1996-09

New motion planning and real-time localization methods using proximity for autonomous mobile robots

Wahdan, Mahmoud A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

NEW MOTION PLANNING AND REAL-TIME LOCALIZATION METHODS USING PROXIMITY FOR AUTONOMOUS MOBILE ROBOTS

by

Mahmoud A. Wahdan

September 1996

Thesis Advisor: Yutaka Kanayama

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
September 1996

3. REPORT TYPE AND DATES COVERED
Doctoral Dissertation

4. TITLE AND SUBTITLE
NEW MOTION PLANNING AND REAL-TIME LOCALIZATION METHODS USING PROXIMITY FOR AUTONOMOUS MOBILE ROBOTS

5. I

6. AUTHOR(S)
Mahmoud A. Wahdan

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

8. PERFORMING ORGANIZATION

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

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

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

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

One of the most difficult theoretical problems in robotics--motion planning for rigid body robots--must be solved before a robot can perform real-world tasks such as mine searching and processing. This dissertation proposes a new motion planning algorithm for an autonomous robot, as well as the method and results of implementing this algorithm on a real vehicle.

This dissertation addresses the problem of safely navigating an autonomous vehicle through free space of a two dimensional, world model with polygonal obstacles from a start configuration (position/orientation) to a goal configuration using smooth motion under the structure of a layered motion planning approach. The approach proposes several new concepts, including *v-edges* and *directed v-edges*, and divides the motion planning problem of a rigid body vehicle into two subproblems: (i) finding a global path using Voronoi diagrams and for a given start and goal configurations planning an optimal global path; the planned path is a sequence of directed v-edges, (ii) planning a local motion from the start configuration, using the aforementioned global path. The problem of how to design a safe and smooth path, is effectively solved by the steering function method and proximity. Another problem addressed here is how to make a smooth transition when the vehicle gets closer to an intersection of two distinct boundaries.

14. SUBJECT TERMS
Robotics, autonomous vehicles, global path planning, local motion planning, steering function, polygon tracking, self localization.

15. NUMBER OF PAGES
240

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

13. This dissertation also presents a robust algorithm for the vehicle to continually eliminate its positional uncertainty while executing missions. This functionality is called *self-localization* which is an essential component of model-based navigation for indoor applications. This algorithm is based on the two-dimensional transformation group. Through this method, the robot can minimize its positional uncertainty, make safe and reliable motions, and perform useful tasks in a partially known world.

All of the proposed algorithms were implemented on an autonomous mobile robot *Yamabico-11* to confirm our theoritical algorithms.

Approved for public release; distribution is unlimited

NEW MOTION PLANNING AND REAL-TIME LOCALIZATION METHODS USING PROXIMITY FOR AUTONOMOUS MOBILE ROBOTS

Mahmoud A. Wahdan
Colonel, Egyptian Army
B.S., Military Technical College, Egypt, 1977
M.S., Cairo Universty, Egypt, 1990

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1996

ABSTRACT

One of the most difficult theoretical problems in robotics–motion planning for rigid body robots–must be solved before a robot can perform real-world tasks such as mine searching and processing. This dissertation proposes a new motion planning algorithm for an autonomous robot, as well as the method and results of implementing this algorithm on a real vehicle.

This dissertation addresses the problem of safely navigating an autonomous vehicle through free space of a two dimensional, world model with polygonal obstacles from a start configuration (position/orientation) to a goal configuration using smooth motion under the structure of a layered motion planning approach. The approach proposes several new concepts, including *v-edges* and *directed v-edges*, and divides the motion planning problem of a rigid body vehicle into two subproblems: (i) finding a global path using Voronoi diagrams and for a given start and goal configurations planning an optimal global path; the planned path is a sequence of directed v-edges, (ii) planning a local motion from the start configuration, using the aforementioned global path. The problem of how to design a safe and smooth path, is effectively solved by the steering function method and proximity. Another problem addressed here is how to make a smooth transition when the vehicle gets closer to an intersection of two distinct boundaries.

This dissertation also presents a robust algorithm for the vehicle to continually eliminate its positional uncertainty while executing missions. This functionality is called *self-localization* which is an essential component of model-based navigation for indoor applications. This algorithm is based on the two-dimensional transformation group. Through this method, the robot can minimize its positional uncertainty, make safe and reliable motions, and perform useful tasks in a partially known world.

All of the proposed algorithms were implemented on an autonomous mobile robot *Yamabico-11* to confirm our theoritical algorithms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

For four long years I have devoted myself to seeking the truth in the sea of knowledge at the Naval Postgraduate School. If I was successful in achieving this goal, my own effort was not the only factor. There were many people behind this endeavor. They deserve my appreciation.

First and foremost, I would like to present my gratitude to my sister Nadia Wahdan. She went through and shared all my frustration and excitement during these years. Without her encouragement and unconditional support, I wouldn't have accomplished this work.

I also wish to express my deepest gratitude to Professor Yutaka Kanayama whose support, guidance, and enthusiasm have been a constant inspiration to me. His door was always open for me and for any other student needing help. His patience and positive attitude were invaluable to this research.

I am deeply indebted to my committee for their patience and wisdom. The comments of Professor Thomas Wu and Professor Cynthia Irvine regarding my writing and scientific approach were exceptionally helpful. Professor Xiaoping Yun provided valuable comments on the overall layout of this dissertation and made insightful suggestions to improve my understanding of the robotic field. Professor Craig Rasmussen generously provided me with badly needed mathematical guidance as well as proofreading the manuscript numerous times. From my deepest heart, I would like to thank Professor Fariba Fahroo for her support and encouragement in hard times. With her support and help, I achieved the highest goal in my whole life.

I appreciate the continuous encouragement and support of the members of the Yamabico research group, Khaled Morsy (Egypt), Ed Mays (USA), and Vasilios Karamanlis (Greece).

I would like to thank my best friends Nabil Khalil, Ashraf Mamdouh and his wife, and Khaled Morsy and his wife for their positive attitude before and after my

final defense. This gave me lots of courage in preparing for my defense. Best wishes to all.

I. INTRODUCTION

A. BACKGROUND

Answering the question "Where am I?" is one of the most elementary tasks for any natural or artificial creature moving through the real world in a goal-oriented fashion. Not only human beings but also animals solve this problem easily and with astonishing accuracy by combining visual, acoustic, and other kinds of perceptions, with vague knowledge about the traveled distance, and spatial knowledge which was gathered and memorized at previous times. To understand and model the mechanisms underlying this skill is one of the challenges for researchers and engineers who want to build autonomous mobile robot vehicles. In the field of robotics, the ultimate goal is to design an autonomous robot that is artificially intelligent. Recent advances in computer processing speed have encouraged the development of increasingly capable mobile robot platforms. Making progress toward autonomous robots is of major practical interest in a wide variety of application domains including manufacturing, construction, waste managemnent, space exploration, undersea work, assistance for the disabled, and medical surgery [49]. Due to the characteristics of reprogrammability and multifunctionality, robots have been used in factories to perform a variety of tasks including material handling, welding, painting, assembly, etc. In addition, it is expected that by the end of this century robots will be able to perform very complex tasks such as construction and maintenance in factories and households [45]. The popular trend in current military applications is to accomplish the required mission with a minimum loss of life. Consequently, many government-sponsored efforts are underway to build systems for fighting fires, handling ammunition, transporting material, conducting underwater search and inspection operations, mine searching and other dangerous tasks now performed by humans [20].

Many of the above tasks require motion of the robot in order to carry out any task. Thus there is a problem known as the motion planning problem. Although the research in robot motion planning can be traced back to the late 1960's, most of the theoretical breakthroughs and practical understandings of the issue have been achieved only in the last decade, and much of the problem is still outstanding. The problem of motion planning for rigid body robots has been considered one of the most difficult theoretical problems in robotics and, obviously, must be solved for a robot to perform real-world tasks such as mine searching and processing. The difficulty of motion planning can best be summarized by J. C. Latombe [49] as follows:

> At first glance motion planning looks relatively simple, since humans deal with it with no apparent difficulty in their everyday lives. In fact, as is also the case with perception, the elementary operative intelligence that people use unconsciously to interact with their environment ... turns out to be extremely difficult to duplicate using a computer-controlled robot. It is true that some naive methods can produce apparently impressive results, but the limitations of these methods quickly become obvious. The unaware reader will be surprised by the amount of nontrivial mathematical and algorithmic techniques that are necessary to build a reasonably general and reliable motion planner.

The level of complexity of the problem of motion planning again depends on how the robot is being modeled and what physical constraints are imposed on it.

Motion planning rather than *path planning* is used, because vehicles considered here are not points, but rigid bodies. In path planning, the result is a series of positions which must be followed by the vehicle. In motion planning, not only is position important, but also the orientation of the vehicle are important as it follows a path.

For an autonomous vehicle, planning motions that avoid known and unknown objects in its environment is the most fundamental functionality. Given an arbitrary mission, for instance, mine searching and clearance, motion planning is an inevitable subproblem that needs to be solved.

Generating collision-free motion of acceptable quality is one of the main concerns in robotics. A typical robot presents an arm manipulator with a fixed base operating in three-dimensional space, or a mobile vehicle operating in two-dimensional space, or a combination of the two. Whatever form it takes, the robot is expected to move purposely and safely in an often complex environment filled with known or unknown obstacles.

Central to the success of robotic systems is the availability of intelligent robot planning systems. With such a system, a robot accepts a goal statement or a task specification (instead of the details of the robot actions) and then it can generate a sequence of robot-level operations. By following these operations, the goal can be accomplished.

The general motion planning problem for a system of autonomous vehicles can be stated as follows: Given (1) an initial state of the vehicles, (2) a desired final state of the vehicles, and (3) any constraints on allowable motions, find a collision-free motion of the vehicles from the initial state to the final state that satisfies the constraints.

Also, for a mobile robot, maintaining exact position information poses a major problem. A key capability of a mobile robot operating in an indoor environment is *localization*, i.e. determination of its current position and orientation (posture). Automated guided vehicles, as used for transportation tasks in factories, still need a network of physical guidelines buried in, or attached to, the floor [17]. Recent developments permit leaving the guideline for short maneuvers, for example at crossings or docking stations. Increased flexibility can be achieved by free-navigating vehicles using dead-reckoning and artificial or natural landmarks for localization. Results of related techniques are reported in [15, 19].

Because of its simplicity and low cost, dead-reckoning is the most commonly used localization technique. However, because of error accumulation in dead-reckoning systems, posture errors grow without bound unless they are reduced by reference measurements. For this purpose, passive sensors like cameras [46] as well as active sensors like sonar [51] and infrared imaging systems [12] have been applied. Natural landmarks, such as walls and edges, or artificial landmarks, such as corner cubes and retro-reflective strips are used as absolute references.

Navigation which is a fundamental requirement of autonomous mobile robots, can be broadly separated into two distinct approaches: reference and dead reckoning. Reference guidance refers to navigation with respect to a coordinate frame based on visible external landmarks. Dead reckoning refers to navigation based on odometry, inertial guidance, or some other "self-contained" sensing. Dead reckoning usually provides the vehicle with an estimate of its position. Its disadvantage is that the position error grows without bound unless an independent reference is used periodically to reduce the error. Reference guidance has the advantage that position errors are bounded, but detection of external references or landmarks and real-time position fixing may not always be possible. Clearly, dead reckoning and reference navigation are complementary and combinations of the two approaches can provide very accurate positioning systems.

Starting from the premise that coping with uncertainty is the most crucial problem a mobile robot must face, we can conclude that the robot must have the following basic capabilities:

- **Sensory interpretation**: The robot must be able to determine its relationship to the environment by sensing. A wide variety of sensing technologies are available: odometry; ultrasonic; infrared and laser range sensing; and monocular, binocular, and trinocular vision have all been explored. The difficulty is in interpreting these data, that is, in deciding what the sensor signals tell us about the external world.

- **Reasoning**: The robot must be able to decide what actions are required to achieve its goal(s) in a given environment. This may involve decisions ranging from what paths to take to what sensors to use.

B. PROBLEM STATEMENT

1. Definitions

This subsection defines a list of terms and concepts used throughout this dissertation.

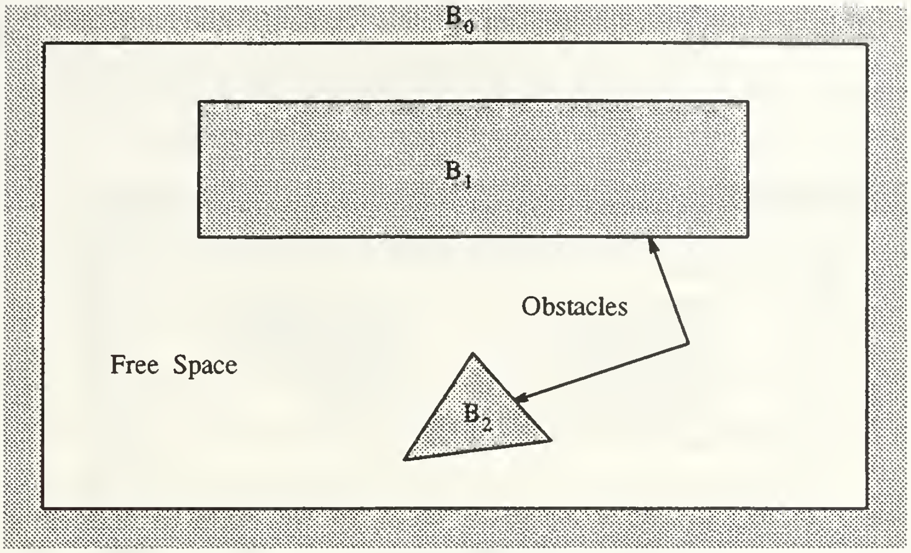


Figure 1. Robot's world space

Let $\mathcal{R}$ denote the set of real numbers. The environment for the motion planning task of this dissertation is a two-dimensional plane $\mathcal{R}^2$ on which a global Cartesian coordinate system is defined.

Let $B_1, \cdots, B_n$ be fixed objects (simple polygons) distributed in $\mathcal{R}^2$. These B_i's are called *obstacles.*

A *world* $\mathcal{W}$ is a set of n simple polygonal obstacles,

$$\mathcal{W} = \{B_0, B_1, \cdots, B_n\}, \quad n > 0$$

where B_0 is the outermost polygonal boundary, $B_1, \cdots B_n$ are polygonal obstacles inside the boundary, and no pair of polygons intersects or touches.

The *free space* free($\mathcal{W}$) is the inside of B_0 minus the union of the n polygons contained in B_0. In other words, the free space is the complement of the union of all polygons in $\mathcal{W}$. We call the free space, together with the set of polygons, the *robot's world* (Figure 1).

We consider path f to be directed curve with natural direction from $f(0)$ to $f(1)$. A *path* f in $\mathcal{W}$ is a continous function

$$f : [0, 1] \rightarrow free(\mathcal{W})$$

with $f(0) \neq f(1)$. The two points $f(0)$ and $f(1)$ are called its *endpoints*, and the path *joins* them. If they are distinct, we usually denote $f(0)$ as a start S and $f(1)$ as a goal G (Figure 2).

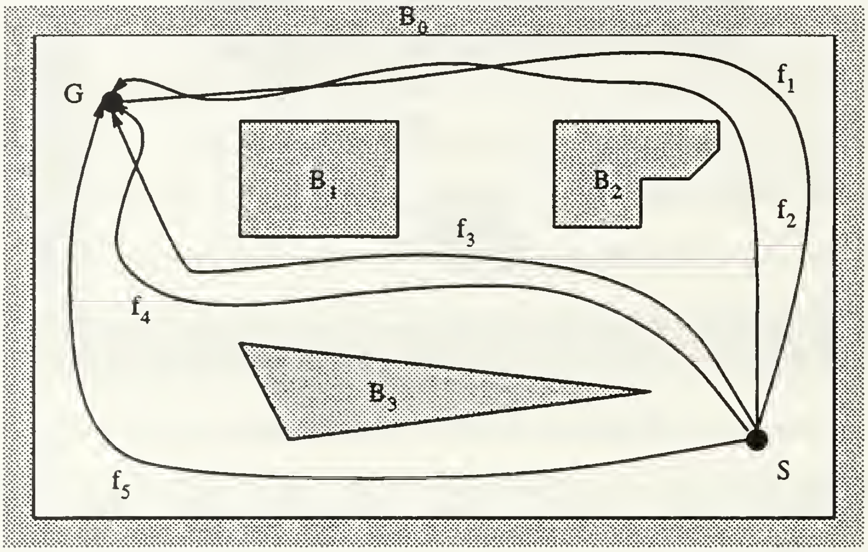


Figure 2. A world and paths

Let q denote the robot's *configuration*. The robot's configuration q is defined by

$$q \equiv (p, \theta, \kappa)$$

where p, θ and κ are its position, orientation, and curvature respectively. The configuration defined in this dissertation is normally used to describe the robot's instantaneous status, either stationary or moving. This configuration is especially useful for specifying a path. For instance, if we use $q = (p, \theta, \kappa)$ to specify a line, this line passes through the point at position p and with orientation θ. When the curvature element is $\kappa = 0$, it is specifying a straight line, otherwise it is a curve.

The motion of the robot is subject to *nonholonomic* kinematic constraints, That is, the robot is able to perform both forward and reverse motion but not sideways motion:

- A finite *curvature* limitation of motion represented by the maximum curvature (κ_{max}) that the vehicle can take.

- A finite *rate of change of curvature* limitation of smooth motion represented by the maximum rate of change of curvature $((\frac{d\kappa}{ds})_{max})$.[1]

2. Problem Description

The purpose of this research is to investigate fundamental theories for navigation to construct an autonomous mobile robots for military and industrial applications. This dissertation is an investigation of one aspect of this goal: the problem of motion planning which allows an autonomous robot to plan its own motion in a known and static two-dimensional environment. Here it is desired to safely navigate an autonomous vehicle through free space using smooth motions.

We consider that the motion planning problem for a rigid body robot must be divided into at least two subproblems: a *global path planning* problem and a *local motion planning* problem. The first is the problem of finding the best path class in terms of homotopy [26]. In that sense, this level is an abstract portion of the whole problem. The second is the problem of finding the best motion when a path class is defined by the first subproblem. We call this method *layered motion planning.*

The problem statements specifically addressed herein are the following:

1. How do we best represent the path class to make local motion planning easier?
2. How do we find a safe local motion planning algorithm?
3. How do we find a robust real-time positional-uncertainty elimination (self-localization) algorithm?

Following theoretical analysis, algorithm design, and simulation, we will implement the resulting algorithms on the autonomous self-contained mobile vehicle Yamabico-11 for testing and evaluation.

---

[1]This limitation is applicable only when we are interested in smooth motion in which the robot is not supposed to stop when moving along a path. If the robot is allowed to stop before maneuvering, then this limitation does not exist and the robot is able to follow any κ_{max}–constrained path so long as there is tangential continuity anywhere on the path.

3. Assumptions

The following assumptions are used throughout this dissertation:

- The world $\mathcal{W}$ is polygonal.
- Although the robot will be operating in a three-dimensional environment, it is assumed that the model reflects the projection of the obstacles onto the plane of the floor on which the robot moves.
- The vehicle and all objects in the robot's world are rigid bodies.
- The obstacles do not intersect or touch each other.
- The robot has complete knowledge of the environment in which it is operating. However, the use of external references to guide its motion other than the physical characteristics of the walls will not be used.
- All obstacles in the environment are stationary.
- All obstacles faces are perpendicular to the plane in which the robot moves. This assumption is required to assure a good sensor return from all objects.

C. PREVIOUS WORK

1. Motion Planning

Several concepts and theories have been developed which may lead to solving the motion planning problem. The "classical" approaches to motion planning can be divided in the following three classes: *roadmap methods*, *cell decomposition methods*, and *potential field methods.* We will briefly introduce these approaches and summarize them below. For a thorough discussion of these approaches see [49, 32].

a. Roadmap and Cell Decomposition Methods

Let $\mathcal{W}$ denote the space of all configurations for the robot, and let free($\mathcal{W}$) be the robot's free configuration space, i.e., the subset of $\mathcal{W}$ in which the robot does not intersect any obstacles. The *roadmap approach* (or *skeleton approach*) consists of capturing the connectivity of free($\mathcal{W}$) in the form of a network of one-dimensional curves, the *roadmap*, lying in free($\mathcal{W}$). After a roadmap ρ has been constructed, the path planning is reduced to connecting the start and goal configurations to ρ, and searching ρ for a path.

The principle of the *cell decomposition approach* is to decompose the robots free configuration space free($\mathcal{W}$) into a collection of non-overlapping regions (cells), whose union is (exactly or approximately) free($\mathcal{W}$). This *cell decomposition* is then used for constructing the *connectivity graph* G which represents the adjacency relation among the constructed cells. Every node in G corresponds to a cell, and two nodes are connected by an edge if and only if their corresponding cells are adjacent. The path planning is then performed by finding a path in G from the node corresponding to the start cell (the cell containing the start configuration) to the node corresponding to the goal cell (the cell containing the goal configuration).

We see that both the roadmap approach as well as the cell decomposition approach consists of constructing a global data structure that can later be used for solving one or more motion planning problems. A strong point of both approaches is that cell-decomposition and roadmap algorithms are typically complete, i.e., whenever a path exists a path will be found. There are two serious drawbacks:

1. The computations of the data structures tend to be very expensive in both time and memory, and

2. They do not seem to be suitable for robots with non-holonomic constraints such as car-like robots or multi-body mobile robots.

b. Potential Field Methods

In the *potential field approach*, no data structure is built. Globally the idea is that the robot (represented by a configuration in configuration space) is treated as a particle under the influence of an *artificial potential field* whose variations are expected to reflect the "structure" of the free configuration space free($\mathcal{W}$). The potential field is typically defined by a function $f : \mathcal{W} \to \mathcal{R}$ that is a weighed sum of an *attractive* potential, pulling the robot towards the goal configuration, and a number of *repulsive* potentials, pushing the robot away from the obstacles. The motion planning is performed by repeatedly computing the most promising direction of motion and moving in this direction by some step size.

A typical problem with potential field methods is that the robot can become stuck in a local minimum of the potential field. That is, the robot reaches a configuration q where the (weighted) sum over all the potentials is equal to the null-vector. Recently, much progress has been made in defining good potential functions with few local minima, and efficient techniques have been developed for escaping from local minima. Currently there exist practical potential-field planners for robots with many degrees of freedom, as well as for some types of non-holonomic robots (see for example [3]). So it seems that the potential-field approach does not have the disadvantages of the former approaches. A major drawback of the potential-field approach, though, is that the concept is unsuitable for learning problems (no start and goal configurations are specified, and the objective is to compute a data-structure, which can later used for queries with arbitrary start and goal configurations), due to the fact that every goal configuration defines a distinct potential field.

c. Other Methods

Several other methods were developed by Lozano-Perez to handle rigid body robots as point robots. The *configuration space approach* is considered as one of global motion planning using the concept of the *vehicle configuration* (x, y, θ) [53]. The idea is to transform the problem of planning the motion of a dimensioned object into the problem of planning the path of a point robot by mapping the obstacles from the physical work space into the configuration space. However, it is known that the computation time for the configuration space approach is larger and also it is difficult to incorporate nonholonomic constraints into the searching algorithm.

Barraquand and Latombe present a method in which the entire configuration space is discrete. A dynamic search in the discrete configuration space uses the number of maneuvers as a cost function is considered. Methods of this type possess conflicts between accuracy and computational costs [2].

Laumond extended the basic motion planning problem defined by Latombe [49] to the case of a point robot with kinematic constraints. He developed a method to break down the planning problem into two phases. In the first phase, the problem is solved by finding a collision-free path while ignoring the orientations of robot's start and goal configurations. Then, in second phase, the path is transformed into a topologically equivalent collision-free path using arcs and tangent line segments. The number of reversals in the path is not limited and the path involving reversals is not smooth [50].

A closely related research direction is to develop algorithms for motion planning using the border concept [47, 9]. Drawbacks of the border approach are several:

- This concept is unsuitable if the shape of the regions is not always simple (as in non convex region).
- The decomposition is not unique.
- The optimum number of borders is still a question.
- This task becomes unduely complex for dynamic environments.

The other global motion planning and local motion planning ideas can be found in other research reports. Some of these focus on motion planning for manipulators [33, 42] and others provide general ideas [23, 32, 65].

2. Self-Localization

Several approaches have been developed relating to robust and precise navigation for an autonomous mobile vehicle using model-sonar based navigation. We will briefly introduce these approaches and summarize them below.

In [14], a method for reducing uncertainty using sonar data interpretation and Kalman filtering is proposed. Line fitting with the sonar data is used.

A technique to estimate the positional and orientational errors and a method to reset them is described in [66].

The problem of landmark tracking over sequences of stereo image pairs is studied in [56, 48]. Both approaches develop multivariate Gaussian error models for the triangulation errors occurring when depth is inferred from stereo images. Kalman filters are used to reduce the uncertainty in the vehicle position as well as in the position of the observed objects.

Use of an Approximate Transformation (AT) framework for robot localization with sonar data is described in [18]. Fifteen ultrasonic range finding transducers arranged in a circular array are used to build dense two-dimensional maps based upon empty and occupied volumes in a cone in front of the sensor.

Rule–based matching of line segments which are extracted from sonar data with precompiled line models of indoor environments is suggested in [16].

In [12], a fast, robust matching algorithm which determines the congruence between range data points (derived from an infrared range-finder) and a two-dimensional map of its environment is investigated.

The localization system of a free-navigating mobile robot is described in [30]. The absolute position and orientation of the vehicle by matching verticle plannar surfaces extracted from a 3D-laser-range-image with corresponding surfaces predicted from a 3D-environmental model are determined. Continuous localization is achieved by fusing single-image localization and dead-reckoning data by means of a statistical uncertainty evolution technique.

The robot "RAMUS" uses an *a priori* map of the environment for mobile robot localization [29]. This environment is cluttered with unknown obstacles and an environmental model is built from ultrasonic readings using clustering to discard false echos.

In [10], a robot automatically maps an office building environment and then smoothly navigates through this environment at a speed of 78 cm per second.

D. ORGANIZATION OF DISSERTATION

The dissertation is organized as follows.

Chapter II discusses the approach used in this dissertation and contrasts it with previous work in the field of autonomous mobile robot motion planning.

Chapter III presents definitions and concepts of polygons and subpolygons. Also, it describes the algorithm of determing image type of any point in a free space on a convex polygon.

Chapter IV describes the theory of a free-space decomposition using Voronoi diagrams. It presents a method to symbolically represent the path classes using a polygonal world.

Chapter V describes how to track any polygon. It describes the algorithm for polygon tracking. It reports the results as implemented on the simulator.

Chapter VI discusses local motion planning in detail. It presents the analysis of the local motion planning tools to be used in this dissertation. It gives a description of the algorithm for planning the robot's motion. It reports the results as implemented on the simulator.

Chapter VII presents the theory of self-localization. It introduces an algorithm for robot odometry correction.

Chapter VIII reports the results of local motion planning algorithm as implemented on an autonomous mobile robot system Yamabico-11 and discusses the implications and consequences of the results. Also, it gives a detailed explanation of an experimental results of applying positional uncertainty elimination in real time using Yamabico-11.

Chapter IX introduces the hardware of the Naval Postgraduate School autonomous mobile robot Yamabico-11. It describes the design of a robotic software system – Model-Based Mobile robot Language (MML).

Chapter X describes recommendations for future research.

Chapter XI summarizes the major contributions of this dissertation.

Appendix A provides a normalization definition.

Appendix B introduces a least square linear fitting method.

II. LAYERED MOTION PLANNING

A. INTRODUCTION

Motion planning is one of the most important areas of robotics research. The complexity of the motion-planning problem has hindered the development of practical algorithms. Not all robotic systems plan the robot's motion in a deliberate fashion. In fact, there exists a wide variety of motion planners including: no plan/no model, a flexible plan, and a rigid, unalterable plan. Many different methods have been developed for motion planning. These methods are variations of a few general approaches: road map, cell decomposition, potential field and mathematical programming [49, 32]. Some of them is widely applicable, whereas others solve only a narrow range of motion planning problems. Unfortunately, none of them is complete in the sense of practical applicability for solving the motion planning problem defined in this dissertation. For example, the robot's motion in the area of the start or goal configuration is more restricted and requires more deliberative planning. Not all robotics systems proposed for motion planning are developed to address this consideration. Also, nonholonomic constraints and kinematic constraints have not taken into consideration in many approaches. Furthermore, most research in motion planning, although theoretically valuable, is not practically useful. For these reasons, we propose a new approach where the motion planning problem for a rigid body robot is attacked through a method called *layered motion planning*. The layered motion planning problem uses *global path planning* and *local motion planning* to solve the original motion planning problem. As the layered motion planning is divided into two parts, the first one (path class determination) is solved by the global path planner, while the second part (path class navigation) is handled by the local motion planning. The global path planner finds the optimal path class in terms of homotopy [26]. In that sense, this level is an abstract portion of the whole problem. The second is the problem of finding the optimal motion when a global path plan is defined by the first planner. Figure 3 shows the layered motion planning structure.

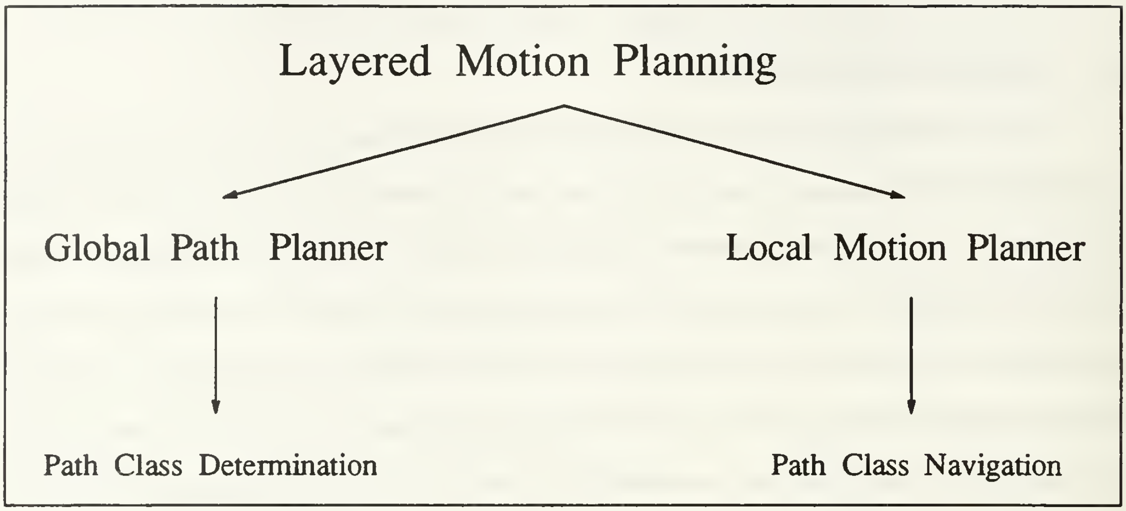


Figure 3. Layered motion planning structure

B. MOTION PLANNER STRUCTURE

The motion-planner structure of the system provides the framework in which each of the above parts interact. Figure 4 provides a depiction of the structure of the motion planner used in Yamabico-11. The motion planner has a layered structure. It consists of a mission planner, a global path planner, a local motion planner and a self-localization module.

1. Mission Planner

The highest level in the framework is mission planner. The mission planner uses knowledge-based inference engines to convert abstract goals into geometric goals and mobility constraints. In this level, high levels of abstraction and long-term memory are used. This level is not a focus of this dissertation.

2. World Model

The world model contains information used by the global path planner and local motion planner. This information is used by the global path planner in constructing a global path plan. Also, lower levels (local motion planner) use that in-

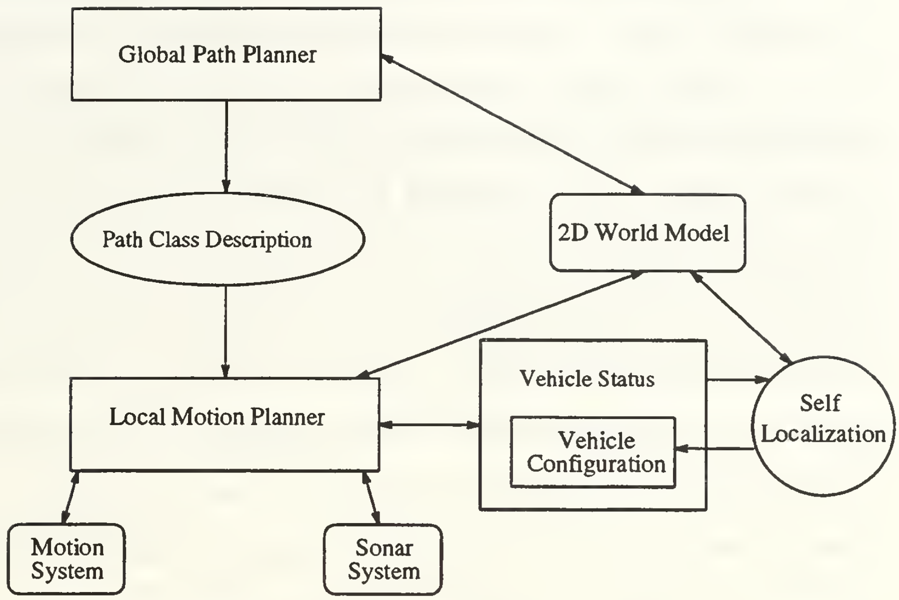


Figure 4. Motion planner/execution architecture

formation to carry out the global path plan. This information serves as a basis for real-time decision process of the local motion planner.

3. Global Path Planner

The global path planner is related to the most abstract aspect of the motion planning problem in robotics, i.e., the connectivity of geometrical objects. It uses the idea of the *Voronoi diagram* to represent the path class. It starts with decomposition of the free space of the given polygonal world. Then a *connectivity graph* is built and searched to determine the required *path class.* This path class represented by a sequence of left and right polygons called a *directed v–edges sequence,* Ξ, which specifies the direction of a possible path for the robot. This path class plays an important role in local motion planning. The details of global path planning will be discussed in Chapter IV.

4. Local Motion Planner

The local motion planner is responsible for following the global path as closely as possible without violating any kinematic, dynamic, or holonomic constraints. So, the task of the local motion planner is to produce a smooth collision-free motion for the robot. The local motion planner is responsible for the following: selecting and initiating a steering function control rule, executing the resultant motion, and monitoring to ensure that the plan is proceeding. The steering function and the principle of the left and right images of the given path class are powerful notions used to find solutions in this layer. This method was implemented first on a simulator, then on the autonomous mobile robot Yamabico-11. This problem is very important in this dissertation because self-localization is executed while robot moving. The local motion planning will be discussed more deeply in Chapters V, VI.

5. Self-Localization Module

A mobile robot can be assisted in its navigation tasks by providing it with *a priori* knowledge about the environment in which it will navigate, usually called a *world model* or a *map*. One of the issues to be addressed in using a stored model as an aid in mobile robot navigation is that of estimating the position and orientation of the robot with respect to the model. Once the robot accurately estimates its location within the model, other navigation tasks can be performed. Most mobile robots are equipped with A key capability of an autonomous mobile robot operating in an indoor environment is *localization*, i.e. determination of its current position and orientation. The usual method for position estimation of a wheeled autonomous mobile robot is *odometry* or *dead reckoning*. However, it has a problem of gradual error accumulation when the robot moves long distances. Unlike the errors in robot manipulators, this problem is crucial in navigation because vehicles' localization errors determined by odometry may be increased indefinitely until the vehicle is not able to move safety. We assume that the vehicle

1. has a geometric model of the static portions of an indoor world,

2. possesses a dead-reckoning capability,
3. executes model-based navigation through these two capabilities, and
4. has sonic sensors.

So, the purpose of the *self-localization* is to find a robust algorithm so that the vehicle can continually eliminate its positional uncertainty while execting missions. Through this method, the robot can minimize its positional uncertainty, can make safe and reliable motions, and can perform useful tasks in a partially-known world. Thus, self-localization is an essential component of model-based navigation for indoor applications. Self-localization will be discussed in detail in Chapter VII.

C. METHODOLOGY

Summarizing, the approach taken in this dissertation will provide a unified approach to the motion planning problem for autonomous vehicles using proximity. It includes descriptions of the following:

1. An image type of a point in free space on a convex polygon algorithm (Chapter III).
2. A path class representation using polygonal world and Voronoi diagram (Chapter IV).
3. A safe local motion planning algorithm (Chapter V, VI).
4. A robust real-time positional-uncertainty elimination (self-localization) algorithm (Chapter VII).

After theoretical analysis, algorithm design, and simulation, we implement the resulting algorithms on the autonomous mobile vehicle Yamabico-11 for testing and evaluation.

III. POLYGONS, SUBPOLYGONS AND IMAGES

Before discussing motion planning, we need to give precise meaning to the concepts that provide the basis for this dissertation. This chapter presents definitions and concepts associated with polygons and subpolygons. Afterwards, a discussion of an algorithm which finds an image of a point in free space on polygon is presented. We will restrict the discussion in this chapter to the Euclidean Plane E^2.

A. GENERAL DEFINITIONS

A *point* p is represented as a pair of coordinates (x, y) in E^2. Given two distinct points p_1 and p_2 in E^2, the linear combination

$$\alpha p_1 + (1-\alpha)p_2 \qquad \alpha \in \mathcal{R}$$

is a *line* in E^2 where $\mathcal{R}$ is the set of *real numbers*. If, in the expression $\alpha p_1 + (1-\alpha)p_2$, we add the condition $0 \leq \alpha \leq 1$, we obtain the *convex combination* of p_1 and p_2, i.e.,

$$\alpha p_1 + (1-\alpha)p_2 \qquad (\alpha \in \mathcal{R}, \quad 0 \leq \alpha \leq 1).$$

This convex combination describes a *line segment* joining the two points p_1 and p_2 [59]. Normally this segment is denoted as $\overline{p_1 p_2}$.

A *topology* [67] on a set S is a collection T of subsets of S (called open sets) having the following properties:

1. The empty set and set S are in T.
2. The union of elements in an arbitrary subcollection of T is in T.
3. The intersection of elements in a finite subcollection of T is in T.

Definition: A *metric* (or *distance function*) [67] on a set S is a function d: $S \times S \rightarrow \mathcal{R}$ that satisfies the following conditions:

1. *Positive definiteness*: $d(x, y) > 0$ for all $x, y \in S$, and $d(x, y) = 0$ if and only if $x = y$.

2. *Symmetry*: $d(x, y) = d(y, x)$ for all $x, y \in S$.

3. *Triangle inequality*: $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in S$.

A *metric space* (S, d) is a set S together with a metric on it. If there is no ambiguity, the metric space can be referred to simply as S. A space S is *connected* if it is not the union of two disjoint, nonempty open sets. Intuitively, this means that S can best be viewed as "one piece", and is in some sense indecomposible. A related idea, and one which is more suitable to our purpose is that of *path connectedness* [25, 60].

Let x_0 and $x_1 \in S$. A *path* f in S from x_0 to x_1 is a continuous function

$$f : [0,1] \longrightarrow S$$

such that $f(0) = x_0$ and $f(1) = x_1$. We say that S is *path connected* if for every pair of points x_0 and x_1 in S, there exists a path between them. Additionally, if a space is path connected, then it is also connected [25, 60].

Two characterizations of sets which are needed for later definitions are whether a set is open or closed, and whether a set is bounded or unbounded. A set is *closed* if and only if it contains its boundary (in other words, if and only if it contains all its limit points). Additionally, the complement of a closed set is *open*, which implies that a set is open if and only if it contains none of its boundary points. Since, a set may contain only a portion of its boundary, it may be neither open nor closed. We give the definition of a bounded set by using the intuitive notion of distance. A set is *bounded* if the distance between any two of its members is finite. A set that is not bounded is said to be *unbounded* [43, 60].

Finally, we introduce the concept of a *hole*. The Jordan Curve Theorem states that a simple closed curve J in the Euclidean plane separates the plane into two open connected sets with J as their common boundary. Exactly one of these open connected sets (the "inner region") is bounded [13]. We define a hole to be one of the open connected sets. We say that the hole is *ccw* if it is bounded, and *cw* if it is unbounded. Sometimes it may be useful to consider the hole along with its boundary, but generally we refer to them separately.

