon detection of red and green tapes.
x_C-y_C:	Camera's coordinate system.
x_M-y_M:	Robot's coordinate system.
x_W-y_W:	World's coordinate system.
θ_W:	Robot's momentary direction with respect to the room.
θ_{Offset}:	Measured robot's direction after self-localization.
$\theta_{Odometer}$:	Measured robot's momentary direction with respect to its coordinate system.
MR:	Initial robot's distance from the middle of the red tape.
MG:	Initial robot's distance from the middle of the green tape.

Figure 4.10: Robot's initial position and orientation.

Note that the values of (θ_G, ρ_G) and (θ_R, ρ_R) are calculated with respect to C_P and the corresponding normals from C_P to the red and green lines. As well, the location of the robot's center M (the origin of the robot's coordinate system) with respect to the center lines of the red and green lines is determined by:

$$MR = \rho_R + MC_P \cos\theta_G \tag{4.6}$$

$$and$$

$$MG = \rho_G + MC_P \sin\theta_G \tag{4.7}$$

where $MC_P = 27cm$ and is determined by geometric measurements.

Consequently, knowing the location of the center lines of the tapes with respect to the room, the exact initial position of the robot with respect to the world coordinate system can be determined.

Finally, if the displacement of the robot along the x_M and y_M axis of its coordinate system, which is measured by the odometer system, is denoted by (x_m, y_m), its displacement along the x_W and y_W axis of the world's coordinate system, denoted with (x_w, y_w), can be evaluated by:

$$\begin{pmatrix} x_w \\ y_w \end{pmatrix} = \begin{pmatrix} \cos(\frac{3\pi}{2} - \theta_{Offset} - \theta_G) & -\sin(\frac{3\pi}{2} - \theta_{Offset} - \theta_G) \\ \sin(\frac{3\pi}{2} - \theta_{Offset} - \theta_G) & \cos(\frac{3\pi}{2} - \theta_{Offset} - \theta_G) \end{pmatrix} \begin{pmatrix} x_m \\ y_m \end{pmatrix} \tag{4.8}$$

Chapter 5

Navigation and Obstacle Avoidance

5.1 Equations of Motion and the Navigation Law

A typical two dimensional (2D) navigation problem is shown in Figure (5.1), where the instantaneous position and orientation of the mobile robot are denoted by $(x(t), y(t))$ and $\theta(t)$ respectively ($\theta(t)$ is measured with respect to the x axis). The initial position and orientation of the mobile robot are denoted with (x_0, y_0) and θ_0. The general form of the equations of motion for a mobile robot can be expressed as follows:

$$\begin{cases} \dot{x}(t) = v(t)\cos\theta(t) \\ \dot{y}(t) = v(t)\sin\theta(t) \\ \dot{\theta}(t) = \omega(t) \end{cases} \tag{5.1}$$

where $v(t)$ and $\omega(t)$ are the mobile robot's linear and angular velocities respectively.

In our work, the linear velocity of the robot is kept constant except around the goal which decreases proportional to its distance to the goal. The navigation law is defined as:

$$\omega(t) = -k_w[\theta(t) - \theta^*(t)] \tag{5.2}$$

where $\theta^*(t)$ is the steering angle and k_w is a positive constant.

5.2 Navigation without Obstacles

In the case where there are no obstacles on the path, as the robot goes forward with the constant speed, it will move toward the goal (Figure (5.2)). In this case, $\theta^*(t) = \theta_t^*(t)$ where

$$\theta_t^*(t) = \begin{cases} \phi(t) + \pi & \phi(t) \leq \theta(t) \\ \phi(t) - \pi & \phi(t) > \theta(t) \end{cases} \tag{5.3}$$

$$\phi(t) = \tan^{-1}\frac{y(t)}{x(t)} \qquad 0 \leq \phi(t) \leq 2\pi \tag{5.4}$$

5.3 Avoidance Behavior

Our algorithm for obstacle detection and avoidance behavior is founded based on the well-known method named the Vector Field Histogram (VFH) [10]. The VFH method enables the mobile

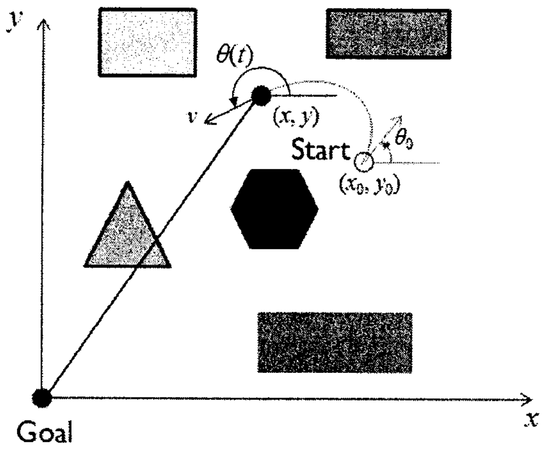


Figure 5.1: Navigation with obstacles.

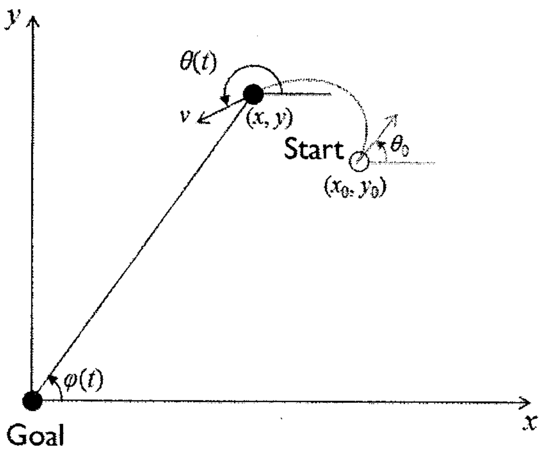


Figure 5.2: Navigation without obstacles.

robot to detect the unexpected random obstacles and detour them effectively as it moves with a continuous and smooth controlled speed toward its target. The VFH is an improved version of the Virtual Force Field method (VFF), which is an obstacle avoidance method proposed by the same research group at the University of Michigan. While the VFF method [8] provides superior real-time obstacle avoidance for fast mobile robots, some limitation concerning fast travel among densely cluttered obstacles were identified in the course of the research group experimental work [10]. Retaining all advantages of the VFF method, the VFH method was proposed to address these limitations.

5.3.1 The VFF Concept

In the VFF method, the environment surrounding the mobile robot is represented as a two dimensional certainty grid, so-called Cartesian Histogram Grid. Each cell (i, j) of the histogram grid contains a certainty value $c_{i,j}$ representing the probability of the existence of an obstacle at the corresponding location. In each range reading, if any of the sonar sensors detects an obstacle, only the cell located on the acoustic axis of the sonar sensor at the measured distance x is incremented (Figure (5.3)).

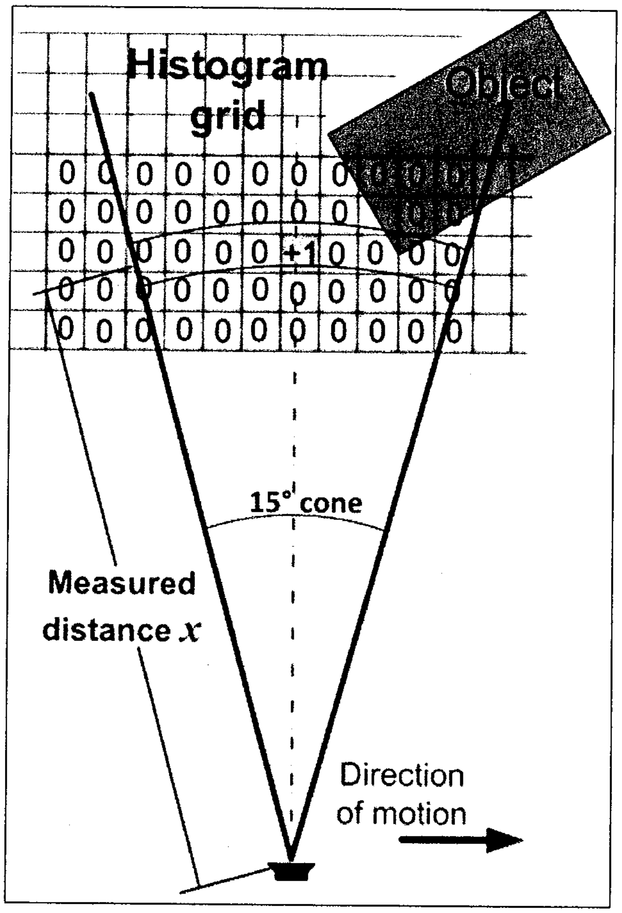


Figure 5.3: For each range reading, only one cell is incremented.

By continuous and rapid sampling, as the robot moves, a probabilistic distribution of the obstacles is obtained in which each cell together with its neighboring cells are repeatedly incremented (Figure (5.4)).

Therefore, the locations of the detected objects can approximately be determined by the cells of the histogram which have high certainty values. Moreover, by applying the potential filed notion to the obtained histogram grid at each instant, the robot can be effectively controlled. In the potential filed method, each occupied cell exerts a repulsive force on the robot, while the goal exerts an attractive force. The steering angle of the robot can be determined by vector summation of all repulsive forces and a target-directed attractive force [29]. Figure (5.5) shows how the VFF algorithm works.

At the startup, a window of $w_s \times w_s$ cells, called "active window", is constructed in such a way that its center overlies the center of the mobile robot. The region of the histogram grid C which is covered by the active window at any position of the robot is called "active region"and is denoted by C^*. As well, those cells of C belonging to C^* are called "active cells"and are denoted by $c^*_{i,j}$. The size of the window is 33×33 cells (with a cell size of 10 cm $\times$ 10 cm) [10]. Note that a circular window would return a better result, but to reduce the complexity of the computation, a square window is preferred.

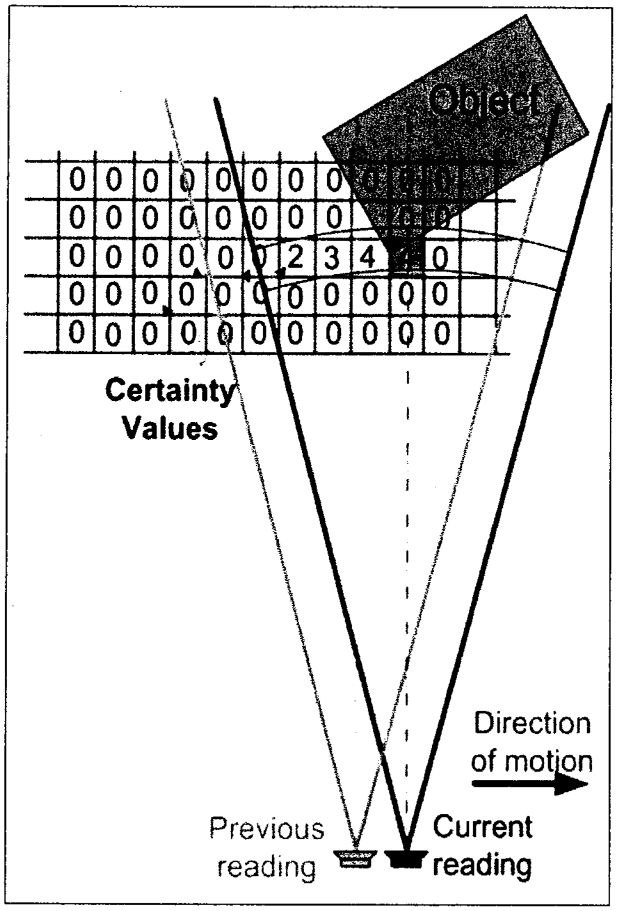


Figure 5.4: By continuous and fast sampling of the sensors, a histogram probability distribution is obtained.

In the VFF method, each active cell exerts a virtual repulsive force $F_{i,j}$ on the robot. The magnitude of $F_{i,j}$ is proportional to the certainty value $c^*_{i,j}$ and inversely to x^2, where x is the distance between the active cell and the center of the active window (center of the robot). At each iteration, the repulsive forces corresponding to all of the active cells are added to obtain a repulsive force F_r. Simultaneously, F_t pulls the mobile robot toward the target. The vector summation of F_r and F_t yields the resulting force vector R whose direction is used as the instantaneous steering angle of the mobile robot.

Shortcomings of the VFF Method

Experimental results reveal that under most conditions, the VFF algorithm controls the mobile robot very well; however VFF suffers some shortcomings which are as follows [10]:

1. To determine R, up to w_s^2 individual repulsive force vectors $F_{i,j}$ should be computed which makes it expensive in term of computation time for large active windows.
2. Whenever the center of the active window moves from one cell to a neighboring cell, because of the discrete nature of the histogram grid, a coarse change in the resultant R may happen which causes considerable fluctuations in the output steering angle.
3. While passing through narrow corridors along the center line, a very slight deviation to either

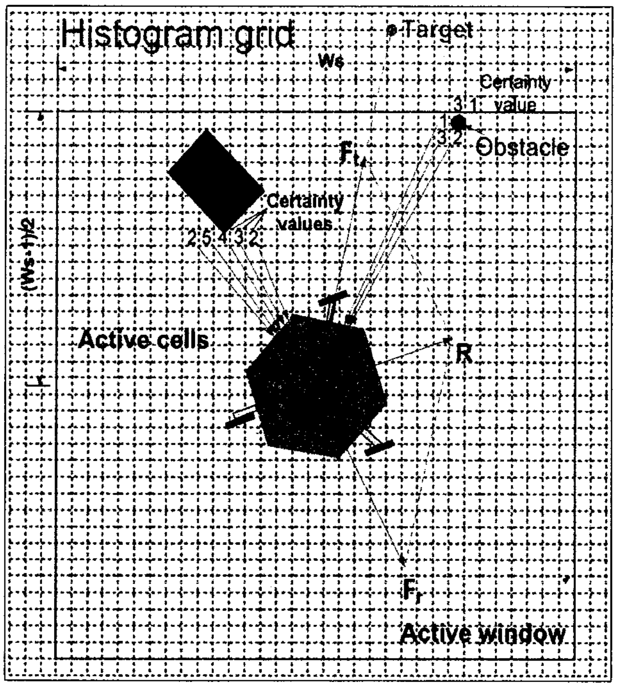


Figure 5.5: The mobile robot and its corresponding *active window* and *active cells.*

side causes the robot to be pushed away to the other side which results in an oscillatory and unstable motion.