B. POLYGON

Given $n \geq 3$ points $v_1, \cdots, v_n$ in the plane, in a certain order, we obtain a *n-sided polygon* or *n-gon* by connecting each point to the next, and the last to the first, with a line segment. The points v_i are the *vertices* and the segments $\overline{v_i v_{i+1}}$ are the *sides* or *edges* of the polygon. Therefore, polygon, B, is defined as:

$$B = \{v_1, v_2, \cdots, v_n\}, \quad n \geq 3$$

When $n = 3$ we have a *triangle*, when $n = 4$ we have a *quadrangle* or *quadrilateral*, and so on [67]. A polygon is represented as a sequence rather than a set of points because the order in which the points are given is very important. Changing the order, even without changing the points themselves, may result in a different polygon. In this dissertation, we will follow a convention that a vertex with the minimum x-coordinate among all the vertices is taken as the first vertex in B. If there is more than one vertex which has the same x-coordinate, we take the one with the least y-coordinate as the first one among them.

Now, how we define how to determine the next or previous vertex from the current one.

Definition: A *simple* polygon [67] is one whose corresponding path does not intersect itself; this means that

1. no consecutive edges are on the same line, in other words, any three consecutive points in the sequence are not colinear and
2. no two edges intersect (except that consecutive edges intersect at the common vertex).

For example, Figure 5a and 5b show two simple quadrilaterals while 5c is not simple. Another example is shown in Figure 6. We will treat only simple polygons.

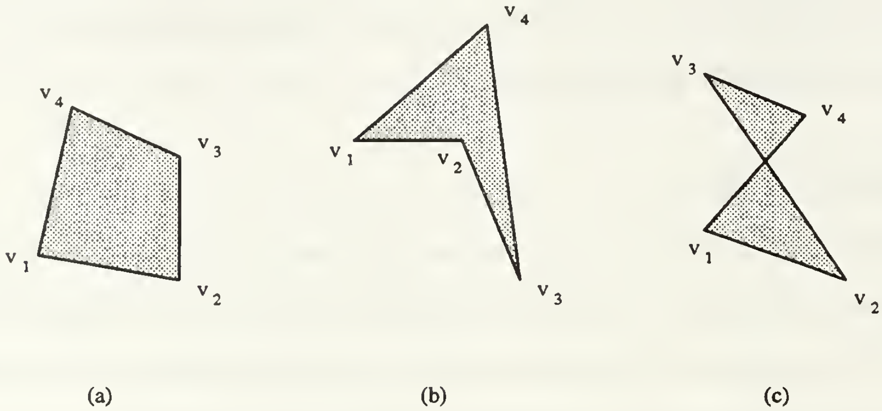


Figure 5. Simple and non-simple polygon (I)

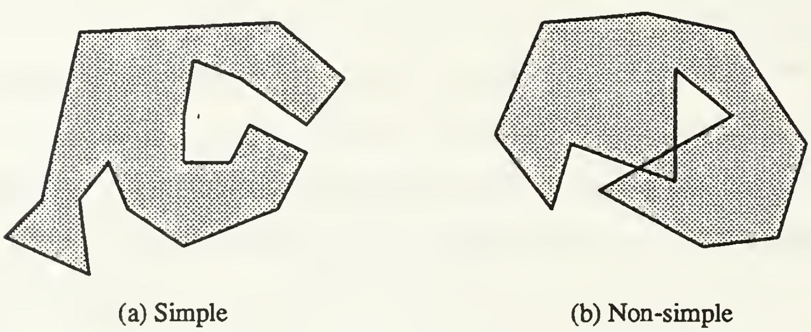


Figure 6. Simple and non-simple polygon (II)

Definition: The *next* function $\varphi : B \to B$ is defined as:

$$\varphi(v_i) = \begin{cases} v_{i+1} & \text{if } \; 1 \leq i \leq n-1 \\ v_1 & \text{if } \; i = n \end{cases} \tag{III.1}$$

The meaning of $\varphi(v)$ is the "next vertex" of v in B. For example, in Figure 5a, the next of v_1 is v_2 and the next of v_4 is v_1.

Proposition III.1 *The function $\varphi : B \to B$ is a one-to-one corresponding or a bijection.*

Proof. The function φ is one-to-one and onto. It is one-to-one since the function takes on distinct values. It is onto since all elements of the codomain are images of elements in the domain. Hence, φ is a bijection. □

Proposition III.2 *: Let the function φ be a one-to-one corresponding from the set B to the set B. The inverse function $\varphi^{-1} : B \longrightarrow B$ exists and is a one-to-one corresponding also.*

Proof. If a function φ is not a one-to-one corresponding, we cannot define an inverse function of φ. When φ is not a one-to-one corresponding, either it is not one-to-one or it is not onto. If φ is not one-to-one, some element v_j in the codomain is the image of more than one element in the domain. If φ is not onto, for some element v_j in the codomain, no element v_i in the domain exists for which $\varphi(v_i) = v_j$. Consequently, if φ is not a one-to-one corresponding, we cannot assign to each element v_j in the codomain a unique element v_i in the domain such that $\varphi(v_i) = v_j$ (because for some v_j there is either more than one such v_i or no such v_i). □

The meaning of φ^{-1} is the "previous vertex" of v. For example, in Figure 5 - part (a), the previous vertex of v_1 is v_4.

When we refer to the *angle* at a vertex v_i we have in mind the *interior* angle. We denote this angle by β_i. In any *n-gon, the sum of the interior angles equals* $2(n-2)\times 90°$; for example, the sum of the angles of a triangle is 180°. The complement of v_i is the *exterior* angle at that vertex. We denote this angle by δ_i (see Figure 8). Let $\Psi(v_i, \varphi(v_i))$ represent the direction from v_i to $\varphi(v_i)$.

Definition: Given two distinct points, $p_1 \equiv (x_1, y_1)$ and $p_2 \equiv (x_2, y_2)$ (Figure 7). we define a direction function $\Psi(p_1, p_2)$ as

$$\Psi(p_1, p_2) \equiv atan2(y_2 - y_1,\ x_2 - x_1)$$

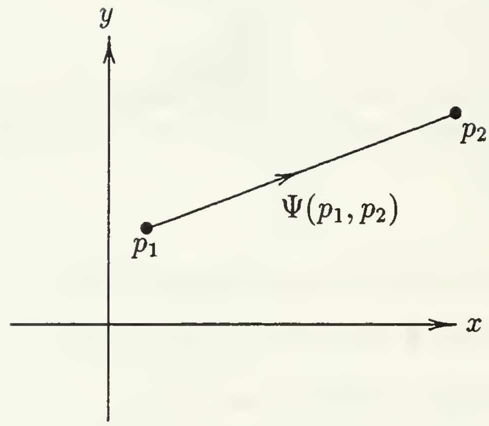


Figure 7. Direction between Two Points

The *exterior* angle, δ_i, at v_i is the angle between one side and the extension of the adjacent side related to v_i [67] (see Figure 8).

$$\delta_i = \Phi\left(\Psi(v_i, \varphi(v_i)) - \Psi(\varphi^{-1}(v_i), v_i)\right)$$

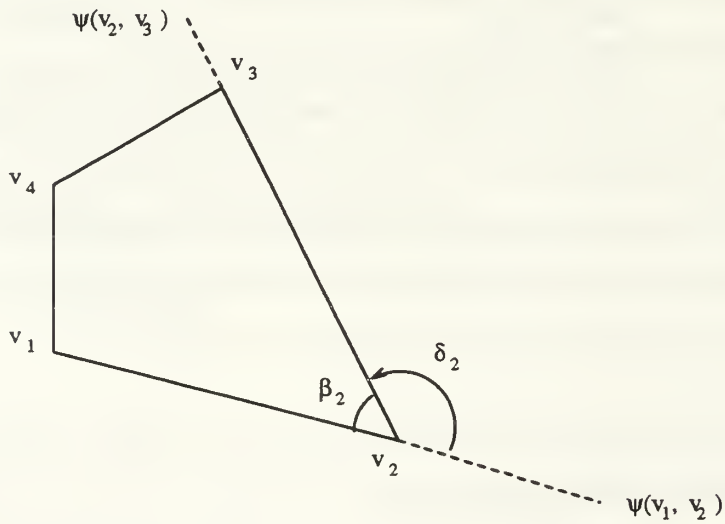
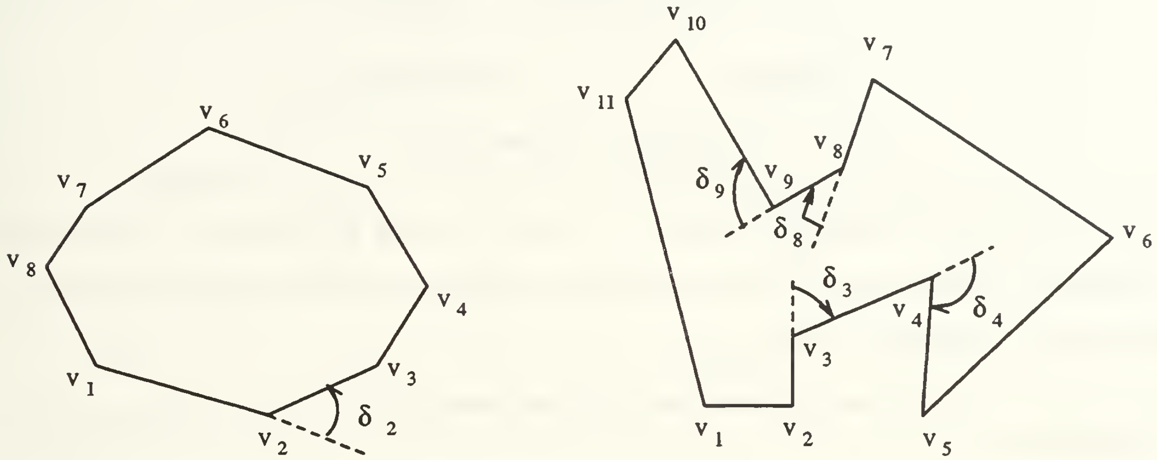


Figure 8. Interior and exterior angle of a simple polygon

Note that the difference between the directions is *normalized* to fall within $[-\pi, \pi]$. (the function Φ is defined in "APPENDIX. NORMALIZING ANGLES").

Definition: A vertex v_i on a simple polygon is said to be a *convex* vertex if $\delta_i > 0$. If $\delta_i < 0$, A vertex v_i is said to be *concave* vertex.



(a) Convex Simple Polygon (b) Concave Simple Polygon

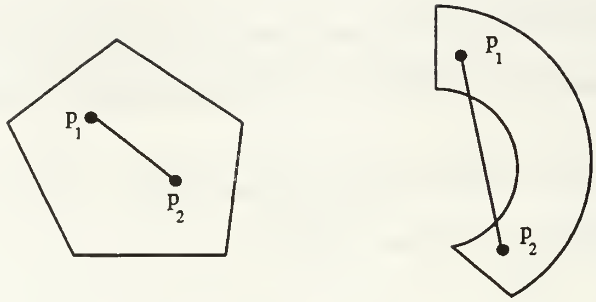
Figure 9. Convex and concave simple polygons

For example, in Figure 9, in part (a), the vertex v_2 is convex because $\delta_2 > 0$. In part (b), the vertex v_3 is concave because $\delta_2 < 0$.

Definition: A domain D in E^2 is *convex* if , for any two points p_1 and p_2 in D, the segment $\overline{p_1 p_2}$ is entirely contained in D (Figure 10(a)). It can be shown that the intersection of convex domains is a convex domain.

Definition: A simple polygon is a *convex* polygon if all of its vertices are convex (Figures 9(a), 10(a)), otherwise it is *nonconvex* polygon (Figures 9(b), 10(b)).

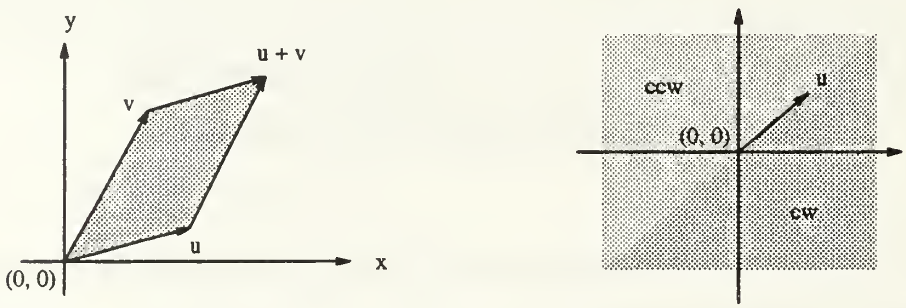
Now, how we can represent any convex or nonconvex polygon. Before doing this, we will define three important predicates, *ccw* (counterclockwise), *cw* (clockwise)



(a) Convex (b) Not convex

Figure 10. Convex set

and *col* (colinear). Consider vectors $\vec{u} = (x_1, y_1)^T$ and $\vec{v} = (x_2, y_2)^T$, shown in Figure 11(a). The *cross product* $\vec{u} \times \vec{v}$ can be interpreted as the signed area of the parallelogram formed by the points $(0,0), u, v$, and $u + v = (x_1 + x_2, y_1 + y_2)$. An equivalent, but more useful, definition gives the cross product as the determinant of a matrix.[1]



(a) (b)

Figure 11. Cross product of vectors

$$\begin{aligned} \vec{u} \times \vec{v} &= \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -\vec{v} \times \vec{u} \end{aligned} \tag{III.2}$$

[1]Actually, the cross product is a three-dimensional concept. It is a vector that is perpendicular to both $\vec{u}$ and $\vec{v}$ according to the "right-hand rule" and whose magnitude is $|x_1 y_2 - x_2 y_1|$. Here, however, it will prove convenient to treat the cross product simply as the value $x_1 y_2 - x_2 y_1$.

If $\vec{\mathbf{u}} \times \vec{\mathbf{v}}$ is positive, then $\vec{\mathbf{u}}$ is clockwise from $\vec{\mathbf{v}}$ with respect to the origin (0, 0); if this cross product is negative, then $\vec{\mathbf{u}}$ is counterclockwise from $\vec{\mathbf{v}}$. Figure 11(b) shows the clockwise and counterclockwise regions relative to a vector $\vec{\mathbf{u}}$. A boundary condition arises if the cross product is zero; in this case, the vectors are collinear, pointing in either the same or opposite directions [11].

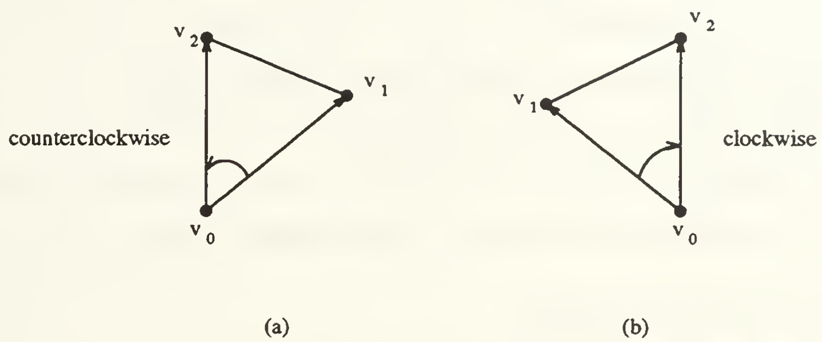


Figure 12. Using the cross product to determine how consecutive line segments $\overline{v_0v_1}$ and $\overline{v_1v_2}$ turn at a point v_1

To determine whether a directed segment $\overline{v_0v_2}$ is clockwise or counterclockwise from a directed segment $\overline{v_0v_1}$ with respect to their common endpoint v_0, we simply translate to use v_0 as the origin (see Figure 12). That is, we let $\mathbf{v_1} - \mathbf{v_0}$ denote the vector $\vec{\mathbf{u}}' = (x'_1, y'_1)$, where $x'_1 = x_1 - x_0$ and $y'_1 = y_1 - y_0$, and we define $\mathbf{v_2} - \mathbf{v_0}$ similarly. We then compute the cross product

$$(\mathbf{v_2} - \mathbf{v_0}) \times (\mathbf{v_1} - \mathbf{v_0}) = (x_2 - x_0)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_0)$$

If the sign of this cross product is negative, then $\overline{v_0v_2}$ is counterclockwise from $\overline{v_0v_1}$; if positive, it is clockwise. The above discussion is very useful for all results related to the area of the polygon.

The *area* of a polygon whose vertices v_i have coordinates (x_i, y_i), for $1 \leq i \leq n$, is the "signed" value of

$$\begin{aligned} area(B) &= \frac{1}{2}(x_1y_2 - x_2y_1) + \cdots + \frac{1}{2}(x_{n-1}y_n - x_ny_{n-1}) + \frac{1}{2}(x_ny_1 - x_1y_n), \\ &= \frac{1}{2}\sum_{i=1}^{n}(x_iy_{i+1} - x_{i+1}y_i), \end{aligned}$$

where in the summation we take $x_{i+1} = x_1$ and $y_{i+1} = y_1$. In particular, for a triangle $B = \{v_1, v_2, v_3\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, let vectors $\vec{\mathbf{u}} = (x_1, y_1)^T$, $\vec{\mathbf{v}} = (x_2, y_2)^T$ and $\vec{\mathbf{w}} = (x_3, y_3)^T$. the "signed" area is defined as

$$\begin{aligned}
\triangle &= \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad \text{(III.3)} \\
&= \frac{1}{2}(x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + x_3y_1 - x_1y_3) \\
&= \frac{1}{2}(\vec{\mathbf{u}} \times \vec{\mathbf{v}} + \vec{\mathbf{v}} \times \vec{\mathbf{w}} + \vec{\mathbf{w}} \times \vec{\mathbf{u}}) \\
&= \frac{1}{2}[(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)]
\end{aligned}$$

Proposition III.3 *For any triangle B,*
(I) If $\triangle > 0$, B is ccw and area of B is equal to $\triangle$.
(II) If $\triangle < 0$, B is cw and area of B is equal to $|\triangle|$.
(III) If $\triangle = 0$, B is col and area of $B = 0$.

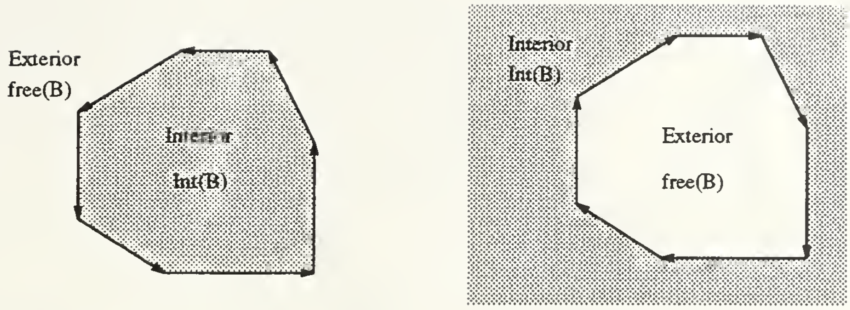
Proof. By using Eq. III.2,

$$\begin{aligned}
\triangle &= \frac{1}{2}(\vec{\mathbf{u}} \times \vec{\mathbf{v}} + \vec{\mathbf{v}} \times \vec{\mathbf{w}} + \vec{\mathbf{w}} \times \vec{\mathbf{u}}) \\
&= \frac{1}{2}\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \begin{vmatrix} x_3 & x_1 \\ y_3 & y_1 \end{vmatrix}\right)
\end{aligned}$$

The sign of $\triangle$ gives us the result. □

Definition: A convex polygon is a polygon whose ordered list of vertices produces a counterclockwise (*ccw*) boundary loop. An nonconvex polygon is a polygon whose ordered list of vertices produces a clockwise (*cw*) directed boundary loop (see Figure 13).

A simple polygon partitions the plane into two disjoint regions, the *interior* (bounded) and the *exterior* (unbounded) that are separated by the polygon (*Jordan curve theorem)* [13].



(a) ccw Polygon (b) cw Polygon

Figure 13. Interior and exterior of a simple polygon

Definition: The set of points in the plane enclosed by a simple polygon forms the *interior* of the polygon, denoted int(B), the set of points on the polygon itself forms its *boundary* , denoted B, and the set of points surrounding the polygon forms its *exterior*, denoted free(B) (see Figure 13).

Therefore, int(B) is defined as the set of points to the left of the boundary and free(B) is defined as the set of points to the right of the boundary. We classify each simple polygon into one of two kinds, *ccw* or *cw*, depending on how its free side defined:

1. for a *ccw* polygon, free(B) is defined as its exterior, and
2. for a *cw* polygon, free(B) is defined as its interior.

Definition: The *convex hull* of a set of points S in E^2 is the boundary of the smallest convex domain in E^2 containing S [59].

C. SUBPOLYGONS

Let $B = \{v_1, v_2, \cdots, v_n\}$, $n \geq 3$ be a polygon. It is desired to decompose B into smaller pieces, called subpolygons. If the polygon is *convex*, i.e., if all the vertices are convex (see Figure 9(a)), we stipulate that the polygon B itself is a unique subpolygon in B. If B is *nonconvex* (see Figure 14), i.e., if there is at least one concave vertex in B, the polygon can be broken up into one or more subpolygons. In that case,

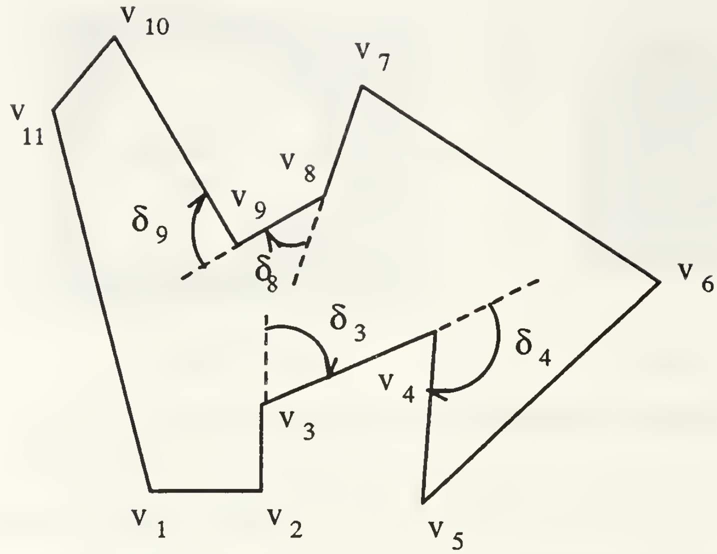


Figure 14. Concave polygon

the first vertex in the subsequence of vertices defining a subpolygon is a concave vertex. The subsequence continues until it encounters another concave vertex, which become the last vertex in the subpolygon's defining subsequence. For example, in Figure 14, v_3 is the first concave vertex ($\delta_3 < 0$) and v_4 is the last concave vertex in the this subsequence. Figure 15 shows the decomposition of the conacave polygon B in Figure 14. Note that B (Figure 14), which is a nonconvex polygon, consists of four subpolygons $\Upsilon_1, \Upsilon_2, \Upsilon_3$ and Υ_4.

Definition: A subsequence

$$\Upsilon = \{v_j, v_{j+1}, \cdots, v_{k-1}, v_k\}, \quad j \leq k$$

of

$$B = \{v_1, v_2, \cdots, v_n\}, \quad n \geq 3$$

is said to be a *subpolygon* of B, if Υ satsifies the following conditions:

1. v_j and v_k are concave, and
2. all the verices $v_{j+1}, \cdots, v_{k-1}$ are convex.

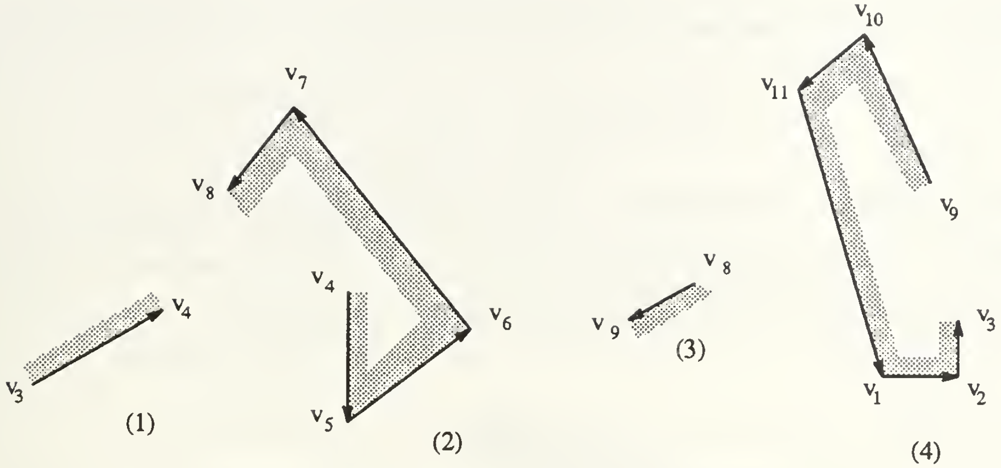


Figure 15. Subpolygons decomposition of concave polygon

where v_j and v_k are said to be the *end-vertices* of the subpolygon Υ.

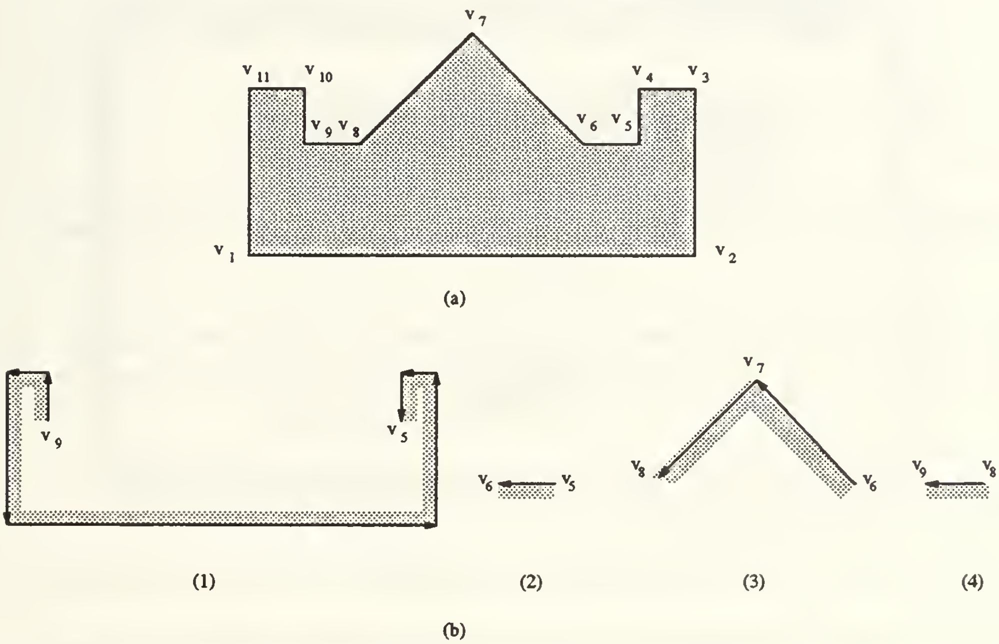


Figure 16. Concave polygon and its subpolygons (I)

Figure 16 is another example of decomposition of a polygon into subpolygons. The *end-vertices* of Υ are disconnected except in the case where the subpolygon consists of only two vertices (see Figure 15). There is a special case where there is only one concave vertex v_i in B. In this case,

$$\Upsilon = \{v_i, v_{i+1}, \cdots, v_i\}$$

is the unique subpolygon. In other word, the nonconvex polygon B consists of only one subpolygon Υ. For example, in Figure 17, vertex v_3 is the only concave vertex in B where the polygon B consists of only one subpolygon Υ ($B \equiv \Upsilon$).

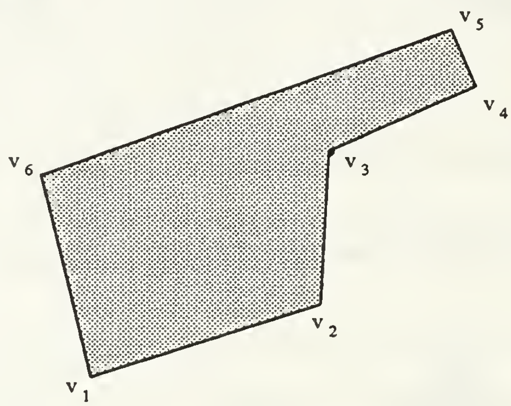


Figure 17. Concave polygon and its subpolygons (II)

The following lemma is the result of the previous discussion of subpolygons.

Lemma III.1 *Any nonconvex polygon B is uniquely divided into a finite number of subpolygons ($\Upsilon_1, \Upsilon_2, \cdots, \Upsilon_n$) in keeping with the order of occurrences of vertices in B. Each convex vertex in B belongs to one and only one subpolygon.*

D. THE ROBOT'S SPACE

We use polygonal models to represent the vehicle's $2D$ world $\mathcal{W}$. Polygons are considered to be *holes* or *obstacles* for robots in this world. We assume that a world $\mathcal{W}$ is encircled by an outermost polygonal boundary (*cw* polygon) and has n polygonal obstacles inside the boundary (*ccw* polygons).

Definition: A *world* $\mathcal{W}$ is a finite set

$$\mathcal{W} = \{B_0, B_1, \cdots, B_n\}, \quad n > 0$$

of polygons which satisfies the following conditions:

1. B_0 is (*cw* polygon),
2. $B_1, \cdots B_n$ are *ccw* polygons, and
3. for any i, j with $0 \leq i < j \leq n$,

$$free(B_i)^c \bigcap free(B_j)^c = \emptyset,$$

where S^c denotes the complement of a set S.

A robot can work only in the *free space* free($\mathcal{W}$) of this world. The free space of $\mathcal{W}$ is the inside of B_0 minus the union of the other n polygons' inside. In other words, the free space is the complement of the union of all the polygons. We call the free space, together with the set of polygons, the *robot's world* (Figure 18).

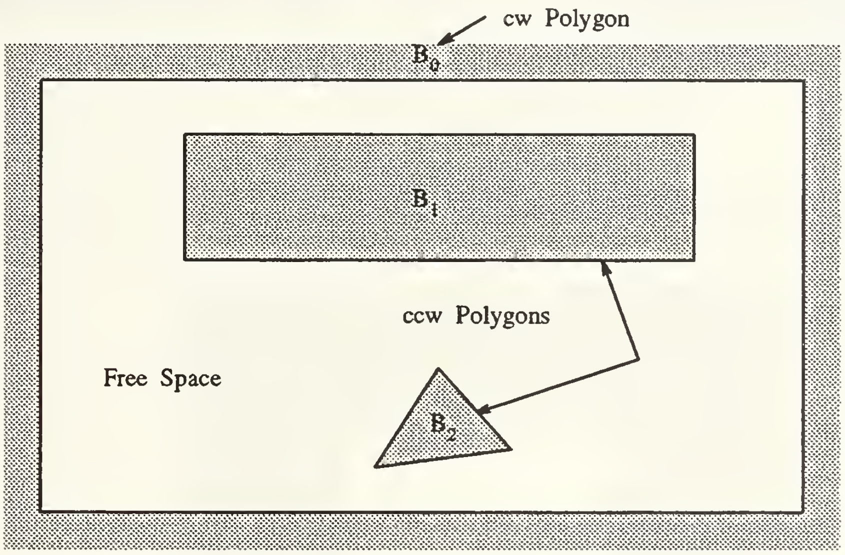


Figure 18. Robot's world space

Definition: In a given world W, the free space and interior of W are defined as follow:

$$\begin{aligned} free(\mathcal{W}) &= \bigcap_{i=0}^{n} free(B_i) \\ &= \mathcal{R}^2 - \mathcal{W} \\ int(\mathcal{W}) &= \bigcup_{i=0}^{n} int(B_i) \end{aligned}$$

Furthermore, we consider the boundary of a polygon to be directed curve which when traversed, puts the polygon to the left. This directed boundary naturally defines the neighbors of a vertex to be the *next vertex*, and the *previous vertex*.

E. IMAGES

We assume a global two-dimensional Cartesion coordinate system in a plane E^2. Given two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in E^2, The Euclidean distance $d(p_1, p_2)$ between them is defined as:

$$d(p_1, p_2) \equiv \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{III.4}$$

Assume that there is an object o in a plane. An object might be a point, a line, an open line segment, a polygon, or other set of points. The shortest distance $d(p, o)$ between a point p and an object o is defined as follows:

$$d(p, o) \equiv \min_{p_1 \in o} d(p, p_1) \tag{III.5}$$

Eq. III.5 generalizes the function d defined by Eq. III.4.

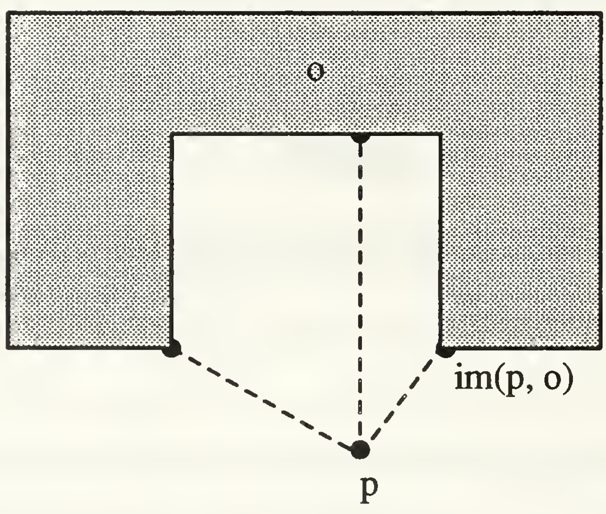


Figure 19. Image on object

Definition: A point p_1 in o which satisfies $d(p, p_1) = d(p, o)$ is said to be an *image* of p on o and is denoted by $im(p, o)$ (Figure 19).

If a world $\mathcal{W}$ has more than one object, an image $im(p, \mathcal{W})$ is defined as the image $im(p, o_i)$ such that $d(p, o_i)$ is the minimum over all objects in $\mathcal{W}$ (Figure 20).

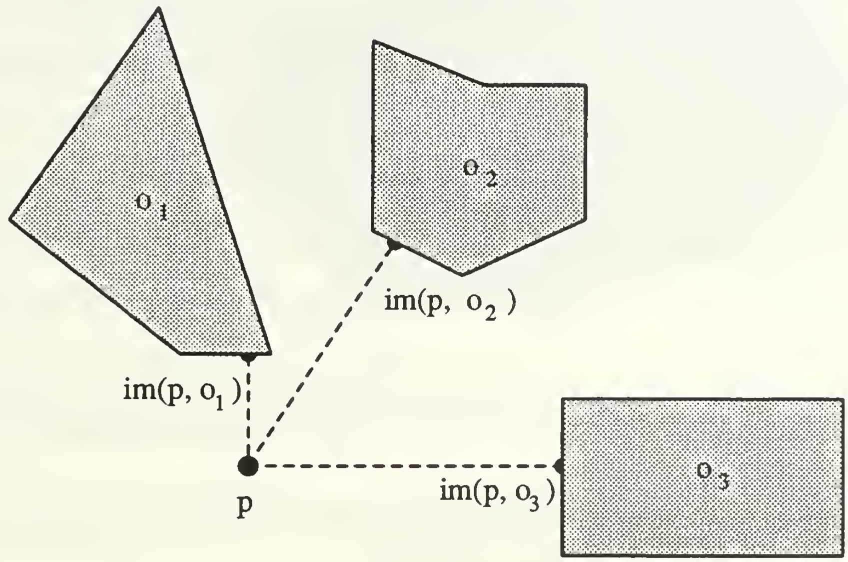


Figure 20. Images on world

Suppose that a vehicle's position in the free space is known. It has its left and right images on the obstacles (polygons). The image may be on an edge or on a vertex of a convex polygon. We shall try to solve the following problem: given a point p in free space and a convex polygon B, determine whether the image from p to B is on an edge or on a vertex of B. In the following subsections, we describe our solution to this problem.

1. Visibility from Point to Polygon

Assume that we are given a convex polygon $B = (v_1, \cdots, v_n)$ and a point $p \in$ free(B). The significant notion for our purpose is the following classification of each vertex v_i of B with respect to the segment $\overline{pv_i}$. Each vertex of B is said to be *visible*, *invisible*, *cw-tangential*, or *ccw-tangential* (we should add with respect to segment $\overline{pv_i}$, but we shall normally imply this qualification) (see Figure 21).

Definition: Let B be a convex polygon, and let a point $p \in$ free(B).

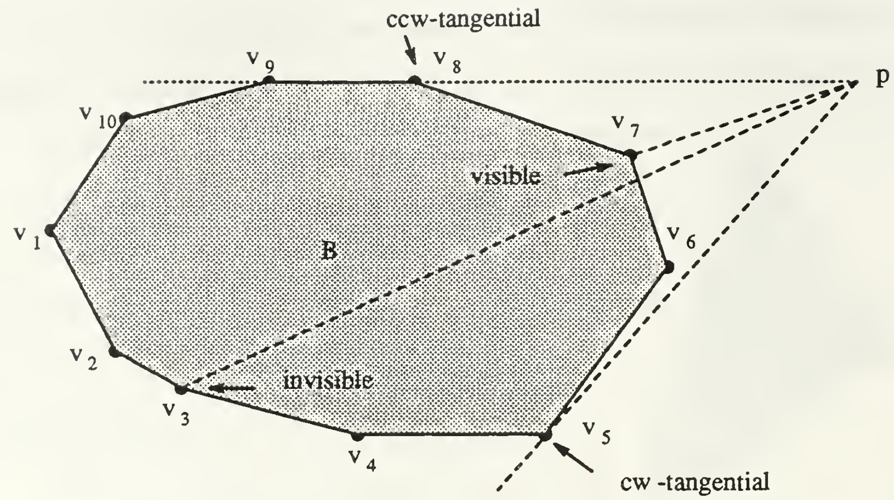


Figure 21. Visibility from point p to convex polygon B (I)

- A vertex v_i is *tangential* from point p if the two vertices adjacent to v_i lie on the same side of the line containing $\overline{pv_i}$.
- A vertex v_i is *visible* if the segment $\overline{pv_i}$ does not intersect the interior of B and the two vertices adjacent to v_i lie on opposite sides of the line containing $\overline{pv_i}$.
- A vertex v_i is *invisible* if the segment $\overline{pv_i}$ intersects the interior of B.

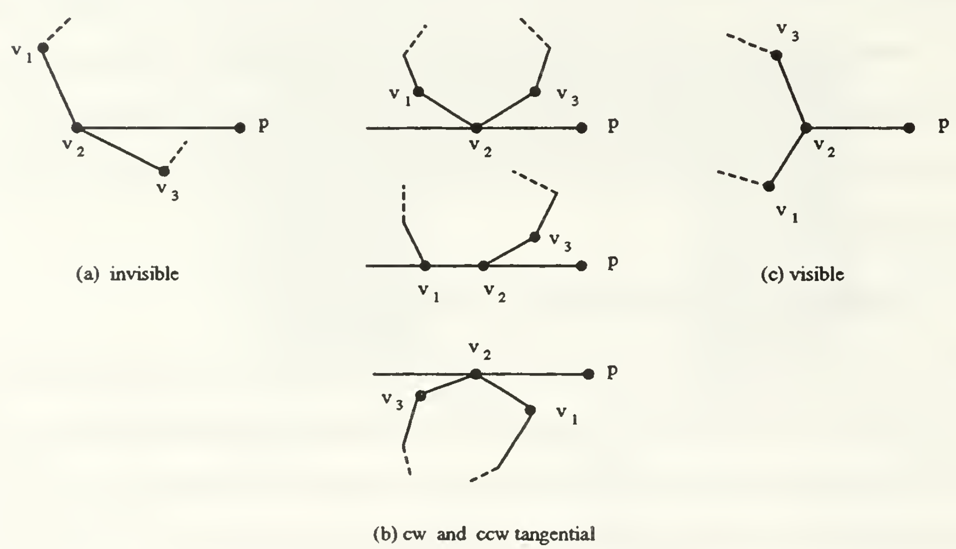


Figure 22. Classifications of vertex v_i of polygon B with respect to a segment $\overline{pv_i}$

Figure 22 shows the classifications of a vertex v_i of polygon B with respect to a segment $\overline{pv_i}$.

Let *cw-tangential*(p, v_i, B) denote that vertex v_i of B is clockwise tangential with respect to the segment $\overline{pv_i}$, *ccw-tangential*(p, v_i, B) denote that vertex v_i of B is counterclockwise tangential with respect to the segment $\overline{pv_i}$, *visible*(p, v_i, B) denote that vertex v_i of B is visible with respect to the segment $\overline{pv_i}$, and *invisible*(p, v_i, B) denote that vertex v_i of B is invisible with respect to the segment $\overline{pv_i}$. It is now easy to establish the following lemma.