4. Sometimes when the mobile robot tries to pass through a doorway, the vector summation of the virtual repulsive forces from both sides of the doorway is higher than the magnitude of the virtual attracting force which causes the mobile robot to be pushed away.

5.3.2 The Vector Filed Histogram Method

The VFH method which is proposed to address the VFF method's shortcomings has three levels of data representation as follows:

1. Highest level: Holds the detailed information of the environment surrounding the robot. The algorithm continuously updates the Cartesian histogram C with the real-time range data collected by the on-board range detectors.

2. Intermediate level: Holds the data of a one-dimensional *polar histogram* H which is constructed around the instantaneous position of the robot. A polar histogram consists of n angular sectors of width α. By mapping the active region C^* onto the polar histogram H, each sector k holds a value h_k which is a representation for the *polar obstacle density* in the direction of that sector.

3. Lowest level: Holds the outputs of the VFH algorithm which are used for the driving and steering controllers of the mobile robot.

Creation of the Polar Histogram

In the first level of the data representation, the algorithm maps the active cells (cells belonging to active region C^* of the histogram C) onto the polar histogram H. In the VFH algorithm, the value of each active cell (i,j) is considered as an obstacle vector toward the center of the mobile robot, as shown in Figure (5.6), which has a magnitude of:

$$m_{i,j} = (c^*_{i,j})^2(a - bd_{i,j}) \tag{5.5}$$

and the direction of $\beta_{i,j}$:

$$\beta_{i,j} = \tan \frac{y_j - y_m}{x_i - x_m} \tag{5.6}$$

where

a, b :	Positive constants,
$c^*_{i,j}$:	Certainty value of active cell (i,j),
$d_{i,j}$:	The distance between the active cell (i,j) and the center of the mobile robot,
x_m, y_m :	The coordinates of the center of mobile robot,
x_i, y_j :	The coordiante of the active cell (i,j).

The quadratic term, $c^*_{i,j}$, in the equation (5.5) reduces the noise effects introduced by the misreading of the range detectors since the occurrence of the actual range reading for the real obstacles is more often (higher certainty value) than that of the misreading.

In addition, since $m_{i,j}$ is proportional to $-d$, the closer active cells to the center of the mobile robot have larger magnitudes of $m_{i,j}$ compared to that of the far away cells. The parameters a and b are chosen such that $a - bd_{max} = 0$, where $d_{max} = \sqrt{2}(w_s - 1)/2$ is the distance between the farthest active cell and the center of mobile robot [10]. Hence, for the farthest active cell, $m_{i,j} = 0$ and it increases linearly as the active cell gets closer to the center of the mobile robot.

Furthermore, the polar histogram H is divided into n equal sectors of $\alpha = 360°/n$ (e.g., $n = 72$ and $\alpha = 5°$ as in our case). Each sector k corresponds to a discrete angle ρ quantized to multiples of α, such that $\rho = k\alpha$ where $k = 0, 1, 2, \ldots, n-1$ [10]. Thus, the corresponding sector k for each $c^*_{i,j}$, is determined by:

$$k = \text{floor}(\beta_{i,j}/\alpha) \tag{5.7}$$

and the polar obstacle density h_k corresponding to each sector k is calculated by:

$$h_k = \sum_{i,j} m_{i,j} \tag{5.8}$$

An example of mapping from C^* into H is depicted in Figure (5.6) in which all of the active cells corresponding to sector k are highlighted in red ($\alpha = 10°$ to clarify the picture).

Due to the discrete nature of the histogram grid, the calculated h_k may appear to be rugged which introduces some errors in the calculated steering angle. To reduce the effect of the errors, the polar obstacle density h_k is smoothed. The *smoothed polar obstacle density* (POD), h'_k, is computed as:

$$h'_k = \frac{h_{k-l} + 2h_{k-l+1} + \ldots + (l+1)h_k + \ldots + 2h_{k+l-1} + h_{k+l}}{2l+1} \tag{5.9}$$

where $l = 5$ yields a satisfactory smoothing result [10].

In Figure (5.7) a typical example of a mobile robot surrounded by three different obstacles and the corresponding smoothed polar histogram of the instantaneous robot's position in the polar form are depicted. The peaks A, B, and C in the polar histogram result from obstacle clusters A, B, and C in the histogram grid.

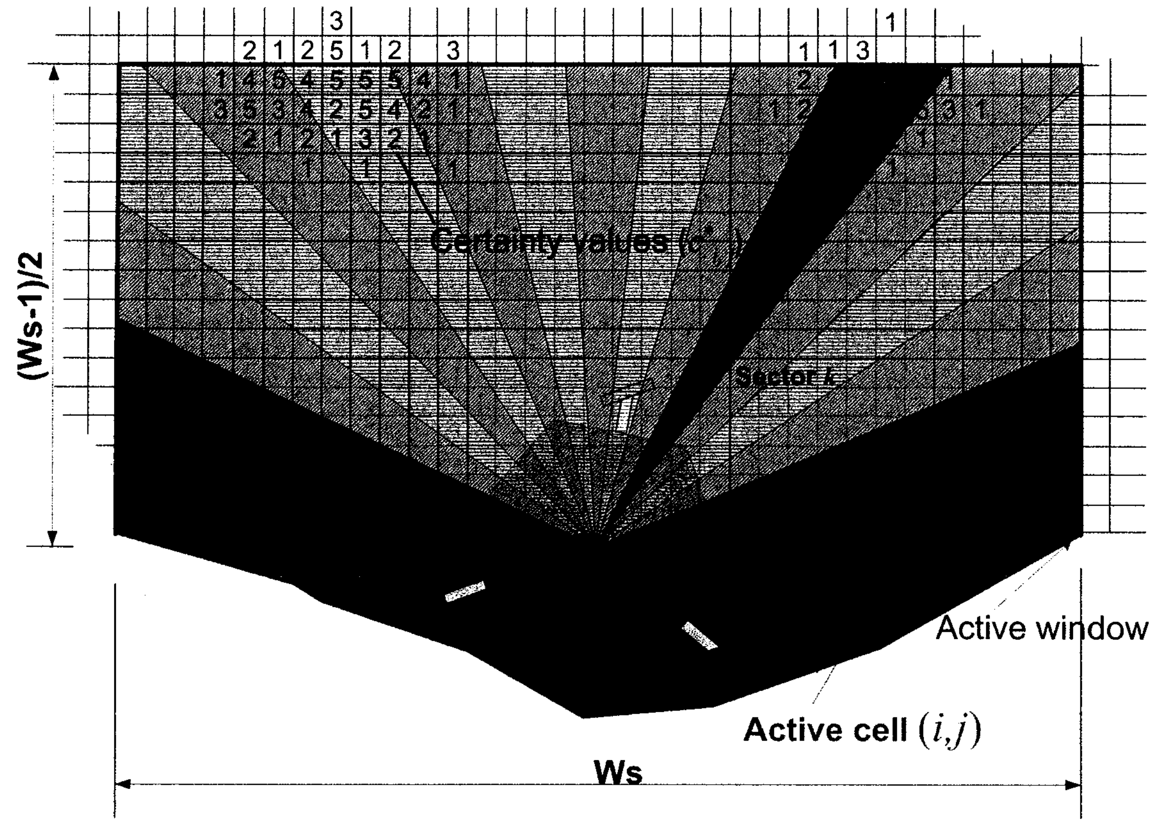


Figure 5.6: Mapping of the active cells onto the polar histogram.

Steering Control

The VFH algorithm selects the candidate valley which is closer to the direction of the target k_{targ}. The steering angle is then determined by selecting the suitable sector within the candidate valley as follows:

First the algorithm measures the width of the selected candidate valley. The result can either be a wide or a narrow valley.

A wide valley has at least S_{max} consecutive sectors below the threshold value (in our case $S_{max} = 18$). In reality, a wide valley is a representation of a wide gap between two obstacles as depicted in Figure (5.8). The sector of the selected candidate valley which is the closest one to k_{targ} represents the near border of the valley to the obstacle and is denoted by k_n. The far border k_f is defined as:

$$k_f = k_n + S_{max} \tag{5.10}$$

and the desired steering direction θ^* is defined as:

$$\theta_{Steer} = \theta^* = \frac{k_f + k_n}{2} \tag{5.11}$$

If the mobile robot is too close to an obstacle on the path (Figure (5.8-a)), θ^* is selected in such a way that takes the robot away from the obstacle. On the other hand, if the robot is too far from an obstacle (Figure (5.8-b)), θ^* will be chosen so that the robot can move closer to the obstacle as it moves towards the target. In the steady-state, when the distance between the robot

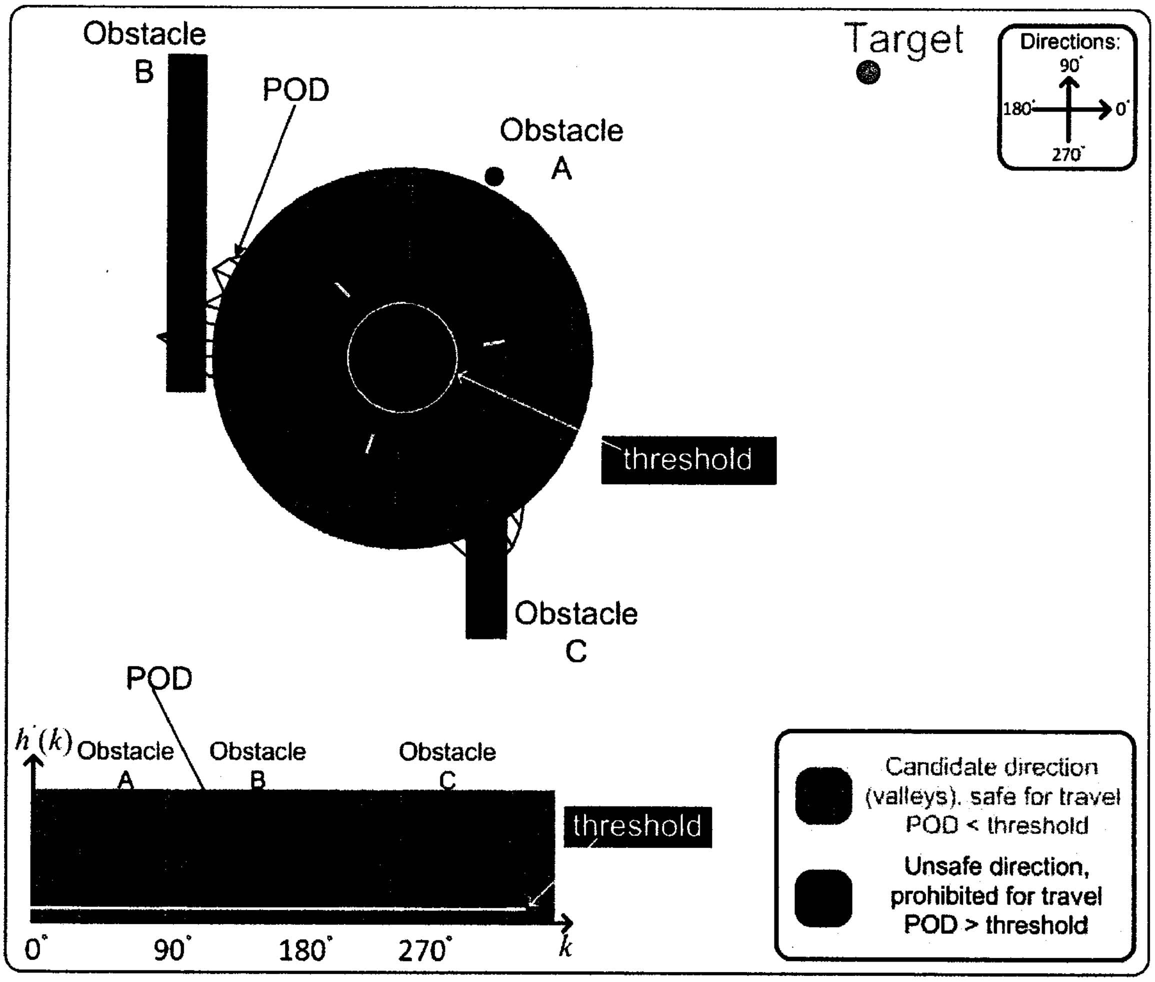


Figure 5.7: The mobile robot surrounded by three obstacles and the corresponding POD established around the location of the robot.

and the obstacle is d_s (Figure (5.8-c)), the robot's direction θ^* is selected parallel to the obstacle and the small disturbances are rejected thanks to the filtering property of equation (5.9).

It is important to note that the distance d_s is mainly determined by S_{max}. If a large distance between the obstacles and the robot in the steady-state is desired, S_{max} should be assigned to a larger value, while a smaller value results in a closer distance to the obstacles in the steady-state.

In the case where obstacles are closely spaced (Figure (5.9)), k_f is less than S_{max} sectors apart from k_n and the steering angle θ^* is determined by equation (5.11). In this case the robot travels on the path which is centered between the obstacles.

Since the VFH method is capable of detecting the narrow valleys and choosing the centered path between the obstacles, it can easily pass the robot through the doorways. The lack of this ability is a shortcoming of the VFF algorithm. Due to the additional averaging effect of the polar histogram and the additional smoothing by (5.9), k_n and k_f (and consequently θ^*) vary very slowly between sampling intervals [10]. Therefore, no low-pass filter in the control loop of the steering is needed and consequently the robot is able to react much faster to unexpected obstacles. In addition, the robot can easily travel through narrow hallways without any oscillation.

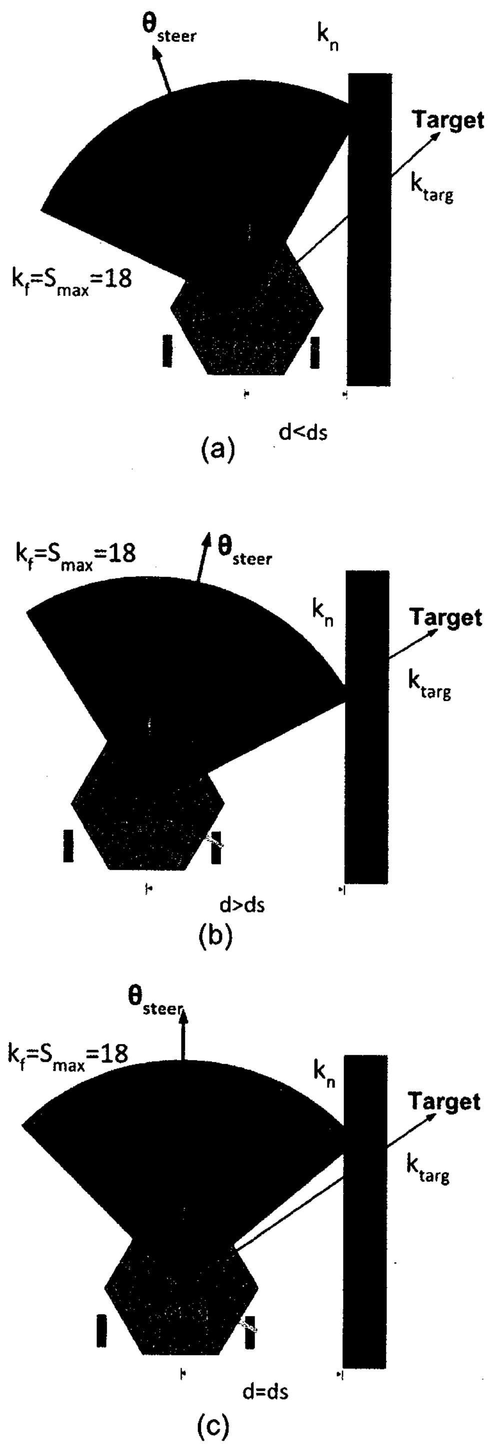


Figure 5.8: Obtaining a safe path while the robot travels alongside an obstacle: (a) The robot is too close to the obstacle and θ_{steer} points away to increase the distance. (b) The robot is far away and θ_{steer} points toward the obstacle to decrease the distance. (c) The robot is at a proper distance d_s with respect to the obstacle and moves alongside it.

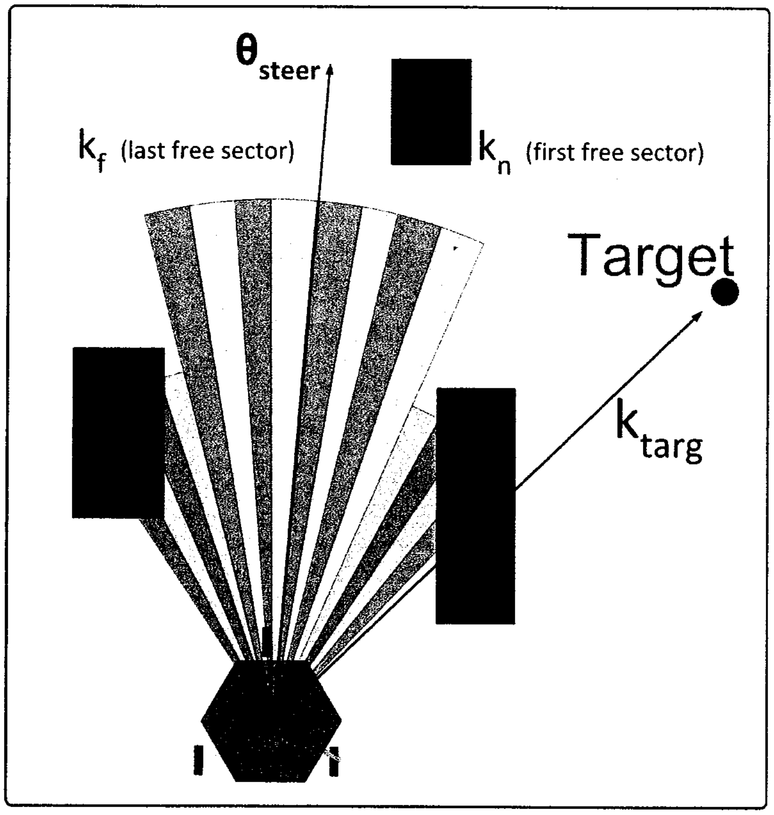


Figure 5.9: The steering angle θ_{steer} when an obstacle obstructs k_{targ}.

Threshold

In the VFH algorithm, the candidate valleys are determined by the threshold value. Unlike many other sensor-based systems, the VFH algorithm is very robust with respect to changes in the threshold value. As depicted in Figure (5.7), decreasing or increasing the threshold even by a factor of 3 or 4 only affects the width of candidate valleys and it has only a small effect on narrow valleys, since the steering direction is chosen in the center of the valley [10]. Moreover, changes in the threshold for wide valleys, causes the steady-state distance d_s to be affected. Severe maladjustments of the threshold, however, will affect the system's behavior as follows: If the threshold value selected is too large, some of the obstacles may be ignored and a collision occurs. On the other hand, if the threshold is chosen too low, some of the potential candidate valleys maybe neglected and the robot will not pass through the space between them.

5.3.3 Combining the Sonar Sensors and Vision System

The VFH method is mainly developed to utilize the sonar sensors as the range detector. However, our experiments reveal that when the surfaces of the obstacles are very smooth and the ultrasonic waves emitted from the sonar sensors make steep angles with the surface of the obstacle, the echoed ultrasonic waves may not be detected by the sonar sensors. To address this problem, our research group suggested to use a vision system as a secondary range detector. In our method a combination of two distinct VFH controllers which work in parallel are used as the controller. The input to the first VFH controller is the data collected by a belt of 24 ultrasonic sensors around the mobile robot (as explained before), while the second controller is fed with the data collected by the vision system. To detect the obstacles in the image plane of the camera, the bottom edges of the objects are painted with a blue color (Figure (5.10) and Figure (5.11)) and the algorithm searches for the blue pixels in the image. Note that no blue pixel should appear on the body of the obstacles, otherwise it will be counted as a pixel belonging to a virtual obstacle which is located a little farther from the original one. Measurements reveal that the pixels belonging to the blue tapes have the following properties:

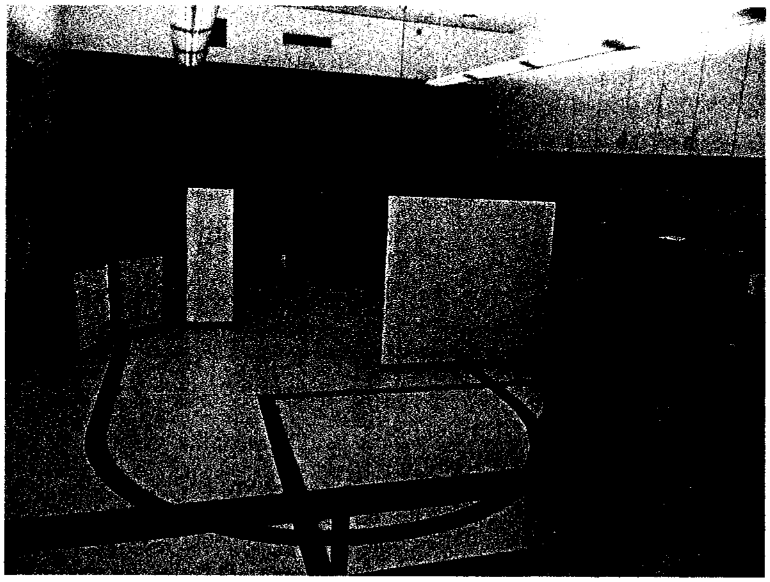


Figure 5.10: The bottom edge of the obstacles are painted in blue color.

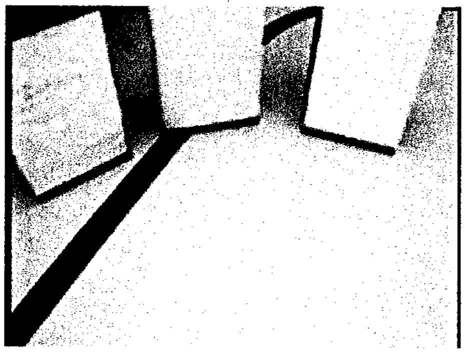


Figure 5.11: Obstacles seen in the image plane of the camera (raw image).

$$\begin{cases} B > 80 \\ B > 1.28R \\ B > 1.10G \end{cases} \tag{5.12}$$

As the robot moves, the camera captures images and each image is searched row by row and column by column and the detected blue pixels, if any, are assigned to a black pixel while the rest are assigned to white pixels (Figure (5.12)). Thereafter, as explained in the previous chapter and by applying equations (3.18) to (3.20), the top view image of each binary image is determined and superposed on the active window of the second VFH subroutine (Figure (5.13)). If the number of the pixels overlying a cell is greater than a certain amount (5 pixels in our application), the value of the corresponding cell is incremented. Moreover, the stray noisy blue pixels that may appear on the floor are also discarded by both the threshold value for the number of blue pixels in a cell and the property of the quadratic term in equation (5.5). After updating the cells' values, each of the active windows is processed with the procedure explained before and the output steering angle corresponding to each of them in each iteration is calculated. The absolute value of the difference between each of these steering angles and θ_t^* is calculated and the VFH controller corresponding to the larger result takes over and its steering angle is used as the θ^* in equation (5.2).

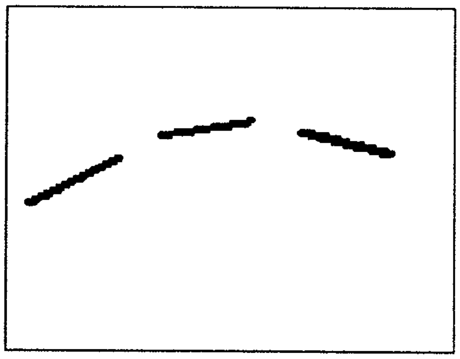


Figure 5.12: The binary image of Figure (5.11).

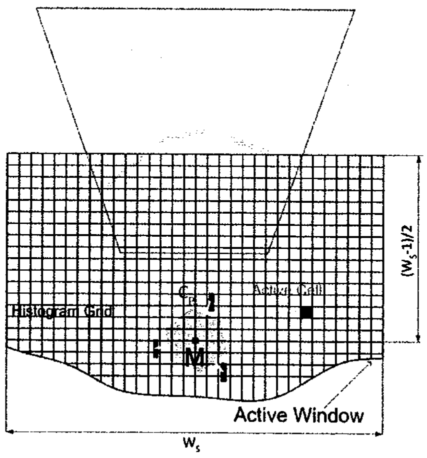


Figure 5.13: The superimposed top view image of Figure (5.12) on the active window.

5.4 Experimental Setup

The mobile robot which is used as our platform is a B21r mobile robot manufactured by iRobot Corporation. It consists of a mobile base, one on-board computer, and a variety of different sensors such as two belts of sonar sensors, tactile sensors, odometer system, and a monocular color vision system.

The B21r mobile robot has a four-wheel synchronous drive system and all of the wheels are mechanically coupled by a heavy-duty timing belt. Three identical DC motors burden the responsibility of generating enough pushing force for driving the mobile robot. Moreover, to have an accurate steering, all of the rubber wheels are mechanically coupled by a second heavy-duty timing belt which is driven by the forth DC motor.

The internal LAN (Ethernet) of the B21r robot facilitates the interfacing of different data-acquisition cards, drive systems and other peripheral devices to the main controlling unit rFLEX. The main controlling unit, rFLEX, is responsible for the traffic of information of all devices on the internal LAN to/from the on-board computer. The on-board computer is connected to the rFLEX via a RS-232 serial link.

In addition, the on-board Pentium III computer system provides a sufficient computation power for sophisticated and demanding projects. The B21r is provided with the Real World Interface's MOBILITY Robot Software Development Package, featuring client/server architecture and *C++* libraries with source code.

5.4.1 External Components of the B21r Mobile Robot

The B21r hardware system consists of three main sections (Figure (5.14)):

- **The Base**, which is mainly used for low level functions such as dead reckoning position integration, motor current sensing and battery voltage sensing. It contains four heavy-duty car batteries, belts, rubber wheels, gears, the mechanical drive and steering components, four motors (three for translation, one for steering) as well as motion-controller boards, a belt of 24 sonar sensors, and tactile sensors (bump detectors).
- **The Enclosure**, which contains the computer, communication equipment, a belt of 24 sonar sensors, a belt of 24 Infrared sensors, the interface console, switches, processing and interface equipment. The enclosure and wheels are mechanically coupled. Therefore, the reference point of the enclosure with respect to the base is always in the direction of the linear motion.
- **The Console**, located on top of the B21r enclosure, houses the selector knob of the rFLEX, emergency kill switches, joystick port, and serial ports. It also provides a physical-mounting platform for a pan-tilt device, cameras, etc.

5.4.2 Internal Components of the B21r Mobile Robot

The major internal components of the B21r mobile robot can be summarized as follows:

- **The Drive System** which consists of a four-wheel synchronous steering and a four-wheel drive system. All of the four wheels are always parallel. To achieve a high positional accuracy the drive system and steering system work independently. To provide enough pushing force, three drive motor run the same timing belt. Steering is handled with only one motor.
- **Electrical System** consists of four removable 12-Volt batteries which supply the motors, the computer, sensors, communication network, camera, and other equipment in the enclosure. Since the enclosure rotates, power and communication between the enclosure and the base travels through a set of rotary contacts [4].
- **The Computer**, which is a self-contained unit, is mounted inside the enclosure. The computer communicates with the rFLEX control system via a RS-232 serial port. The computer is also equipped with ten RS-232 serial ports, one parallel port, and Ethernet as standard equipment. The sonar, tactile, and infrared sensors on the base and enclosure are controlled by the STI boards mounted on each door of the base and enclosure (Figure (5.15)). The operating system of the computer is a Red Hat Linux 6.2 with iRobot's Mobility Robot Integration Software.