Lemma III.2 *Given a convex polygon B and a point $p \in free(B)$, the vertex v_i is one of the following four types:*

$$\textit{cw-tangential}(p, v_i, B) \equiv \sim ccw\left(p, v_i, \varphi^{-1}(v_i)\right) \wedge \sim ccw\left(p, v_i, \varphi(v_i)\right) \tag{III.6}$$

$$\textit{ccw-tangential}(p, v_i, B) \equiv \sim cw\left(p, v_i, \varphi^{-1}(v_i)\right) \wedge \sim cw\left(p, v_i, \varphi(v_i)\right) \tag{III.7}$$

$$\textit{visible}(p, v_i, B) \equiv ccw\left(p, v_i, \varphi^{-1}(v_i)\right) \wedge cw\left(p, v_i, \varphi(v_i)\right) \tag{III.8}$$

$$\textit{invisible}(p, v_i, B) \equiv cw\left(p, v_i, \varphi(v_i)\right) \wedge ccw\left(p, v_i, \varphi^{-1}(v_i)\right) \tag{III.9}$$

Proof.

For the first part (Eq. III.6), v_i is *cw* tangential if the two vertices adjacent to v_i, $\varphi^{-1}(v_i)$ and $\varphi(v_i)$, lie on the same side of the line containing $\overline{pv_i}$. we have the following three cases.

- case 1: $cw\left(p, v_i, \varphi^{-1}(v_i)\right) \wedge cw\left(p, v_i, \varphi(v_i)\right)$
  If $\overline{pv_i}$ and $\overline{v_i\varphi^{-1}(v_i)}$ make a right turn at v_i, $\overline{p\varphi^{-1}(v_i)}$ is clockwise from $\overline{pv_i}$, and $\overline{pv_i}$ and $\overline{v_i\varphi(v_i)}$ make a right turn at v_i, $\overline{p\varphi(v_i)}$ is clockwise from $\overline{pv_i}$, then v_i is *cw-tangential.*
- case 2: $col\left(p, v_i, \varphi^{-1}(v_i)\right) \wedge cw\left(p, v_i, \varphi(v_i)\right)$
  If p, v_i, and $\varphi^{-1}(v_i)$ are collinear and $\overline{pv_i}$ and $\overline{v_i\varphi(v_i)}$ make a right turn at v_i, $\overline{p\varphi(v_i)}$ is clockwise from $\overline{pv_i}$, then v_i is *cw-tangential.*

- case 3: $cw\,(p, v_i, \varphi^{-1}(v_i)) \wedge col\,(p, v_i, \varphi(v_i))$
  If $\overline{pv_i}$ and $\overline{v_i\varphi^{-1}(v_i)}$ make a right turn at v_i, $\overline{p\varphi^{-1}(v_i)}$ is clockwise from $\overline{pv_i}$, and p, v_i, and $\varphi(v_i)$ are collinear, then v_i is *cw-tangential.*

This gives a proof of Eq. III.6. in other words, v_i is *cw-tangential* from p (see Figures 21, 22).

The second part (Eq. III.7) is proven similarly.

For the third part (Eq. III.8), since the two vertices adjacent to v_i lie on the opposite side of the line containing $\overline{pv_i}$ and $\overline{pv_i}$ does not intersect the interior of B, therefore $\overline{pv_i}$ and $\overline{v_i\varphi^{-1}(v_i)}$ make a left turn at v_i, $\overline{p\varphi^{-1}(v_i)}$ is counterclockwise from $\overline{pv_i}$, and $\overline{pv_i}$ and $\overline{v_i\varphi(v_i)}$ make a right turn at v_i, $\overline{p\varphi(v_i)}$ is clockwise from $\overline{pv_i}$. This gives a proof of Eq. III.8 (see Figure 21, Figure 22).

For the last part (Eq. III.9), since $\overline{pv_i}$ intersects the interior of B, therefore $\overline{pv_i}$ and $\overline{v_i\varphi^{-1}(vi)}$ make a right turn at v_i, $\overline{p,\varphi^{-1}(v_i)}$ is clockwise from $\overline{pv_i}$, and $\overline{pv_i}$ and $\overline{v_i\varphi(v_i)}$ make a left turn at v_i, $\overline{p\varphi(v_i)}$ is counterclockwise from $\overline{pv_i}$. This gives a proof of Eq. III.9 (see Figure 21, Figure 22). □

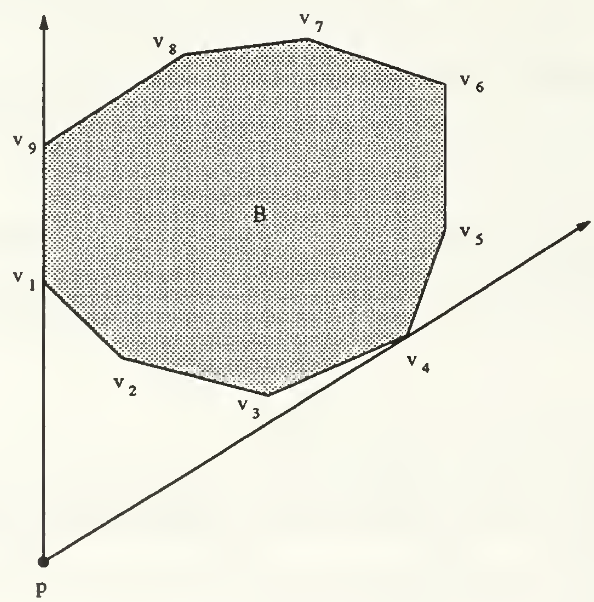


Figure 23. Visibility from point p to convex polygony B (II)

For example, in Figure 23, vertex v_1 is *cw-tangential*, vertex v_2 is *visible*, vertex v_4 is *ccw-tangential* and vertex v_7 is *invisible.*

2. Type of an Image from a Point to a Convex Polygon

Let B denote a convex polygon with n vertices. Let a point $p \in \text{free}(B)$. The image of p may be on an edge or a vertex of convex polygon. If an image of p is on an edge, the image moves when p moves slightly. However, if the image of p is on a vertex, it does not move when p moves slightly. The following theorem determines the image occurs either on an edge or on a vertex.

Theorem III.1 *Let $B = \{v_1, \cdots, v_n\}$ be a convex polygon, and let p be a point in free(B) and define θ, θ_1, and θ_2 by*

$$\begin{aligned} \theta &= \Psi(v_i, \varphi(v_i)) + \frac{\pi}{2}, \\ \theta_1 &= \Psi(p, v_i), \\ \theta_2 &= \Psi(p, \varphi(v_i)). \end{aligned}$$

Let vertex v_j be cw-tangential from point p. There exists a vertex v_i ($i = j$ or $i \neq j$) such that the image of p on B is of one of the following two types.

(I) If

$$(\theta_1 > \theta) \ \wedge \ (\theta_2 < \theta) \tag{III.10}$$

then the image lies on an edge $\overline{v_i\varphi(v_i)}$ of polygon B,

(II) If

$$\theta_1 \leq \theta \ \wedge \ (\theta_2 \leq \theta) \tag{III.11}$$

then the image of p is on a vertex v of polygon B.

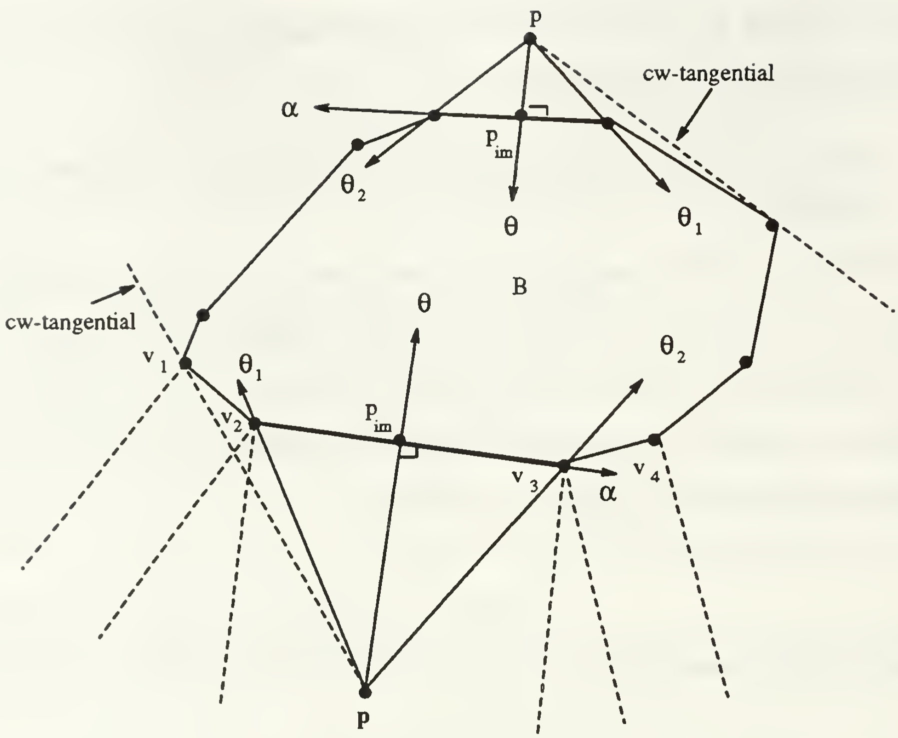


Figure 24. Image of point p lies on an edge of convex polygon B

Proof:

Consider two straight lines, one joining p with v_i and the other joining p and $\varphi(v_i)$. The orientations of these two lines are θ_1 and θ_2 respectively. Also, denote by α the orientation from v_i to $\varphi(v_i)$ and by $\theta = \alpha + \frac{\pi}{2}$ the perpendicular from p to $\overline{v_i\varphi(v_i)}$.

For the first part of the proof (Eq. III.10), let p_{im} be the intersection of two lines whose orientations are α and θ (see Figure 24). Assume that the hypothesis of Eq. III.10 is true. Since $\theta_1 > \theta$, then $\overline{pp_{im}}$ and $\overline{p_{im}v_i}$ make a left turn at p_{im}. Also, $\theta_2 < \theta$, then $\overline{pp_{im}}$ and $\overline{p_{im}\varphi(v_i)}$ make a right turn at p_{im}. It follows that p_{im} is visible from p by lemma III.2. This means that v_i and $\varphi(v_i)$ are on opposite sides of p_{im}. Therefore, p_{im} lies on the boundary of B. In other words, p_{im} lies on an edge $\overline{v_i\varphi(v_i)}$ of B.

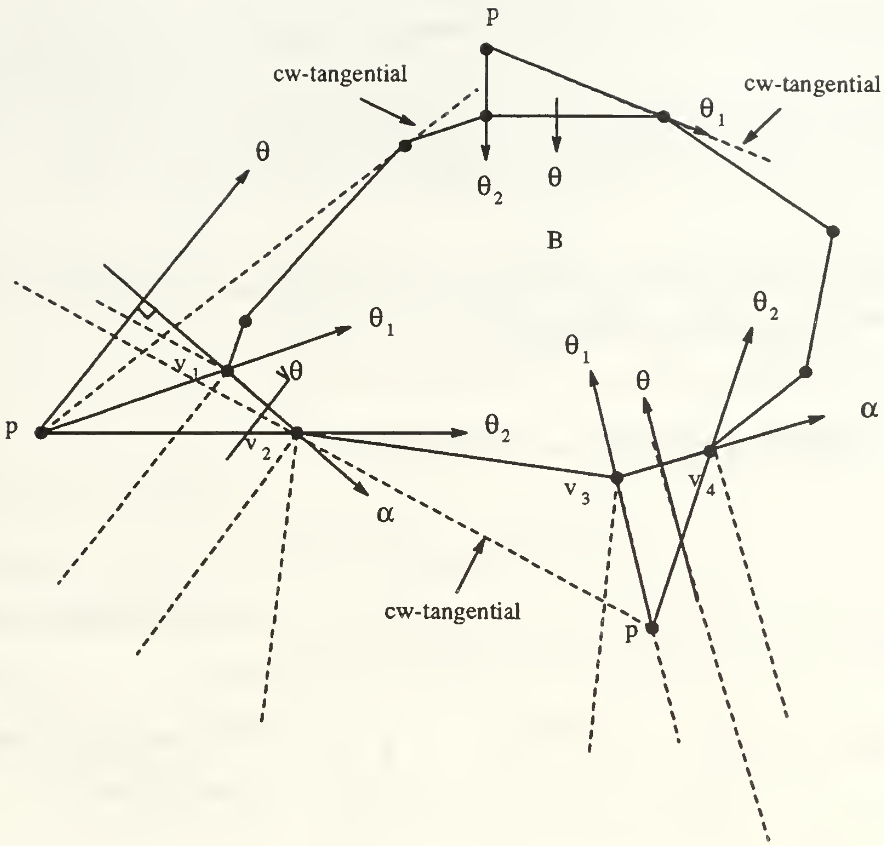


Figure 25. Image of point p lies on vertex v_i of convex polygon B

For the second part (Eq. III.11), assume that the hypothesis of Eq. III.11 is true. we have the following three cases (see Figure 25).

- Case 1: $\theta_1 < \theta \wedge \theta_2 < \theta$

  Since $\theta > \theta_1$, and $\theta > \theta_2$. Therefore the image of p does not lie on the edge $\overline{v_i\varphi(v_i)}$. But $\theta_1 < \theta_2$, since $\varphi(v_i)$ is the next vertex to v_i. Then v_i is a closed point from p. Therefore, the image of p is a vertex v_i.

- Case 2: $\theta_1 = \theta \wedge \theta_2 < \theta$

  Since $\theta = \theta_1$ and $\theta > \theta_2$, then the image of p does not lie on the edge $\overline{v_i\varphi(v_i)}$. But $\theta_1 < \theta_2$, since $\varphi(v_i)$ is the next vertex to v_i. Then v_i is a closed point from p. Therefore, the image of p is a vertex v_i.

- Case 3: $\theta_1 < \theta \wedge \theta_2 = \theta$

  Since $\theta > \theta_1$ and $\theta = \theta_2$, then the image of p does not lie on the edge $\overline{v_i\varphi(v_i)}$. But $\theta_1 < \theta_2$, since $\varphi(v_i)$ is the next vertex to v_i. Then $\varphi(v_i)$ is a closed point from p. Therefore, the image of p is a vertex $\varphi(v_i)$.

This gives a proof of Eq. III.11. in other words, p_{im} occurs on a vertex of B. $\square$

Since there are no vertices in the interior of a convex polygon B, then by Theorem III.1 we obtain the following corollary.

Corollary III.1 *For any point $p \in free(B)$ and a convex polygon B, there exists only one image from p to a convex polygon B.*

3. The Image Type Algorithm

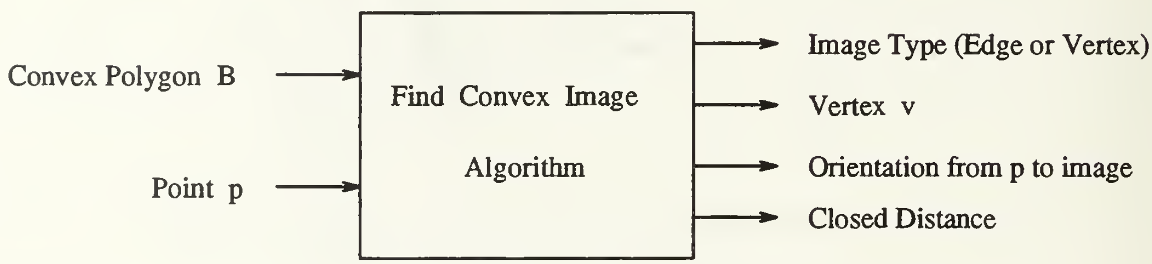


Figure 26. Image type

We now describe the construction of an algorithm for image type. The block diagram for finding image type is shown in Figure 26. The inputs are a convex polygon B and a point $p \in \mathrm{free}(B)$. The outputs are an image type (vertex type or edge type), a vertex v_i, the orientation from p to its image, and the closed distance from p to the image. For a vertex type image, vertex v_i is the image of p on B, but for an edge type image, the image of p on B lies on an edge $\overline{v_i\varphi(v_i)}$. In pseudo–code the method is as follows:

Convex_Image$(p,\ B)$

Input: point p ($\in$ free(B))
convex polygon $B = (v_1, \cdots, v_n)$

Output: *image* image type (vertex-type or edge-type)
vertex v
orient (the orientation from p to image)
closed (the distance from p to image)

begin
1. $v :=$ first-vertex(B)
2. *** find clockwise tangential(v) ***
3. **while** $(\sim ccw(p, v, \varphi^{-1}(v)) \wedge \sim ccw(p, v, \varphi(v)))$
4. $v = \varphi(v)$
5. *** find image type ***
6. **while**(1)
7. $\theta = \Psi(v, \varphi(v)) + \frac{\pi}{2}$
8. $\theta_1 = \Psi(p, v)$
9. $\theta_2 = \Psi(p, \varphi(v))$
10. **if** $((\theta_1 \leq \theta)\ \wedge\ (\theta_2 \leq \theta))$
11. **then**
12. $image.type$ = VERTEX
13. $image.posi = v$
14. $image.orient = \theta_1$
15. $image.closed =$ **Compute_Euclidean_Distance**$(p,\ v)$
16. **else**
17. **if** $((\theta_1 > \theta)\ \wedge\ (\theta_2 < \theta))$
18. **then**
19. $image.type$ = EDGE
20. $image.posi = v$
21. $image.orient = \theta$
22. $image.closed =$ **Compute_Dist**$(p,\ v)$
23. **else**
24. $v = \varphi(v)$
25. **return** *image*

end

The algorithm simply loops until the *image* is reached (line 25). First, the algorithm loops until *cw-tangential* vertex is reached (lines 3-4). Hence, in each loop (line 6), we check the condition for vertex type (line 10). If the condition is not satisfied, the condition for edge type is checked (line 17). Also, if it is not satisfied, we take the next vertex (line 24). We continue in this process until one of the above conditions (line 10 or line 17) is satisfied.

The subroutine **Compute_Euclidean_Distance** computes the distance between two points (see Eq. III.4). The subroutine **Compute_Dist** computes the closest distance from p to its image which lies on an edge. The subroutine for **Compute_Dist** is as shown below.

Compute_Dist(p, v)
Input: point p ($\in$ free(B))
v first vertex of edge where the image on it

Output: *closed* the closet distance from p to *image*

begin
$area$ = **Compute_Area_Triangle**$(p, v, \varphi(v))$
$dist$ = **Compute_Euclidean_Distance**$(v, \varphi(v))$
$closed = \frac{2 \times area}{dist}$
return *closed*
end

The subroutine **Compute_Area_Triangle** computes the area of triangle (see Eq. III.3).

a. Proof of Correctness of the Algorithm

To prove the correctness of the above algorithm, we want to show that the algorithm always returns an *image* structure when the **while-loop** in line 6 is executed. In other words, the **while-loop** in line 6 is never executed forever.

Assume that v_1 is the starting vertex of polygon B (Figure 27). Since v_3 is *cw-tangential*, the while-loop in line 3 returns $v = v_3$. It follows that, at the beginning of the while-loop in line 6, v will be checked to determine the image type.

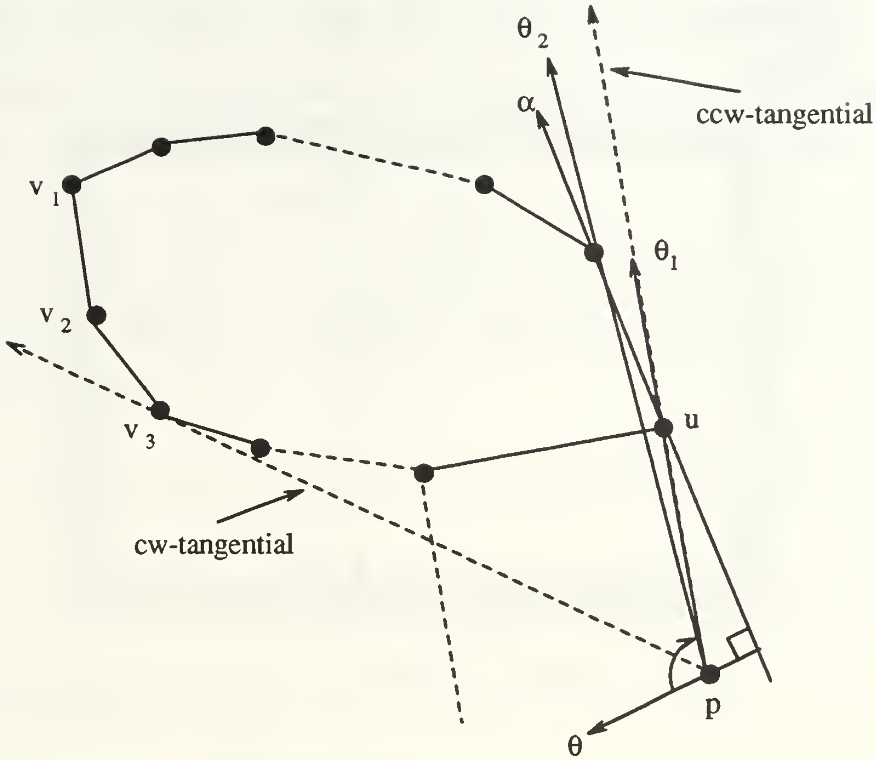


Figure 27. Correctness of image type algorithm

If the conditions in lines 10 and 17 are not satisfied, we take the next vertex, as shown in line 24. In the worst-case, we continue in this process until vertex $v = u$. Vertex v is *ccw-tangential*, but the condition in line 10 will be satisfied ($\theta_1 < \theta \wedge \theta_2 < \theta$). It follows that the algorithm returns the image type of point p as vertex type and vertex v. This proves that the while-loop in line 6 is always terminated.

b. Analysis of the Worst-Case Time Complexity of the Algorithm

The operations in lines 1, 4, and 7-25 each takes $O(1)$ time. The loop from lines 3 through 4 will be taken $O(n)$ time in the worst-case. The loop from lines 6 through 25 will be taken $O(n)$ time in the worst-case. The overall running time of the algorithm is $O(n)$.

F. FINDING AN IMAGE ON A NONCONVEX POLYGON

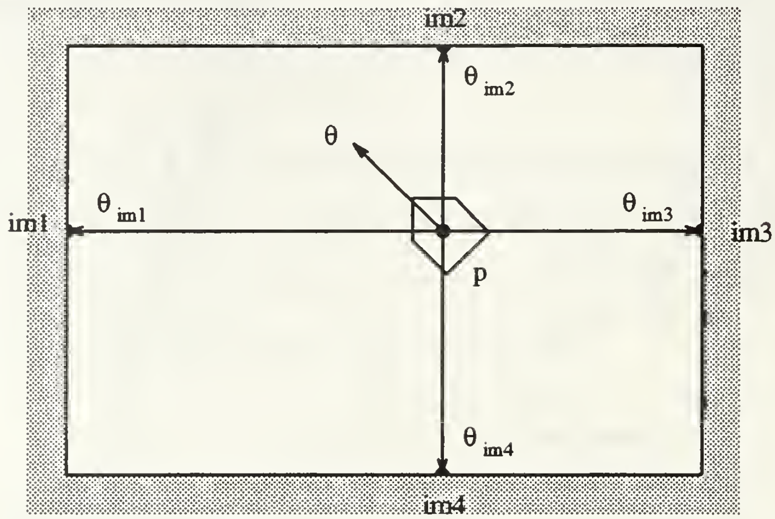


Figure 28. Image of a point p on *cw* concave polygon B

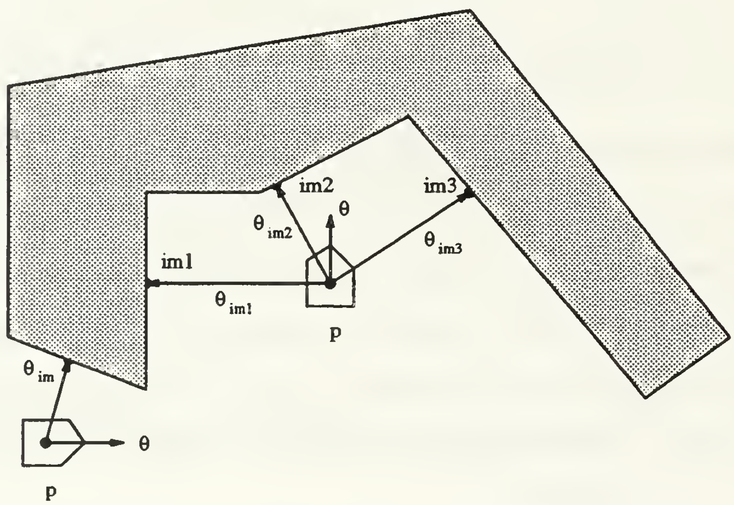


Figure 29. Image of a point p on *ccw* concave polygon B

Suppose we have an outermost nonconvex *cw* polygonal boundary (Figure 28) or nonconvex *ccw* polygon obstacle inside the boundary (Figure 29). Let a point $p \in$ free(B). In the case of an outermost nonconvex *cw* polygon, there is more than one image. The image always lies on an edge of B. In the case of nonconvex *ccw* polygon, there may be one or more images depending upon the position of the robot. The image may be one of the vertices of B or it may lie on an edge of B. We have the following observations. First, the image may be behind the vehicle. For instance, in Figure 28, p_{im3} and p_{im4} are behind the vehicle. In this case, this can not be an image.

The following remark illustrates how we can know whether the image is behind a vehicle. Let θ denote a vehicle's heading (the direction from p) and let $\Psi(p, p_{im})$ denote the direction from p to p_{im}.

Remark III.1 *Given a nonconvex polygon B and a point $p \in free(B)$.*

(I) If

$$|\Phi\left(\theta - \Psi(p, p_{im})\right)| \leq \frac{\pi}{2} \tag{III.12}$$

then the image of p is usable.

(II) If

$$|\Phi\left(\theta - \Psi(p, p_{im})\right)| > \frac{\pi}{2}$$

then the image of p is behind the vehicle.

The second observation, for the usable image (Eq. III.12). The following remark illustrates how we can know if the image is on the right, left or front of the vehicle.

Remark III.2 *The real image is of one of the following three types.*

(I) If

$$\Phi\left(\theta - \Psi(p, p_{im})\right) > 0$$

then the image of p is on the right of the vehicle.

(II) If

$$\Phi\left(\theta - \Psi(p, p_{im})\right) < 0$$

then the image of p is on the left of the vehicle.

(III) If

$$\Phi\left(\theta - \Psi(p, p_{im})\right) = 0$$

then the image of p is on the front of the vehicle.

To summarize: in the case of a nonconvex polygon, we conclude that

1. We need an another algorithm to find the image(s).
2. We need another data structure for the image. In this case, we may have one or more images. Therefore, we need an array of image structures. The size of this array is the maximum numbers of images.
3. If the initial orientation, θ, of the vehicle is in the opposite direction to the desired motion of the vehicle, then we cannot use lemma III.1 to reject the image which lies behind the robot.

According to above, the use of subpolygons when the world has nonconvex polygons will let us use the same algorithm for convex polygons (see Subsection 3) and the same data structure for image (see Chapter VIII).

IV. PATH CLASS REPRESENTATION USING VORONOI DIAGRAMS

The *global path planning* problem is the problem of finding the optimum path class to connect given *start* and *goal* configurations. The idea of Voronoi diagrams plays an important role in solving this problem. This chapter presents a method to symbolically represent the path classes in a polygonal world. It is developed with the objective of providing useful information to the local motion planning, with an emphasis on *safely* navigating through free space with smooth motions. The discussion and analysis given in this chapter are related to one of the most important aspects of the motion planning problem in robotics, i.e., the connectivity of geometrical objects. The motivation of this approach arises from the following observation. Steering-function control rules exist for line, circle and parabola tracking, as well as for two lines, two points, and vertex tracking (see Chapter VI). Parallels exist between these rules and physical obstacles from which the sensors obtain returns when the robot travels down an office corridor. A vehicle moving in hallways recognizes the left and right walls. This traversal path can be described in terms of left and right obstacles. Since closer obstacles present the most immediate threat to the robot's safety, then we should be most concerned with these. This will also aid in focusing the attention of sensors on those obstacles. The Voronoi boundary gives us the idea that the motion will be considered safer if it stays further away from obstacles. The motivation behind this method is to try to link the path class definition to the major obstacles of the world that the robot sensors would use. Prior to examining this method, background information on path classes and Voronoi diagrams will be addressed. Second, the path class representation using directed v-edges squence is developed as a decomposition for use with a local motion planner. Third, the short comings of using a polygonal world, and their solution using the idea of subpolygons, will be discussed. Last, the advantages of using the path class representation using the directed v-edges sequence are presented.

A. PATH CLASSES

A *path* f in a world $\mathcal{W}$ is a continuous function

$$f : [0,1] \rightarrow free(\mathcal{W})$$

with $f(0) \neq f(1)$. We consider a path f to be a directed curve with natural direction from $f(0)$ to $f(1)$. The two points $f(0)$ and $f(1)$ are called its *endpoints* and we say that the path *joins* them. We usually denote $f(0)$ as a start S and $f(1)$ as a goal G. Figure 30 is an example of a world with three *ccw* polygons B_1, B_2 and B_3, one *cw* polygon B_0, and paths from S to G.

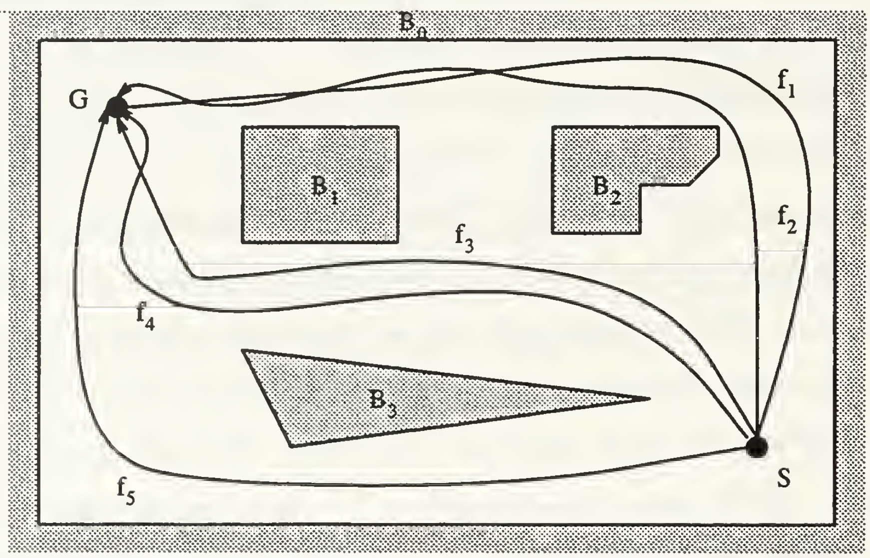


Figure 30. A world and paths

It is clear that, in any connected space, the set of paths between any two points is infinite. In order to simplify the problem of choosing a path, we want to group paths that are, in some sense, alike. Before we give a formal definition, we present an intuitive idea of what makes two paths similar. In Figure 30, we see that paths f_1 and f_2 are somewhat similar in that they both go to right of B_1, B_2 and B_3. Another observation is that there is no polygon between them. Notice, however, that

B_1 and B_2 are between f_1 and f_3. Based on these observations, we might conclude that f_1 and f_2 should be grouped together, and also f_3 and f_4, but f_5 should be in a group by itself. The relation of *homotopy* provides a formal method for making these groupings [13].

Consider two paths in the robot's world, say f and g, with common endpoints. We say that f is homotopic to g, written $f \cong g$, provided there is a continuous function

$$H : [0,1] \times [0,1] \rightarrow free(\mathcal{W})$$

which satisfies these equations:

$$\begin{aligned} H(t,0) &= f(t) \ \forall t \in [0,1] \\ H(t,1) &= g(t) \ \forall t \in [0,1] \\ H(0,s) = f(0) &= g(0) \ \forall s \in [0,1] \\ H(1,s) = f(1) &= g(1) \ \forall s \in [0,1]. \end{aligned}$$

In other words, H is a function that allows us to continuously deform one path into the other without crossing an obstacle, with both endpoints fixed. Furthermore, homotopy defines an equivalence relation on the set of paths which partitions them into a collection of *homotopy classes* or *path classes* [13]. We will use this relation to reduce the problem of path selection by considering a finite set of path classes rather than an infinite set of paths. In Figure 30, $f_1 \cong f_2$ and $f_3 \cong f_4$.

The concept of homotopy class or path class is essential in motion planning [26]. Consider typical missions for an autonomous vehicle such as

- Given start and goal configurations, a vehicle finds the best path class and executes a motion in the path class,
- A vehicle is hugging right (or left) walls,
- A vehicle is browsing randomly in the free area,
- A vehicle is following a walking person, or
- A vehicle is looking for an office that has the light on.

In each of these missions, one path class is found through some algorithm. We consider the problem of how to symbolically represent path classes. In order to symbolically represent path classes and to make the navigation task easier, one of the following methods can be used to decompose the world $\mathcal{W}$:

1. Borders (see [36, 47])
2. Generalized Voronoi diagram (we will discuss this method in this cxhapter)
3. Shortest paths

B. THE LOCUS APPROACH TO PROXIMITY PROBLEMS: VORONOI DIAGRAM

Proximity or closeness is one of the most essential concepts in robotics. This concept, for instance, is related to safe motion of a robot in a given environment. In a simple hallway, its "center line" has the obvious meaning. A Voronoi boundary is a generalized version of a center line in a complex geometrical configuration. Our interest in this dissertation is in using the idea of Voronoi diagram to simplify the planning of collision–free paths for a robot among obstacles. The Voronoi diagram, as usually defined, is a *strong deformation retract* of free space so that free space can be continuously deformed onto the diagram. This means that the diagram is complete for path planning, i.e., searching the original space for paths can be reduced to a search on the diagram. Reducing the dimension of the set to be searched usually reduces the time complexity of the search. Secondly, the diagram leads to robust paths, i.e., paths that are maximally clear of obstacles.

1. Definitions

Assume there are $n > 1$ different polygons in a world $\mathcal{W}$:

$$\mathcal{W} = \{B_1, \cdots, B_n\} \quad n > 1$$

Definition: The *Voronoi region* $V(B_i)$ of polygon B_i in $\mathcal{W}$ is the the set of points whose images are on it.

Definition: The union of all region boundaries is called the *Voronoi diagram* of a world.

$$V(\mathcal{W}) = \bigcup_{B_i \in \mathcal{W}} V(B_i) \quad 1 < i \leq n$$

Definition: The boundary of the Voronoi regions is called *Voronoi boundary.* Therefore the Voronoi boundary of a world is the set of points that have at least two images on distinct objects.

Definition: The common boundary of two Voronoi regions is a *Voronoi edge.*

Definition: Two Voronoi edges meet at a *Voronoi vertex*; such a point has three or more nearest neighbors in the world $\mathcal{W}$.

We know that

1. the Voronoi boundary of two points is a line,
2. the Voronoi boundary of two lines is one or two lines, and
3. the Voronoi boundary of a point and a line is a parabola.

For more details, see [36, 59, 44]. In the following subsection, we are going to show the Voronoi diagram of a world $\mathcal{W}$ consisting of a polygon.

2. Voronoi Diagram of Polygon

We consider a world $\mathcal{W}$ that has only one *ccw* polygon B. An image $im(p, B)$ of a point $p \in \text{free}(B)$ to B is the closest point from p on B. The image is a *vertex* on B or on an *open edge* e in B (an open edge does not include both endpoints) (see Figure 31). In this case, a polygon is regarded as the union of vertices and open edges.

Each point $p \in \text{free}(B)$ can be characterized by whether the image $im(p, B)$ is one of the vertices of B or on any edge of B. The Voronoi region of a vertex,

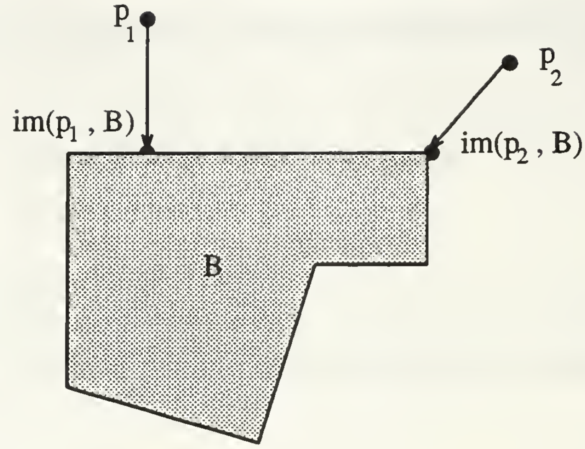


Figure 31. Images on a polygon

such as $V(v_1)$ in Figure 32, is said to be *vertex type*, and that of an open segment, such as $V(e_1)$ in Figure 32, is said to be *edge type*. Suppose p is the position of a moving vehicle. Then its image moves when p is in an edge type region, but the image does not move when p is in a vertex region. This fact is important in local motion planning. An example of the Voronoi diagram of a *ccw* polygon is shown in Figure 32. In this polygon, there are five vertices and five edges, and hence there are ten Voronoi regions.

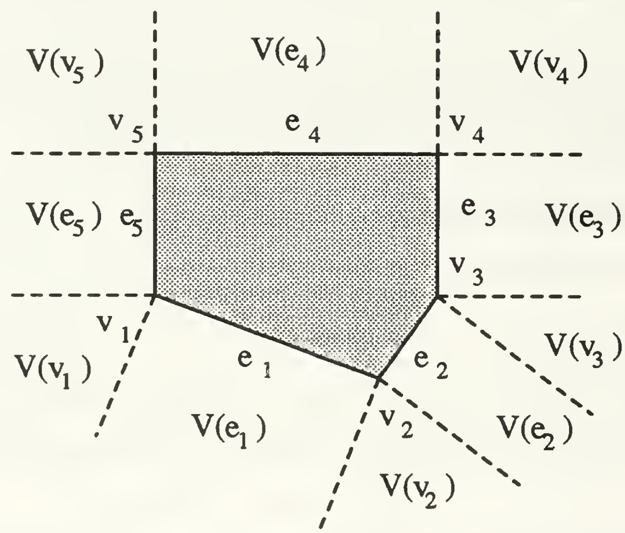


Figure 32. Voronoi diagram of a *ccw* polygon

If a world $\mathcal{W}$ consists of *cw* polygon B, its Voronoi diagram is shown in Figure 33. Another example is shown in Figure 34. For a concave vertex v, its Voronoi region $V(v)$ is the empty set.

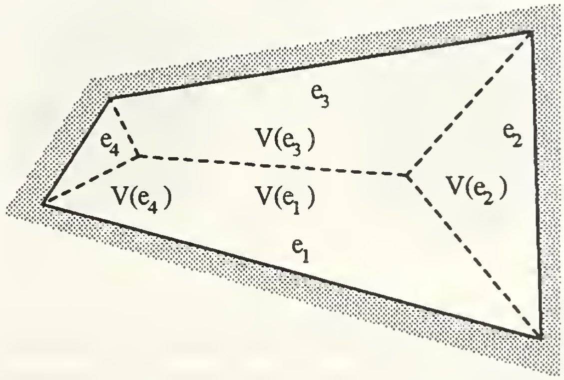


Figure 33. Voronoi diagram of a *cw* polygon (I)

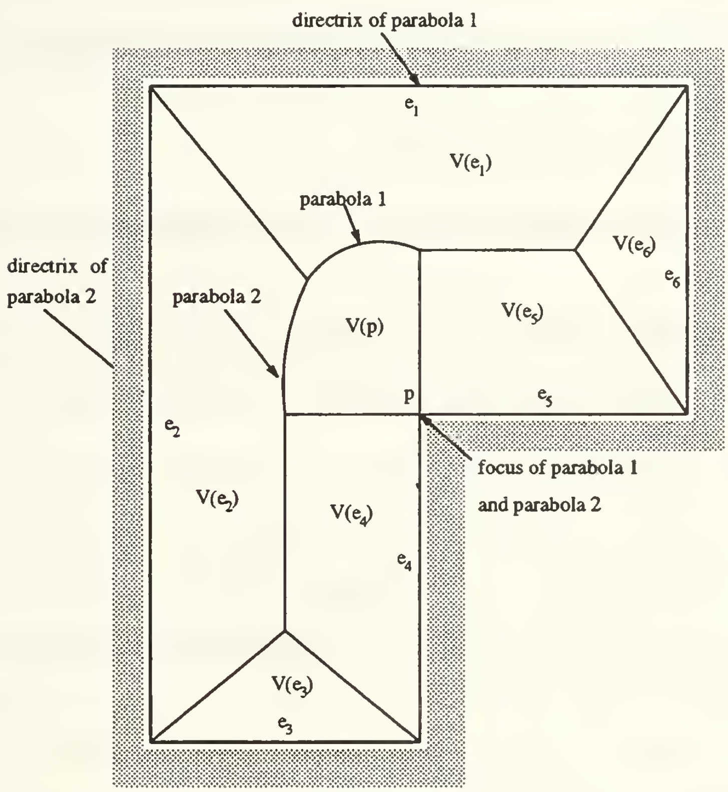


Figure 34. Voronoi diagram of a *cw* polygon (II)

C. POLYGONAL WORLD AND PATH CLASSES

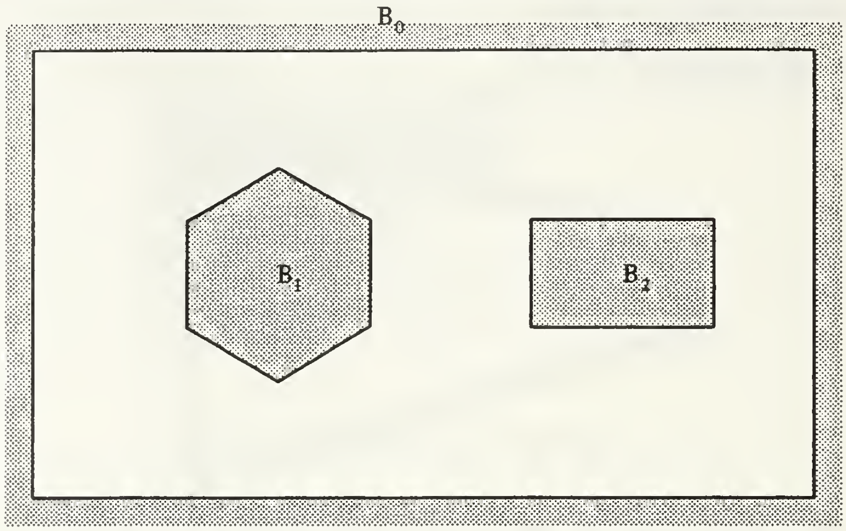


Figure 35. Polygonal world

Consider a world $\mathcal{W}$ which consists of a finite number of polygons n, i.e.,

$$\mathcal{W} = \{B_0, B_1, \cdots, B_n\}, \quad n > 0,$$

where B_0 is a *cw* polygon, and *ccw* polygons $B_1, \cdots, B_n$ are considered to be obstacles for the robot (see Figure 35).