Each STI board is responsible for collecting information from the Infrared, sonar, and tactile sensors. Each STI board is also connected to the internal LAN to send its data to the rFLEX

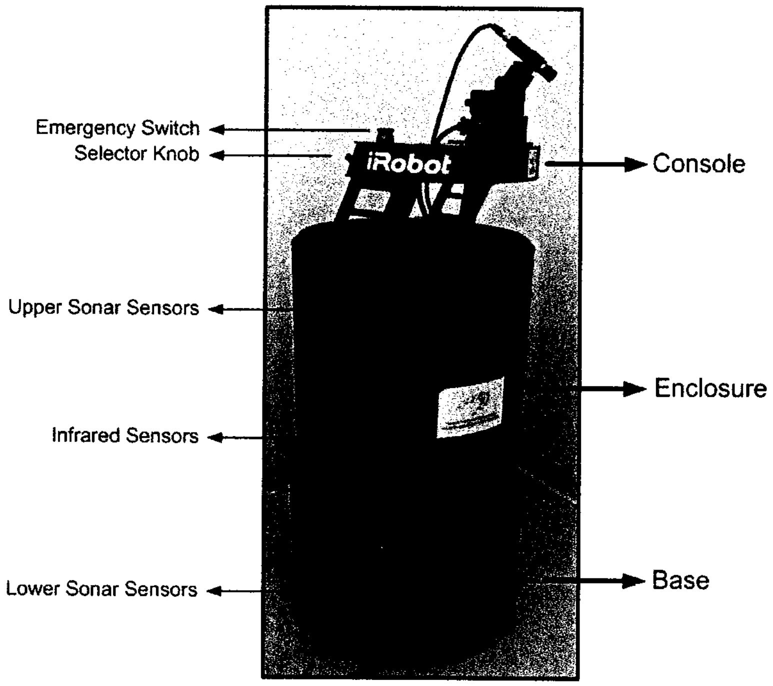


Figure 5.14: Different sections of the B21r mobile robot [63].

control system. The rFLEX control system collects the information sent from each STI board, puts them in a single packet and sends it to the computer via a RS-232 serial port. It also receives controlling information for the drive system from the computer via the same RS-232 serial port.

5.4.3 Sensors

Sonar Sensing

The B21r mobile robot is equipped with two belts of sonar sensors (each composed of 24 sonar sensors). The sonar sensors are mounted on the panels of the enclosure and base. Each sonar sensor is connected to the ISP card corresponding to' that panel (Figure (5.15)). One of the responsibilities of the ISP cards is to run the sonar sensors and collect their information.

Once working with the sonar sensors, the following facts should be taken into consideration [41]:

- While sonar sensor can detect the objects within the range of $20cm$ to $4m$, it has no way of knowing exactly where, in its $15°$ and wider cone of attention, an obstacle actually is.
- The sonar sensor has no way of knowing the relative angle of an obstacle. Obstacles at steep angles might bounce their echoes off in a completely different direction, leaving the sonar sensor ignorant of their existence, as it never receives an echo.
- The sonar sensor can be fooled if its ping bounces off an obliquely-angled object onto another object in the environment, which then, in turn, returns an echo to the sonar sensor. This effect, called specular reflection, can cause errors;. the sonar sensors overestimate the distance between the robot and the nearest obstacle.
- Extremely smooth walls presented at steep angles, and glass walls, can seriously mislead the sonar sensors.

The direction of each of the sonar sensors of the upper belt (mounted on the enclosure) with respect to the world's coordinate system varies as the robot steers, while the direction of the sonar sensors corresponding to the lower belt (mounted on the base) is always fixed.

Accessing Sonar Sensors in Software

The *ActiveSystemComponent*, which is called "Hardware I/O", is a subclass of *ActiveSystem-Component* that is designed to talk to a particular set of robot hardware, in this case, the sonar sensors. The "Hardware I/O"component is configured to connect to several *StateObjects* that provide raw sonar data going to/from the robot hardware. When the "Hardware I/O"component updates the raw sonar data, contained within a class of *StateObject* called *FVectorState*, the owner of the "raw"data is notified by a *StateChangeHandler* interface. The *Sonar* object, which is a subclass of the *CompositeStateObject*, processes this notification and uses its database of sonar geometry to update its abstract representations of the robot state. Since the robot-specific geometry is handled within the *Sonar* component, the abstract representations are more useful than raw information and user does not need to depend as much on specific robot hardware.

The following code is an implementation which allows reading the value of each sonar sensor of the enclosure in *C++* (the base sonar sensors can be accessed in a similar way):

```
//This is a buffer for object names.
char pathName[255];

//Hold -robot command line option.
char *robotName;

//Look for robot name option so we know which one to run.
robotName = mbyUtility::get_option(argc, argv, "-robot");
if (robotName == NULL)
{
  fprintf(stderr, "Need a robot name to use. \n");
  return -1;
}

//All Mobility servers and clients use CORBA and this initialization
//is required for the C++ language mapping of CORBA.
pHelper = new mbyClientHelper(argc, argv);

//Build a pathname to the component we want to use to get sonar data.
sprintf(pathname, "%s/Sonar/Segment", robotName);

//Use robot name arg.
//Locate the component we want.
ptempObj = pHelper ->find_object(pathName);

//Find the sonar data (we're going to use sensor feedback).
//The XX_var variable is a smart pointer for memory management.
MobilityGeometry::SegmentState_var    pSonarSeg;
MobilityGeometry::SegmentData_var    pSegData;

//Request the interface we want from the object we found
try
```

```
{
  pSonarSeg = MobilityGeometry::SegmentState::_narrow(ptempObj);
}
catch(...)
{
  return -1;        //We are through if we cannot use sensors.
}

//To sample the sonar segmented data
pSegData = pSonarSeg ->get_sample(0);
```

The following example uses the sonar data to find the objects whose distances to the robot are less than a predefined value *MinDistance*:

```
for(int index = 0; index < pSegData->org.length(); index ++)
{
  //Compute segment lengths.
  tempDist = sqrt( (pSegData->org[index].x - pSegData->end[index].x)*
                   (pSegData->org[index].x - pSegData->end[index].x)+
                   (pSegData->org[index].y - pSegData->end[index].y)*
                   (pSegData->org[index].y - pSegData->end[index].y) );

  if(tempdist < MinDistance)
    printf("An objected detected at a distance of %f m \n", tempDist);
}
```

Infrared Sensing

The B21r mobile robot is equipped with infrared sensors. These sensors can be used as an indicator of the proximity of an obstacle by emitting light and measuring the intensity of the reflection from the obstacles surface. Infrared light, the part of the electromagnetic spectrum of radiation above 0.75 millimeters in wavelength, is not visible to humans. Infrared sensors provide information that can be useful for building up a picture of the environment. The infrared sensors can detect obstacles located in the range of $10cm$ to $1m$. The infrared sensors can be accessed in a similar way as that of the sonar sensors and the reader is referred to [41]. These types of sensors are not used in our application.

Robotic Tactile Sensing

Tactile sensors, also known as contact sensors, are used to stop the mobile robot when it hits an object (if none of the other protections works). Each panel of the B21r mobile robot is equipped with four tactile sensors located at its four corners (Figure (5.15)). When the mobile robot collides with a solid obstacle, the corresponding panel is pushed inside which causes the tactile sensor function. The users program is responsible for checking the status of the tactile sensors to determine whether a collision happened. Similar to sonar sensors, in software level, the robot's tactile sensors are represented by a *CompositeSystemComponent* that contains several *StateObjects* that represent the states of each of the robot's tactile sensors. The tactile sensor object contains a *BVectorState* object that represents the raw tactile sensor readings. This information is provided primarily for debugging and testing use. The tactile sensor object also provides a *PointState* view of the contact points of the robot. The tactile sensors can be accessed in a similar way as that of the sonar sensors and the reader is referred to the source code. These types of sensors are not used in our application.

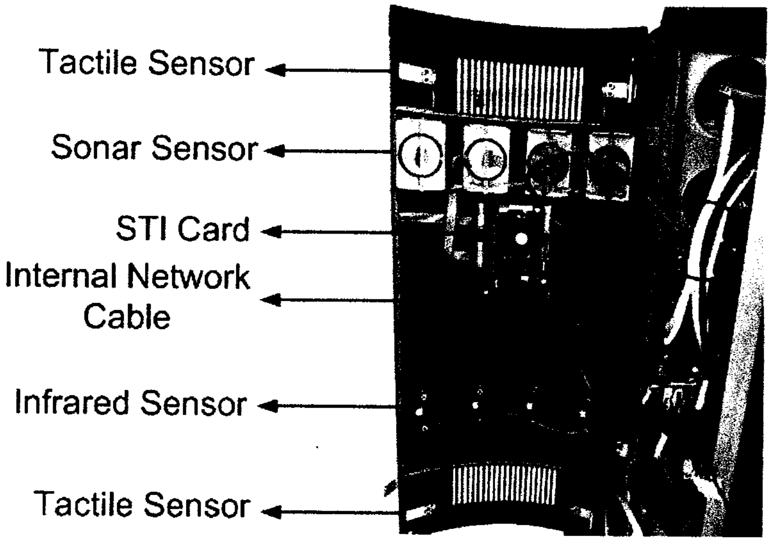


Figure 5.15: A STI card, sonar sensors, infrared sensors, and tactile sensors [63].

5.4.4 Odometer System

The base of the B21r mobile robot is equipped with rotary encoders to keep track of translation and rotation motions. It is also equipped with another rotary encoder which measures the relative angle of rotation of the enclosure with respect to the base which is called *SkirtPose.*

The rFLEX, which is responsible for integrating all of the measurements, keeps track of the incremental changes in the rotary encoders and provides information in a raw format. The raw data are further processed by the *Mobility* in higher software levels. The robot odometry object is a *CompositeSystemComponent.* It contains several *StateObjects* representing the state of robot's odometer. A *TransformState* object represents the current location of the robot. An *FVectorState* object represents the robots current velocity. Mobility provides the robot position in X,Y, and theta relative to the startup location. The units for these measured parameters at the software level are in meters and radians and the velocity outputs are in meters/second and radians/second. These parameters are updated with a rate of 10-15Hz and each new sample is provided with an updated time stamp. The odometer data can be accessed in a similar way as that of the sonar sensors and the reader is referred to the source code.

5.4.5 Actuators

The actuators of the robot are controlled by the rFLEX. The rFlex receives commands such as linear and angular velocities and accelerations from the on-board PC and converts them to appropriate digital values to be sent as set points to the actuators.

The robot drive object is also a *CompositeSystemComponent* containing several *StateObjects* that represent the state of the robot drive system and the current drive command. An *ActuatorState* object represents the state of the robot current drive system. Another *ActuatorState* object represents the state of the current drive command.

The following source code shows how the actuators can be accessed in *C++* programming language:

```
//Build a pathname to the component we want to use to drive the robot.
sprintf(pathname, "%s/Drive/Command", robotName);
```

```
//Use robot name arg.
//Locate object within robot.
ptempObj = pHelper ->find_object(pathName);

//Find the drive command (we're going to drive the robot around).
//The XX_var variable is a smart pointer for memory management.
MobilityActuator::ActuatorState_var    pDriveCommand;
MobilityActuator::ActuatorData_var     OurCommand;

//Request the interface we need from the object we found
try
{
  pDriveCommand = MobilityActuator::ActuatorState::_duplicate(
                        MobilityActuator::ActuatorState::_narrow(ptempObj);
}
catch(...)
{
  return -1;        //We are through if we cannot use drive command.
}
```

The following code sets the linear and angular velocities of the robot to $0.5m/s$ and $-0.8rad/sec$ respectively.

```
OurCommand.velocity[0] = 0.5;
OurCommand.velocity[1] = -0.8;
pDriveCommand->new_sample(OurCommand,0);
```

5.4.6 Communication with B21r Mobile Robot

The B21r mobile robot works as a web server and has two options to communicate with outside world: TCP/IP Ethernet communication or RS-232 serial communication. Once connected to the B21r through TCP/IP, a TELNET link between the B21r and a remote computer can be established and the user can gain control of the robot through the remote computer. To take advantage of the B21r mobile robot's capabilities, the user can use the libraries utilizing the MOM (Mobility Object Manager) interface to move the robot (as described above). Since these libraries are located on the hard disk of the B21r's on-board computer, the user source code, written in *C/C++*, should be uploaded to that hard disk and compiled there. If the user's code is saved in a file named "MyProg.c", an script file called "Makefile" which contains the following script code is used to compile the user's code:

```
#------------------------------------------------
#
# Makefile for Mobility example: MyProg
#
#------------------------------------------------
# Do you want every command printed, or just errors? (Q=Quiet)
ifndef Q
Q = @
#Q =
endif
# Sources to compile
```

```
CPP_SRCS =
C_SRCS =
# Libraries used by these test programs
LDLIBS += -lmby -lidlmby
# define the name of the executables (up to 3)
PROG = MyProg
# include the master Mobility library makefile
include $(MOBILITY_ROOT)/etc/scripts/cppexec.mf
```

To compile the user's code, the command "make" should be entered in the command line.

After a successful compile, the user's program "MyProg" can be executed remotely by entering "MyProg" in the command line of the remote session which is opened through the TELNET link. Before running the user's codes, the servers which are responsible for driving the robot and reading the sensors should be activated. An application called "base" is responsible for this task and can be executed remotely. Similarly the servers for the pan-tilt system and the vision system which are called "dppserver" and "v4lserver" respectively should be activated before running the user applications.

The user's program should stay in contact with the remote computer during the run. In the case that the communication link is disconnected, the B21r mobile robot will continue its motion with the latest executed command from the user's program and halts after 3 seconds if the communication link is not reestablished (this period can be changed if desired).