For a point $p \in \text{free}(\mathcal{W})$, the distance $d(p, B_i)$ from p to a polygon B_i is defined in Eq. III.5. The Voronoi region $V(B_i)$ of a polygon B_i in $\mathcal{W}$ is defined as

$$V(B_i) = \{p \in \mathit{free}(\mathcal{W}) \mid (\forall j)[(i \neq j \wedge 1 \leq j \leq n) \rightarrow [d(p, B_i) < d(p, B_j)]]\} \quad \text{(IV.1)}$$

For instance, Eq. IV.1 means that any point within free($\mathcal{W}$) has its image on the two polygons. The Voronoi diagram of world $\mathcal{W}$ consisting of three polygons is shown in Figure 36. The Voronoi boundaries of $\mathcal{W}$ shown in Figure 36 consists of line segments and parabolic arcs. Note that the intersection where three or more of Voronoi boundary segments meet is called a *v-node*. A Voronoi boundary segment(s) between two v-nodes is called a *v-edge*. For example, there are two v-nodes and three v-edges as shown in Figure 36.

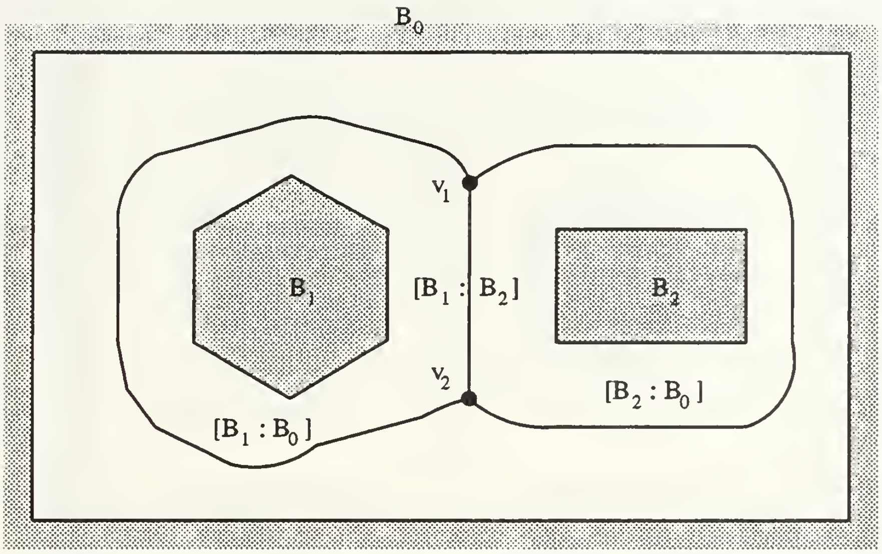


Figure 36. Voronoi diagram of polygonal world (I)

Each undirected v-edge ξ is the boundary of two Voronoi regions, $V(B_i)$ and $V(B_j)$. We denote an undirected v-edge ξ by

$$\xi = [B_i : B_j],$$

where $[B_i : B_j]$ and $[B_j : B_i]$ are considered the same. For example, in Figure 36, the undirected v-edge between the two v-nodes v_1 and v_2 is $\xi = [B_1 : B_2]$ or $\xi = [B_2 : B_1]$. In Figure 36, there are three undirected v-edges $[B_1 : B_0], [B_1 : B_2]$, and $[B_2 : B_0]$. Another example is shown in Figure 37. In this example, a world $\mathcal{W}$ consists of five polygons B_0, B_1, B_2, B_3 and B_4. There are five *v-nodes* and eight undirected v-edges $[B_1 : B_0], [B_1 : B_2], [B_2 : B_0], [B_2 : B_3], [B_1 : B_4], [B_4 : B_0], [B_3 : B_4]$, and $[B_3 : B_0]$.

1. Directed v-edge

Each undirected v-edge is the boundary of two Voronoi regions, $V(B_i)$ and $V(B_j)$. In this case,

$$[B_i : B_j] \equiv [B_j : B_i].$$

Now, we consider the *directed* v-edge. Once the directed v-edge is given, the concepts of left and right images take on meaning. This will aid in using the world

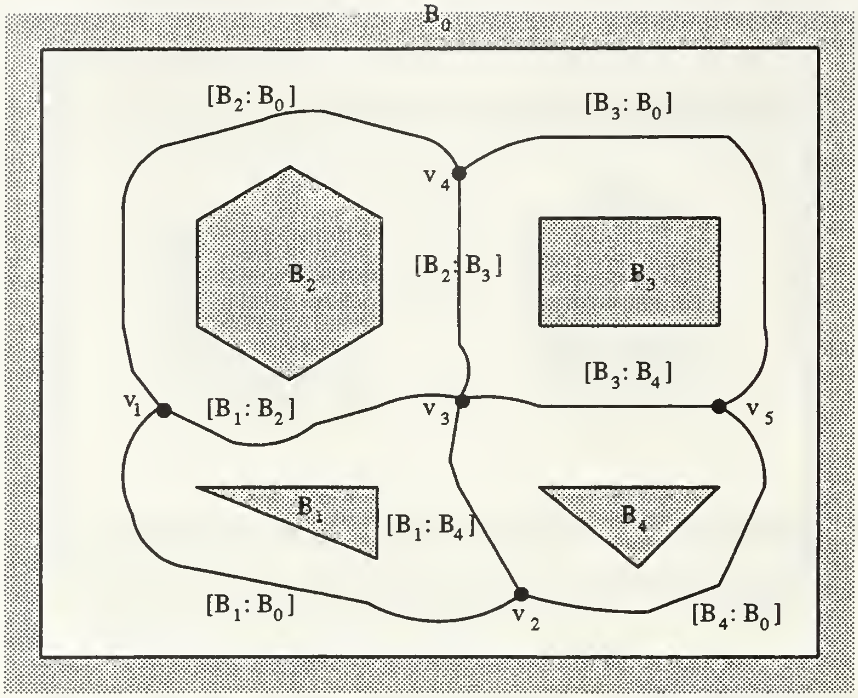


Figure 37. Voronoi diagram of polygonal world (II)

data to capture the spatial relationship between the objects in the world. We have two types of directed boundaries:

1. Directed boundaries of two polygons are the same (*ccw*):

   There are two opposite directions on an undirected v-edge $[B_i : B_j]$. One direction goes *ccw* with B_i and *cw* with B_j. The other direction goes *cw* with B_i and *ccw* with B_j (see Figure 38).

2. Directed boundaries of two polygons are different (*ccw* and *cw*):

   There are two opposite directions on an undirected v-edge $[B_i : B_j]$. One direction goes *ccw* with B_i and *cw* with B_j. The other direction goes *cw* with B_i and *ccw* with B_j (see Figure 39).

Now, we denote *directed* v-edge ξ by

$$\xi = [B_i/B_j],$$

where B_i and B_j refer to the left and right polygons respectively. It is clear that $[B_i/B_j]$ and $[B_j/B_i]$ are not the same. Although the assignment of left and right is

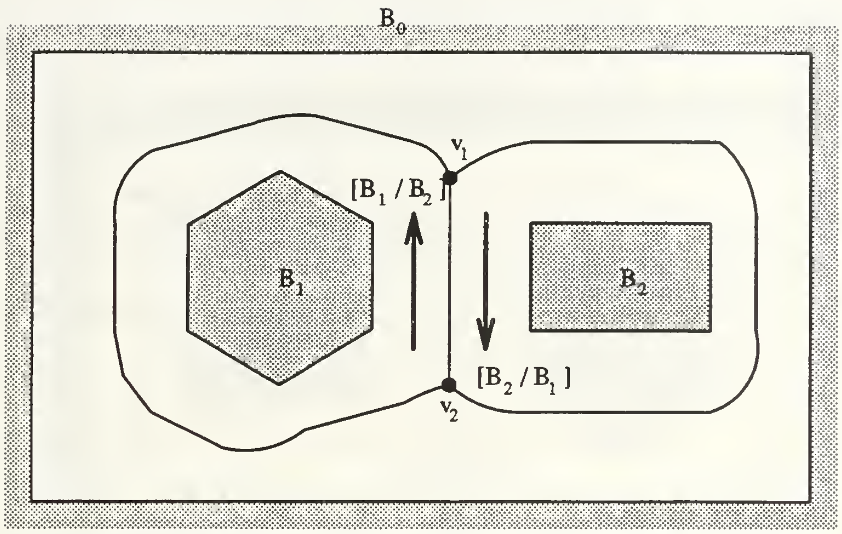


Figure 38. Defining directed v-edge for the same directed boundaries (*ccw* polygons)

arbitrary, it is fixed for all times once set. For consistency in this dissertation, left and right polygons will be the first and second terms in directed v-edges, respectively.

The following is the result of the previous discussion of directed v-edge.

Lemma IV.1 *In a polygonal world $\mathcal{W}$, where $\mathcal{W}$ is encircled by an outermost cw polygonal boundary and has n $(n \geq 1)$ ccw polygonal obstacles inside the boundary, a directed v-edge consists of two different polygons.*

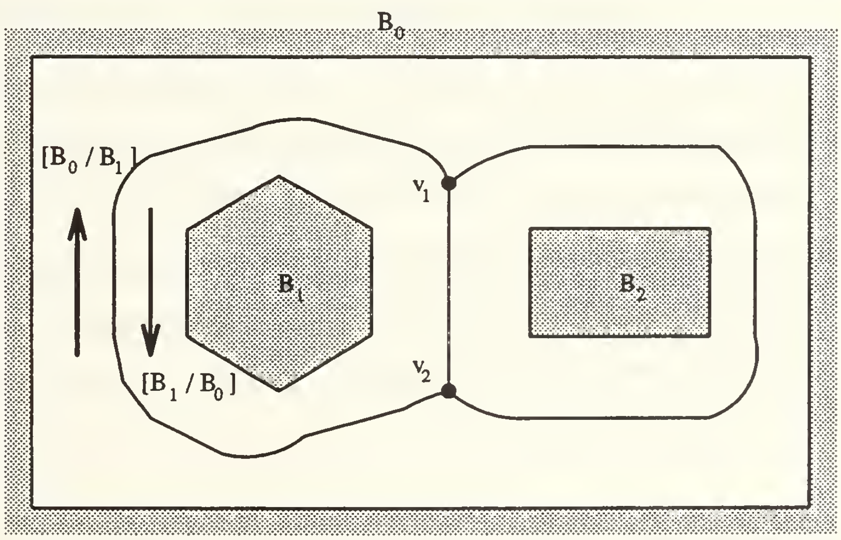


Figure 39. Defining directed v-edge for different directed boundaries (*cw* and *ccw*)

2. Canonical Paths and Directed v-edges Sequences

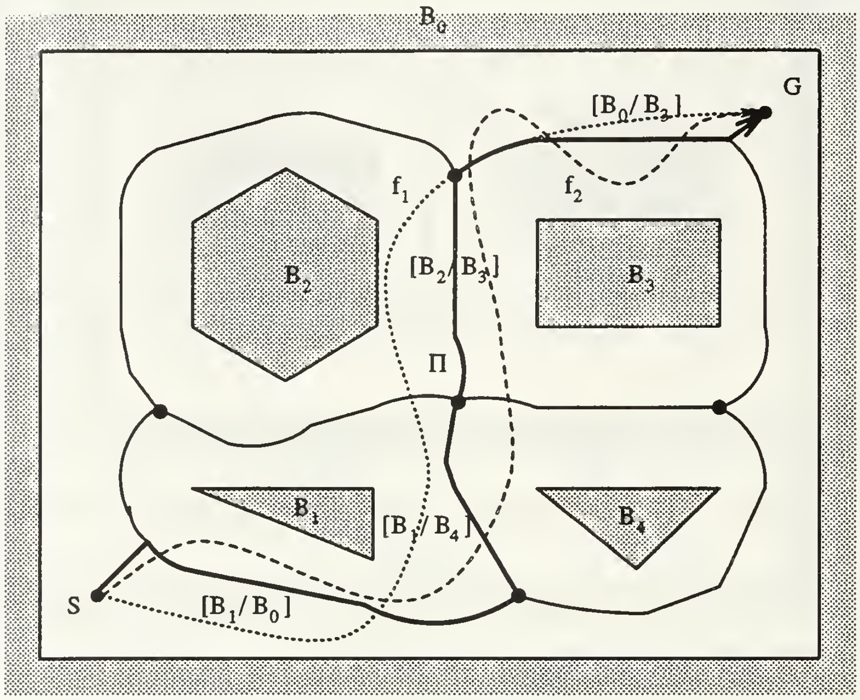


Figure 40. Paths and canonical paths

A robot can work only in the free space, free($\mathcal{W}$). A *path* f in a world $\mathcal{W}$ is a continuous function

$$f : [0,1] \rightarrow free(\mathcal{W}) \qquad \text{(IV.2)}$$

Consider the problem of finding a path from a start configuration, S, to a goal configuration, G in a polygonal world $\mathcal{W}$ (see Figure 40, where *ccw* polygons B_1 and B_2 are considered as obstacles for robot in this world and a world has one *cw* polygon B_0). It is desired to connect the start configuration, S, to the goal configuration, G, using a continuous, smooth path. There are infinitely many distinct paths connecting S and G. However, actually, we need to compare only paths which satisfy a special property.

Definition: A path Π is called a *canonical path* if there exists a sequence of directed v-edges such that

$$\Pi = s_s\, \xi_1\, \cdots\, \xi_k\, s_g \quad k \geq 1, \tag{IV.3}$$

where

- the right hand side of Eq. IV.3 is the concatenation of $k+2$ subpaths,
- the subpath s_s is the shortest path from S to ξ_1,
- $\xi_1 \cdots \xi_k$ is the sequence of directed v-edges, and
- the subpath s_g is the shortest path from ξ_k to G.

For example, in Figure 40,

$$\Pi = s_s[B_1/B_0][B_1/B_4][B_2/B_3][B_0/B_3]s_g.$$

The following is the result of the previous discussion of the canonical path.

Lemma IV.2 : *For a given $\mathcal{W}$, S, and G, a canonical path Π is the only one among all the paths in a homotopy class which satisfies the following conditions:*

1. the subpath connecting S to first directed v-edge is the shortest one,
2. sequential pieces from one directed v-edge to the next, and
3. the subpath connecting the last directed v-edge to G is the shortest one.

Proposition IV.1 : *For a given $\mathcal{W}$, S, and G, for paths f_1 and f_2 in a homotopy class, if $f_1 \rightarrow \Pi_1$ and $f_2 \rightarrow \Pi_2$ then $\Pi_1 \equiv \Pi 2$.*

Proof. Assume that the hypothesis is true. Since f_1 and Π_1 are homotopic, there is a continous function H which transforms f_1 into Π_1. Also, there is a continous function H which transforms f_2 into Π_2. By Lemma IV.2, there is only one canonical path Π among all paths in a homotopy class. It follows that $\Pi_1 \equiv \Pi 2$. □

Definition: A *directed v-edges sequence* Ξ is a finite sequence of directed v-edges such that no subsequence of $[B_i/B_j]$ $[B_j/B_i]$ is a part of it.

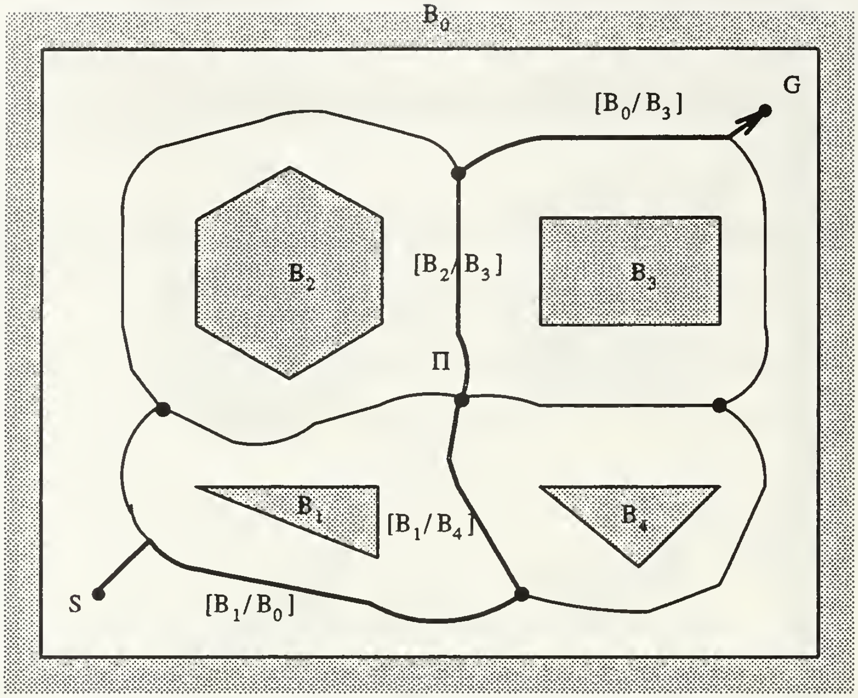


Figure 41. Interpretation of canonical path as directed v-edges sequence

By definition, if Π is a canonical path, then $\Pi = s_s \; \Xi \; s_g$ (See Figure 41), where Ξ is $\xi_1 \; \cdots \; \xi_k$.

Several examples of directed v-edges sequences are illustrated in Figures 42 and 43. For example, the directed v-edges sequences for the above figures are as follows:

$$\Xi = [B_1/B_0][B_1/B_4][B_2/B_3][B_0/B_3] \quad (Figure\ 42)$$

$$\Xi = [B_1/B_0][B_4/B_0][B_3/B_0] \quad (Figure\ 43)$$

Proposition IV.2 : *In a homotopy class, for all paths f_1 and f_2, $f_1 \cong f_2$ if and only if $\Xi_1 = \Xi_2$.*

Proof.

First prove the sufficiency. Assume $\Xi_1 = \Xi_2$. If $\Xi_1 = \Xi_2$, each path has a sequence of the same directed v-edges. Furthermore, in a homotopy class, both paths have the same left and right polygons. Each path is a concatenation of pieces. These

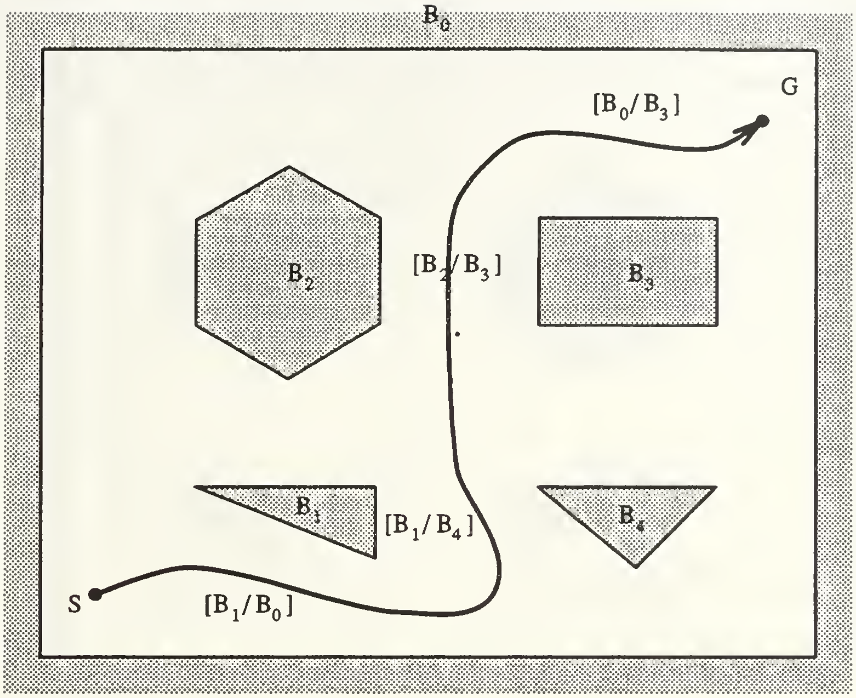


Figure 42. Directed v-edges sequence (I)

pieces connect the start configuration to the first directed v-edge in Ξ. the sequential pieces from one directed v-edge to the next, and the last directed v-edge to the goal configuration. We can easily construct H to transform f_1 into f_2 piece by piece without running over any obstacles. The transformation, H, is the composition of the sequences of the transformations shown. Hence, the paths are homotopic.

To prove the necessity, assume $f_1 \cong f_2$. We are given a path f_1. Consider a directed v-edges sequence Ξ_1 of f_1. Since f_1 and f_2 are homotopic, there is a continuous function H which transforms f_1 into f_2. Since $H(s,t)$ is a continous function, each directed v-edge ξ, which has left and right polygons, continuously concatenates with the next ξ over s as t moves when transforming f_1 into f_2. However, there is no way in which f_2 can eliminate, insert or repeat any ξ other than in the monotonic sequence of f_1. $H(s,t)$ can neither destroy existing nor create any new ξ, because $H(s,t) \in$ free(W) and $H(s,t)$ is continuous. Therefore $\Xi_1 = \Xi_2$. □

From above, we can conclude that

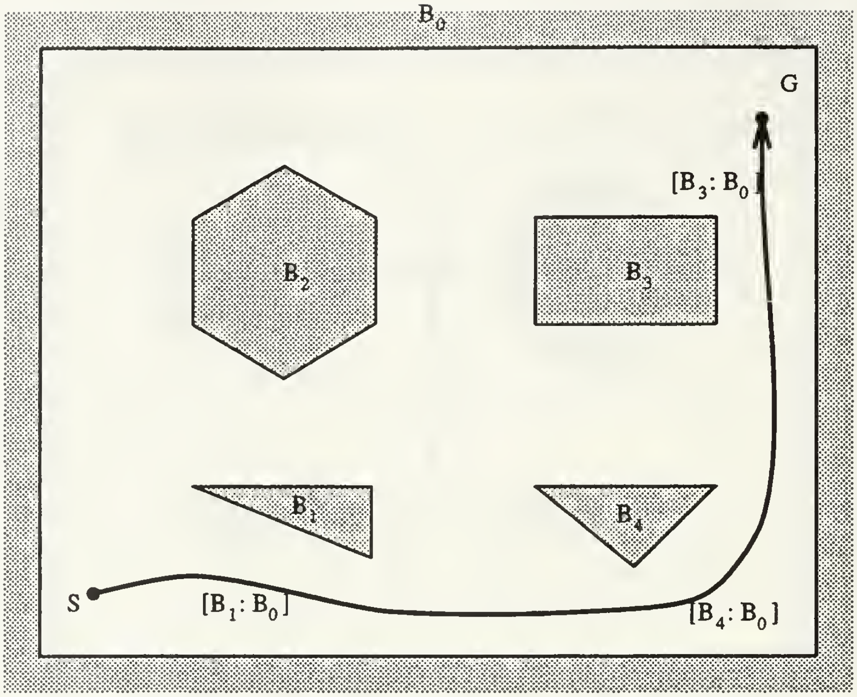


Figure 43. Directed v-edges sequence (II)

1. A directed v-edges sequence Ξ is unique for paths which are not homotopic.
2. A directed v-edges sequence Ξ is a symbolic representation.

In Chapter VI, we will show that the advantage of using directed v-edges sequence Ξ for local motion planning.

3. Connectivity Graph

We make the following observations about the world in Figure 36. Three Voronoi boundary segments intesect in one node (v-node). There is one line segment between two v-nodes (v-edge). Each v-node operates in both directions, and no v-node has a v-edge to itself.

Definition: A *basic connectivity graph* $G = (V, E)$ consists of V, a nonempty set of v-nodes, and E, a set of unordered pairs of distincts elements of V called undirected v-edges. Consequently this figure can be modeled using a basic connectivity graph, consisting of vertices which represent v-nodes, and undirected edges, which represent undirected v-edges, where each edge connects two distinct vertices.

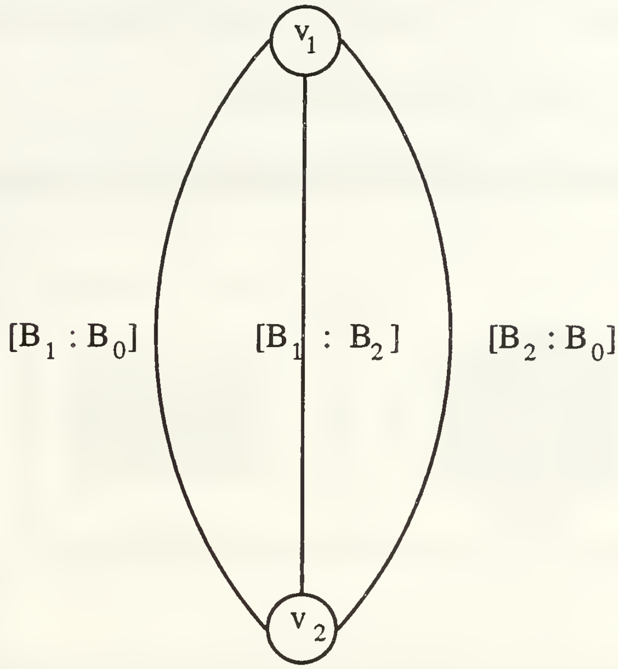


Figure 44. Basic connectivity graph of a polygonal world (I)

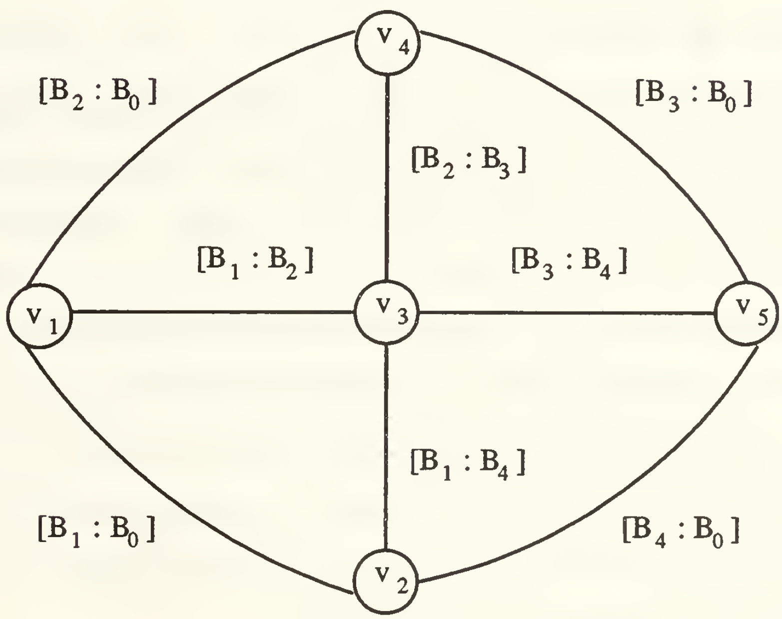


Figure 45. Basic connectivity graph of a polygonal world (II)

The basic connectivity graphs generated by the world in Figures 36 and 37 are shown in Figures 44 and 45.

Now we will explain how to represent a path class (see subsection 4).

4. Path Class Representation

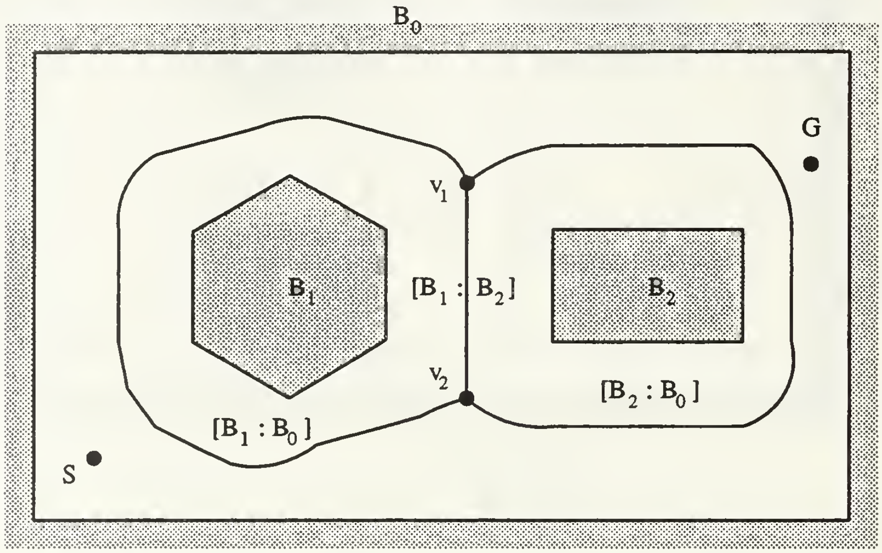


Figure 46. Polygonal world (I)

Consider the problem of finding a path from a start configuration, S, to a goal configuration, G in a polygonal world $\mathcal{W}$ (Figure 46). It is desired to connect the start configuration, S, to the goal configuration, G, using a continuous, smooth path. In Figure 46, there are four different path classes. Consider the problem of how to symbolically represent each path class. A method based on directed v-edges is presented. Given start and goal configurations, we add two new nodes, S and G, to the basic connectivity graph to obtain an *augmented connectivity graph.* The augmented connectivity graph generated by the world in Figure 46 is shown in Figure 47. In Figure 47, there are four different path classes. In its most general form, a path class, π, is symbolically represented by a sequence of directed v-edges. For instance, four typical path classes in Figure 47 are represented by:

$$\pi_1 = [B_0/B_1]\,[B_0/B_2]$$

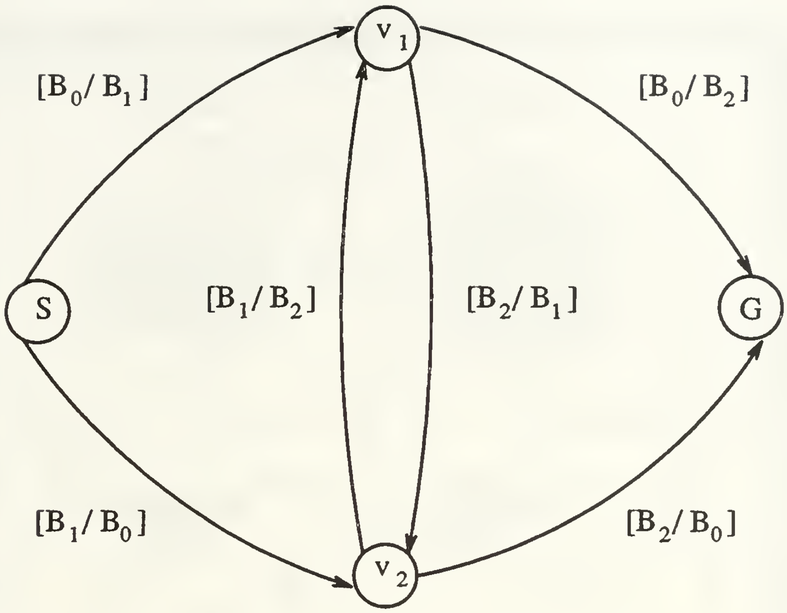


Figure 47. Augmented connectivity graph of a polygonal world (I)

$$\pi_2 = [B_0/B_1]\,[B_2/B_1]\,[B_2/B_0]$$
$$\pi_3 = [B_1/B_0]\,[B_1/B_2]\,[B_0/B_2]$$
$$\pi_4 = [B_1/B_0]\,[B_2/B_0]$$

Another example is shown in Figure 48. The augmented connectivity graph generated by the world in Figure 48 is shown in Figure 49. In Figure 49, there are twelve different path classes which connect S with G:

$$\pi_1 = [B_0/B_1]\,[B_0/B_2]\,[B_0/B_3]$$
$$\pi_2 = [B_0/B_1]\,[B_0/B_2]\,[B_3/B_2]\,[B_3/B_4]\,[B_3/B_0]$$
$$\pi_3 = [B_0/B_1]\,[B_0/B_2]\,[B_3/B_2]\,[B_4/B_1]\,[B_4/B_0]\,[B_3/B_0]$$
$$\pi_4 = [B_0/B_1]\,[B_2/B_1]\,[B_2/B_3]\,[B_0/B_3]$$
$$\pi_5 = [B_0/B_1]\,[B_2/B_1]\,[B_3/B_4]\,[B_3/B_0]$$
$$\pi_6 = [B_0/B_1]\,[B_2/B_1]\,[B_4/B_1]\,[B_4/B_0]\,[B_3/B_0]$$
$$\pi_7 = [B_1/B_0]\,[B_1/B_4]\,[B_1/B_2]\,[B_0/B_2]\,[B_0/B_3]$$

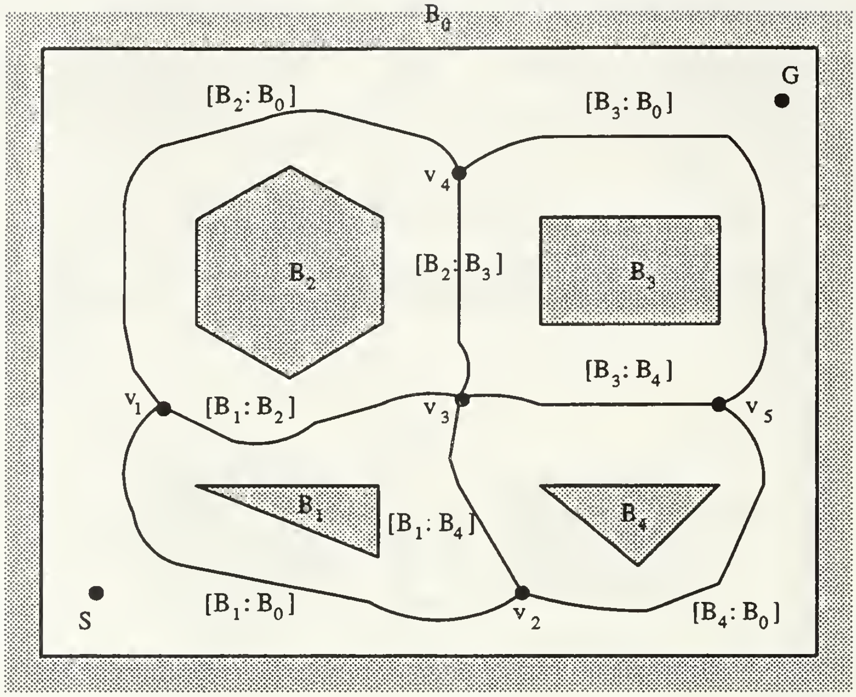


Figure 48. Polygonal world (II)

$$\pi_8 = [B_1/B_0]\,[B_1/B_4]\,[B_2/B_3]\,[B_0/B_3]$$

$$\pi_9 = [B_1/B_0]\,[B_1/B_4]\,[B_3/B_4]\,[B_3/B_0]$$

$$\pi_{10} = [B_1/B_0]\,[B_4/B_0]\,[B_3/B_0]$$

$$\pi_{11} = [B_1/B_0]\,[B_4/B_0]\,[B_4/B_3]\,[B_2/B_3]\,[B_0/B_3]$$

$$\pi_{12} = [B_1/B_0]\,[B_4/B_0]\,[B_4/B_3]\,[B_1/B_2]\,[B_0/B_2]\,[B_0/B_3]$$

5. Finding the Best Path Class

In this subsection we outline how to find the best path class. Finding the best path class from start configuration, S, to goal configuration, G, in the world is transformed into the minimum cost path finding problem from S to G in the augmented connectivity graph. The augmented connectivity graph uses a weighted edge whose value depends upon the mission-based cost function associated with the v-edge. For instance, a cost for the edge is defined as the energy (or time) for the vehicle to make a motion from one v-node v_i to another v-node v_j. This cost not only reflects the distance, but the turns needed to make the motion. It may also

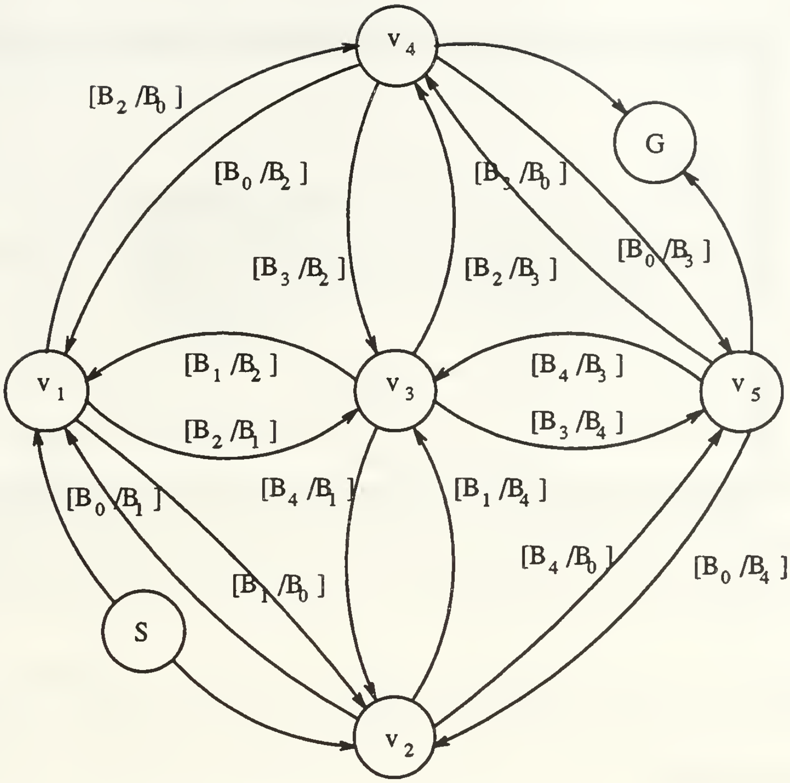


Figure 49. Augmented connectivity graph of a polygonal world (II)

reflect the safety (i.e., if the region is narrow, the cost is high). Dijkstra's algorithm, or a All-pairs shortest paths, can be perfectly applied to this global motion planning problem. As its result, the best path class in terms of a sequence of directed v-edges is obtained. The computation time is $O((n+m) log\, n)$ using a priority queue, where n and m are the numbers of v-nodes and the number of the directed v-edges in the augmented connected graph respectively. Once the path class is found, it is passed to a routine which ensures the vehicle will follow the path class to reach the goal.

6. Following the Path Class

Once the path class is found, it is passed to a routine which ensures that the vehicle will follow the path class to reach the goal. The choice of the mission type ultimately affects which steering function (for steering function definition, see Chapter V) is used to guide the vehicle. For example, one mission is to travel down the center of a hallway and remain at a user-specified distance from the corners when executing a turn into another corridor.

D. PATH CLASSES AND SUBPOLYGONS

The objective of path classes using polygonal world is to provide useful information for local motion planning. The directed v-edges sequences, Ξ, of a world $\mathcal{W}$ which consists of a finite number of polygons n is independent of the position of the vehicle inside the free($\mathcal{W}$). For example, in Figure 50, suppose the path class $\pi = [B_1/B_0]\ [B_2/B_0]$ and the start configuration of the vehicle are given as shown. Also, we know any point within free($\mathcal{W}$) has its left and right images on the two polygons. We proved in Chapter III that there is only one image of a point which lies in free space to a convex polygon and more than one image for a non convex polygon. When representing the path class using a polygonal world, we have the following disadvantages:

1. In Figure 50, B_1 and B_2 are *ccw* convex polygons and a B_0 is *cw* nonconvex polygon. When the vehicle navigates the given path π, left image is im_3 and its right images are im_1 and im_2. Since the start orientation of the vehicle is θ, as shown in Figure 50, the right images are im_1 and im_4, but im_2 is behind the vehicle.

2. If there is *ccw* horse–shoe polygon in the world, how can we know which image is on the left and which is on the right on the same polygon (see Figure 51)? In this case, $\xi = [B_i : B_i]$.

3. We can not construct the connectivity graph if a world $\mathcal{W}$ consists of two polygons B_0 and B_1, where $\mathcal{W}$ is encircled by an outermost *cw* polygonal boundary B_0 and has one *ccw* polygonal obstacle B_1 inside the boundary (Figure 52), since every v-node of the connectivity graph is the common intersection of three or more Voronoi boundary segments.

Due to the above shortcomings, we need more information when we represent the path classes in order to simplify local motion planning. The use of the subpolygons (see Chapter III) will solve the above problems and give more information for the local motion planning task.

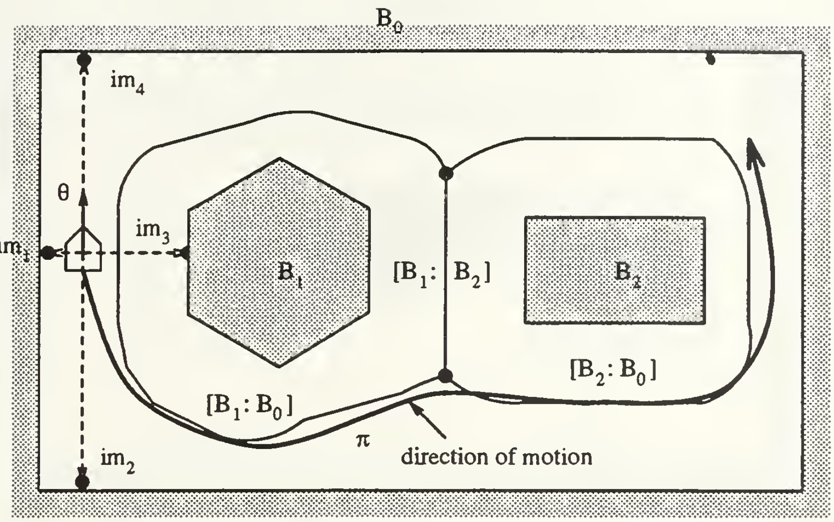


Figure 50. Problem 1: initial orientation of a vehicle is different from the direction of a motion

Consider the same world $\mathcal{W}$ in Figure 36. The nonconvex polygon B_0 can be broken into four subpolygons B_{00}, B_{01}, B_{02}, and B_{03} (see Figure 53). The basic connectivity graph generated by the world in Figure 53 is shown in Figure 54. There are six v-nodes $(v_1, \cdots, v_6)$ and seven undirected v-edges:

$$[B_1 : B_{00}], [B_1 : B_{01}], [B_1 : B_{03}], [B_2 : B_{01}], [B_2 : B_{02}], [B_2 : B_{03}, [B_1 : B_2].$$

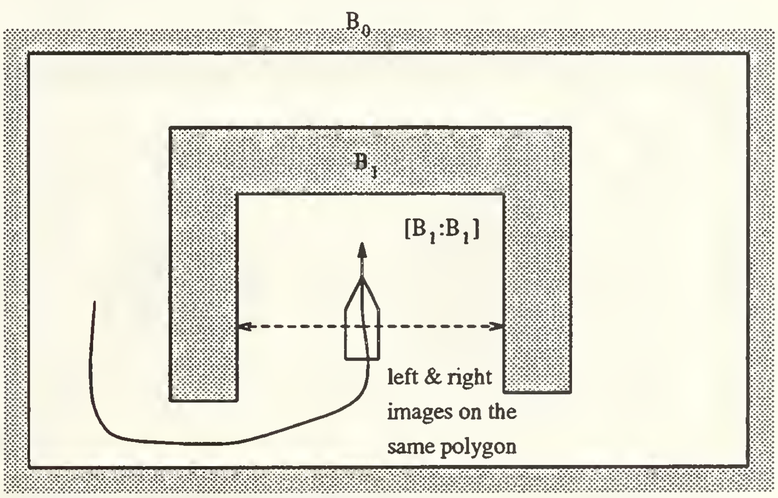


Figure 51. Problem 2: directed v-edge of a concave polygon

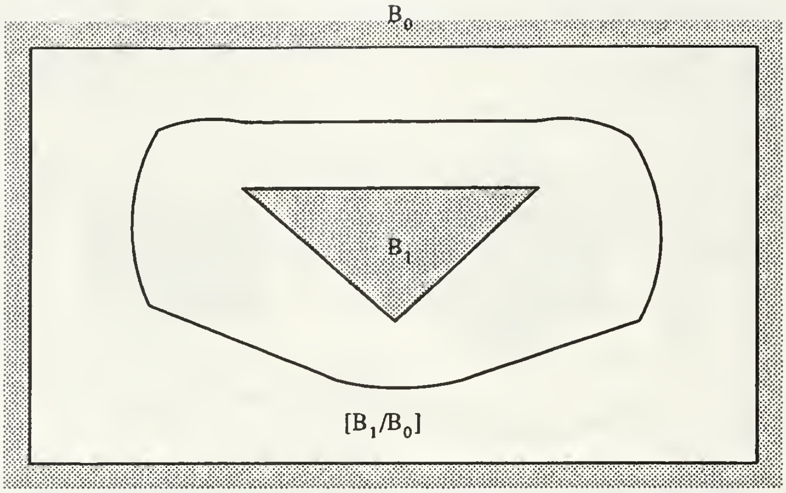


Figure 52. Problem 3: Voronoi diagram of polygonal world consisting of two polygons (*ccw* polygon inside *cw* polygon boundary)

Now, assume that a start configuration, S, and a goal configuration, G, are given in free($\mathcal{W}$) (see Figure 53). The augmented connectivity graph generated by this world is shown in Figure 55. In Figure 55, there are four different path classes represented by a directed v-edges sequences as follows:

$$\pi_1 = [B_{00}/B_1]\,[B_{03}/B_1]\,[B_{03}/B_2]\,[B_{02}/B_2]$$

$$\pi_2 = [B_{00}/B_1]\,[B_{03}/B_1]\,[B_2/B_1]\,[B_2/B_{01}]\,[B_2/B_{02}]$$

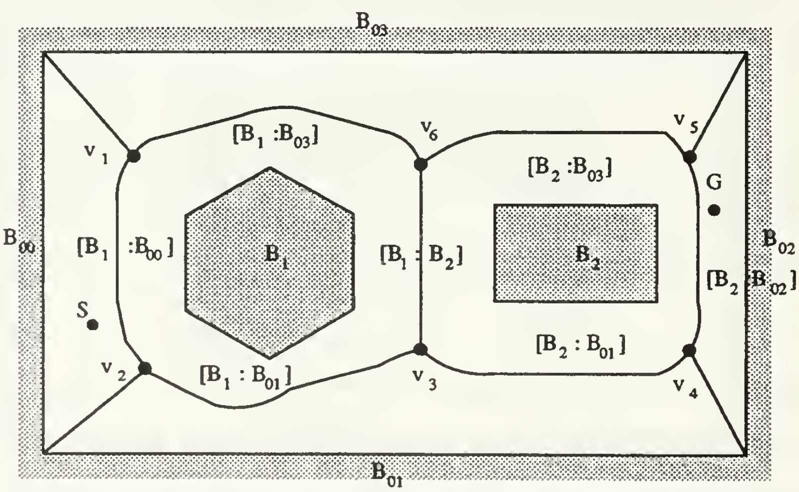


Figure 53. Solution of probelm 1: Voronoi diagram of a subpolygonal world

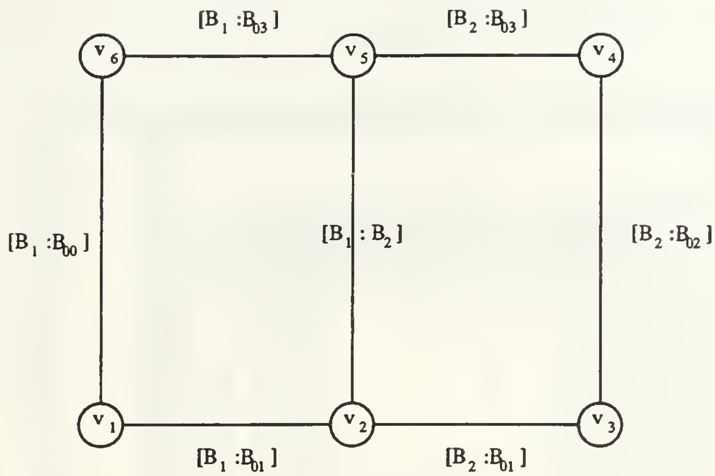


Figure 54. Basic connectivity graph of a subpolygonal world

$$\pi_3 = [B_1/B_{00}]\,[B_1/B_{01}]\,[B_1/B_2]\,[B_{03}/B_2]\,[B_{02}/B_2]$$

$$\pi_4 = [B_1/B_{00}]\,[B_1/B_{01}]\,[B_2/B_{01}]\,[B_2/B_{02}]$$

As a result, the use of subpolygons solves the problem when the start orientation of the vehicle is different from the direction of the motion. In other words, path classes represented by subpolygons possesses more information for local motion

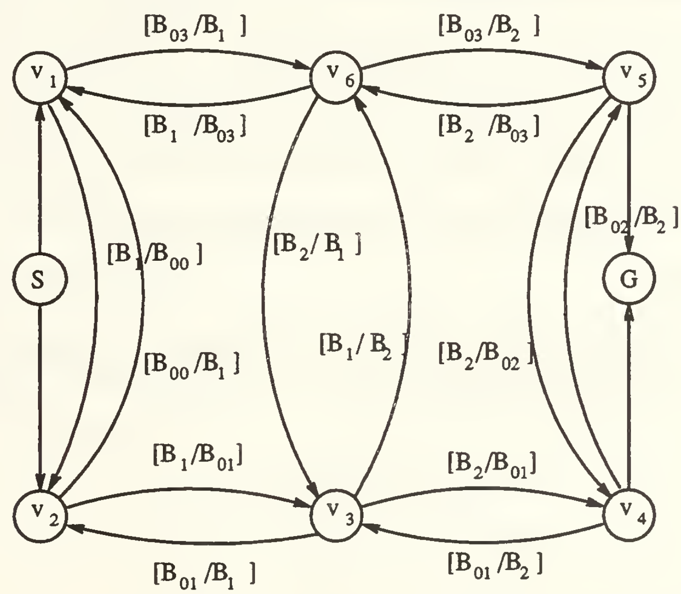


Figure 55. Augmented connectivity graph of a subpolygonal world

planning than do those represented by polygons.

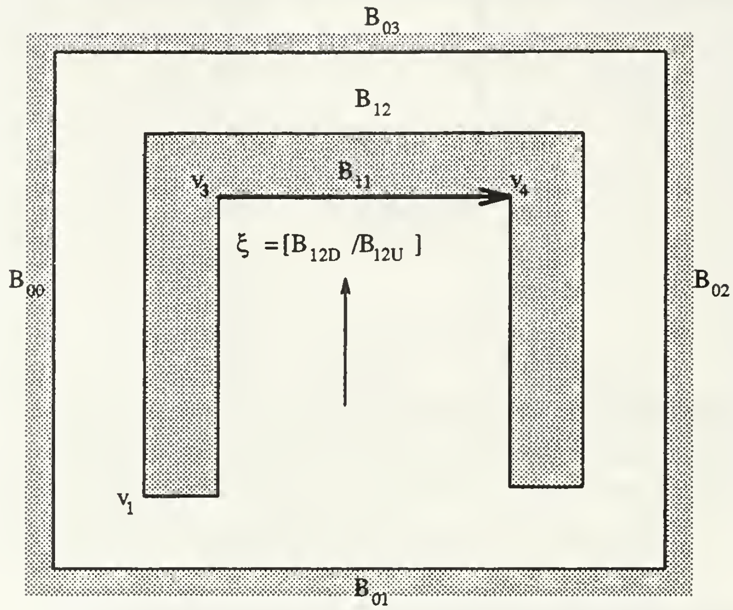


Figure 56. Solution of problem 2: up and down directed v-edges (I)

Now, we will discuss how we can solve the problem of a horseshoe-shaped polygon in the world. In Figure 51, polygon B_1 is decomposed into subpolyogons B_{11} and B_{12} (see Figure 56). In Figure 56,

$$B_{11} = \{v_3,\ v_4\}$$

and

$$B_{12} = \{v_4,\ v_5,\ v_6,\ v_7,\ v_8,\ v_1,\ v_2,\ v_3\} \tag{IV.4}$$

are two subpolygons.

Another example is shown in Figure 57. Polygon B_1 is decomposed into four subpolyogons B_{11}, B_{12}, B_{13}, and B_{14} where:

$$\begin{aligned} B_{11} &= \{v_4,\ v_5\} \\ B_{12} &= \{v_5,\ v_6\} \\ B_{13} &= \{v_6,\ v_7\} \\ B_{14} &= \{v_7,\ v_8,\ \cdots,\ v_4\} \end{aligned} \tag{IV.5}$$

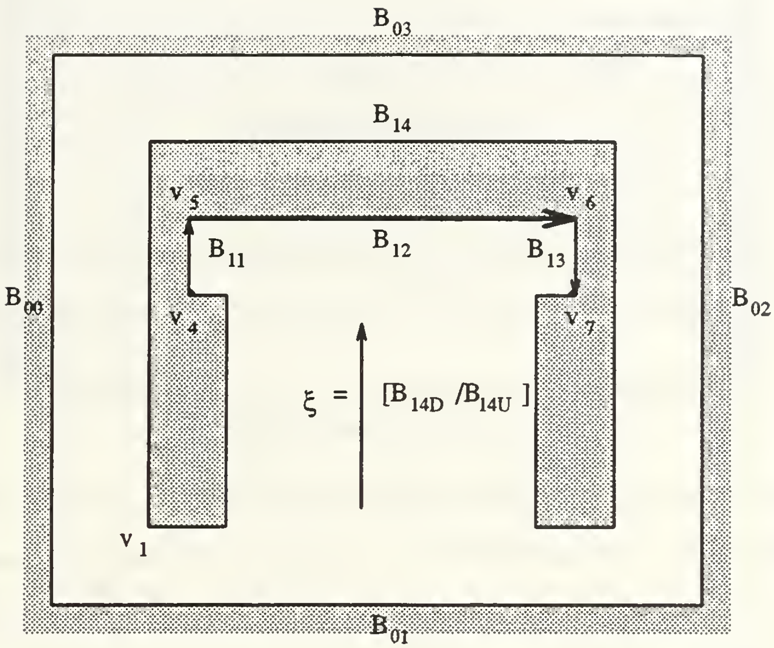


Figure 57. Solution of problem 2: up and down directed v-edges (II)

We have the following observations. In Eq. IV.4 (Figure 56), The first and last vertices of subpolygon B_{12} are v_4 and v_3 respectively. The right image is on the edge whose first vertex is v_4 $(\overline{v_4\varphi(v_4)})$. The left image is on the edge whose second vertex is v_3 $(\overline{\varphi^{-1}(v_3)v_3})$. In Eq. IV.5 (Figure 57), The first and last vertices of subpolygon B_{14} are v_7 and v_4 respectively. The right image is on the side whose first vertex is v_7. The left image is on the side whose second vertex is v_4. According to above observations, we have the following definition:

Definition: If left and right images are on the same subpolygon, then the directed v–edge is defined as follows:

$$\xi = [B_{iD}/B_{iU}]$$

or

$$\xi = [B_{iU}/B_{iD}]$$

where B_{iU} is subpolygon i associated with its first vertex and B_{iD} is subpolygon i associated with its last vertex.

For instance, in Figure 56,

$$\xi = [B_{12D}/B_{12U}]$$

where B_{12D} is the left side of subpolygon B_{12} (subpolygon B_{12} and last vertex v_3) and B_{12U} is the right side of subpolygon B_{12} (subpolygon B_{12} and first vertex v_4). In Figure 57,

$$\xi_1 = [B_{14D}/B_{14U}]$$

where B_{14D} is the left side of subpolygon B_{14} (subpolygon B_{14} and last vertex v_4) and B_{14U} is the right side of subpolygon B_{14} (subpolygon B_{14} and first vertex v_7).

The problem of constructing a connectivity graph when a world $\mathcal{W}$ consists of only two polygons B_0 and B_1 is solved by using the idea of subpolygons (see Figure 58). In Figure 58, there are two different path classes:

$$\begin{aligned}\pi_1 &= [B_{01}/B_1]\,[B_{00}/B_1]\,[B_{03}/B_1]\\ \pi_2 &= [B_1/B_{01}]\,[B_1/B_{02}]\,[B_1/B_{03}]\end{aligned}$$

E. ADVANTAGES OF PATH CLASS REPRESENTAION USING DIRECTED V-EDGES SEQUENCES

There are several advantages. They include:

1. A unique representation of a path class. In other words, this representation is unambiguous since a directed v-edge is defined by the "closest" two obstacle features.

   For example, in Figure 59, the directed v-edges sequence Ξ is

   $$\Xi = [B_1/B_0][B_1/B_4][B_2/B_3][B_0/B_3].$$

   In directed v-edge $\xi = [B_1/B_4]$, the directed boundaries of B_1 and B_2 are the same (ccw). The path direction goes ccw with left polygon B_1 and cw with right polygon B_4, then a left turn is required.

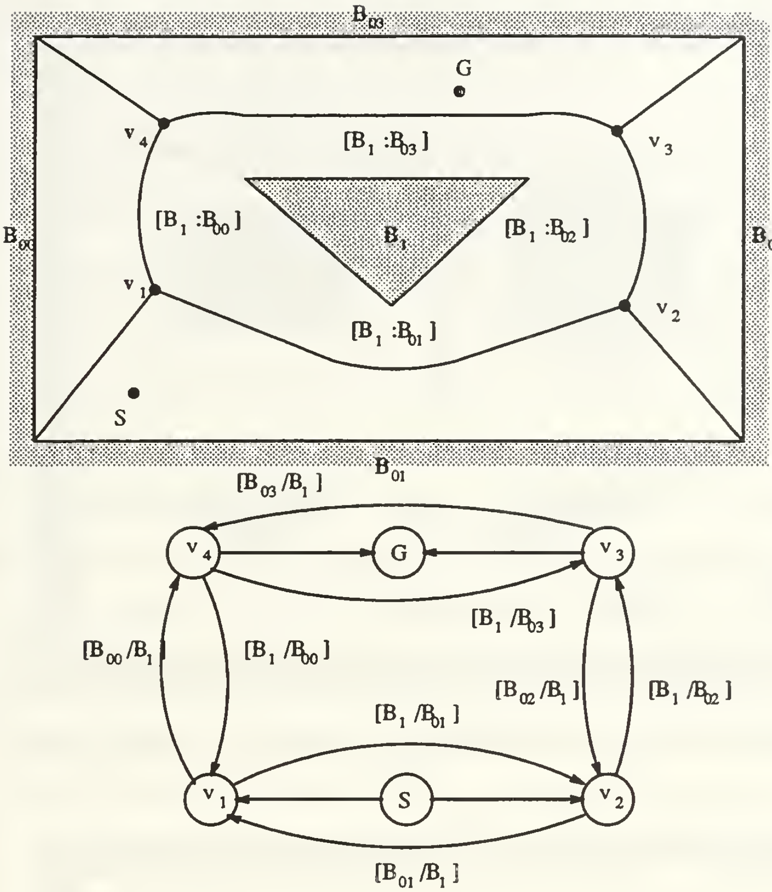


Figure 58. Solution of problem 3: world and augmented connectivity graph

In Figure 60, the directed v-edges sequence Ξ is

$$\Xi = [B_1/B_0][B_4/B_0][B_3/B_0].$$

In directed v-edge $\xi = [B_4/B_0]$, the directed boundaries of B_2 and B_0 are different (*ccw* and *cw*). The path direction goes *ccw* with left polygon B_4 and *cw* with right polygon B_0, and no turn is required.

2. It is an exact free space decomposition, so that if a path exists, the local motion planning should be able to find it.

3. It simplifies the planning of collision-free paths for a robot among obstacles once the directed v-edge sequence in which the robot is located is identified.

4. The local motion planning problem becomes simpler if a path class representing by directed v-edge sequence is given.

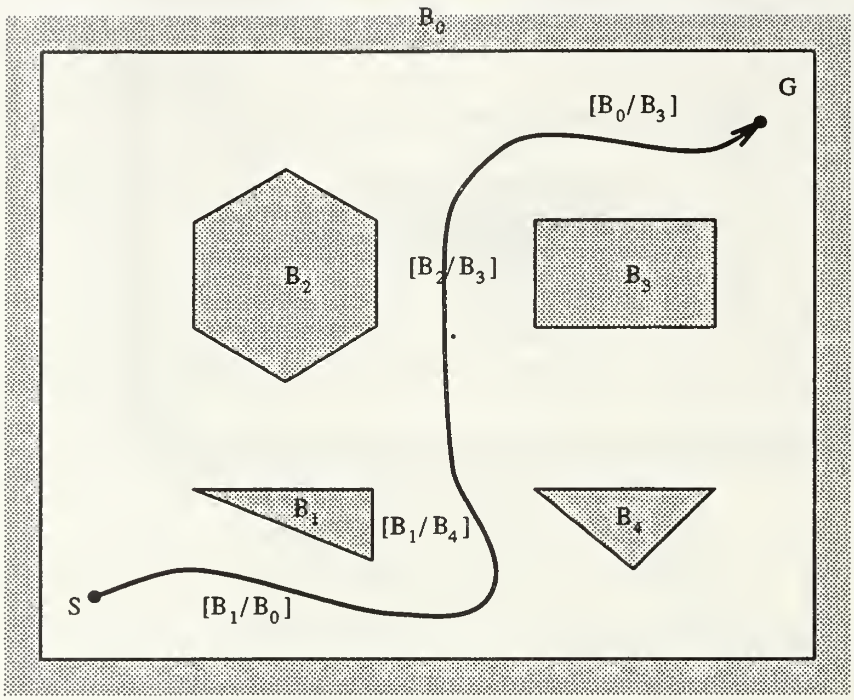


Figure 59. Directed v-edges sequence (left turn is required)

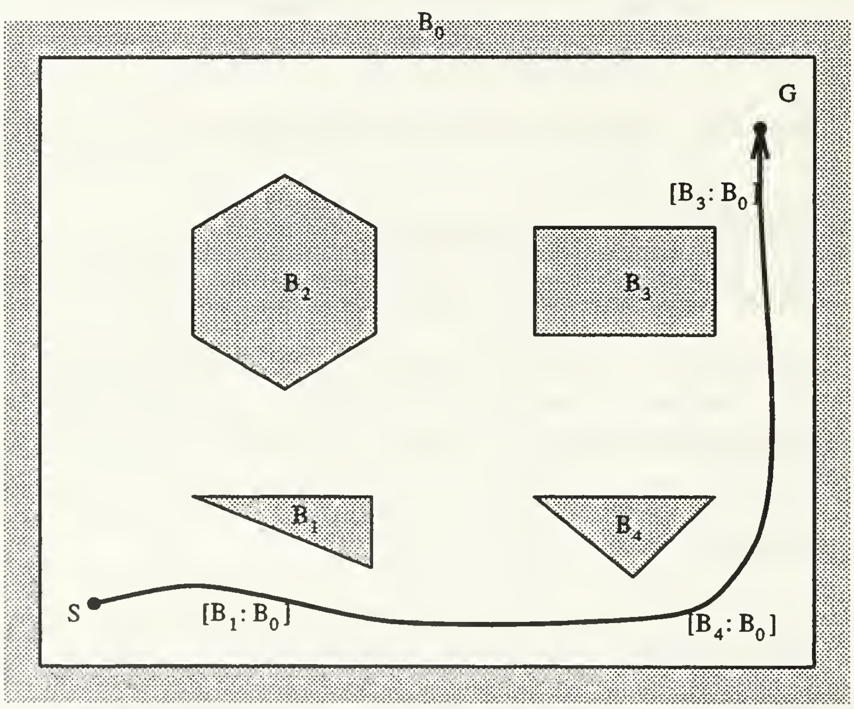


Figure 60. Directed v-edges sequence (no turn is required)

V. POLYGON TRACKING MOTION

This chapter addresses an approach to the tracking of polygons. This new method is based on the fact that obstacles are present in the working environment and they exhibit edges and corners (vertices). When a vehicle is moving, it recognizes its images on these obstacles and we can know the distance between the vehicle and those obstacles using a function called *steering function*, which takes data such as the distances, directions to its image on the boundary, and the desired curvature (the concept of steering function will be discussed in Section B). Therefore, it is possible for a vehicle to travel in the free space along the outer boundaries of obstacles and to keep a certain safety clearance (safety clearance function is defined in Section C). Since keeping a clearance from objects is important in polygon tracking motion, the robot will travel along a polygon's outer edges with clearance required. But when a vertex is eventually met, the robot needs to change its orientation to keep following the object. While the robot is changing its heading orientation, it is traveling past the vertex of a polygon, trying to keep the required clearance from the object so that it can continue to perform the same motion when an edge is available again. This Chapter proposes a few measurements which can be used in order to choose among several alternative paths (see Section D). The problem of how to make smooth motion when the vehicle gets close to the intersection of two distinct segments will be discussed in Section E. We have three different tracking techniques:

1. Edge–Convex Vertex Tracking (see Section F),
2. Convex Vertex Tracking (see Section G), and
3. Edge–Concave Vertex Tracking (see Section H).

A. PROBLEM STATEMENT

Given a *ccw* (*cw*) polygon B, the initial configuration $q = (p, \theta, \kappa)$ of a vehicle (p, θ, and κ are its position, orientation and curvature respectively), a safety clearance

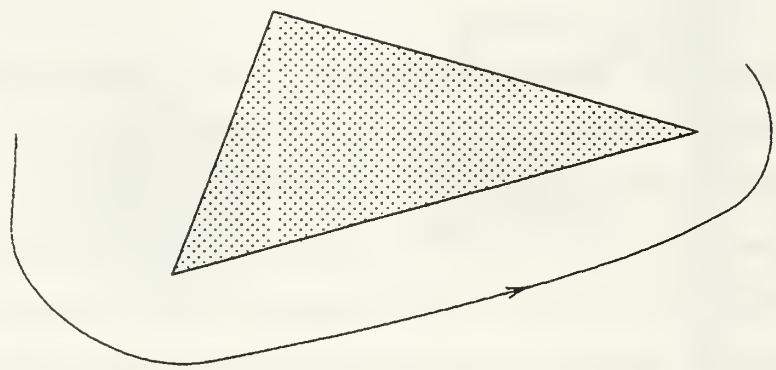


Figure 61. ccw tracking direction

$d_0 > 0$, and path direction (*ccw* or *cw*) (see Figures 61 and 62), we are trying to find a path of the vehicle starting from q (Figure 63) satisfying the following conditions:

1. Its path curvature is continuous, and
2. The total safety cost of the path is minimized (see Section D).

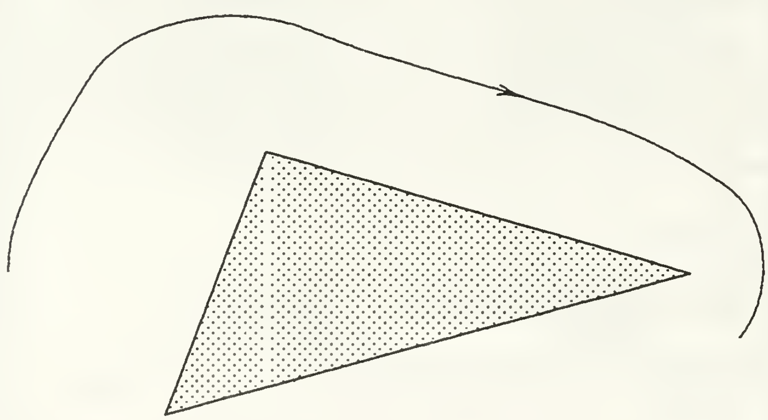


Figure 62. cw tracking direction

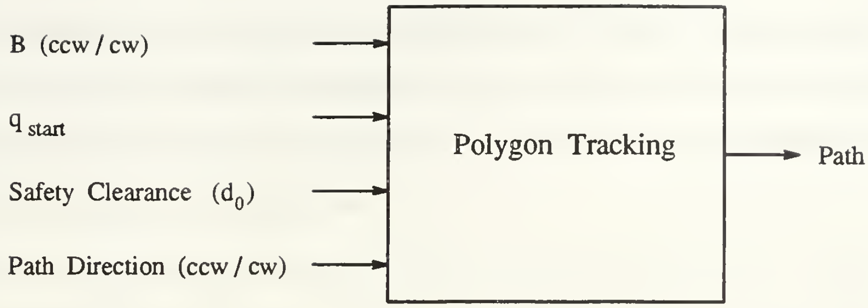


Figure 63. Block diagram for polygon tracking

B. GENERAL CONCEPTS OF THE STEERING FUNCTION

The mathematical framework that is used while working with steering functions is now described. First, only curves in the two-dimensional plane are considered, using the Euclidean space E^2 as the work space. A path will be described by a curve C which is a function of path length, s. By the fundamental existence and uniqueness theorem for plane curves, if $\kappa(s)$ is an arbitrary continuous function on a closed interval $[a, b]$, then there exists one and only one curve C for which $\kappa(s)$ is the curvature and s is a natural parameter along C. Hence, the curve is completely and uniquely described by the initial position, orientation, and curvature κ [27, 31, 63].

Second, a vehicle's *configuration* q is defined as

$$q(s) \equiv (p(s), \theta(s), \kappa(s)) \tag{V.1}$$

where $p(s)$, $\theta(s)$ and $\kappa(s)$ are its position, orientation, and curvature.

Each non-holonomic vehicle has two degrees of freedom: the translational speed v and rotational speed ω. Since a non-holonomic robot's heading orientation θ is always equal to the trajectory's tangent orientation, the vehicle's rotational speed ω is equal to κv, where κ is the path curvature (because $\omega = d\theta/dt = (d\theta/ds)(ds/dt) = \kappa v$, where t is time and s is the traveling length of the robot). Therefore, the smooth motion planning of a robot vehicle is designing (κ, v) or (ω, v) as functions of t or s. This control model with curvature is useful for vehicles with any kinematics [35].

In a real vehicle's path, it is well-known that the vehicle heading direction and the curvature must be continous [37]. The local motion planning problem is therefore the problem of how to control the curvature κ. One obvious method is to compute the curvature directly as a function of the geometrical constraints and the mission. However, one drawback of this method is that when some of the input has a discontinuity from the previous value, the output κ also tends to be discontinuous. As widely known, rigid body motion with a discontinuous curvature function is not physically realizable. Curvature continuity is essential in the local motion planning because a discontinuity in vehicle acceleration may cause wheel slippage which will add to odometery errors. In order to solve this problem, we take the derivative of the curvature $d\kappa/ds$ instead of the curvature κ itself as a control variable. As long as $d\kappa/ds$ takes on a finite value, the curvature continuity is guaranteed and the trajectory becomes smooth. Therefore, the "optimal" function f in an equation

$$\frac{d\kappa}{ds} = f(E, M, q)$$

for a rigid body vehicle is called a *steering function*, where E is the current environment, M the mission, and q the vehicle configuration. After computing this value $d\kappa/ds = f$, the curvature κ is updated through the incremental movement Δs. As long as f is the value of finite, a vehicle's trajectory obtained is "smooth" in the sense that the tangent orientation, curvature and derivative of curvature exist on every point on the trajectory. In this mathematical model, we understand the vehicle's curvature is not rapidly changed, hence, we include κ in the vehicle's configuration as shown in Eq. V.1. We adopt the following general form for the steering function that works in all situations we have applied:

$$\begin{aligned} \frac{d\kappa}{ds} &= -(a\Delta\kappa + b\Delta\theta + c\Delta d) \\ &\equiv -(a(\kappa - \kappa_d) + b(\theta - \theta_d) + c\Delta d), \end{aligned} \tag{V.2}$$

where a, b, and c are positive constants. Also, κ is the path curvature, θ the vehicle's heading (which is equal to the path tangential direction), κ_d the desired curvature, and θ_d the desired heading direction. This steering function can be applied to various motion planning situations. The definitions of κ_d, θ_d, and Δd are defined according to situations as we will see in the Sections F, G, and H. The meanings of these variables, $\Delta\kappa$, $\Delta\theta$, and Δd, are as follows:

1. $\Delta\kappa$ is the difference between the current vehicle's curvature κ and the desired curvature κ_d.
2. $\Delta\theta$ is the difference between the current vehicle's orientation θ and the desired orientation θ_d.
3. Δd is the difference between the current and desired positions and is a signed number. For instance, if the robot is tracking a directed reference path, Δd is the signed distance from the vehicle position to the directed path.

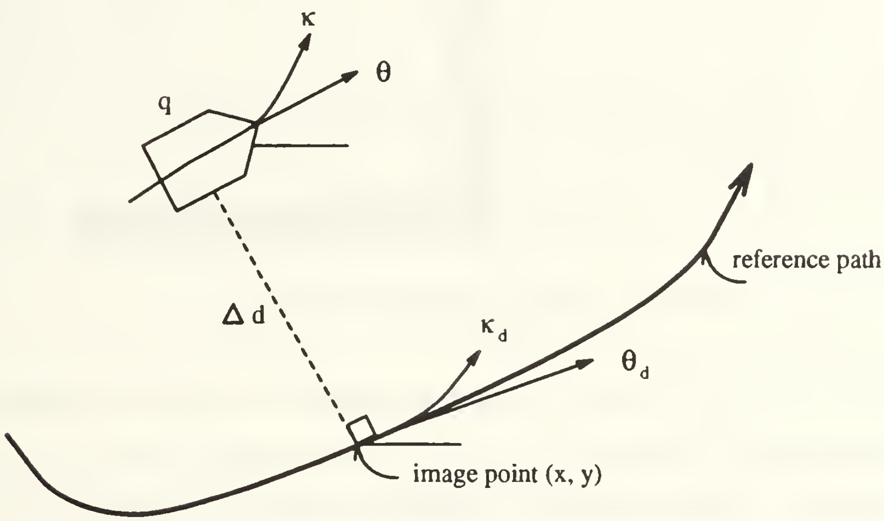


Figure 64. Geometrical concepts of steering function

Figure 64 illustrates the geometric concepts involved with a steering function used to follow a reference path. The closest point on the reference path from the robot's configuration is called the *image point*. A signed distance value, Δd, is used to represent the shortest distance between the robot's current configuration and the image located in the reference path. The sign of Δd depends on the robot's position relative to the reference path. When $\Delta d > 0$, the robot is to the left of the reference path and when $\Delta d < 0$, the robot is to the right of the reference path. Therefore, Δd is a signed distance indicating how far the robot is located from a reference path.

For details on the steering function and an argument as to why the steering function works, see [36].

C. CLEARANCE DEFINITION

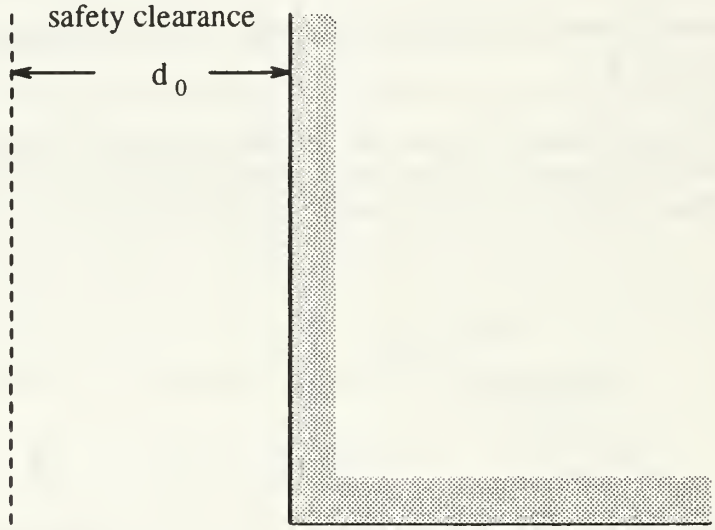


Figure 65. Robot's safety clearance (I)

In this dissertation, we take safety as the single characteristic of motions to be optimized. The polygon tracking problem is the one of planning a motion for a vehicle to track a flat wall in parallel to it with a given safety clearance (see Figure 65). If the distance between the robot and polygon is less than this safety clearance, the robot must try to make the distance to the left/right boundaries greater than this safety clearance using non-linear safety clearance function $g(d)$ (see Figure 66).

Definition: the clearance d_1 is defined as the distance from the robot's outside edge of the wheels to the object (Figure 67).

If d_1 is supplied by sensors instead of as information extracted from the model, the clearance d_1 indicates how far the object is from the sensor.

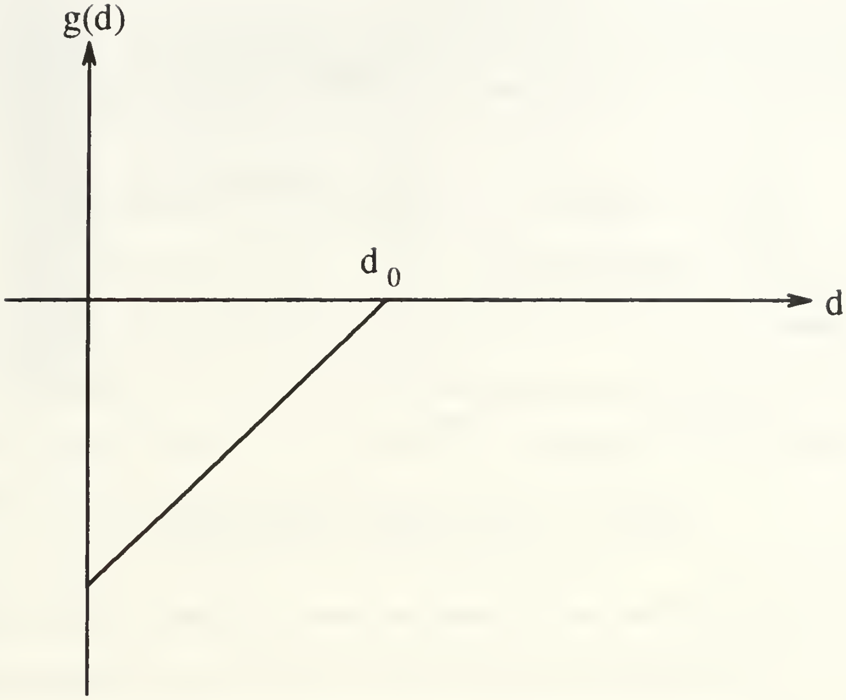


Figure 66. Non-linear safety clearance function

Definition: the robot's *safety clearance* d_0 is defined by

$$d_0 = d_1 + \frac{1}{2} width, \tag{V.3}$$

where width is the robot's width. See Figure 67.

Definition: Let d be the distance between the robot and polygon. The safety clearance function $g(d)$ is defined by

$$g(d) = \begin{cases} d - d_0 & \text{if } d < d_0 \\ 0 & \text{otherwise} \end{cases}, \tag{V.4}$$

where $g : \mathcal{R} \rightarrow \mathcal{R}$ is a nonlinear function defined as in Figure 66.

D. COMPARING PATH ALTERNATIVES

Currently, a quantitative technique for comparing alternative paths is needed. Our problem is: to compare two or more alternative paths in order to select the best

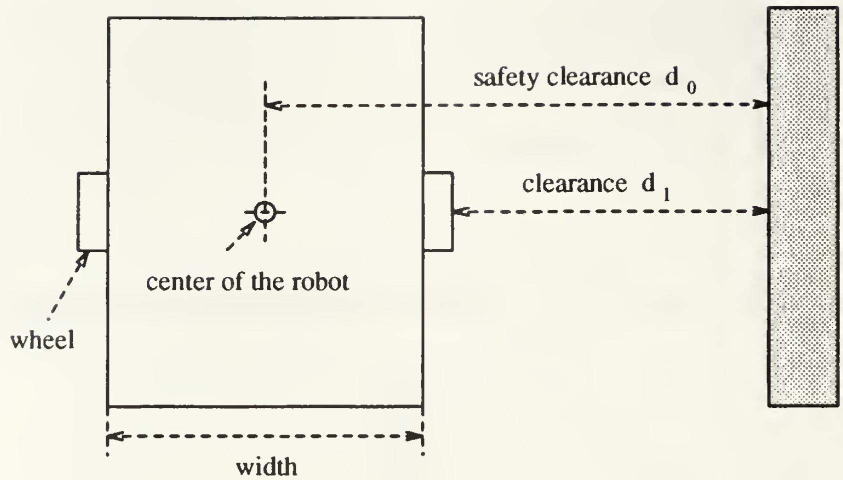


Figure 67. Robot's safety clearance (II)

one. Only a few attributes may be used to describe a path. These attributes include length, smoothness and safety. Path safety is the most important property, and path smoothness is desirable to ameliorate odometry errors and to decrease travel time along the path due to the ability to use higher velocities on paths with lower curvatures. Based on the stipulated mission parameters, the cost function for path comparison may be found. By evaluating the penalties associated with path attributes, the path which minimizes the cost function can be chosen as the best of the alternative paths for a given mission.