5.5 Experimental Results

In our experiment, the linear speed of the robot in equation (5.1) is kept constant ($v(t) = 10$ cm/sec); however, within a circle centered at the goal with a radius of 50 cm, the linear speed is proportional to the distance between the goal and the center of the mobile robot and if the distance becomes less than 5 cm, the robot will stop. The parameter k_ω in equation (5.2) is kept constant equal to 1.6 sec^{-1}. Decreasing k_w by a significant amount causes the steering angle to settle down late and in some cases collision may happen, while increasing it causes sharp curves in the trajectory which may not be desirable for some applications. In our experiments, we found that acceptable results are achieved if $k_w/v(t) = 0.16$.

Figure (5.16) shows the top view of our LAB. The mobile robot is placed in a random location within the area surrounded by the green and red tapes. There are some obstacles in the way of the robot as it progresses towards its goal and no information about any of them is given to the robot in advance. In the first experiment, only the VFH controller corresponding to the sonar sensors is activated and its threshold value is assigned to 70,000.

In Figure (5.17) some of the navigation parameters such as the position of the robot in x and y directions, the orientation of the robot $\theta(t)$, the angular velocity of the robot $w(t)$, the difference between the output steering angle of each of the VFH controllers and $\theta_t^*(t)$, denoted by "Vision θ_{Steer}"and "Sonar θ_{Steer}", as well as the corresponding VFH controller that takes over the instantaneous steering control are plotted.

The initial values of the parameters of the robot with respect to the world coordinate system, after a successful self-localization, are determined as follows:

$$\theta_0 = 209.67^\circ$$
$$x_0 = 5.19m$$
$$y_0 = 2.67m$$

Then, the mobile robot starts moving towards the goal. If there is no obstacle in its path, the output of the VFH controller will be equal to $\theta_t^*(t)$ which is evaluated by equation 5.3. Upon detection of an object by the sonar sensors, the corresponding steering angle will be calculated and substituted in equation 5.2 as the new direction of motion.

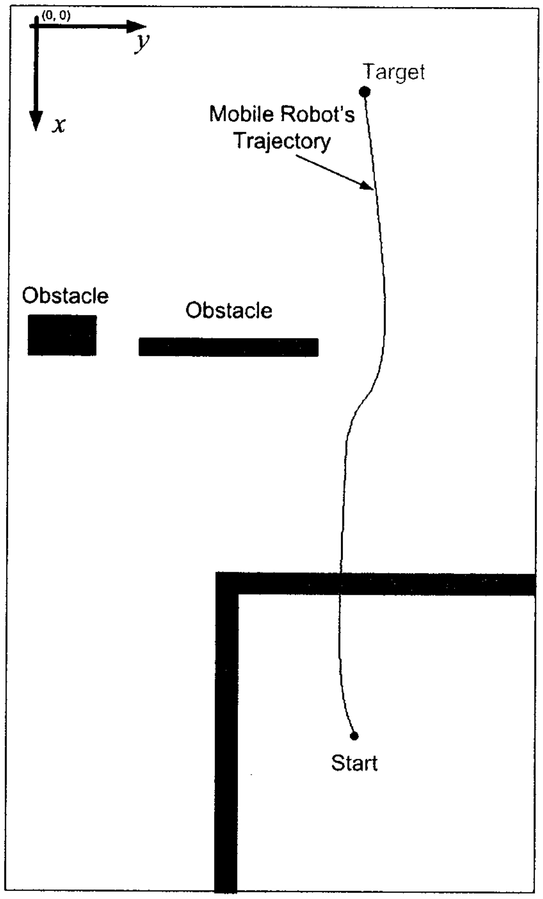


Figure 5.16: Top view of the LAB and the trajectory of the mobile robot.

The obtained histogram grid from the VFH controller after a successful navigation is depicted in Figure (5.18). In our implementation certainty values (CV's) range from 0 to 50. CV = 0 indicates no sensor reading has been projected onto the cell during the run, while CV's > 0 corresponds to the increasing confidence in the existence of an object at that location.

In the next experiment, only the vision-based VFH controller is activated and its corresponding results are depicted in Figure (5.19) to Figure (5.21). In this experiment, the initial values of the robot's parameters are measured as follows:

$$\theta_0 = 208.69^\circ$$
$$x_0 = 5.19m$$
$$y_0 = 2.63m$$

The threshold value of the VFH controller is also assigned to 40,000. Since this threshold value is less than that of for previous experiment (which was set to 70,000), the mobile robot detects the same object faster comparing to the previous experiment and avoids it in a wider field.

Figure (5.22) shows the top view of the LAB with a different setup for the obstacles and the goal. The obtained initial values of the parameters of the robot in this experiment are as follows:

$$\theta_0 = 222.12°$$
$$x_0 = 5.54m$$
$$y_0 = 3.31m$$

In this experiment, both of the VFH controllers are activated. The threshold value for the vision-based VFH controller is set to 40,000 while for the sonar-based one is set to 70,000. The navigation parameters and the obtained histogram grid of combining both of the VFH controllers after a run through the obstacles are depicted in Figure (5.23) and Figure (5.24) respectively. According to the results, since the condition |Vision θ_{Steer}| > |Sonar θ_{Steer}| is always satisfied, the vision-based VFH controller always dominates.

Figure (5.25) shows a new layout of the obstacles achieved by adding an obstacle to the previous layout. All the settings for the new experiment are the same as that of for previous one and the initial values of the robot's parameters are calculated as follows:

$$\theta_0 = 222.70°$$
$$x_0 = 5.49m$$
$$y_0 = 3.34m$$

The resulted navigation parameters are plotted in Figure (5.26). As it can be seen, in some parts of the path |Vision θ_{Steer}| < |Sonar θ_{Steer}|, which causes the sonar-based VFH controller takes over the control at those locations. The histogram grid of combining both of the VFH controllers are also presented in Figure (5.27).

A more complex layout of the obstacles and the trajectory of the mobile robot is depicted in Figure (5.28). This experiment shows how our algorithm can make the robot manoeuver between obstacles and avoid collision. The obstacles which are in black color can only be detected by the sonar sensors while the blue objects can be detected by the vision system (and by the sonar sensors if they are tall enough). The corresponding perceived map of the obstacles is depicted in Figure (5.29) as well.

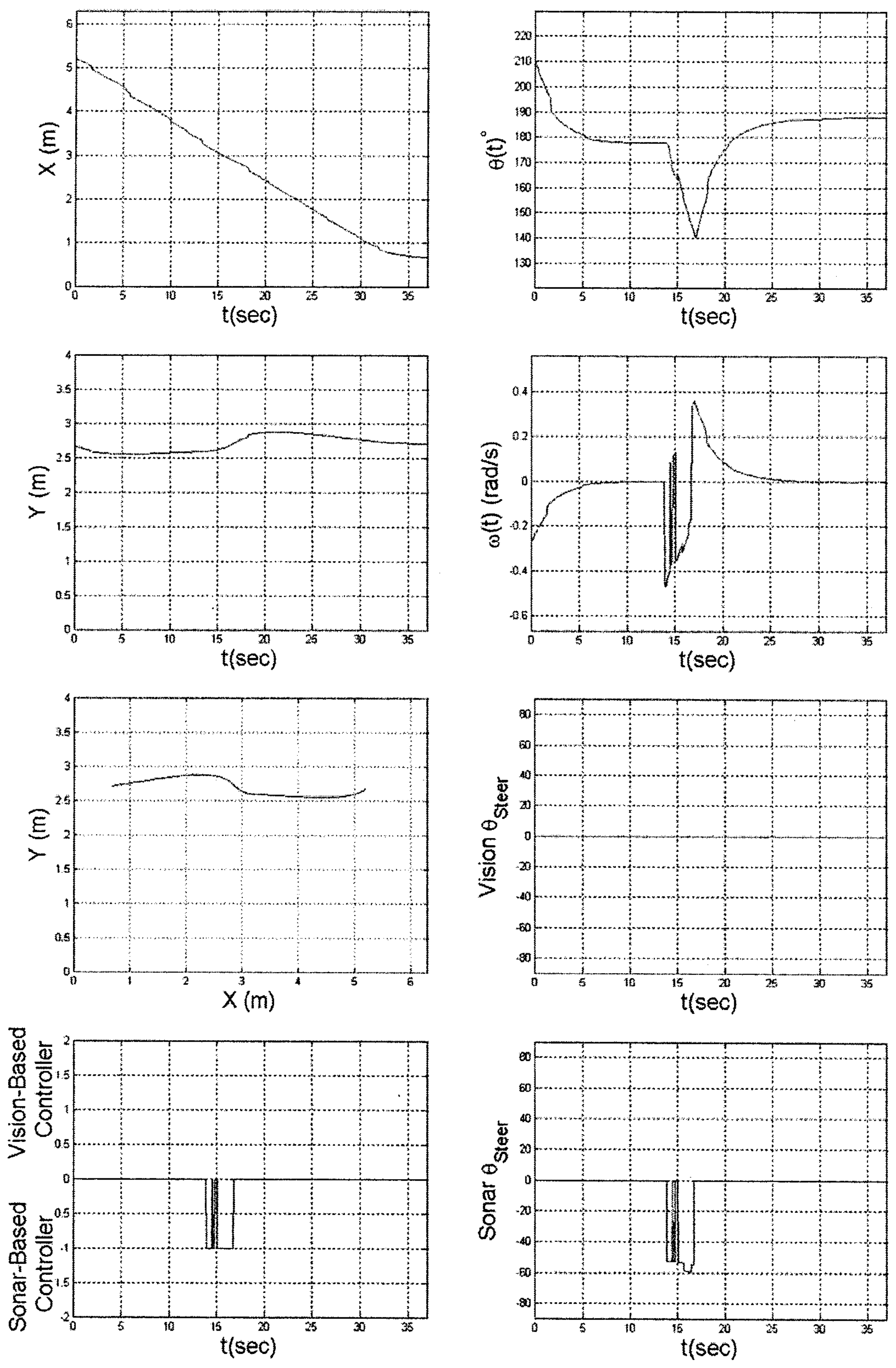


Figure 5.17: Parameters of the navigation (the vision-based VFH controller is inactive).

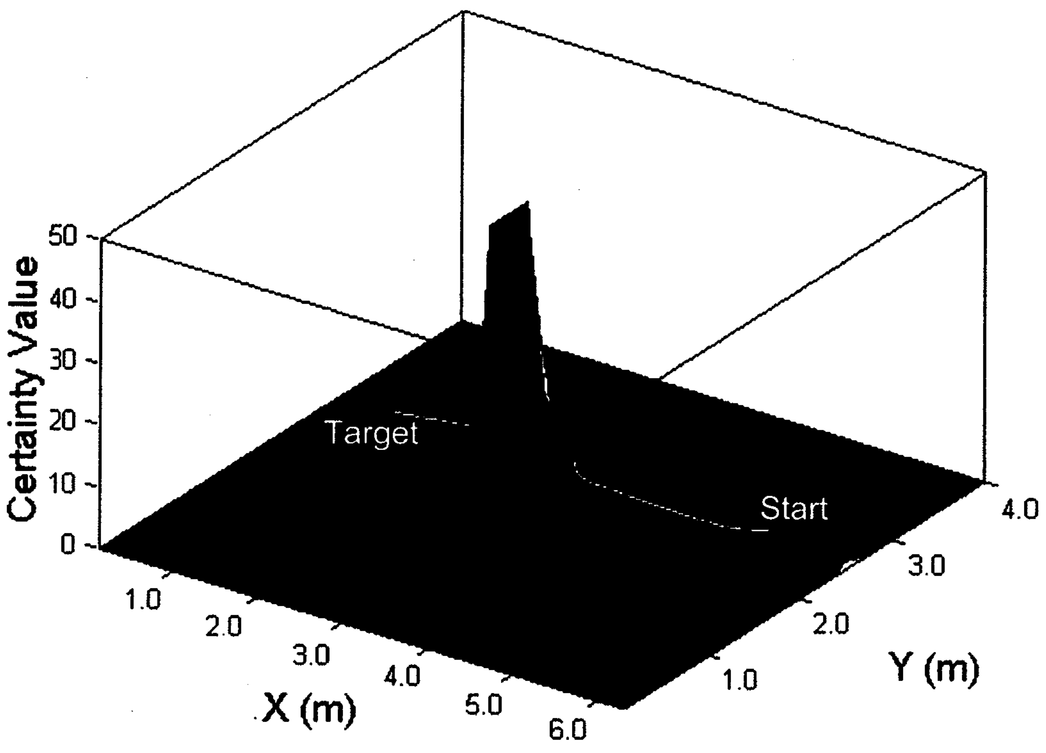


Figure 5.18: The obtained histogram grid after a successful run (the vision-based VFH controller is inactive).

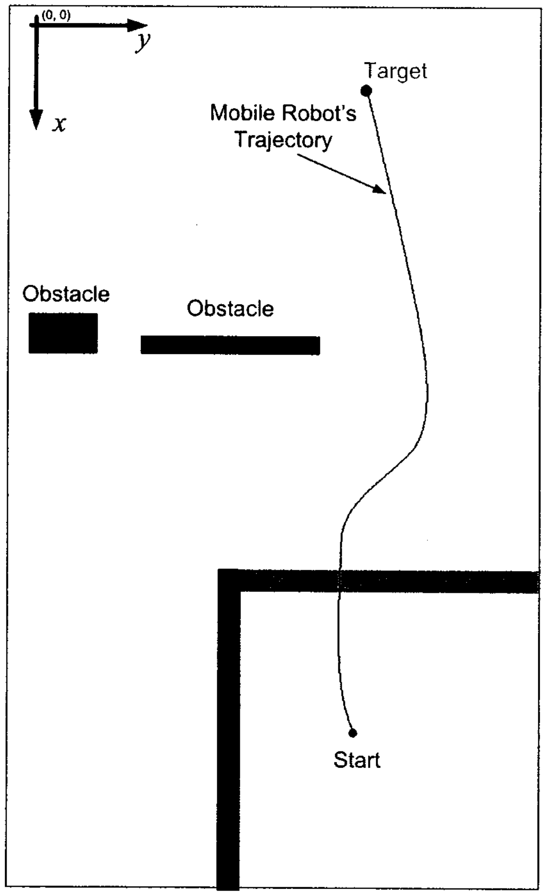


Figure 5.19: Top view of the LAB and the trajectory of the mobile robot.