1. Safety Cost Function

Generally, path safety is a function of the distance of the vehicle to an obstacle. As the distance decreases, the safety decreases. The safest path is one in which the distance to the obstacle is maximum. In many cases, a vehicle should not approach closer to the obstacle than the given safety range. A path is unsafe if the distance to the obstacle is less than or equal to zero.

One way of planning safer paths is to maintain a constant clearance for every point on a path [57, 54]. However, the constant clearance method is still not ideal for two reasons:

1. when the vehicle is moving in a tight space, a smaller clearance may be tolerated. On the other hand, when the vehicle is moving in a wider space, a larger clearance may be required in order to move the vehicle faster and to ease positional control.

2. the initial position of the vehicle may be with null clearance.

Another approach is using a cost for safety [64]. Here, the cost of a path is defined as the sum of costs for its length and for its safety. The safety component of the cost is a function of the integration of the distance between a point on a path and the center line. Therefore, this algorithm does not give any solution if the area is not delimited by a center line or by a Voronoi boundary.

In this dissertation, we use the following approach. A path in free space is a pair (s_1, f) consisting of a positive real number s_1 and a continuous function f. The length of path from the point $p(0)$ to a point $p(s)$ along a path(s_1, f) is equal to s if $0 \leq s \leq s_1$. Let $\gamma(p)$ denote the distance between a point p to a polygon B. Let $p(s)$ denote a vehicle position at s on the path. The total safety cost of a path(s_1, f) is given by a positive cost function $\Gamma : \mathcal{R} \rightarrow \mathcal{R}$ defined by

$$\Gamma = \int_0^{s_1} \left[\gamma(p(s)) - d_0\right]^2 ds, \tag{V.5}$$

Generally, a path farther from obstacles is safer, but tends to be longer. Therefore, we need to strike a balance between smoothness and safety of a path. There is a positive parameter σ in the steering function, which controls the smoothness of the resultant trajectory. If a smaller σ is used, the trajectory becomes sharper and the path becomes safer, and if a larger σ is used, the trajectory becomes smoother and the path becomes more dangerous. As the smoothness parameter σ becomes large, the path converges to the smoothest path. Thus, we obtain a class of paths with different weight between safety and smoothness in an equivalent class.

2. Smoothness Cost Function

Smoothness of path is essential for mobile robot navigation because unsmooth motions may cause slippage of wheels which degrades the robot's dead reckoning ability. A path that does not posses tangential or curvature continuity surely is not smooth. These types of paths will not be allowed as alternative paths due to the severity of the lack of smoothness. In order to control smoothness of paths, we define the cost of a path for smoothness. A unit cost for smoothness at a point $p(s)$ on a path is proposed as the square of the derivative of its curvature [37]. The total smoothness cost of a path is given by a positive cost function $\Sigma : \mathcal{R} \rightarrow \mathcal{R}$ defined by

$$\Sigma = \int_0^{s_1} \left(\frac{d\kappa}{ds}\right)^2 ds, \tag{V.6}$$

E. COMBINING STEERING FUNCTIONS

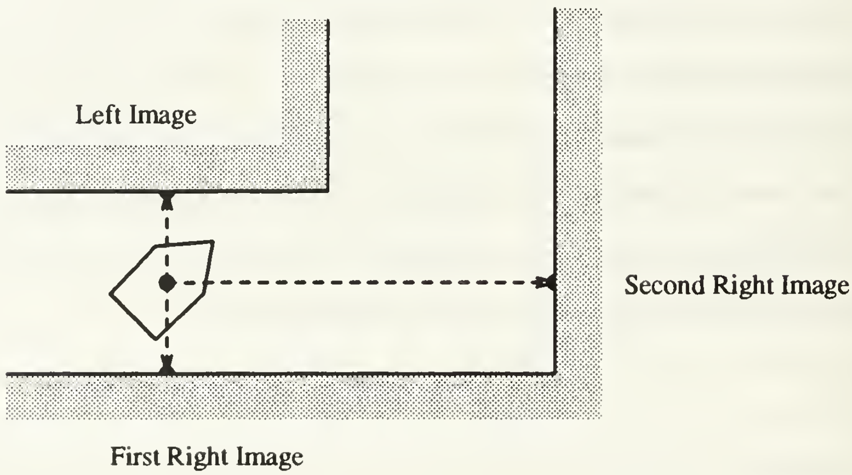


Figure 68. First and second images

The new problem to be solved in this dissertation is that of how to achieve a smooth motion when the vehicle gets close to the intersection of two distinct subpaths (for instance from a line segment to a circle segment). In order to solve this problem, we will watch second images in the forward portion of a left or right boundary, and will make a smooth motion by evaluating the steering function using not only the left/right first images, but the left/right second images too (see Figure 68). That is, we evaluate two steering functions with the first and the second images and take a value by combining these two function results. Thus, resulting paths will be "smoothed" using an appropriate smoothness σ.

First, let the weighting functions ω_1 and ω_2 are defined as:

$$\omega_1 = \exp\left[-\frac{d_1}{\sigma}\right] \quad \text{(V.7)}$$

$$\omega_2 = \exp\left[-\frac{d_2}{\sigma}\right] \quad \text{(V.8)}$$

where d_1 and d_2 are the distance between p and its first left (right) and second left (right) images respectively. These weighting functions are dimensionless.

If a second image is far from the vehicle, the effect of its steering function is very small. When a second image gets closer, its steering function effect increases. We evaluate two steering functions with the first and the second images and take a value by combining these two function results by using the above weighting functions. For instance, consider a situation where the first left image occurs on an edge of left obstacle and the first and second right images occur on an edge of the right obstacle(s) also. Let f_l, f_{r1}, and f_{r2} denote the steering functions of the left, first, and second right images respectively. By combining the first and second right steering functions, we obtain

$$f_r = \frac{\omega_1}{\omega_1 + \omega_2} f_{r1} + \frac{\omega_2}{\omega_1 + \omega_2} f_{r2}, \quad \text{(V.9)}$$

where f_r is right steering function obtained by combining f_{r1} and f_{r2}.

Now, the steering function f for left and right images is obtained by

$$f = f_l + f_r.$$

F. EDGE–CONVEX VERTEX TRACKING

While an image of a vehicle's position occurs on an edge of polygon and the vehicle is trying to keep itself away from the edge with a safety distance d_0, it is following an edge of the polygon. We say that the vehicle in *Edge–Convex Vertex Tracking Mode.* The vehicle has two distinct images $p_{im1} = (x_{im1}, y_{im1})$ and $p_{im2} = (x_{im2}, y_{im2})$ and the vehicle looks at p_{im1} and p_{im2} as the first and second images respectively (see Figures 69, 70). Because an edge is a straight line, the vehicle is supposed to track a directed straight line. By applying the steering function in Eq. V.2, we will evaluate two steering functions for the first and second images and take a new steering function value by combining these two function results using Eqs. V.7,V.8 and V.9. Now, we will explain how to formulate $\frac{d\kappa}{ds}$, the steering function, in Eq. V.2 for each image.

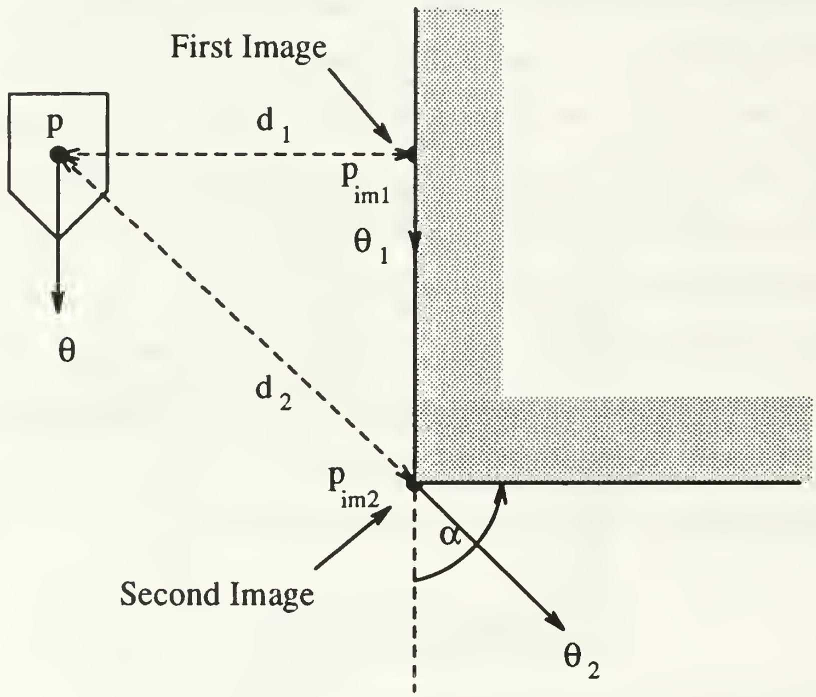


Figure 69. ccw tracking in Edge-Convex Vertex Tracking Mode

Let the current configuration of a vehicle be defined as

$$q = (p, \theta, \kappa), \tag{V.10}$$

where p, θ and κ describe the robot's current position, orientation, and curvature, respectively.

For the first image p_{im1}, the variables κ_{d1}, θ_1, and $d1$ in the steering function (Eq. V.2) can be computed as follows.

The desired curvature of the edge is zero because we assume the edge is flat like a line.

$$\kappa_{d1} = 0.$$

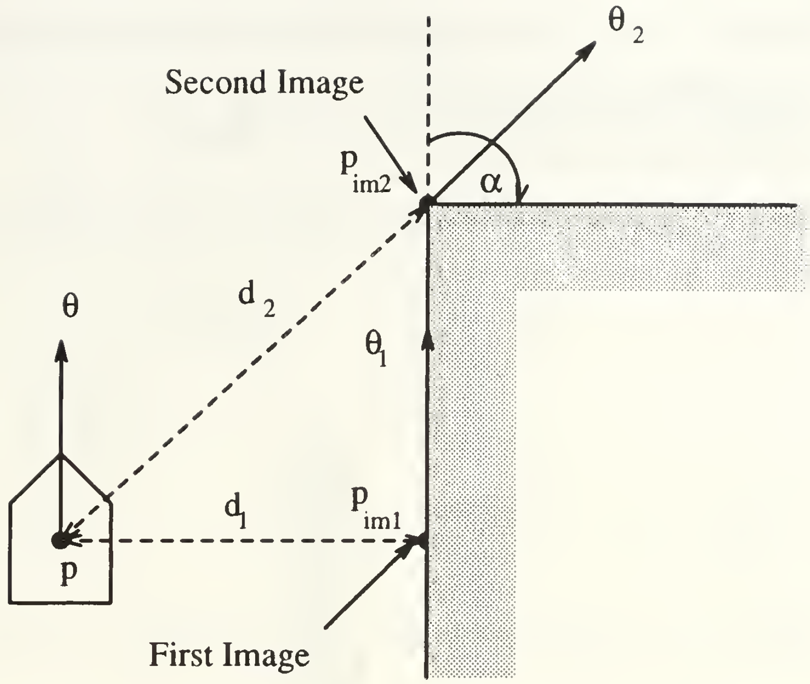


Figure 70. cw tracking in Edge-Convex Vertex Tracking Mode

Let $\Psi(p, p_{im1})$ denote the orientation from p to p_{im1}. The desired orientation θ_1 is evaluated as following:

1. If the image of p on the edge is on the right of the vehicle (Figure 70), then

$$\theta_1 = \Psi(p,\ p_{im1}) + \frac{\pi}{2}.$$

2. If the image of p on the edge is on the left of the vehicle (Figure 69), then

$$\theta_1 = \Psi(p,\ p_{im1}) - \frac{\pi}{2}.$$

The distance, d_1, is the signed distance from the vehicle position p to its image p_{im1}. This signed distance satisfies the condition that $d_1 < 0$ if the edge is on the vehicle's left side while $d_1 > 0$ if the edge is on the vehicle's right side. In Chapter III, Section E, we showed how to evaluate the distance between any point in free space to its image on an obstacle d_1. By Eq. V.4, we calculate the safety clearance function $g(d_1)$ as follows:

1. if the image of p on the edge is on the right of the vehicle (Figure 72), then

$$g(d_1) = \begin{cases} d_1 - d_0 & \text{if } d_1 < d_0 \\ 0 & \text{otherwise} \end{cases}$$

2. if the image of p on the edge is on the left of the vehicle (Figure 71), then

$$g(d_1) = \begin{cases} d_1 + d_0 & \text{if } |d_1| < d_0 \\ 0 & \text{otherwise} \end{cases}$$

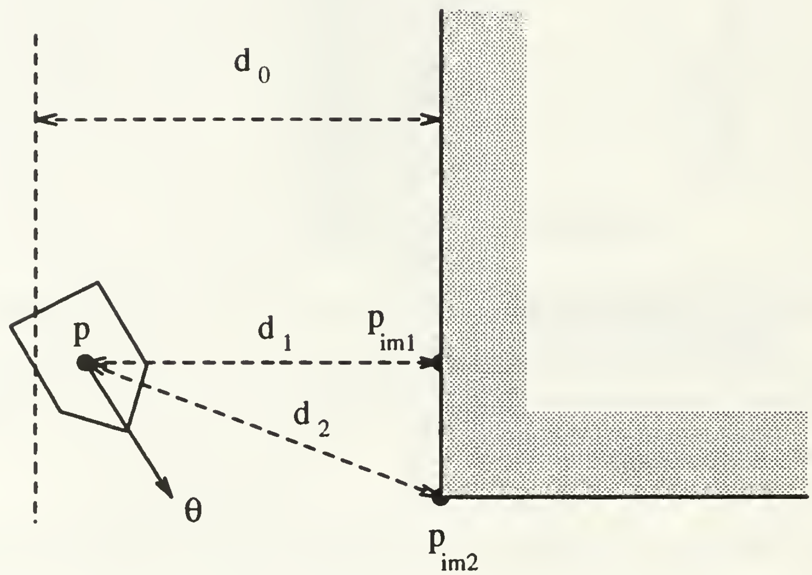


Figure 71. Calculate safety clearance function of ccw tracking

Thus the steering function in Eq. V.2 becomes

$$f_1 = -(a\,\kappa + b(\theta - \theta_1) + c\,g(d_1)).$$

For the second image p_{im2}, the variables κ_{d2}, θ_2, and $d2$ in the steering function (Eq. V.2) can be computed similarly (see Figures 69, 70).

The desired curvature κ_{d2} is

$$\kappa_{d2} = 0. \tag{V.11}$$

The desired orientation θ_2 is evaluated as following:

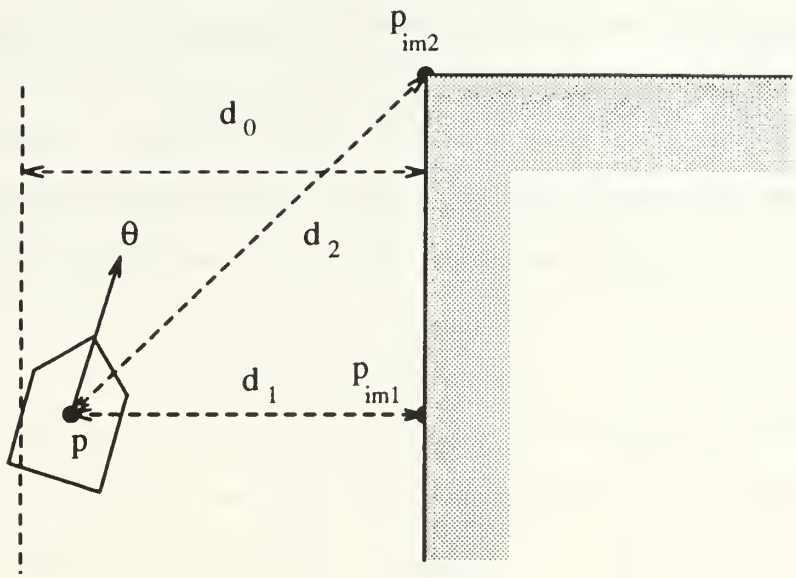


Figure 72. Calculate safety clearance function of cw tracking

1. If the image of p on the edge is on the right of the vehicle (Figure 70), then

$$\theta_2 = \theta_1 - \frac{\alpha}{2}.$$

2. If the image of p on the edge is on the left of the vehicle (Figure 69), then

$$\theta_2 = \theta_1 + \frac{\alpha}{2}.$$

where θ_1 is the desired orientation of the first image and α is the exterior angle induced at p_{im2}, the second image, (see Figure 69, 70).

Similarly, we compute the distance, d_2, and safety clearance function, $g(d_2)$, as

1. If the image of p on the edge is on the right of the vehicle (Figure 72), then

$$g(d_2) = \begin{cases} d_2 - d_0 & \text{if } d_2 < d_0 \\ 0 & \text{otherwise} \end{cases}$$

2. If the image of p on the edge is on the left of the vehicle (Figure 71), then

$$g(d_2) = \begin{cases} d_2 + d_0 & \text{if } |d_2| < d_0 \\ 0 & \text{otherwise} \end{cases}$$

Thus for the second image, the steering function in Eq. V.2 becomes

$$f_2 = -\left(a\,\kappa + b(\theta - \theta_2) + c\,g(d_2)\right).$$

Now, by combining f_1 and f_2 using Eqs. V.7,V.8 and V.9, we obtain the total steering function value while the robot is in Edge–Convex Vertex Tracking Mode:

$$f = \frac{\omega_1}{\omega_1 + \omega_2} f_1 + \frac{\omega_2}{\omega_1 + \omega_2} f_2 .$$

Figure 73 shows some numerical simulation results. The following simulation results are obtained using different smoothness values. The effect of using distinct values of smoothness with $\sigma = 5, 10, 20$ and 40 is clearly shown in the figure. As σ increases, the safety cost function defined in Eq. V.5 increases.

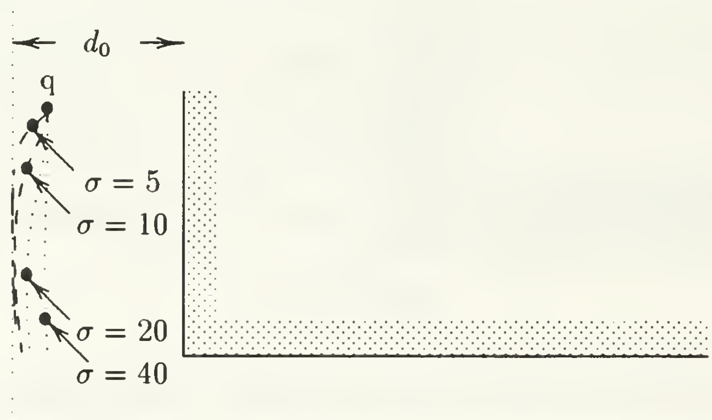


Figure 73. Different trajectories corresponding to their safety cost function values in Edge-Convex Vertex Tracking Mode

G. CONVEX VERTEX TRACKING

When the vehicle is coming to the end of an edge, an image of the vehicle's position occurs on a vertex of polygon. In this case, to keep the desired safety clearance from the polygon, the vehicle needs to turn around the vertex in a circular motion taking the vertex as its center and safety distance d_0 as its radius. Here the vehicle is defined to be in *Vertex Tracking Mode.* In this mode, the vehicle has one image $p_{im} = (x_{im}, y_{im})$, and the vehicle looks at p_{im} on its left or right as the first and second images (see Figures 74, 75). We will evaluate two steering functions for the first and second images and take a new steering function value by combining these two function results using Eqs. V.7,V.8 and V.9. Now, we will explain how to formulate $\frac{d\kappa}{ds}$, the steering function, in Eq. V.2 for each image.

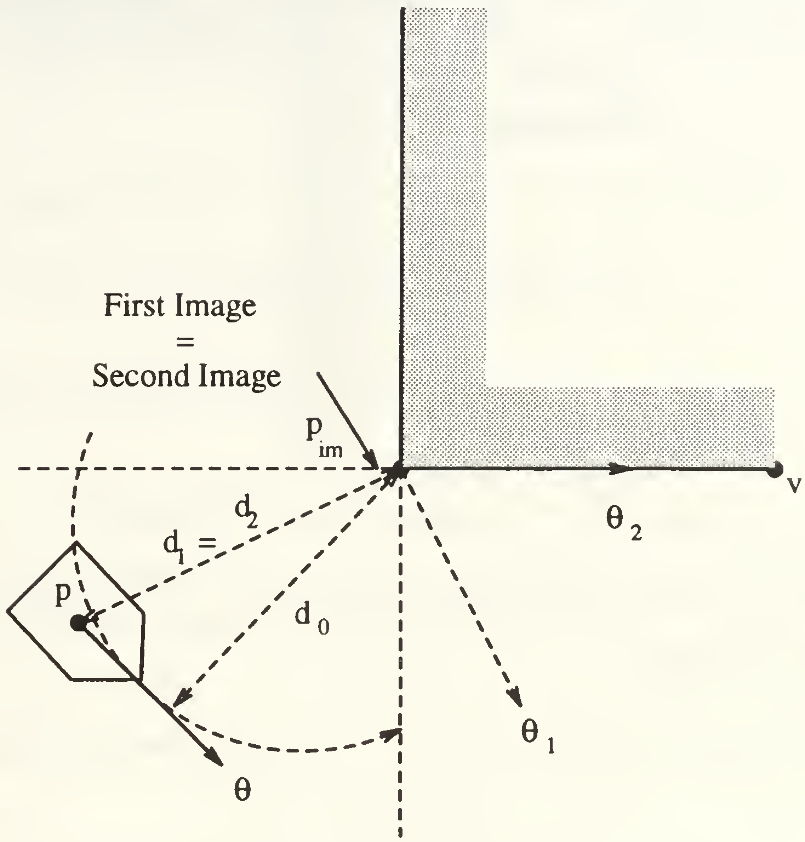


Figure 74. ccw tracking of Vertex Tracking Mode

For the first image p_{im}, the variables κ_{d1}, θ_1, and $d1$ in steering function (Eq. V.2) can be computed as follows.

The desired curvature is the circle's radius d_0 because the vehicle needs to turn around the vertex in a circular motion taking the vertex as its center.

$$\kappa_{d1} = 1/d_0. \tag{V.12}$$

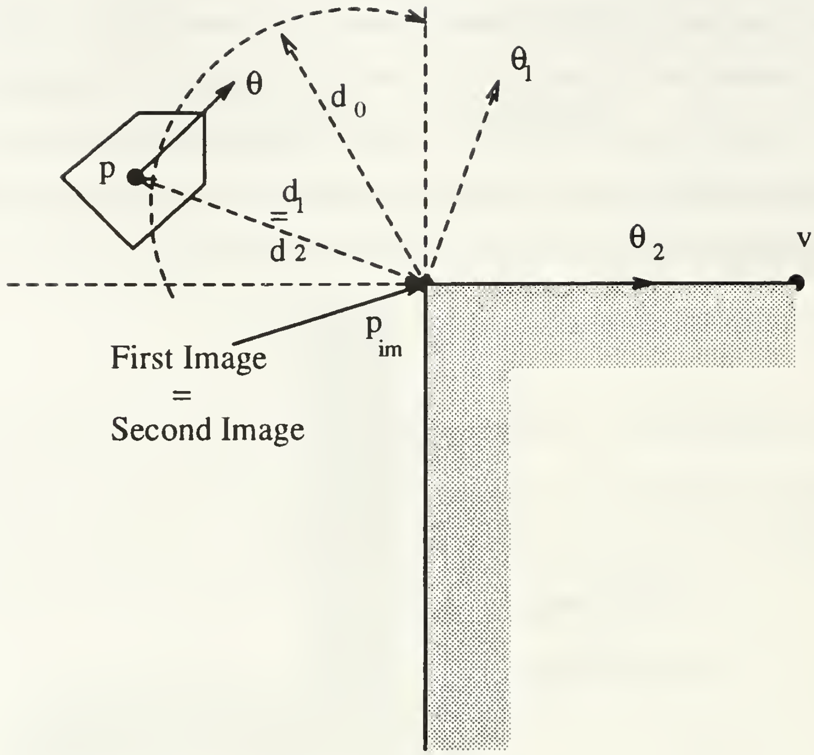


Figure 75. cw tracking of Vertex Tracking Mode

Let $\Psi(p, p_{im})$ denote the orientation from p to p_{im}. The desired orientation θ_1 is evaluated as following:

1. If the image of p on the vertex is on the right of the vehicle (Figure 75), then

$$\theta_1 = \Psi(p,\ p_{im}) + \frac{\pi}{2}.$$

2. If the image of p on the vertex is on the left of the vehicle (Figure 74), then

$$\theta_1 = \Psi(p,\ p_{im}) - \frac{\pi}{2}.$$

The distance, d_1, is the signed distance from the vehicle position p to its image p_{im}.

1. If the image of p on the vertex is on the right of the vehicle (Figure 75), then

$$d_1 \equiv \sqrt{(p.x - p_{im}.x)^2 + (p.y - p_{im}.y)^2}.$$

2. If the image of p on the vertex is on the left of the vehicle (Figure 74), then

$$d_1 \equiv -\sqrt{(p.x - p_{im}.x)^2 + (p.y - p_{im}.y)^2}.$$

The safety clearance function $g(d_1)$ is calculated as following:

1. If the image of p on the vertex is on the right of the vehicle, then

$$g(d_1) = \begin{cases} d_1 - d_0 & \text{if } d_1 < d_0 \\ 0 & \text{otherwise} \end{cases}$$

2. If the image of p on the vertex is on the left of the vehicle, then

$$g(d_1) = \begin{cases} d_1 + d_0 & \text{if } |d_1| < d_0 \\ 0 & \text{otherwise} \end{cases}$$

Thus the steering function in Eq. V.2 becomes

$$f_1 = -(a(\kappa - \kappa_{d1}) + b(\theta - \theta_1) + c\,g(d_1))\,.$$

For the second image p_{im}, the varaibles κ_{d2}, θ_2, and $d2$ in the steering function (Eq. V.2) have another meaning (see Figures 74, 75).

The desired curvature κ_{d2} is zero. In this case, we assume that p_{im} is on the edge $\overline{p_{im}v}$, where v is $\varphi(p_{im})$.

$$\kappa_{d2} = 0.$$

The desired orientation θ_2 is evaluated as following:

$$\theta_2 = \Psi(p_{im}, v).$$

The distance d_2 and safety clearance function $g(d_2)$ are the same as the first image.

Thus for the second image, the steering function in Eq. V.2 becomes

$$f_2 = -(a\,\kappa + b(\theta - \theta_2) + c\,g(d_2))\,.$$

By combining the above two steering function values f_1 and f_2 using Eqs. V.7,V.8 and V.9, we obtain the total steering function value while the robot is in vertex tracking mode:

$$f = \frac{\omega_1}{\omega_1 + \omega_2} f_1 + \frac{\omega_2}{\omega_1 + \omega_2} f_2.$$

Figure 76 hows some numerical simulation results. The following simulation results are obtained using different smoothness values. The effect of using distinct values of smoothness with $\sigma = 5, 10$ and 20 is clearly shown in the figure. As σ increases, the safety cost function defined in Eq. V.5 increases.

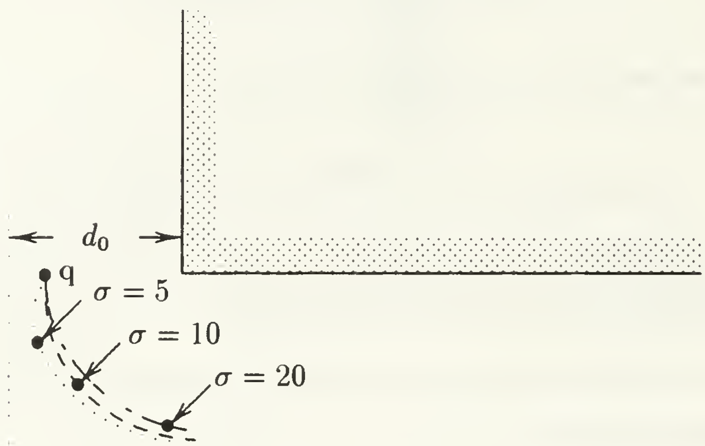


Figure 76. Different trajectories corresponding to their safety Cost Function Values in Vertex Tracking Mode

H. EDGE–CONCAVE VERTEX TRACKING

Suppose a vehicle is heading to a concave vertex (Figures 77, 78). While the vehicle is trying to keep itself away from the edge with a safety distance d_0, it is following an edge of the polygon. The image of a vehicle's position always lies on an edge. We say that the vehicle is in *Edge–Concave Vertex Tracking Mode.* The vehicle has two distinct images $p_{im1} = (x_{im1}, y_{im1})$ and $p_{im2} = (x_{im2}, y_{im2})$ such that the vehicle looks at p_{im1} and p_{im2} as the first and second images, respectively (see

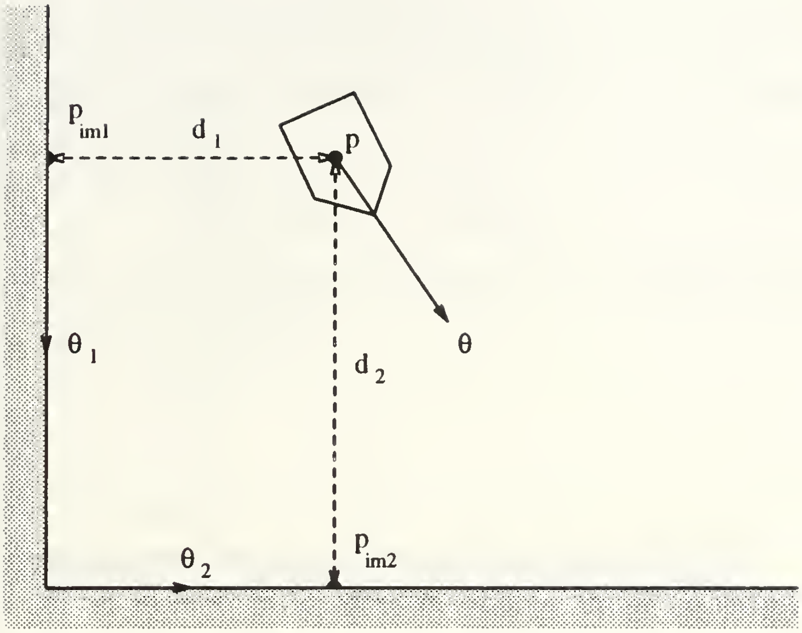


Figure 77. ccw tracking in Edge-Concave Vertex Tracking Mode

Figures 77, 78). Because an edge is a straight line, the vehicle is supposed to track a directed straight line. By applying the steering function in Eq. V.2, we will evaluate two steering functions for the first and second images and take a value by combining these two function results using Eqs. V.7,V.8 and V.9. Now, we will explain how to formulate $\frac{d\kappa}{ds}$, the steering function, in Eq. V.2 for each image.

For both images, we compute the variables κ_d, θ_d, and d in steering function (Eq. V.2) as follows.

For the first image p_{im1},

$$\kappa_{d1} = 0.$$

- If the image of p on the edge is on the right of the vehicle (Figure 77), then

$$\theta_1 = \Psi(p,\ p_{im1}) + \frac{\pi}{2},$$

$$g(d_1) = \begin{cases} d_1 - d_0 & \text{if } d_1 < d_0 \\ 0 & \text{otherwise} \end{cases}.$$

- If the image of p on the edge is on the left of the vehicle (Figure 77), then

$$\theta_1 = \Psi(p,\ p_{im1}) - \frac{\pi}{2},$$

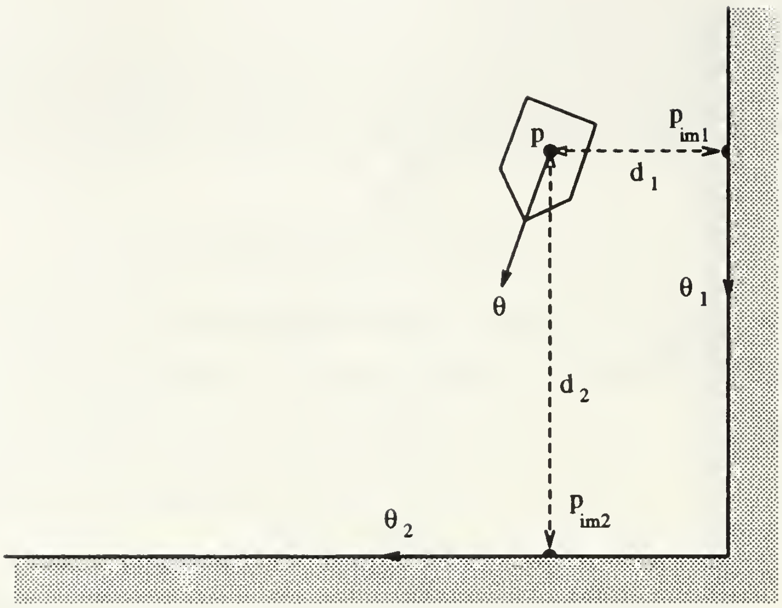


Figure 78. cw tracking in Edge-Concave Vertex Tracking Mode

$$g(d_1) = \begin{cases} d_1 + d_0 & \text{if } |d_1| < d_0 \\ 0 & \text{otherwise} \end{cases} .$$

For the second image p_{im2},

$$\kappa_{d2} = 0.$$

- If the image of p on the edge is on the right of the vehicle (Figure 77), then

$$\theta_2 = \Psi(p,\ p_{im2}) + \frac{\pi}{2},$$

$$g(d_2) = \begin{cases} d_2 - d_0 & \text{if } d_2 < d_0 \\ 0 & \text{otherwise} \end{cases} .$$

- If the image of p on the edge is on the left of the vehicle (Figure 77), then

$$\theta_2 = \Psi(p,\ p_{im2}) - \frac{\pi}{2},$$

$$g(d_2) = \begin{cases} d_2 + d_0 & \text{if } |d_1| < d_0 \\ 0 & \text{otherwise} \end{cases} .$$

Thus

$$\begin{aligned} f_1 &= -(a\,\kappa + b(\theta - \theta_1) + c\,g(d_1))\,, \\ f_2 &= -(a\,\kappa + b(\theta - \theta_2) + c\,g(d_2))\,, \end{aligned}$$

where f_1 and f_2 are the steering functions of the first and second images, respectively.

By combining f_1 and f_2 using Eqs. V.7,V.8 and V.9, we obtain the total steering function value:

$$f = \frac{\omega_1}{\omega_1 + \omega_2} f_1 + \frac{\omega_2}{\omega_1 + \omega_2} f_2.$$

Figure 79 shows the result of different trajectories. If σ increase, the safety cost function defined in Eq. V.5 increases.

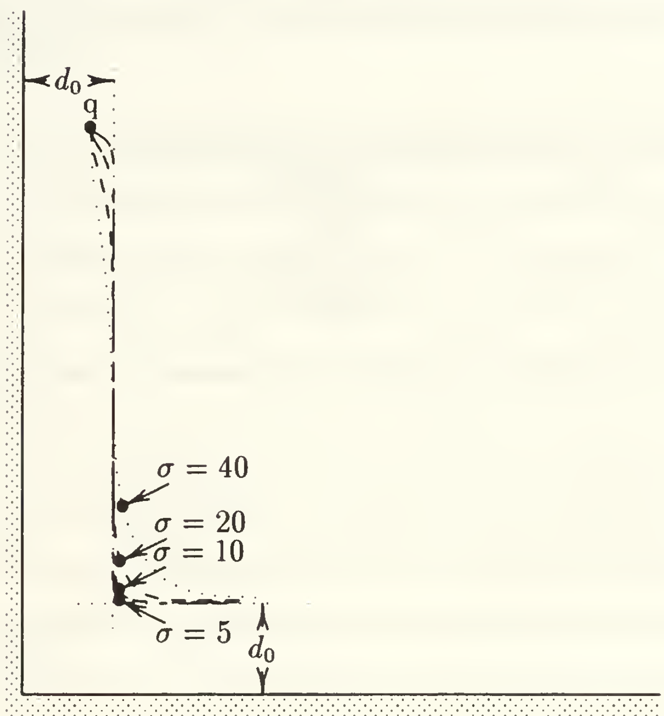


Figure 79. Different trajectories corresponding to their safety Cost Function Values in Edge-Concave Vertex Tracking Mode

I. SIMULATION RESULT ANALYSIS

In this section, several numerical simulation results are demostrated.

In Figures 80, 81 and 82, the vehicle is supposed to track a *ccw* polygon with *ccw* direction, where its initial configuration $q_0 = ((63, 450), -\pi/2, 0)$ and the safety clearance $d_0 = 80$. The effect of using distinct values of smoothness with $\sigma = 5, 10, 20$, and 40 is clearly shown in these figures. From this simulation, we found that there is a close relationship between the smoothness σ and the safety cost function Γ. In order to minimize Γ to obtain safer motion, a smaller σ should be used, and hence, bigger curvature is obtained. Therefore, slower-motion execution is needed. On the other hand, if less safe motions are allowed, a larger σ makes the trajectories smoother, and hence, smaller curvatures will be used. Therefore, faster motion execution is possible. But, in this case, the safety cost function Γ will increase. Table I shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different values of σ.

σ	safety cost function value Γ	smoothness cost function value Σ
5	23.7225	0.03262
10	33.9674	0.00181
20	45.5073	0.00027
40	54.1786	0.00008

Table I. Relation between smoothness and safety cost function values for polygon tracking (I)

In Figure 83, the vehicle is supposed to track a *ccw* polygon with *cw* direction, where its initial configuration $q_0 = ((63, 350), \pi/2, 0)$ and the safety clearance $d_0 = 80$. The effect of using distinct values of smoothness with $\sigma = 10$, and 40 is shown in this figure. Table II shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different values of σ.

Another example is shown in Figure 84. The vehicle is supposed to track a *ccw* polygon with *ccw* direction, where its initial configuration $q_0 = ((103, 450), -\pi/2, 0)$ and the safety clearance $d_0 = 80$. The effect of using distinct values of smoothness

σ	safety cost function value Γ	smoothness cost function value Σ
10	33.9674	0.00181
40	54.1786	0.00008

Table II. Relation between smoothness and safety cost function values for polygon tracking (II)

with $\sigma = 5, 10, 20$, and 40 is shown in this figure. Table III shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different values of σ.

σ	safety cost function value Γ	smoothness cost function value Σ
5	41.6822	0.00834
10	44.3815	0.00118
20	49.8532	0.00025
40	54.7353	0.00008

Table III. Relation between smoothness and safety cost function values for polygon tracking (III)

In Figure 85, the vehicle is supposed to track a *cw* polygon with *cw* direction, where its initial configuration $q_0 = ((60, 500), -\pi/2, 0)$ and the safety clearance $d_0 = 80$. The effect of using distinct values of smoothness with $\sigma = 10, 20$, and 40 is shown in this figure. Table IV shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different σ.

σ	safety cost function value Γ	smoothness cost function value Σ
10	22.0447	0.00275
20	33.0122	0.00027
40	57.2302	0.00003

Table IV. Relation between smoothness and safety cost function values for polygon tracking (IV)

The example in Figure 86 shows the result of the trajectory if the polygon is not rectlinear. This means that our algorithm is applicable to any polygon. the vehicle is supposed to track a *ccw* polygon with *cw* direction, where its initial configuration $q_0 = ((63, 350), \pi/2, 0)$ and the safety clearance $d_0 = 80$. The effect of using distinct values of smoothness with $\sigma = 10, 20$, and 40 is shown in this figure. Table V shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different σ.

σ	safety cost function value Γ	smoothness cost function value Σ
10	42.7962	0.00154
20	56.5925	0.00029
40	64.3274	0.00008

Table V. Relation between smoothness and safety cost function values for polygon tracking (V)

The polygon tracking algorithm was also implemented on Yamabico after being successfully developed on a simulator (see Chapter VIII).

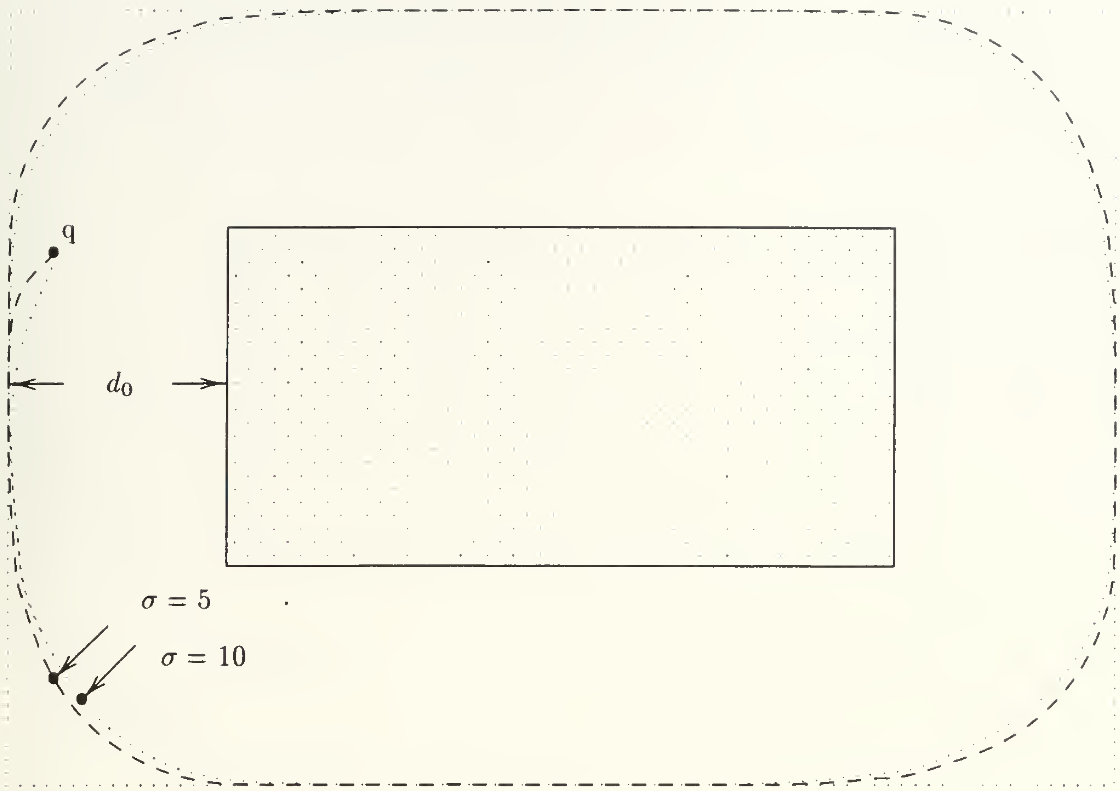


Figure 80: Different trajectories of *ccw* motion corresponding to their safety cost function values for *ccw* polygon (I)

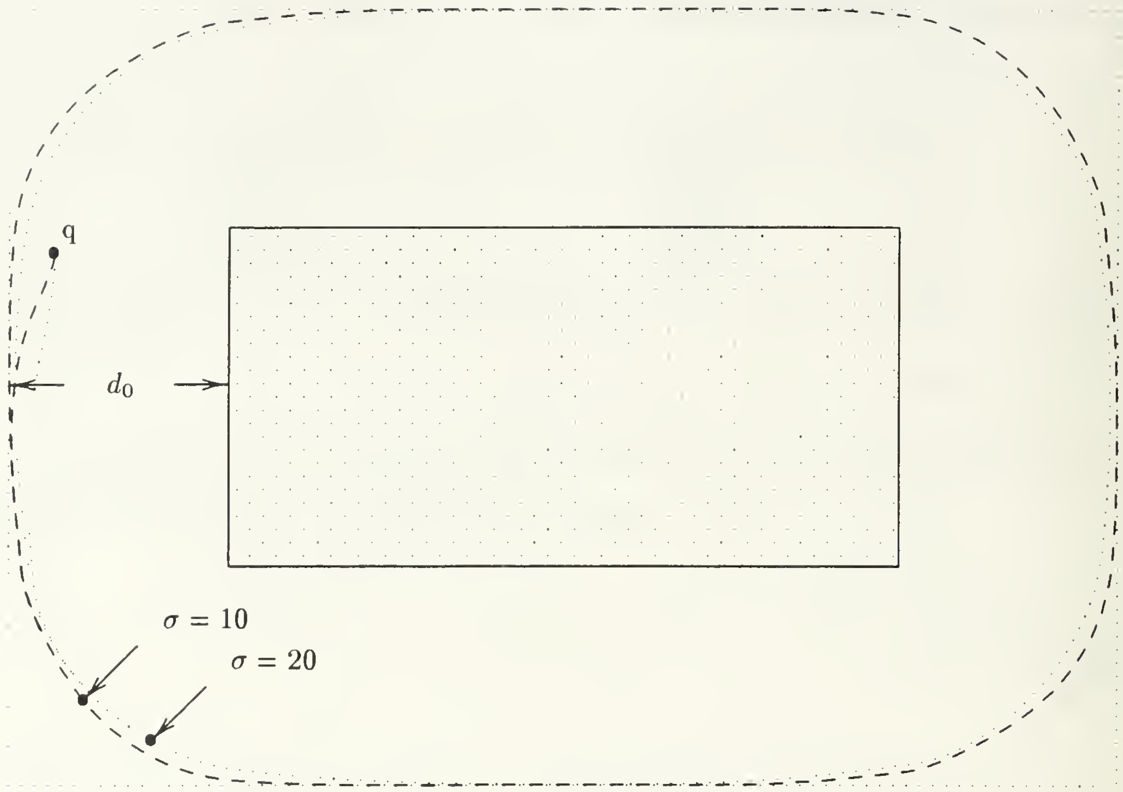


Figure 81: Different trajectories of *ccw* motion corresponding to their safety cost function values for *ccw* polygon (II)

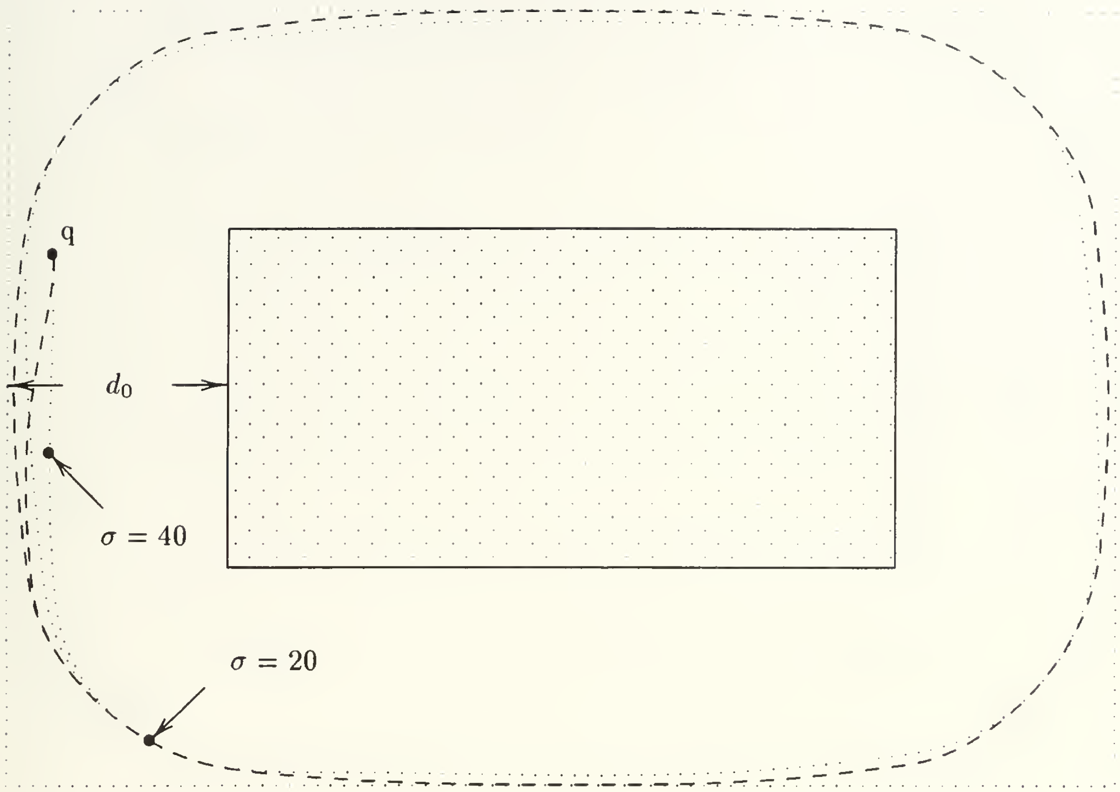


Figure 82: Different trajectories of *ccw* motion corresponding to their safety cost function values for *ccw* polygon (III)

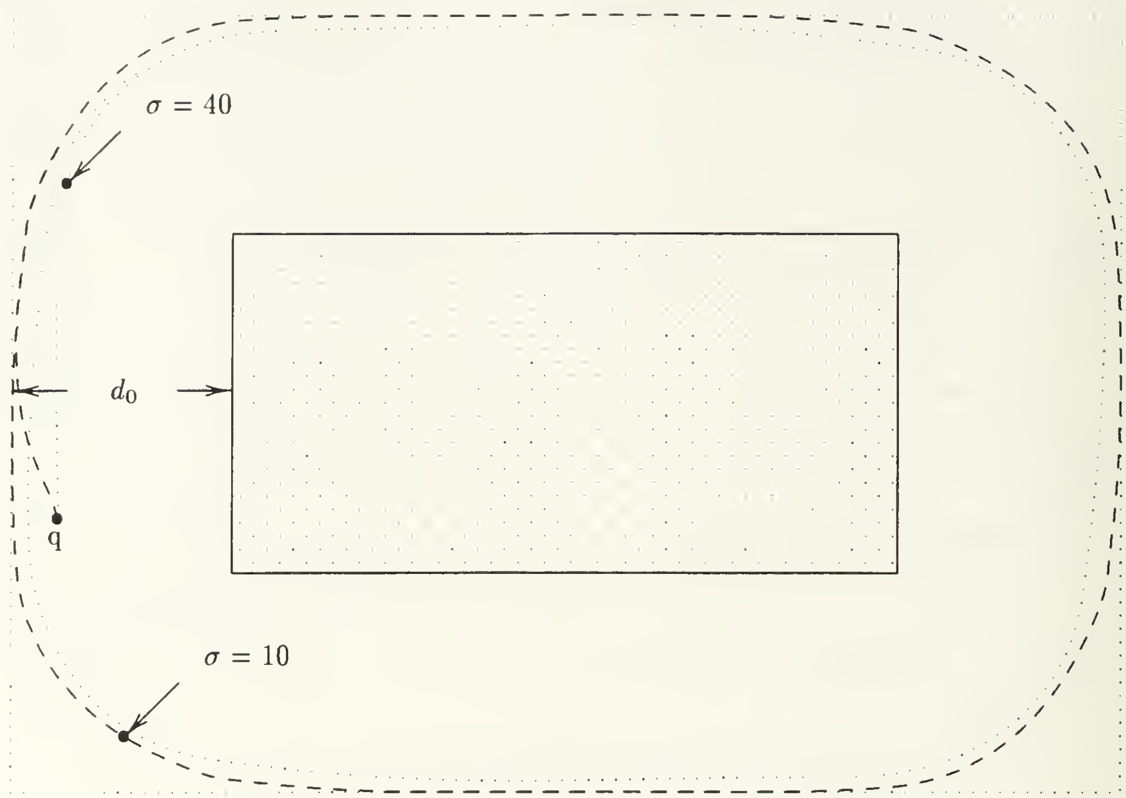


Figure 83: Different Trajectories of *cw* Motion Corresponding to their Safety Cost Function Values for *ccw* Polygon (IV)

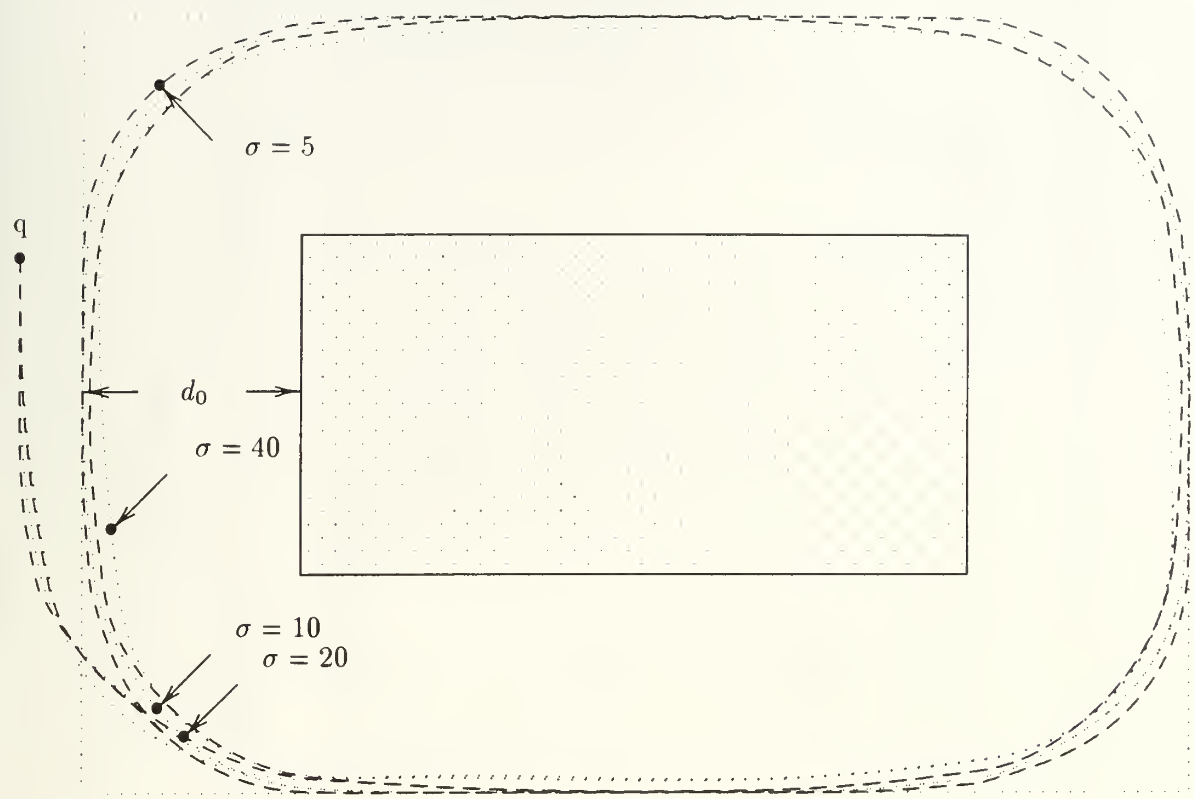


Figure 84: Different trajectories of *ccw* motion corresponding to their safety cost function values for *ccw* polygon (V)

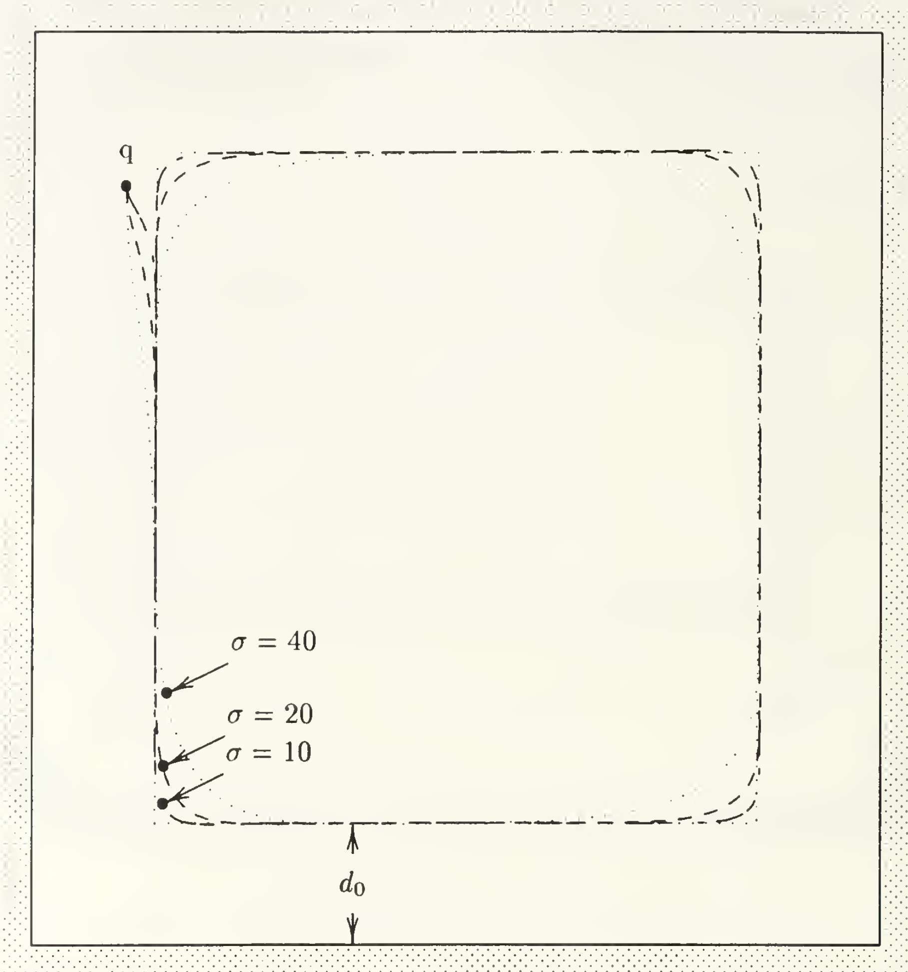


Figure 85: Different trajectories of *cw* tracking corresponding to their safety cost function values for *cw* polygon (VI)

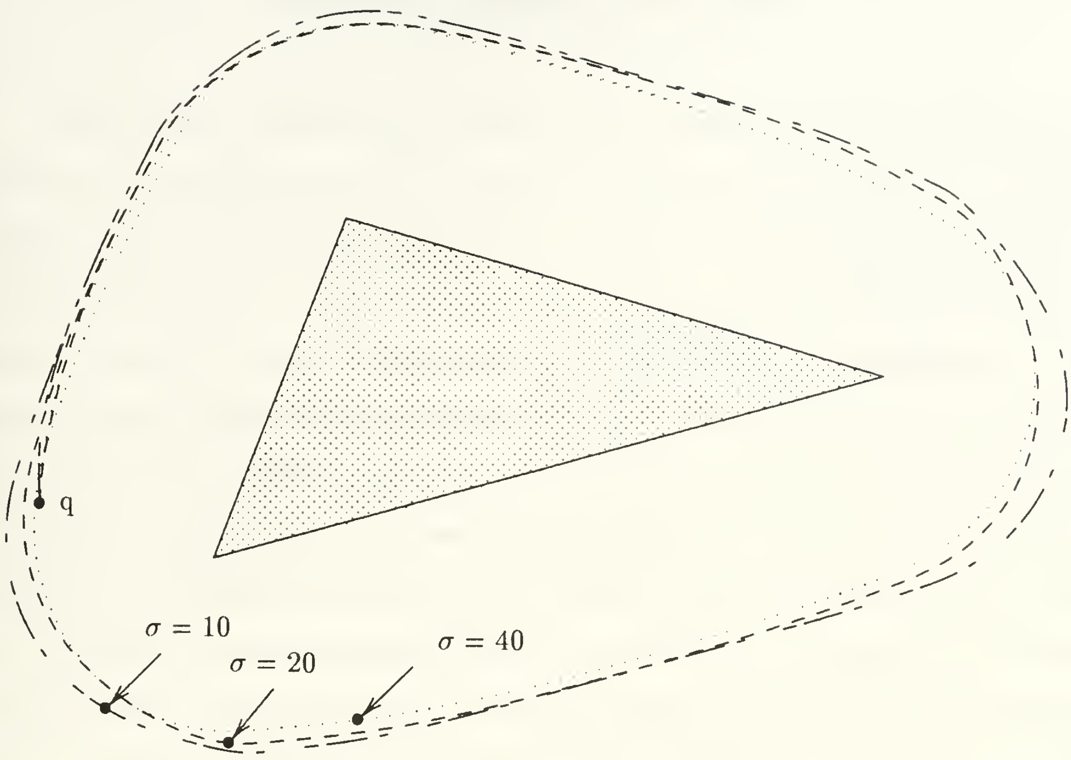


Figure 86: Different trajectories of *cw* motion corresponding to their safety cost function values for *ccw* polygon (VII)

VI. SAFE LOCAL MOTION PLANNING WITH SMOOTHING

This chapter addresses an approach to local motion planning. This approach provides the fundamental concepts to be used in local motion planning of this dissertation. The path class represented by a directed v-edges sequence (Chapter IV) provides information for rough robot navigation. The problem of finding the optimal motion in the path class is called the *local motion planning*. This problem is very important in this dissertation because *self–localization* is executed while the vehicle is moving. How do we define the optimality? In this dissertation, we take *safety* as the one property characteristic of motions to be optimized. Thus, the task of local motion planning is to produce the safest motion in a given path class with smooth motions where both safety and smoothness must be made precise. In Section A, we state the local motion planning problem. Sections B and C describe the safety clearance approach and the generalized safety cost function respectively. In Section D, The concept of local motion planning approach is presented. Sections E and F discuss the usefulness of directed v-edges sequence to local motion planning. In Section G, the local motion planning algorithm is described.

A. PROBLEM STATEMENT

We are given a world, $\mathcal{W}$; a path class represented by directed v-edges sequence Ξ; an initial configuration $q = (p, \theta, \kappa)$ of a vehicle (p, θ, and κ are its position, orientation and curvature respectively); and a safety clearance $d_0(> 0)$ (see Section B in Chapter V) (Figure 87). The problem of local motion planning is to plan a safe motion for a rigid body robot in a given path class, with smooth motions which avoids collisions with obstacles in the environment and satisfying the following conditions:

1. Its path curvature is continuous, and
2. The total safety cost of the path is minimized (see Section C).

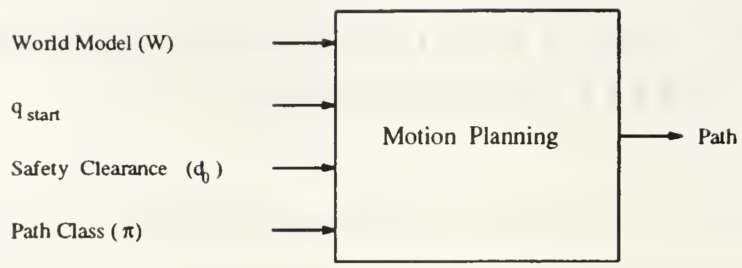


Figure 87. Block diagram for motion planning

B. SAFETY CLEARANCE CONCEPT

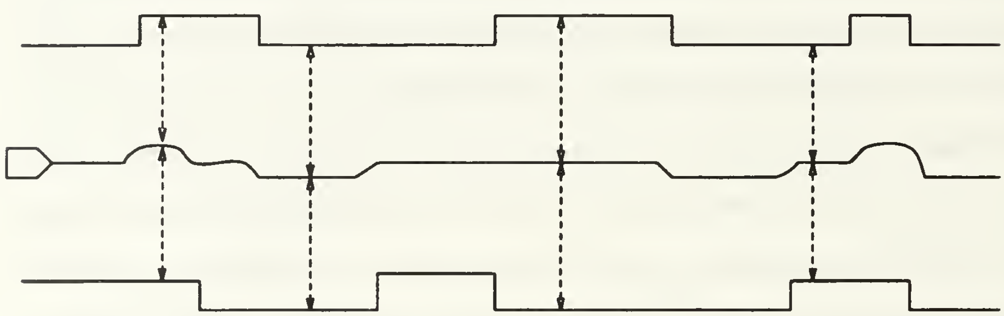


Figure 88. Tracking with exact Voronoi boundary

In this dissertation, we take safety as the single characteristic of motions to be optimized. The vehicle is supposed to move through a region lying between two distinct given images $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, in such a way that the vehicle looks at p_1 and p_2 on its left and right, respectively. When the left and right images are on an edge of the world boundary, the vehicle tries to make the distances to the left and right boundaries equal; in other words, its trajectory is eventually on the directed bisector of the two images (Voronoi boundary). But tracking the exact Voronoi boundary is not an appropriate approach (see Figure 88). We can loosen the strict Voronoi boundary tracking requirement in order to reduce the frequency of lateral transitions. One method is that the vehicle keeps safety clearance from the left/right boundaries (see Figure 89). If the distance between the robot and its left/right boundaries is less than this safety clearance, the robot must try to make the distance to the left/right boundaries greater than this safety clearance using the safety clearance function $g(d)$ (see Eq. V.4). Figure 90 shows that using safety clearance d_0 and safety clearance function $g(d)$ do not cause lateral motion of the vehicle.

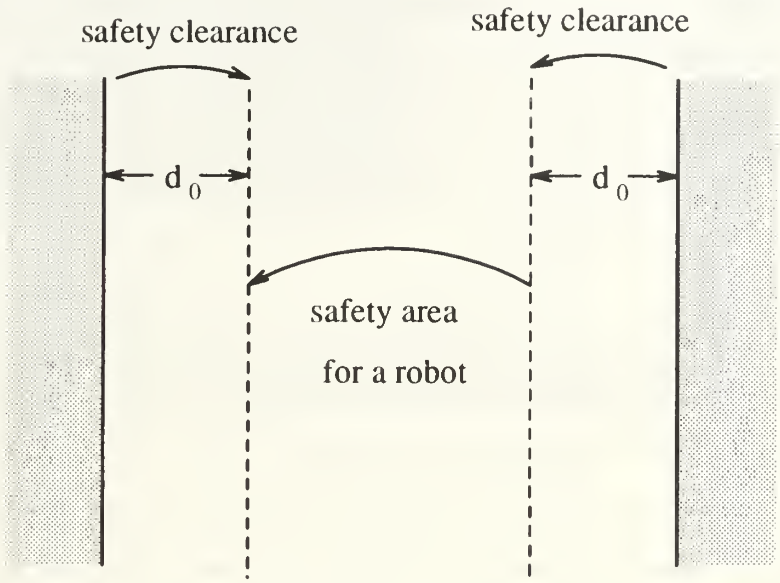


Figure 89. Safety clearance

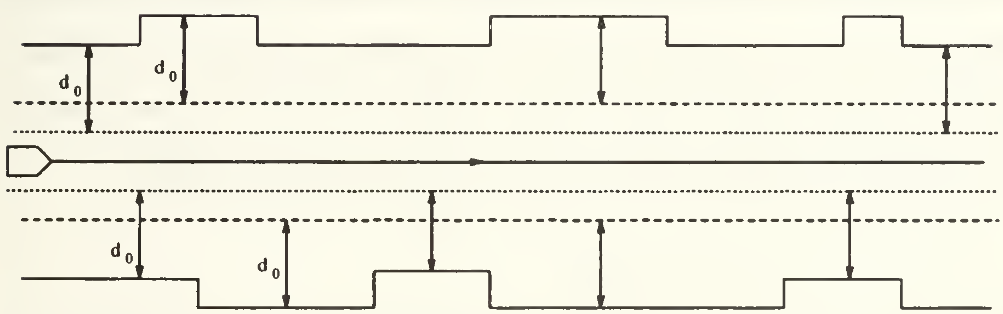


Figure 90. Tracking with safety clearance

C. GENERALIZED SAFETY COST FUNCTION

In Chapter V Section D, we discussed the concept of the safety cost function if we have only one polygon. Now, we will generalize this definition.

Consider a world $\mathcal{W}$ that consists of a finite number of polygons $B_0, B_1, \cdots, B_n$, i.e.,

$$\mathcal{W} = \{B_0, B_1, \cdots, B_n\}, \quad n > 0,$$

where $\mathcal{W}$ has one *cw* polygon B_0 and the n *ccw* polygons $B_1, \cdots, B_n$ are considered to be obstacles for the robot. A path in free space is a pair (s_1, f) consisting of a positive real number s_1 and a continuous function f. The length of path from the point $p(0)$ to a point $p(s)$ along a path(s_1, f) is equal to s if $0 \leq s \leq s_1$. Let $\gamma(p, B_i)$ denote the distance between a point p to a polygon B_i. Let $p(s)$ denote a vehicle position at s on the path. The total safety cost of a path(s_1, f) is given by a positive cost function $\Gamma : \mathcal{R} \rightarrow \mathcal{R}$ defined by

$$\Gamma = \int_0^{s_1} \left[\min_{B_i \in \mathcal{W}} \gamma(p(s), B_i) - d_0 \right]^2 ds, \tag{VI.1}$$

where d_0 is the robot's safety clearance (see Eq. V.3.

Generally, a path farther from obstacles is safer, but it tends to be longer. Therefore, we need to strike a balance between smoothness and safety of a path. There is a positive parameter, σ, in the steering function, which controls the smoothness of the resultant trajectory. If a smaller σ is used, the trajectory becomes sharper and the path becomes safer, and if a larger σ is used, the trajectory becomes smoother and the path becomes more dangerous. As the smoothness parameter σ becomes large, the path converges to the smoothest path. Thus, we obtain a class of paths with different weight between safety and smoothness in an equivalent class.

D. PLANNING APPROACH

The global path class is the input to local motion planning. It provides useful information in directing the robot to accomplish its mission. The task of local motion planning is to provide a smooth, collision-free motion for the robot, based on the global path class generated by the global path planner. Because the safety of an autonomous vehicle navigation is determined by the clearance between the vehicle and obstacles. Path safety is a function of the distance from the robot to an obstacle. As the distance decreases, the safety decreases. The safest path is one in which the distance to the obstacle is maximized. In many cases, a robot should not approach closer to the obstacle than a given safety range (see Figure 91).

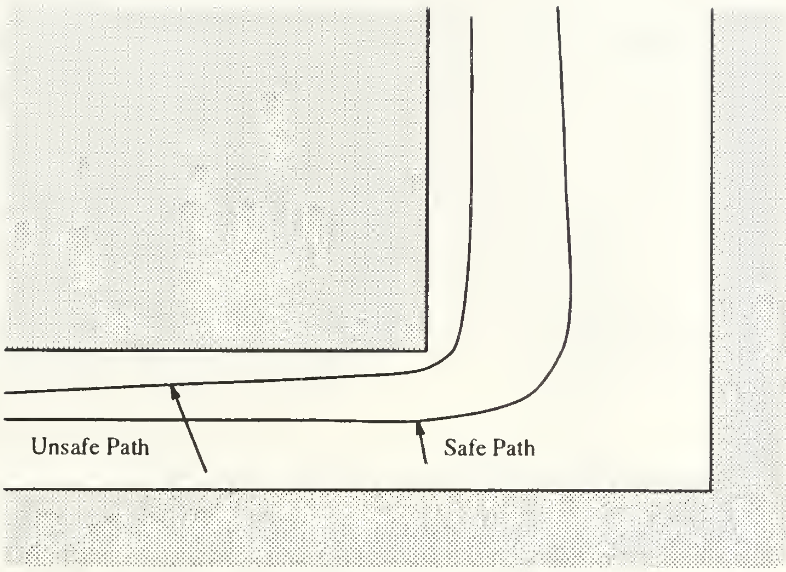


Figure 91. Safe and unsafe paths

Because a Voronoi boundary is the set of points locally maximizing the clearance from obstacles, safety is maximized on such a boundary. Unfortunately, the naive plan of just tracking the Voronoi boundary does not work, because:

1. A Voronoi boundary may have discontinuity in either its tangential direction or its curvature. It is known that a nonholonomic rigid body robot cannot track such a reference path. For example, in Figure 92, there is a discontinuity in its curvature when there is a transition from a line segment to a parabolic arc. Also, there is a discontinuity in its tangential direction when there is a transition from a parabolic arc to another.
2. It is time-consuming and, actually, is not necessary to compute the Voronoi boundary and to track it.
3. A complex data structure is needed to represent Voronoi boundaries.
4. This task becomes unduely complex for dynamic environments.

However, the Voronoi boundary gives us the idea that the motion will be considered safer if it stays further away from objects.

Instead of tracking the Voronoi boundary, the vehicle tries to make the distances to the left and right boundaries using a *steering function* which uses data such as the distances, directions to left and right images, and the desired curvature.

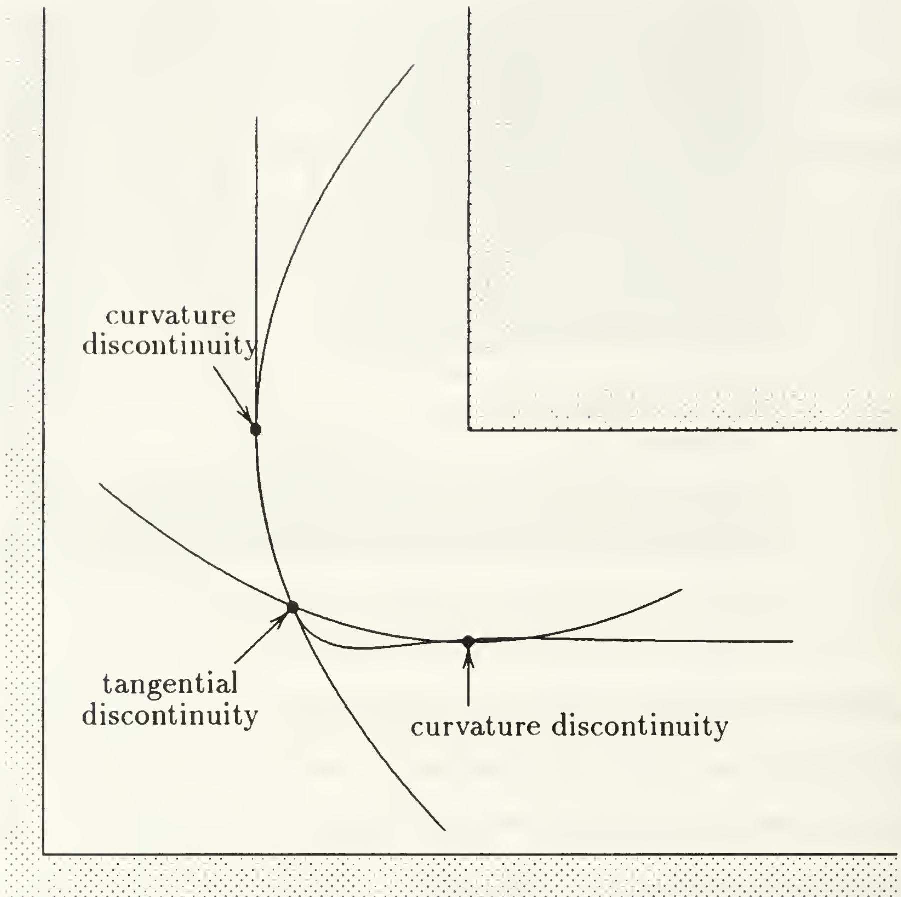


Figure 92. Discontinuity where two distinct Voronoi boundary intersect

The new problem to be solved in this dissertation is how to achieve a smooth motion when the vehicle gets closer to an intersection of two distinct segments (for instance from a line segment to a circle segment). In order to solve this problem, we will use the fact that the proximity relation changes at such an intersection (see Figure 93). Therefore, we will watch second images in the forward portion of a left or right boundary, and will make a smooth motion by evaluating the steering function using not only the left/right first images, but the left/right second images too. That is, when a second image gets closer, we evaluate two steering functions with the first and the second images and take a value by mixing these two function results. Thus, resultant motion paths will be "smoothed" using an appropriate smoothness σ. The *smoothness* σ is parameter in the steering function, which controls the smoothness of the resultant trajectory. If a smaller σ is used, the trajectory becomes sharper, and if a larger σ is used, the trajectory becomes smoother. For more details, see [36].

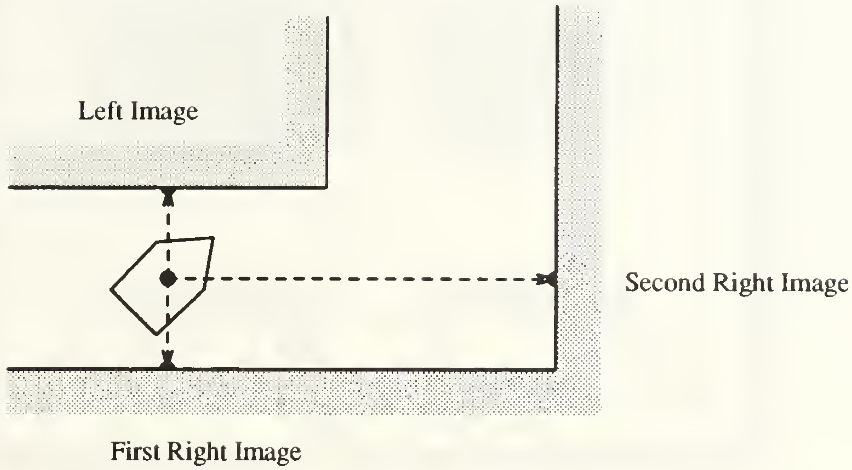


Figure 93. Both left and right images are on edges

As a summary of the above, the safe motion planning is done by the general algorithm stated above. We will confirm the validity of the method of using the left and right images for tracking the smoothed path. Also, we need to find a robust algorithm for making smooth motion from one boundary segment to another. A striking advantage of this method is that is effective in more dynamic environments. This method may be useful even in unknown worlds as well, because the images can be taken by sensors instead of information extraction from the model.

E. THE USEFULNESS OF DIRECTED V-EDGES SEQUENCE TO LOCAL MOTION PLANNING

This section describes how the directed v-edges sequence Ξ is useful for local motion planning. Once the global plan represented by directed v-edges sequence is found, it is passed to a routine which ensures the vehicle will follow the global plan in order to reach the goal. Beacuse the directed v-edge ξ is defined by the two closest polygons, these polygons are used for the selection of the features which are used to

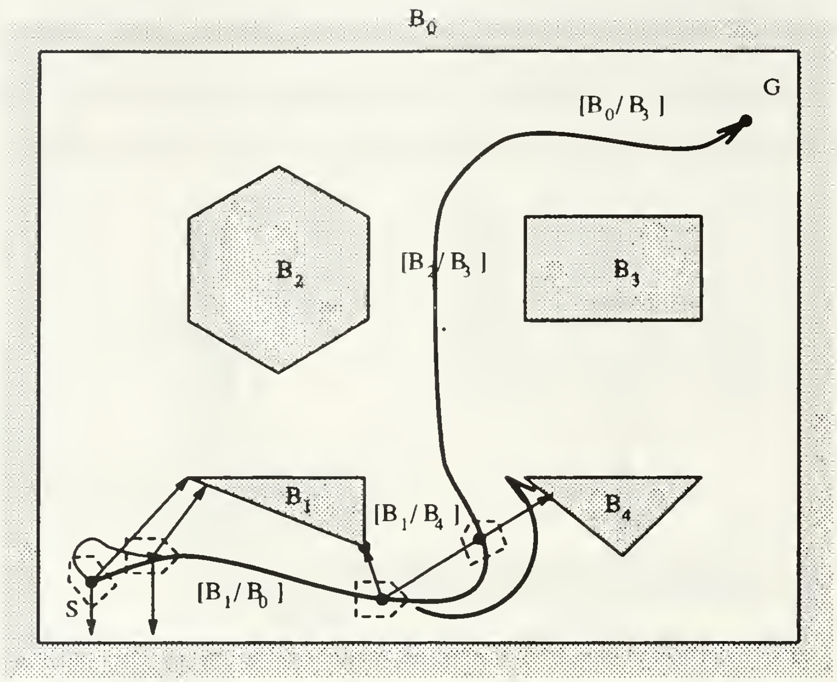


Figure 94. Directed v-edges sequence to local motion planning (left turn is required)

calculate the desired control values. For example, in Figure 94, the directed v-edges sequence Ξ is defined as

$$\Xi = [B_1/B_0][B_1/B_4][B_2/B_3][B_0/B_3] \tag{VI.2}$$

In Eq. VI.2, the first directed v-edge is $\xi_1 = [B_1/B_0]$. This mean that, the vehicle recognizes B_1 and B_0 as the left and right obstacles respectively. Although the start orientation of the vehicle is different from the direction of the motion as shown in Figure 94, the vehicle steers in the direction of motion since B_1 is the left obstacle. In the second directed v-edge, $\xi_2 = [B_1/B_4]$, the vehcile recognizes B_4 as the right obstacle. Then the vehicle will make left turn.

On the other hand, does the following directed v-edges sequence Ξ produce another motion?

$$\Xi = [B_1/B_0][B_4/B_0][B_3/B_0] \tag{VI.3}$$

In the second directed v-edge, $\xi_2 = [B_4/B_0]$, the vehicle recognizes B_4 as the left obstacle (see Figure 95). Then no turn is required.