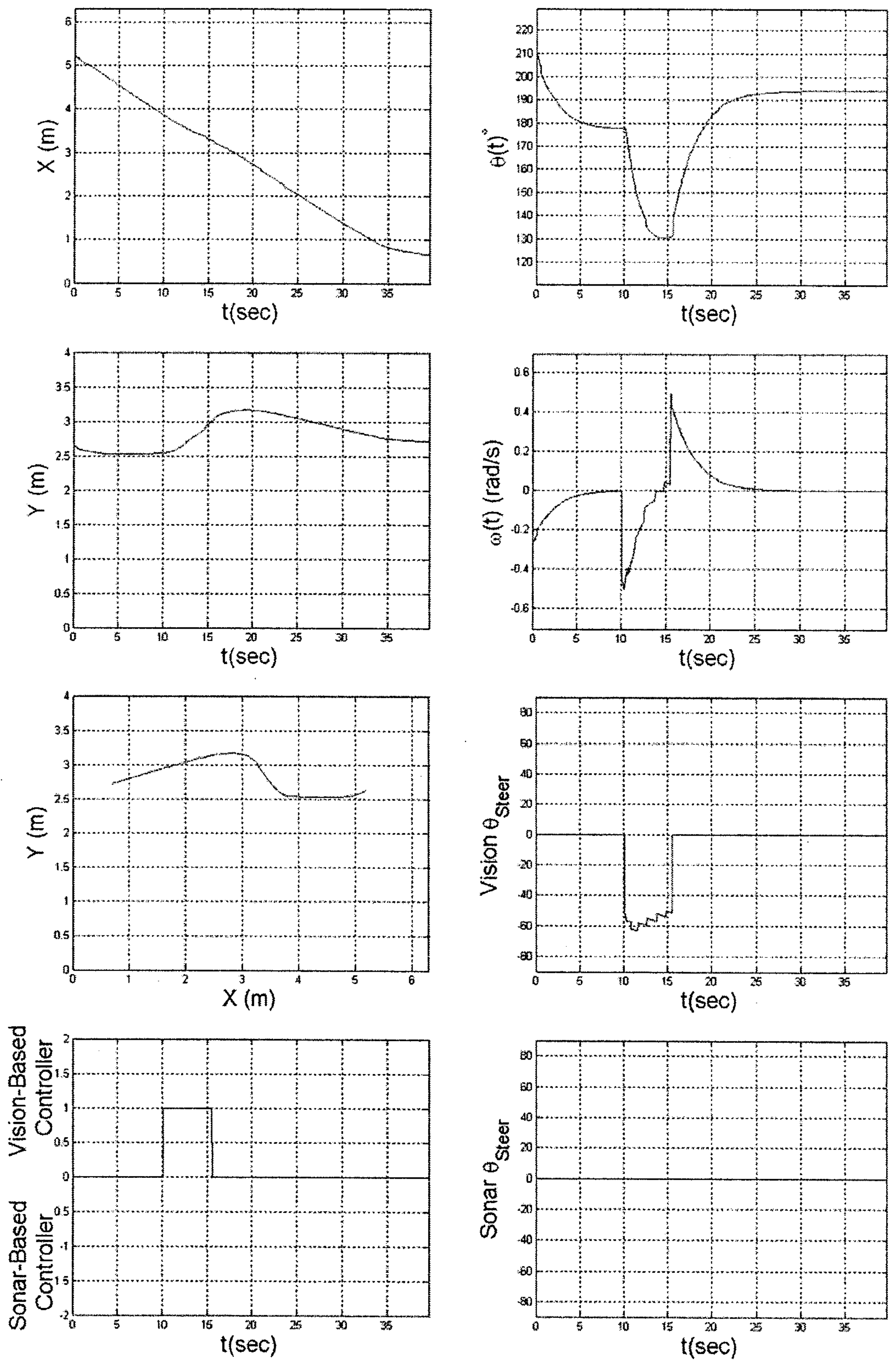


Figure 5.20: Parameters of the navigation (the sonar-based VFH controller is inactive).

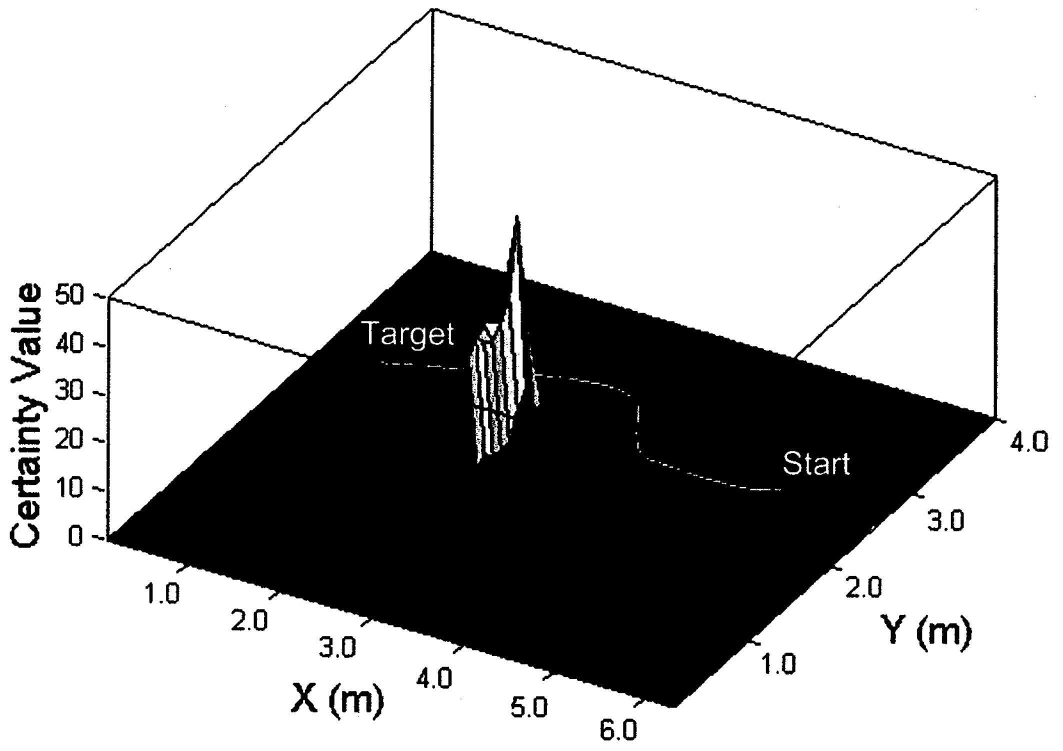


Figure 5.21: The obtained histogram grid after a successful run (the sonar-based VFH controller is inactive).

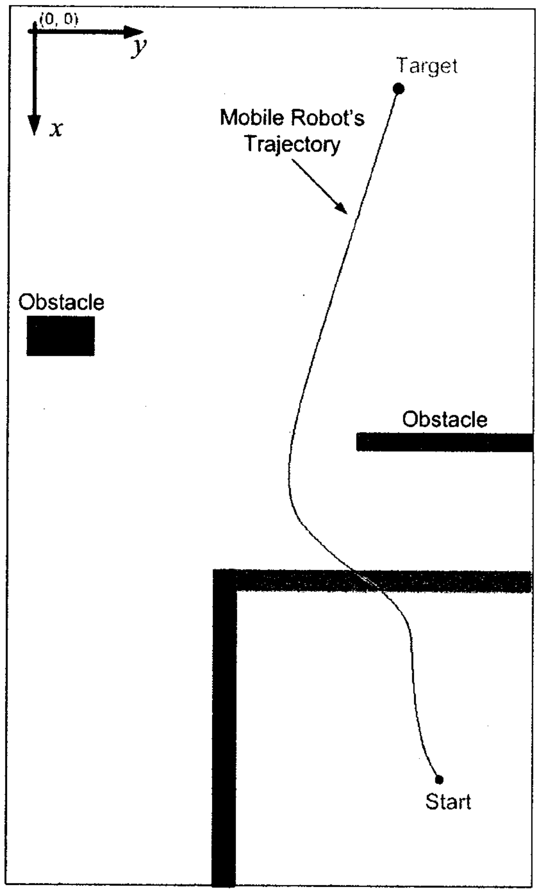


Figure 5.22: Top view of the LAB with a different goal and obstacle setup and the trajectory of the mobile robot.

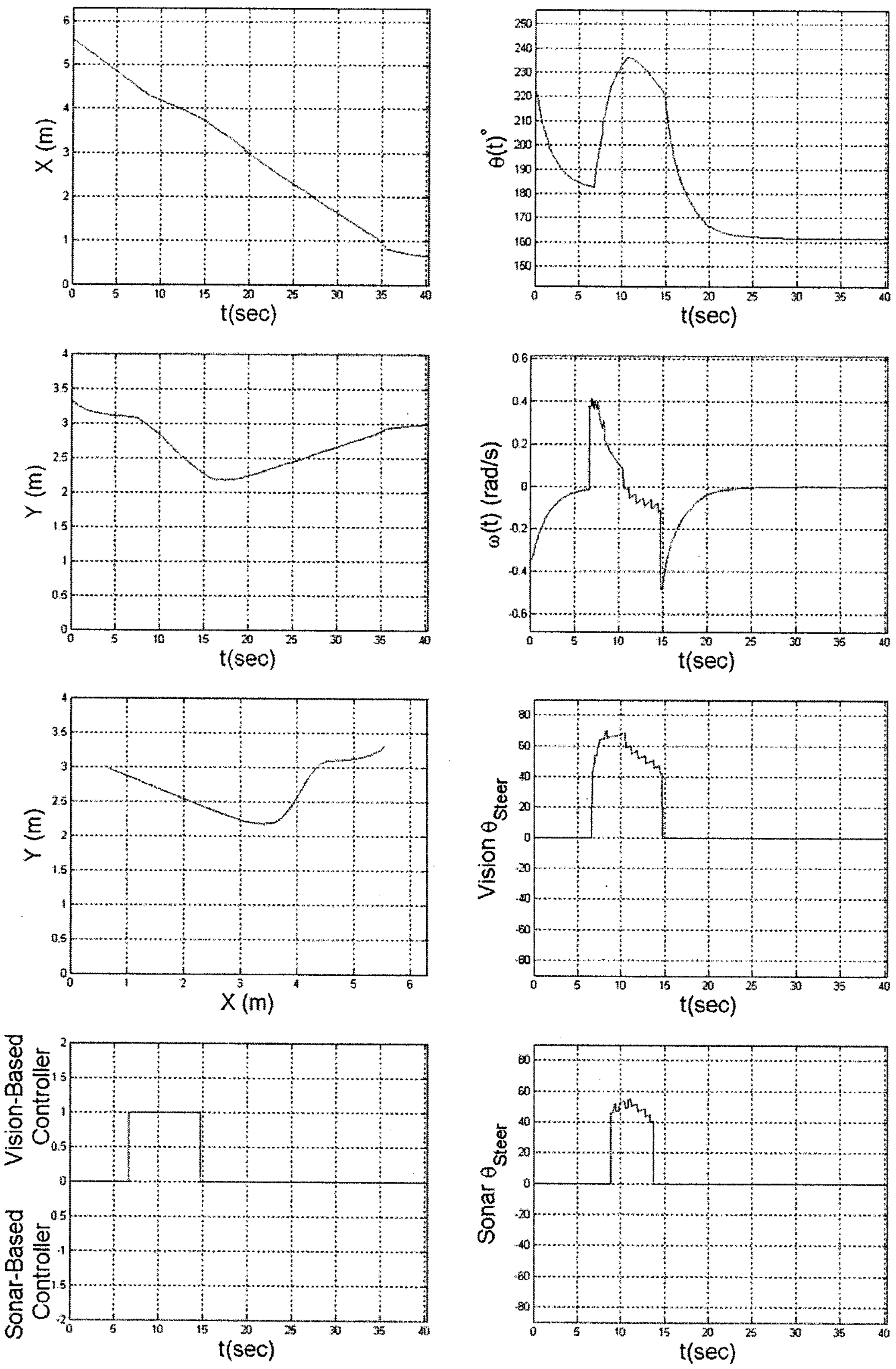


Figure 5.23: Parameters of the navigation.

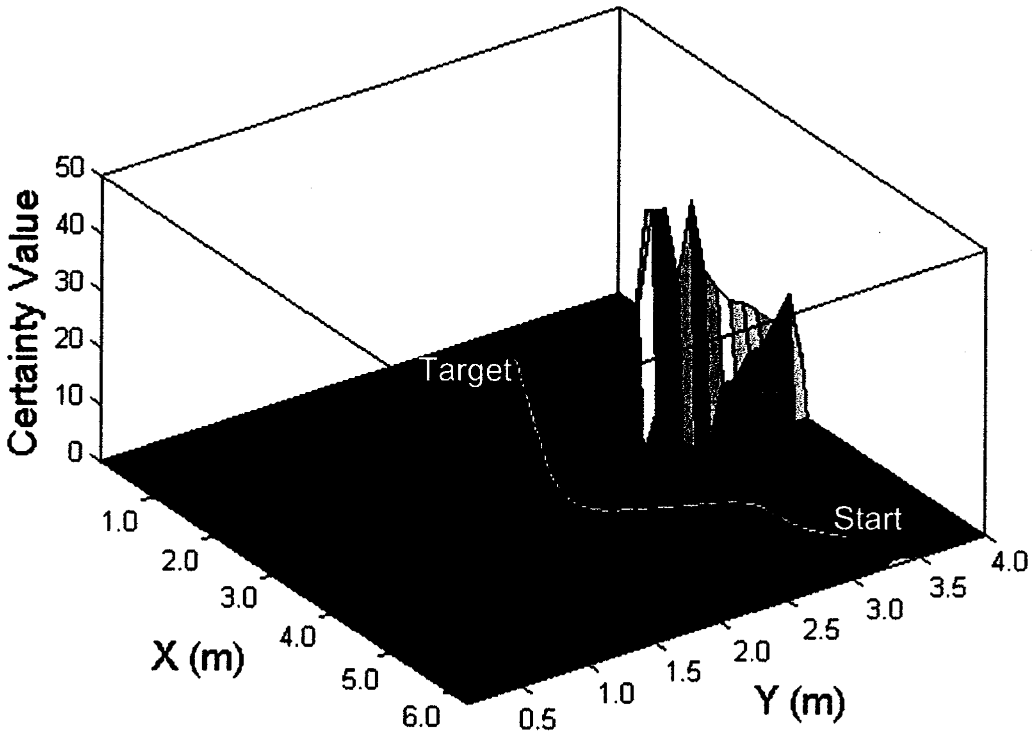


Figure 5.24: The combined histogram grid of both VFH controllers after a successful run.

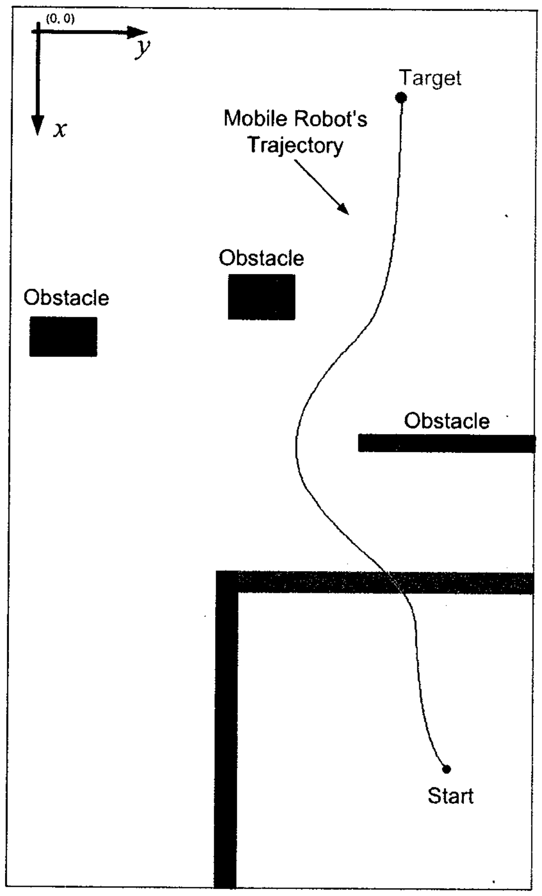


Figure 5.25: Top view of the LAB with a different obstacle setup and the trajectory of the mobile robot.

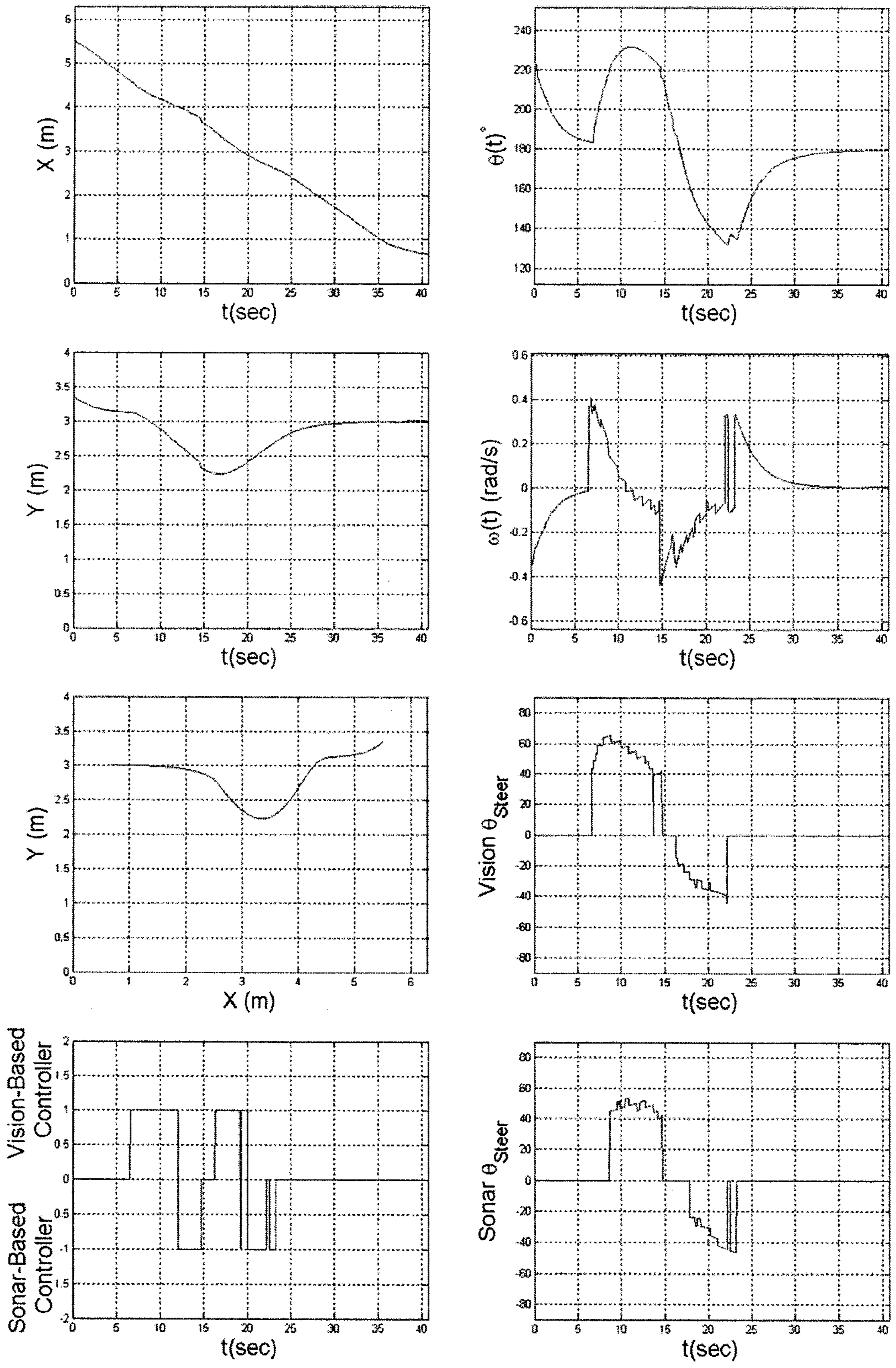


Figure 5.26: Parameters of the navigation.

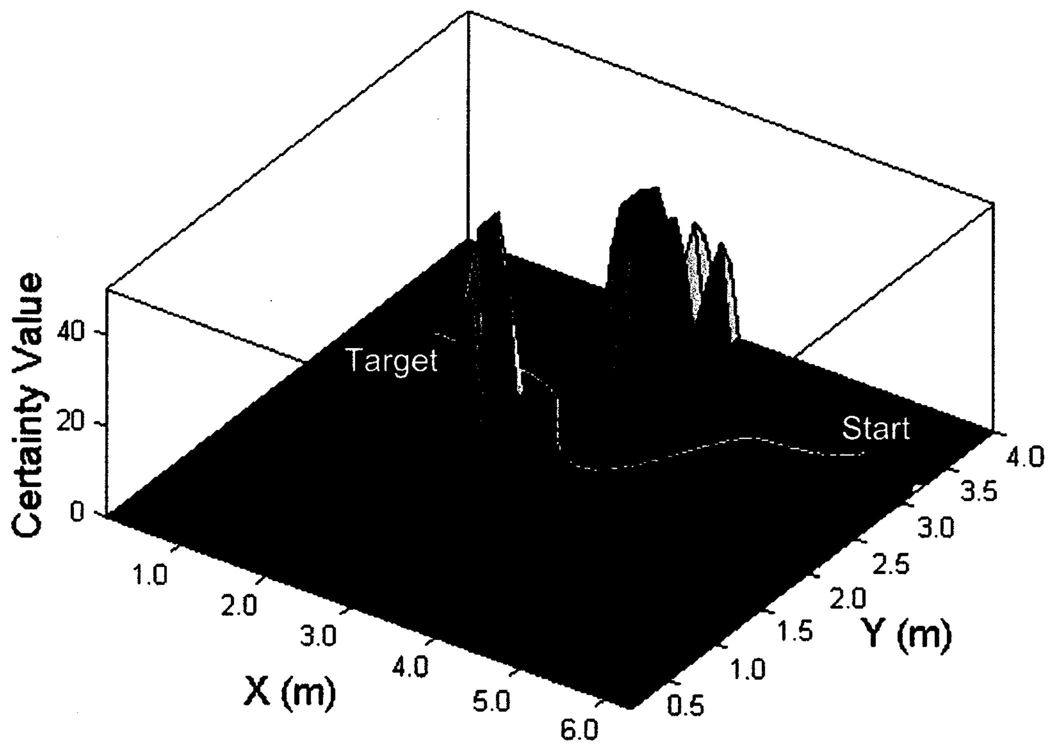


Figure 5.27: The combined histogram grid of both VFH controllers after a successful run.

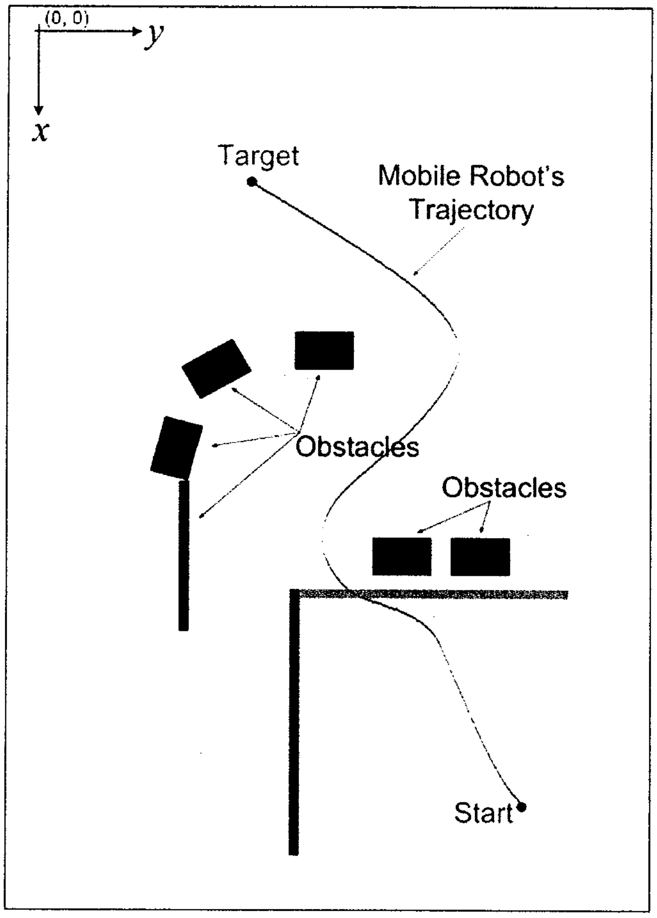


Figure 5.28: Top view of the environment surrounding the mobile robot.

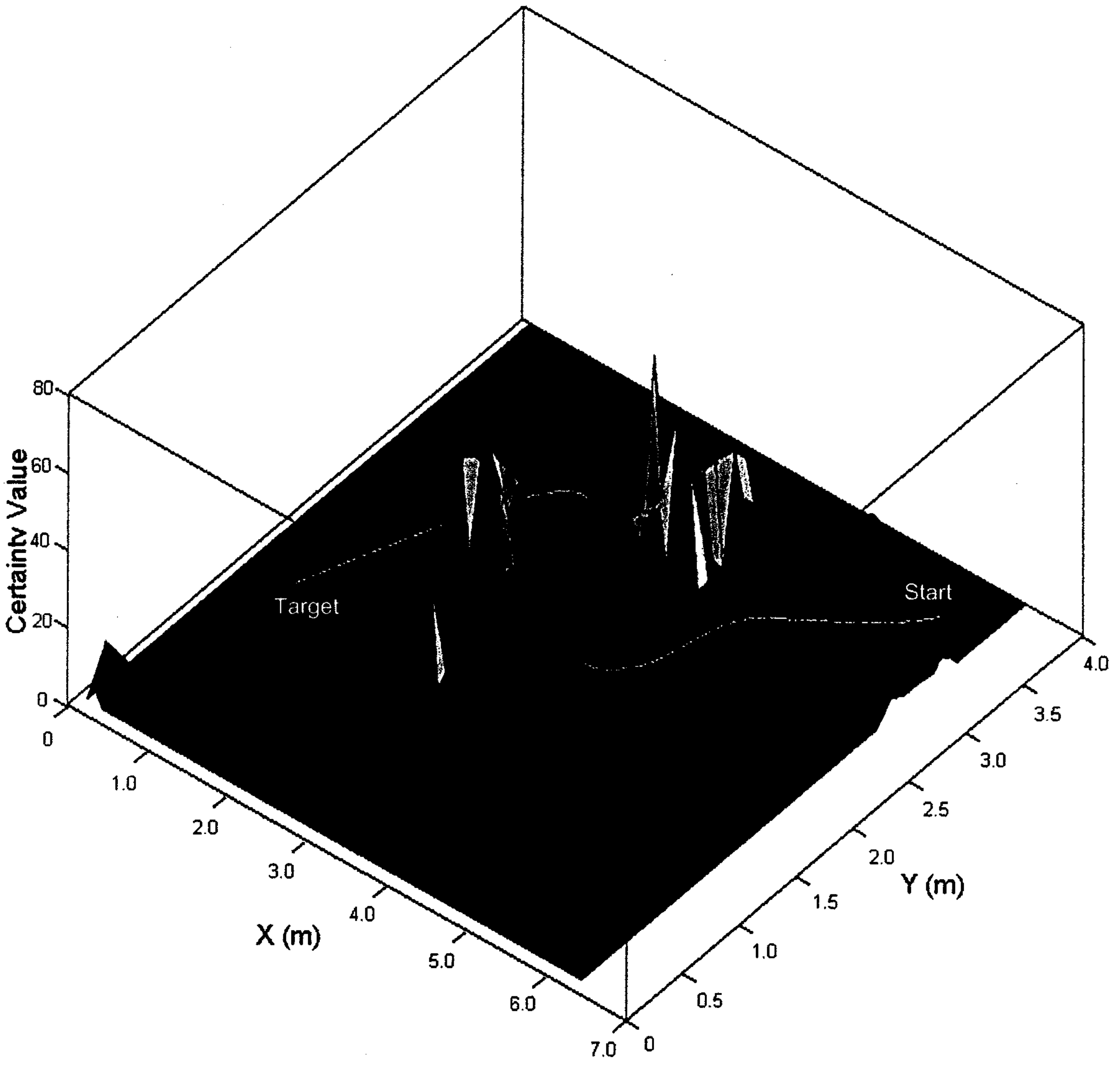


Figure 5.29: The combined histogram grid of both VFH controllers after a successful run.

Chapter 6

Conclusions and future work

An effective method for vision based self-localization using a monocular vision system has been proposed and successfully tested on our research mobile robot "iRobot". Experimental results reveal a high accuracy in the self-localization.

By combining two VFH controllers which work in parallel and use different types of range finders (sonar sensors and vision systems) as input, the robustness of the navigation is considerably improved. This method is computationally efficient, insensitive to misreadings, and allows the mobile robot to navigate towards the target with a continues and smooth speed. The proposed method makes the robot capable of passing between very densely cluttered obstacles while avoiding them (the space between the obstacle can be about twice the diameter of the robot). Moreover, the sensor fusion method helps the robot to detect very short obstacles which may not be detected by the sonar sensors as well as transparent obstacles (such as glass doors) that can not be detected by the vision system. The proposed method has been developed and successfully tested on our experimental mobile robot B21r.

To make our method suitable for realistic conditions, the following aspects should be considered:

The obstacle avoidance algorithm should be improved in such a way that the robot does not get trapped. It can also be modified so that when the robot does not see any obstacle in a far distance, it travels with a fast linear speed and when it gets close to the obstacles, reduces the linear speed as a function of angular velocity. The obstacle avoidance algorithm can also be improved to avoid dynamic obstacle such as humans that may walk on its path.

References

[1] S. Atiya and G.D. Hager, "Real-Time Vision-Based Robot Localization," *IEEE Trans. Robotics and Automation*, Vol. 9, No. 6, pp. 785-8–, Dec. 1993.

[2] N. Ayache and O. D. Faugeras, "Maintaining Representations of the Environment of a Mobile Robot," *IEEE Trans. Robotics and Automation*, Vol. 5, No. 6, pp. 804-819, 1989.

[3] N. Ayache and P.T. Sander, *Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception*, MIT Pree, 1991.

[4] "B21r Mobile Robot User's Guide," *iRobot*, Part Number 2838.

[5] J. Benton and A. Brik, "Hierarchical route planner," *Proceedings of the 1990 Army Science Conference*, Vol. 1, pp. 87-97, 1990.

[6] H. Blum, "A transformation for extracting new descriptors of shape," *Proc. Symp. Models for the Perception of Speech and Visual Form*, Cambridge, MA: MIT Press, 1964.

[7] J. Borenstein, H.R. Everett, and L. Feng, *Navigating Mobile Robots: Systems and Techniques*, A. K. Peters, Ltd., Wellesley, MA, 1996.

[8] J. Borenstein and Y. Koren, "Real-Time Obstacle Avoidance for Fast Mobile Robots," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 19, No. 5, pp. 1179-1187, 1989.

[9] J. Borenstein and Y. Koren, "Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," *Proc. IEEE Int. Conf. Robotics and Automation*, 1990.

[10] J. Borenstein and Y. Koren, "The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots," *IEEE Transaction on Robotics And Automation*, Vol. 7, No. 3, pp. 278-288, June 1991.

[11] M. Born and E. Wolf, *Principles of Optics*, New York: Peramon, 5th Edition, 1975.

[12] B. Browning and D. Govindaraju, *Fast, Robust Techniques for Colored Object Detection in Variable Lighting Conditions*, United State Army, 2003.

[13] W. Burgard, A.B. Cremers, D. Fox, D. Hänel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence*, Vol. 114, pp. 3-55, 1999.

[14] I.J. Cox, "Modeling a Dynamic Environment Using a Bayesian Multiple Hypothesis Approach," *Artificial Intelligence*, Vol. 66, No. 2, pp. 311-344, Apr. 1994.

[15] D. Crandall and J. Luo, "Robust Color Object Detection using Spatial-Color Joint Probability Functions," *Proceedings of IEEE Computer Society*, 2004.

[16] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo Localization for Mobile Robots," *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1322-1328, May 1999.

[17] W. Deng, S. S. Iyengar, and N. E. Brener, "A Fast Parallel Thinning Algorithm for The Binary Image Skeletonization," *The International Journal of High Performance Computing Applications*, Vol. 14, No. 1, pp. 65-81, Spring 2000.

[18] Richard O. Duda and Peter E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Journal of the ACM*, Vol. 15, No. 1, pp. 11-15, January 1972.

[19] N. Duffy, J. Crowley, and G. Lacey, "Object Detection Using Color," *Proceedings of International Conference on Pattern Recognition*, Vol. 1, pp. 1700, Washington, DC, USA, 2000.

[20] S. Florczyk, *Robot Vision: Video-based Indoor Exploration with Autonomous and Mobile Robots*, Wiley-VCH Verlag GmbH & Co. KGaA, 2005.

[21] R. Garcia-Campos, J. Battle, and R. Bischoff, "Architecture of an object-based tracking system using colour segmentation," *3rd International Workshop in Signal and Image Processing*, 1996.

[22] P. Gaussier, C. Joulain, S. Zrehen, and A. Revel, "Visual Navigation in an Open Environment without Map," *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, pp. 545-550, Sept. 1997.

[23] Guilherme N. DeSouza and Avinash C. Kak, "Vision for Mobile Robot Navigation: A Survey," *IEEE Transactions on pattern analysis and machine inteligence*, Vol. 24, No. 2, pp. 237-267, Feb. 2002.

[24] M. Hashima, F. Hasegawa, S. Kanda, T. Maruyama, and T. Uchiyama, "Localization and Obstacle Detection for a Robot for Carrying Food Trays," *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, pp. 345-351, Sept. 1997.

[25] E. Huber and D. Kortenkamp, "Using Stereo Vision to Pursue Moving Agent with a Mobile Robot," *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 3, pp. 2340-2346, May 1995.

[26] M. Isard and A. Blake, "CondensationConditional Density Propagation for Visual Tracking," *Int. Journal of Computer Vision*, Vol. 29, No. 1, pp. 5-28, 1998.

[27] S.D. Jones, C. Andresen, and J.L. Crowley, "Appearance Based Processes for Visual Navigation," *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, pp. 551-557, Sept. 1997.

[28] C. Joulian, P. Gaussier, A. Revel, and B. Gas, "Learning to Build Visual Categories from Perception-Action Associations," *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, pp. 857-864, 1997.

[29] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Proc. IEEE Int. Conf. Robotics Automat*, pp. 500-505, Mar. 1985.

[30] D. Kim and R. Nevatia, "Representation and Computation of the Spatial Environment for Indoor Navigation," *Proc. Int. Conf. Computer Vision and Pattern Recognition*, pp. 475-482, 1994.

[31] D. Kim and R. Nevatia, "Symbolic Navigation with a Generic Map," *Proc. IEEE Workshop Vision for Robots*, pp. 136-145, Aug. 1995.

[32] D. Kim and R. Nevatia, "Recognition and Localization of Generic Objects for Indoor Navigation Using Functionality," *Image and Vision Computing*, Vol. 16, No. 11, pp. 729-743, Aug. 1998.

[33] J. Kittler, J. llingworth, and J. Föglein, "Threshold selection based on a simple image statistic," *Computer Vision, Graphics, and Image Processing*, Vol. 30, pp. 125-147, 1985.

[34] A. Koschan, "A comparative study on color edge detection," *2nd Asian Conference on Computer Vision*, Vol. 3, pp. 574-578, 1995.

[35] A. Kosaka and A.C. Kak, "Fast Vision-Guided Mobile Robot Navigation Using Model-Based Reasoning and Prediction of Uncertainties," *Computer Vision, Graphics, and Image ProcessingImage Understanding*, Vol. 56, No. 3, pp. 271-329, 1992.

[36] G. Kraetzschmar, S. Sablatnög, S. Enderle, H. Utz, S. Steffen and G. Palm, "Integration of multiple representation and navigation concepts on autonomous mobile robots," H.-M. Gro, K. Debes, and H.-J. Böhme, Editor, *Workshop SOAVE 2000 Selbstorganisation von adaptivem Verhalten*, pp. 1-13, Düsseldorf, 2000, VDI GmbH.

[37] T. Lalanne and C. Lempereur, "Color Recognition With A Camera: A Supervised Algorithm For Classification," *IEEE Southwest Symposium on Image Analysis and Interpretation*, 1998.

[38] Y. Matsumoto, M. Inaba, and H. Inoue, "Visual Navigation Using View-Sequenced Route Representation," *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 1, pp. 83-88, Apr. 1996.

[39] M. Meng and A.C. Kak, "Mobile Robot Navigation Using Neural Networks and Nonmetrical Environment Models," *IEEE Control Systems*, pp. 30-39, Oct. 1993.

[40] M. Meng and A.C. Kak, "NEURO-NAV: A Neural Network Based Architecture for Vision-Guided Mobile Robot Navigation Using Non-Metrical Models of the Environment," *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 2, pp. 750-757, 1993.

[41] "Mobility Robot Integration Software Users Guide," *iRobot*, Part Number 2841.

[42] R. Mohr and B. Triggs, "Projective geometry for image analysis," *http://lear.inrialpes.fr/people/triggs/pubs/isprs96/isprs96.html*, July 1996.

[43] U. Montanari, "Continuous skeletons from digitized images," *Journal of the ACM*, Vol. 16, No. 4, pp. 534-549, 1969.

[44] H. Moravec, " Visual Mapping by a Robot Rover," *Proceedings of IJCAI*, pp. 598-600, Tokyo, 1979.

[45] T. Nakamura and M. Asada, "Motion Sketch: Acquisition of Visual Motion Guided Behaviors," *Proc. 14th Int. Joint Conf. Artificial Intelligence*, Vol. 1, pp. 126-132, Aug. 1995.

[46] T. Nakamura and M. Asada, "Stereo Sketch: Stereo Vision-Based Target Reaching Behavior Acquisition with Occlusion Detection and Avoidance," *Proc. IEEE Int. Conf. Robotics and Automation*, Vol 2. pp. 1314-1319, April 1996.

[47] T. Ohno, A. Ohya and S. Yuta, "Autonomous Navigation for Mobile Robots Referring Pre-Recorded Image Sequence," *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, Vol. 2, pp. 672-679, Nov. 1996.

[48] G. Oriolo, G. Ulivi, and M. Vendittelli, "On-Line Map Building and Navigation for Autonomous Mobile Robots," *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 2900-2906, May 1995.

[49] J. Pan, D.J. Pack, A. Kosaka, and A.C. Kak, "FUZZY-NAV: A Vision-Based Robot Navigation Architecture Using Fuzzy Inference for Uncertainty-Reasoning," *Proc. IEEE World Congress Neural Networks*, Vol. 2, pp. 602-607, July 1995.

[50] R. P. Paul, *Robot Manipulators*, MIT Press, 7th Edition, 1986.

[51] M. Reece and K. D. Harris, "Memory for places: A navigational model in support of Marrs theory of hippocampal function," *Hippocampus*, Vol. 6, pp. 735-748, 1996.

[52] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi, "Divergent Stereo for Robot Navigation: Learning from Bees," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition*, 1993.

[53] N. Sawasaki, T. Morita, and T. Uchiyama, "Design and Implementation of High-Speed Visual Tracking System for Real-Time Motion Analysis," *Proc. IAPR Int. Conf. Pattern Recognition*, Vol. 3, pp. 478-483, 1996.

[54] Linda G. Shapiro and Geroge C. Stockman, *Computer Vision*, Prentice Hall, 2001.

[55] R. Simmons and S. Koenig, "Probabilistic Robot Navigation in Partially Observable Environments," *Proc. Int. Joint Conf. Artificial Intelligence*, pp. 1080-1087, Aug. 1995.

[56] M. Stoksik, D. T. Nguyen, and M. Czernkowski, "A Neural Net Based Color Recognition System," *Second International Conference on Artificial Neural Networks*, 1991.

[57] S. Thrun, "Learning Metric-Topological Maps for Indoor Mobile Robot Navigation," *Artificial Intelligence*, Vol. 99, No. 1, pp. 21-71, Feb. 1998.

[58] S. Thrun, "Probabilistic Algorithms in Robotics," *Technical Report CMU-CS-00-126*, Carnegie Mellon Univ., 2000.

[59] V. Vinod, H. Murase, and C. Hashizume, "Focused Color Intersection with Efficient Searching for Object Detection and Image Retrieval," *Proceedings of International Conference on Multimedia Computing and Systems*, 1996.

[60] J.S. Weszka, "A survey of threshold selection techniques," *Computer Graphics and Image Processing*, Vol. 7, pp. 259-265, 1978.

[61] Yun Xia, "Skeletonization via the realization of the fire fronts propagation and extinction in digital binary shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-11, pp. 1076-1086, 1989.

[62] Y. Yagi, S. Kawato and S. Tsuji, "Real-Time Omnidirectional Image Sensor (COPIS) for Vision-Guided Navigation," *IEEE Trans. Robotics and Automation*, Vol. 10, No. 1, pp. 11-22, Feb. 1994.

[63] X. Yang, "Vision-Based Trajectory Tracking Algorithm with Obstacle Avoidance for a Wheeled Mobile Robot," Master's thesis, Lakehead University, Aug. 2005.

[64] Z. Zhang and O. Faugeras, "A 3D World Model Builder with a Mobile Robot," *Int. Journal of Robotics Research*, Vol. 11, No. 4, pp. 269-285, 1992.

[65] J.Y. Zheng, M. Barth, and S. Tsuji, "Autonomous Landmark Selection for Route Recognition by a Mobile Robot," *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 2004-2009, 1991.