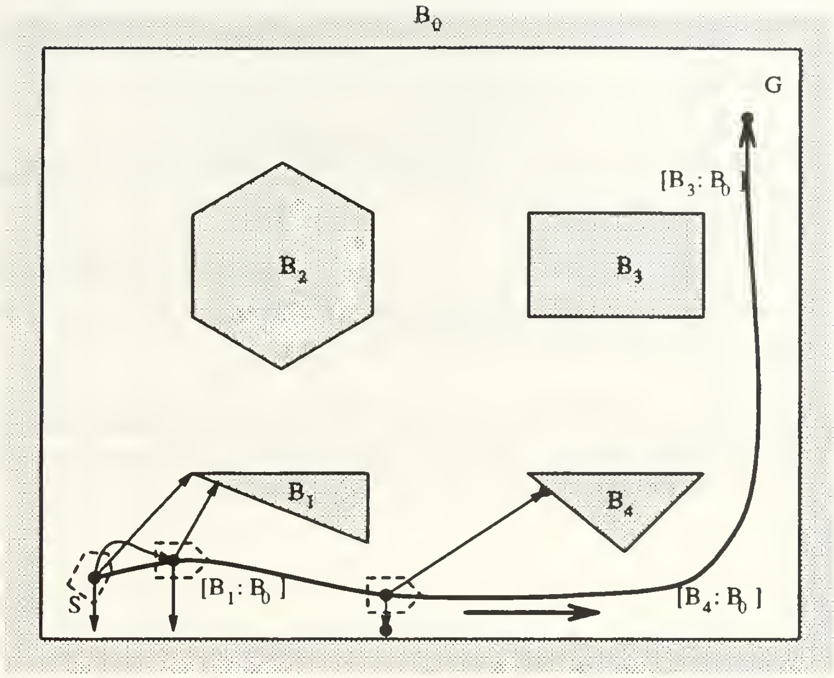


Figure 95. Directed v-edges sequence to local motion planning (no turn is required)

From the above, we can conclude that a directed v-edges sequence is useful for both local motion planning and global path planning.

F. DIFFERENT TYPES OF POLYGON TRACKING IN DIRECTED V-EDGES SEQUENCE

The essential idea is based on the fact that obstacles present in the working environment and when a vehicle is moving, it recognizes the left and right images on these obstacles. Therefore, it is possible for a vehicle to travel in the free space along obstacles's outer boundary and to keep certain safety clearance.

When a vehicle is moving, it recognizes not only the left/right images, but also left/right second images. Therefore, we will watch second images in the forward portion of a left or right boundary, and we will evaluate the steering function using not only the left/right first images, but the left/right second images too. Because path class is defined by a directed v-edges sequence Ξ and each directed v-edge ξ is defined by the two closest polygons (subpolygons), these polygons are used for the selection of the features which are used to calculate the steering function values. We have the following types of tracking:

- The left and right polygons (subpolygons) in current and next directed v-edge are not identical (see Figures 96).
- The left polygons (subpolygons) in current and next directed v-edge are identical, but the right polygons (subpolygons) in current and next directed v-edge are not identical (see Figures 97, 98, 99, 100).
- The left polygons (subpolygons) in current and next directed v-edge are not identical, but the right polygons (subpolygons) in current and next directed v-edge are identical (see Figures 101, 102, 103, 104).

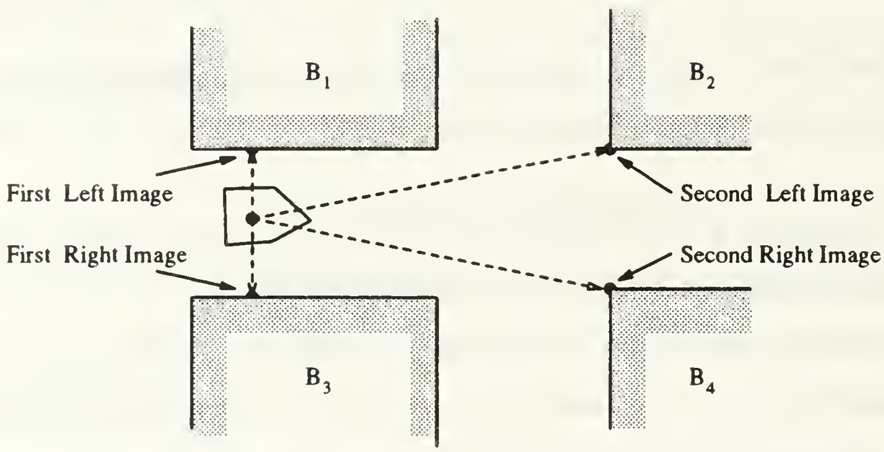


Figure 96. Left and right current and next polygons are not identical in directed v-edges sequence Ξ

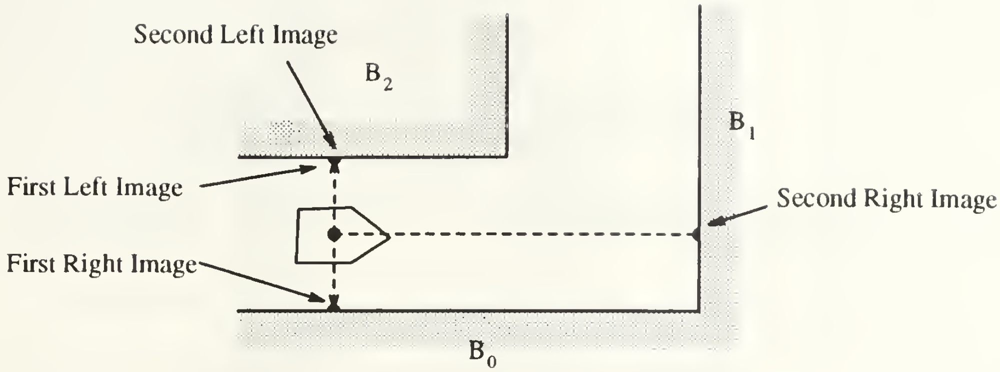


Figure 97. Left current and next left polygons are identical but right current and next right polygons are not identical in directed v-edges sequence Ξ (I)

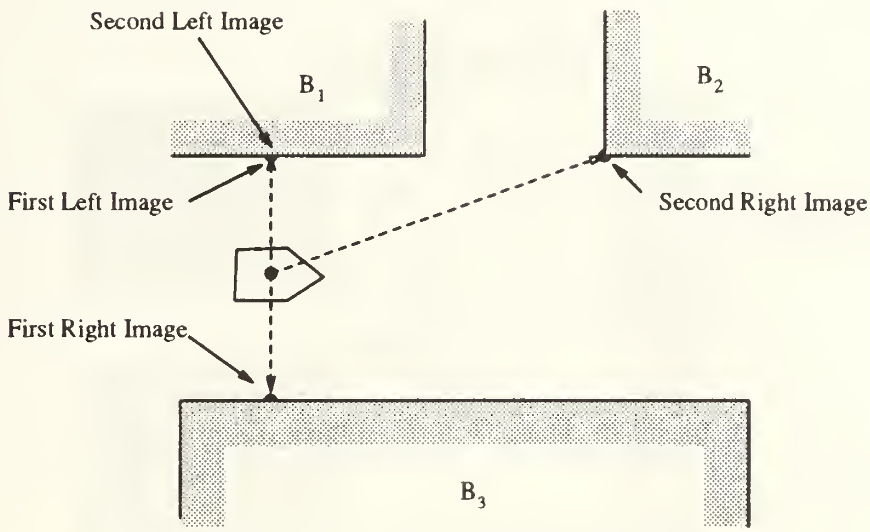


Figure 98. Left current and next left polygons are identical but right current and next right polygons are not identical in directed v-edges sequence Ξ (II)

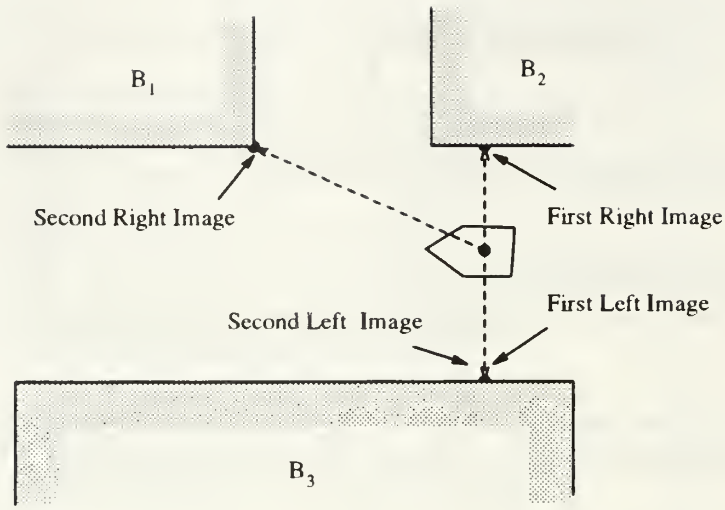


Figure 99. Left current and next left polygons are identical but right current and next right polygons are not identical in directed v-edges sequence Ξ (III)

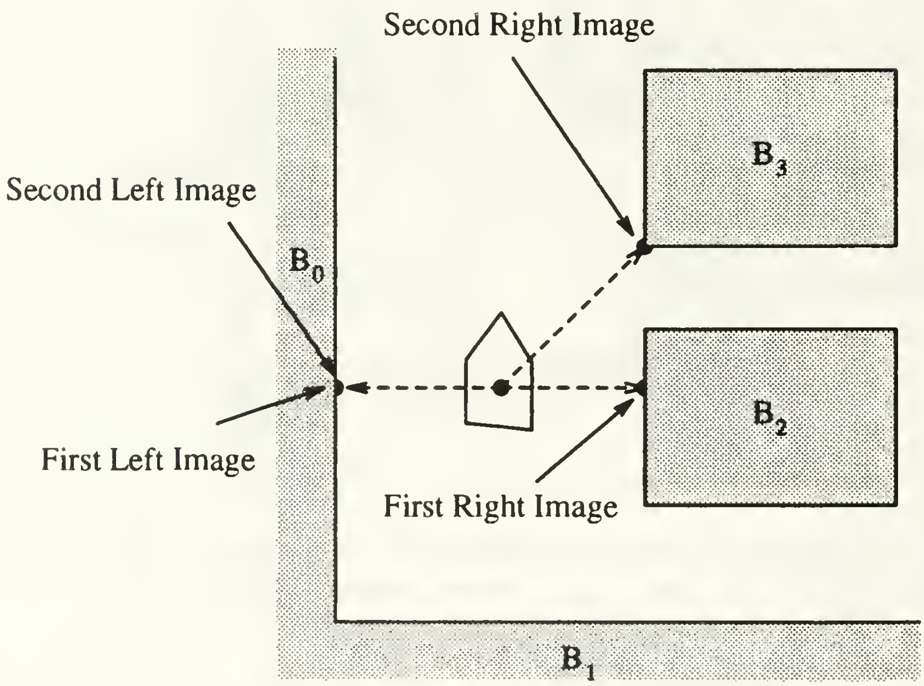


Figure 100. Left current and next left polygons are identical but right current and next right polygons are not identical in directed v-edges sequence Ξ (IV)

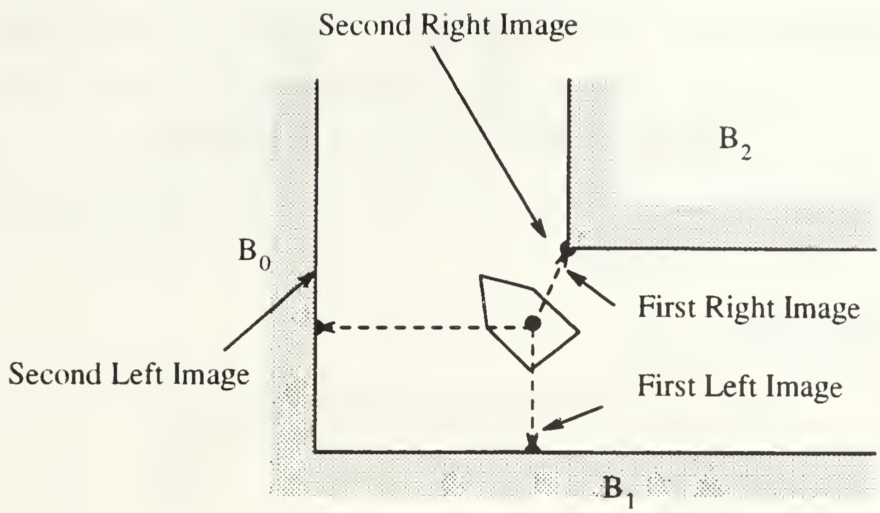


Figure 101. Left current and next left polygons are not identical but right current and next right polygons are identical in directed v-edges sequence Ξ (I)

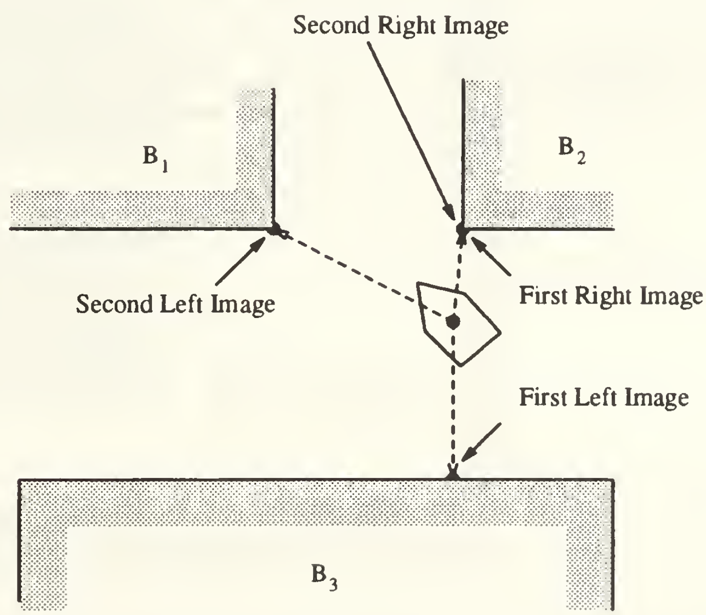


Figure 102. Left current and next left polygons are not identical but right current and next right polygons are identical in directed v-edges sequence Ξ (II)

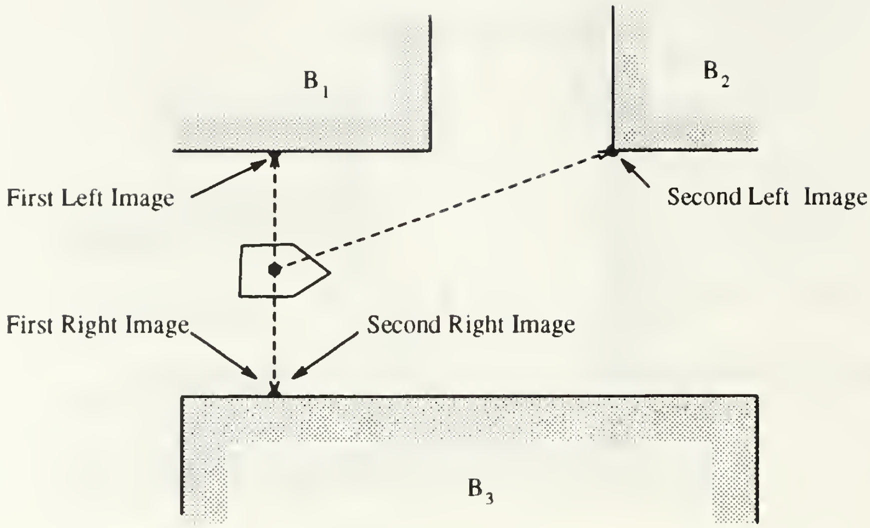


Figure 103. Left current and next left polygons are not identical but right current and next right polygons are identical in directed v-edges sequence Ξ (III)

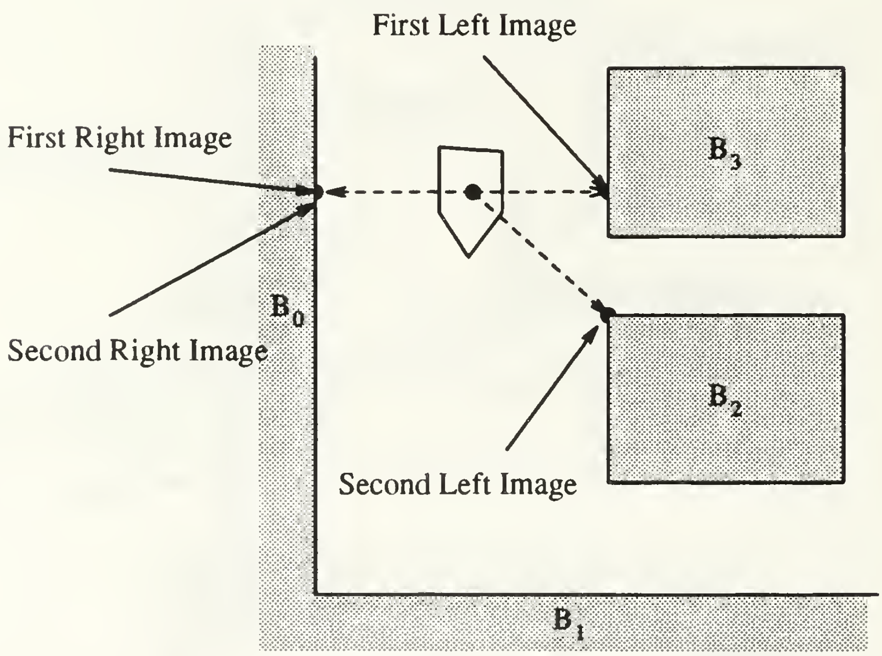


Figure 104. Left current and next left polygons are not identical but right current and next right polygons are identical in directed v-edges sequence Ξ (IV)

G. LOCAL MOTION PLANNING ALGORITHM

The previous section analyzed the different types of polygon tracking possible in a directed v-edges sequence. We summarize that analysis into motion rules based on the type of polygon tracking (see Chapter V). The rule selection is based on the current and next directed v-edge in the directed v-edges sequence Ξ.

1. If both the current and next left polygons in the directed v-edges sequence Ξ are *ccw* and they are identical and both the current and next right polygons in Ξ are *cw* (*ccw*) and they are not identical, then a left turn is required. In this case, both the current and next left images are identical and the direction of tracking left polygon is *ccw* but the current and next right images are not identical and the direction of tracking both right polygons is *cw*. For example, in Figure 105, the sequence Ξ is given as

$$\Xi = [B_2/B_0][B_2/B_1],$$

   and in Figure 106, Ξ is given as

$$\Xi = [B_1/B_3][B_1/B_2].$$

2. If both the current and next left polygons in the directed v-edges sequence Ξ are *cw* (*ccw*) and they are not identical and both the current and next right polygons in Ξ are *ccw* and they are identical, then a right turn is required. In this case, both the current and next left images are not identical and the direction of tracking both left polygons is *ccw* but the current and next right images are identical and the direction of tracking right polygon is *cw*. For example, in Figure 107, the sequence Ξ is given as

$$\Xi = [B_1/B_2][B_0/B_2],$$

   and in Figure 108, Ξ is given as

$$\Xi = [B_3/B_2][B_1/B_2].$$

3. If both the current and next left polygons in the directed v-edges sequence Ξ are *cw* and they are not identical and both the current and next right polygons in Ξ are *ccw* (*cw*) and they are identical, then no turn is required and we follow the right side of the corridor. In this case, both the current and next left images are not identical and the direction of tracking both left polygons is *ccw* but the current and next right images are identical and the direction of tracking right polygon is *cw*. For example, in Figure 109, the sequence Ξ is given as

$$\Xi = [B_1/B_3][B_2/B_3],$$

and in Figure 110, Ξ is given as

$$\Xi = [B_3/B_0][B_2/B_0].$$

4. If both the current and next left polygons in the directed v-edges sequence Ξ are *ccw* (*cw*) and they are identical and both the current and next right polygons in Ξ are *ccw* and they are not identical, then no turn is required and we follow the left side of the corridor. In this case, both the current and next left images are identical and the direction of tracking left polygon is *ccw* but the current and next right images are not identical and the direction of tracking both right polygons is *cw*. For example, in Figure 111, the sequence Ξ is given as

$$\Xi = [B_3/B_2][B_3/B_1],$$

and in Figure 112, Ξ is given as

$$\Xi = [B_0/B_2][B_0/B_3].$$

5. If both the current and next left polygons in the directed v-edges sequence Ξ are *ccw* and they are not identical and both the current and next right polygons in Ξ are *ccw* and they are not identical, then no turn is required and we follow the left (right) side of the corridor. In this case, both the current and next left images are not identical and the direction of tracking both left polygons is *ccw* but the current and next right images are not identical and the direction of tracking both right polygons is *cw*. For example, in Figure 113, the sequence Ξ is given as

$$\Xi = [B_1/B_3][B_2/B_4].$$

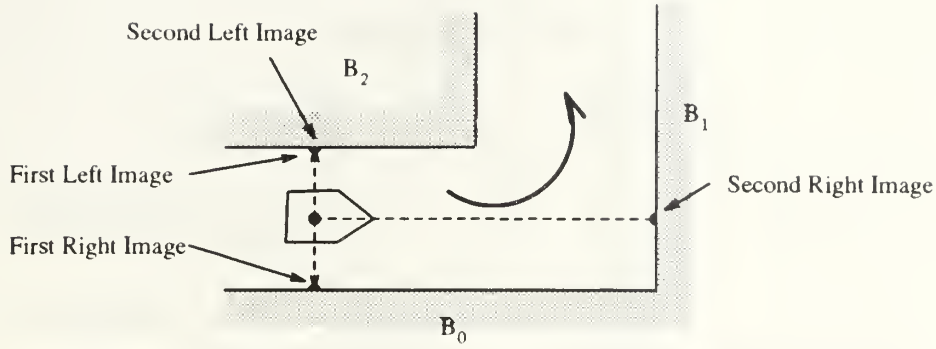


Figure 105. Left turn is required (I)

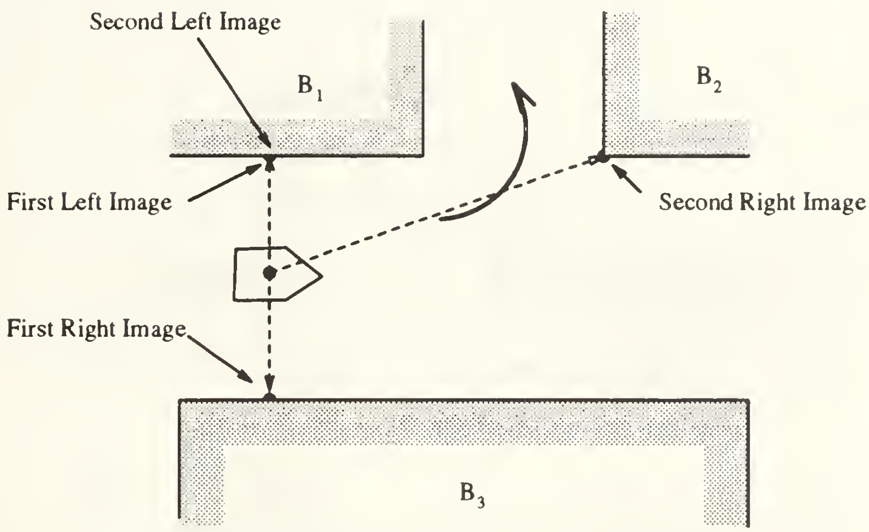


Figure 106. Left turn is required (II)

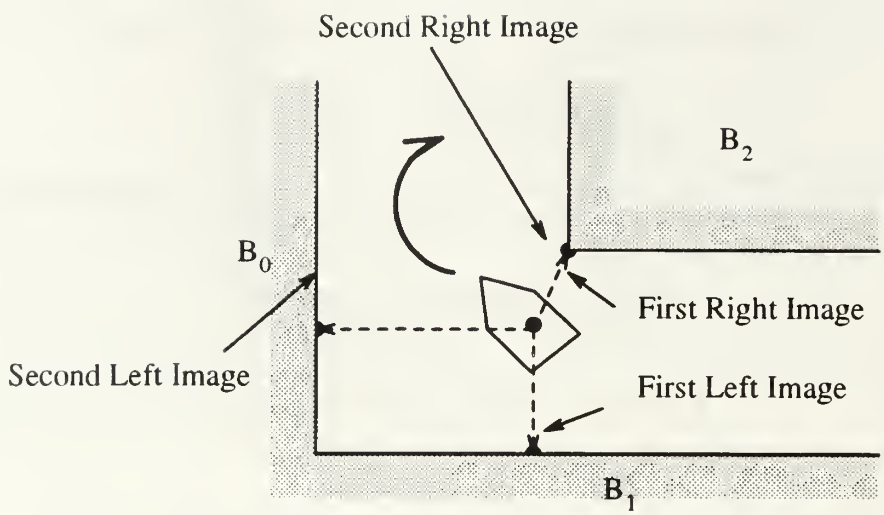


Figure 107. Right turn is required (I)

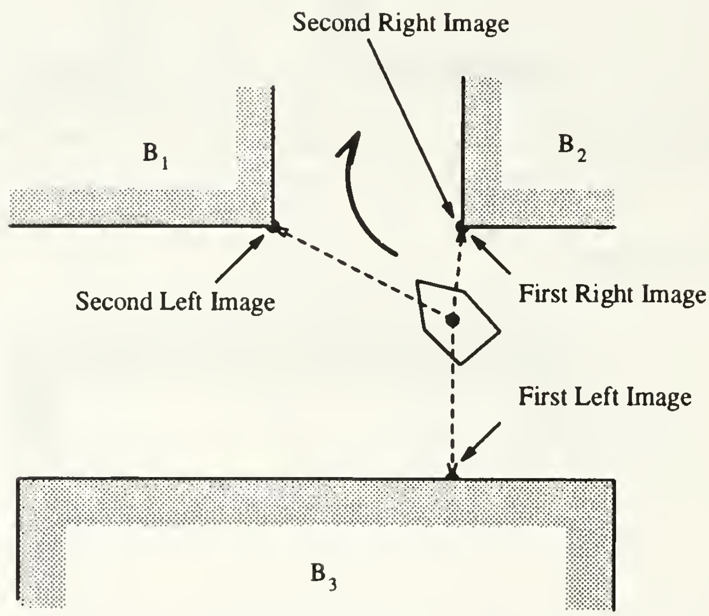


Figure 108. Right turn is required (II)

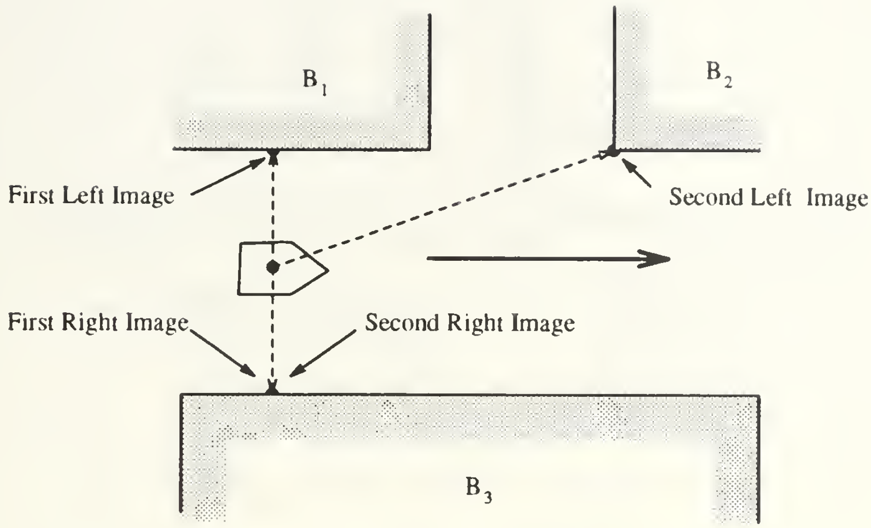


Figure 109. No turn is required (I)

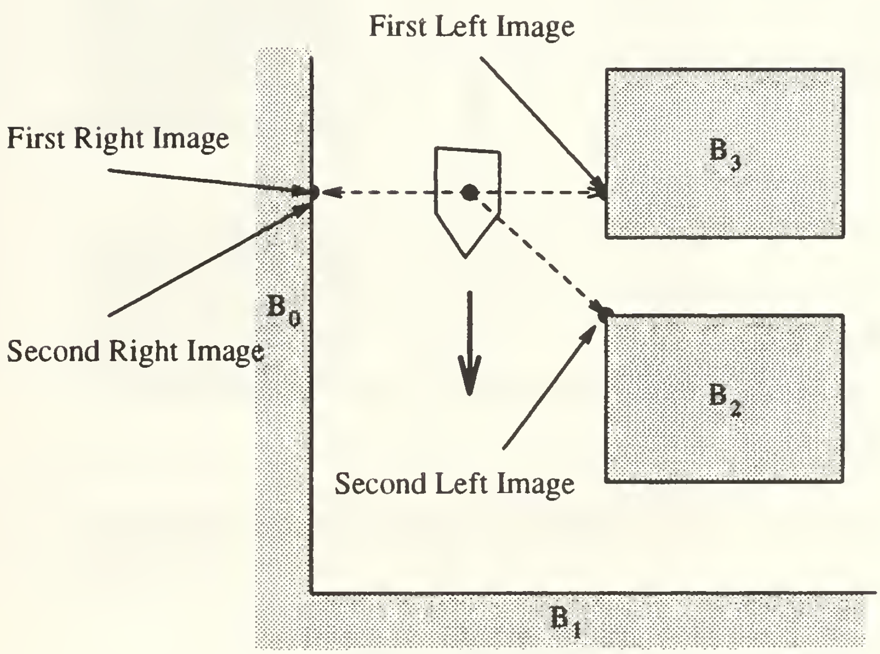


Figure 110. No turn is required (II)

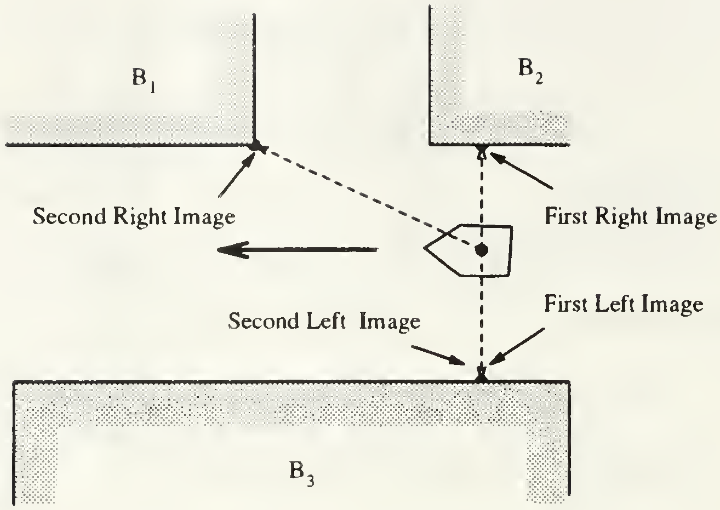


Figure 111. No turn is required (III)

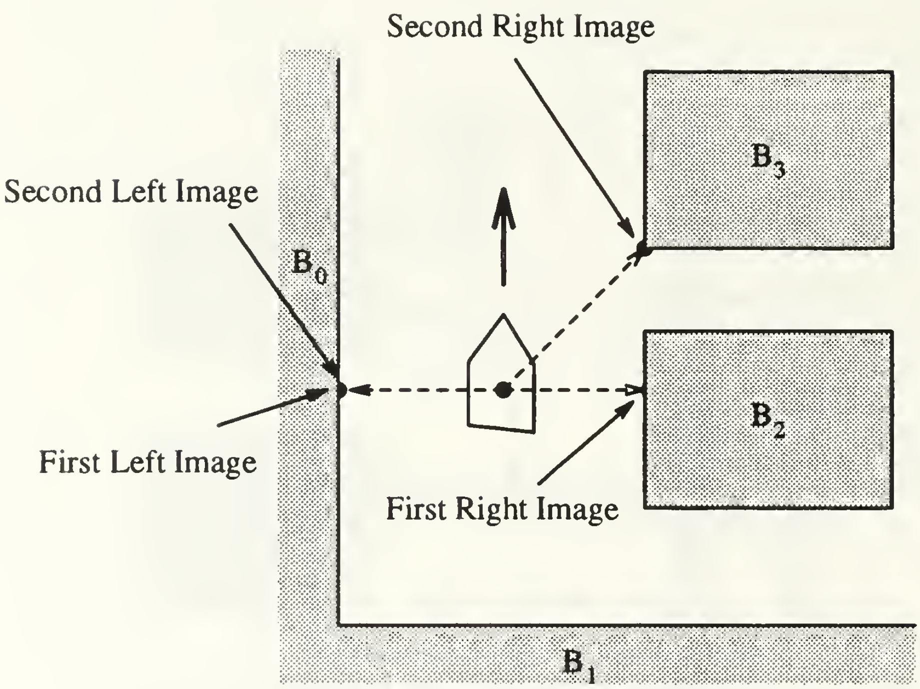


Figure 112. No turn is required (IV)

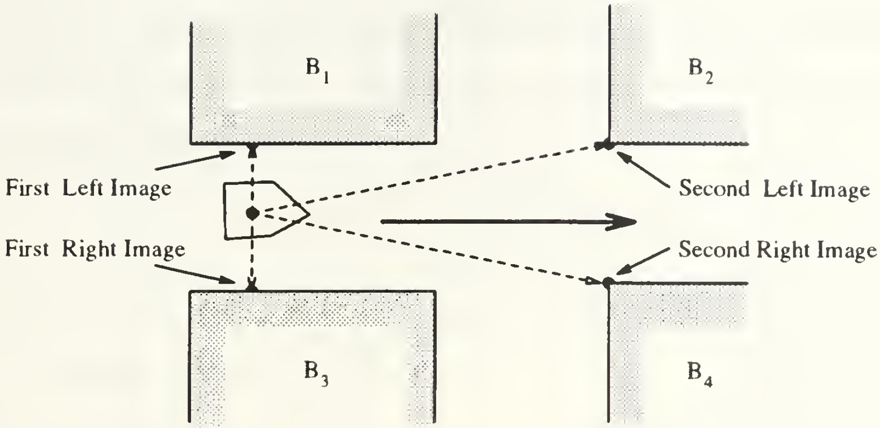


Figure 113. No turn is required (V)

H. SIMULATION RESULT ANALYSIS

In this section, several numerical simulation results are shown.

Consider the problem of finding a path from a start configuration, S, to a goal configuration, G in a polygonal world $\mathcal{W}$ (Figure 114). It is desired to connect the start configuration, S, to the goal configuration, G, using a continuous, smooth path. There are four different path classes. Each path class is symbolically represented by directed v-edges sequence.

$$\begin{aligned}
\pi_1 &= [B_4/B_0]\ [B_4/B_5]\ [B_2/B_5]\ [B_3/B_5] \\
\pi_2 &= [B_4/B_0]\ [B_5/B_0]\ [B_5/B_3]\ [B_5/B_2] \\
\pi_3 &= [B_0/B_4]\ [B_1/B_4]\ [B_2/B_4]\ [B_2/B_5] \\
\pi_4 &= [B_0/B_4]\ [B_1/B_4]\ [B_2/B_4]\ [B_5/B_4]\ [B_5/B_0]\ [B_5/B_3][B_5/B_2]
\end{aligned}$$

In Figure 115, the initial configuration of the vehicle is $q_0 = ((90, 450), -\pi/2, 0)$ and safety clearance is $d_0 = 80$. The path class representing by the directed v-edges sequence is given as

$$\pi_1 = [B_4/B_0]\ [B_4/B_5]\ [B_2/B_5]\ [B_3x/B_5]$$

Table VI shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different σ. The effect of using distinct values of smoothness with

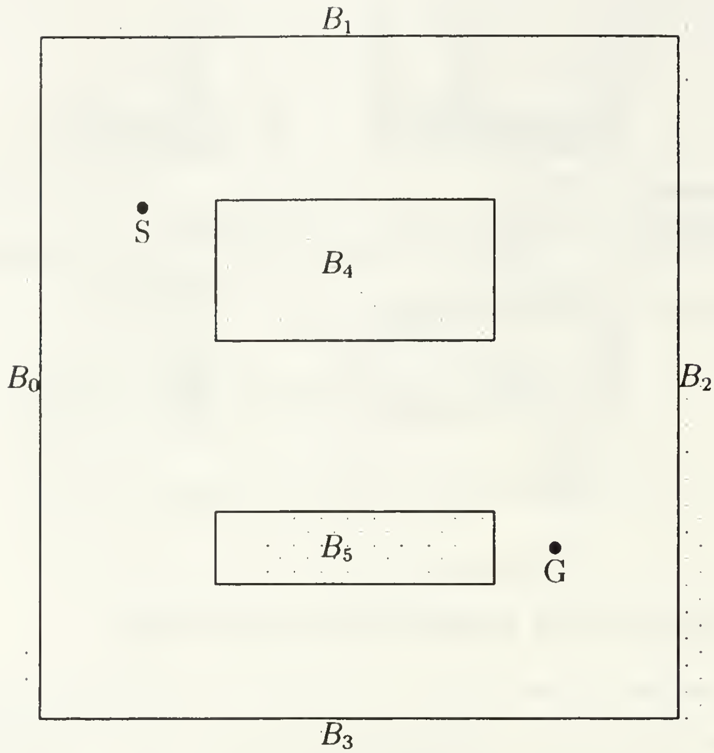


Figure 114. World of motion planning

$\sigma = 5, 10, 20$, and 40 is clearly seen. From this simulation, we found that there is a close relationship between the smoothness σ and the safety cost function Γ. In order to minimize Γ to obtain safer motion, a smaller σ should be used, and hence, bigger curvature is obtained. Therefore, slower-motion execution is needed. On the other hand, if less safe motions are allowed, a larger σ makes the trajectories smoother, and hence, smaller curvatures will be used. Therefore, faster motion execution is possible. But, in this case, the safety cost function Γ will increase.

σ	safety cost function value Γ	smoothness cost function value Σ
5	37.7319	0.08189
10	48.1742	0.00511
20	58.7558	0.00049
40	67.5781	0.00007

Table VI. Relation between smoothness and safety cost function values for motion planning (I)

In Figure 116, the initial configuration of the vehicle is $q_0 = ((90, 450), -\pi/2, 0)$ and the safety clearance is $d_0 = 80$. The path class representing by the directed v-edges sequence is given as

$$\pi_2 = [B_4/B_0]\ [B_5/B_0]\ [B_5/B_3]\ [B_5/B_2]$$

Table VII shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different σ.

σ	safety cost function value Γ	smoothness cost function value Σ
5	55.0527	0.45522
10	57.6073	0.00207
20	60.8893	0.00022
40	66.3729	0.00003

Table VII. Relation between smoothness and safety cost function values for motion planning (II)

In Figure 117, the initial configuration of the vehicle is $q_0 = ((90, 350), \pi/2, 0)$ and the safety clearance is $d_0 = 80$. The path class representing by the directed v-edges sequence is given as

$$\pi_3 = [B_0/B_4]\ [B_1/B_4]\ [B_2/B_4]\ [B_2/B_5]$$

Table VIII shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different σ.

σ	safety cost function value Γ	smoothness cost function value Σ
5	33.0391	0.06152
10	37.4319	0.00313
20	45.1034	0.00027
40	53.0906	0.00003

Table VIII. Relation between smoothness and safety cost function values for motion planning (III)

In Figure 118, the initial configuration of the vehicle is $q_0 = ((90, 350), \pi/2, 0)$ and the safety clearance is $d_0 = 80$. The path class representing by the directed v-edges sequence is given as

$$\pi_4 = [B_0/B_4]\ [B_1/B_4]\ [B_2/B_4]\ [B_5/B_4]\ [B_5/B_0]\ [B_5/B_3][B_5/B_2]$$

Table IX shows the values for both safety cost function Γ and smoothness cost function Σ corresponding to different σ.

σ	safety cost function value Γ	smoothness cost function value Σ
5	61.9985	0.11397
10	68.9123	0.00733
20	78.6803	0.00083
40	89.3738	0.00013

Table IX. Relation between smoothness and safety cost function values for motion planning (IV)

Another example is shown in Figure 119. The vehicle is supposed to track the following path class where its initial configuration $q_0 = ((90, 450), -\pi/2, 0)$ and the safety clearance $d_0 = 80$.

$$\pi = [B_4/B_0]\ [B_4/B_5]\ [B_4/B_2]\ [B_4/B_1]\ [B_4/B_0]\ [B_4/B_5]\ [B_4/B_2]\ [B_4/B_1]$$

The effect of using distinct values of smoothness with $\sigma = 5, 10, 20$, and 40 is shown in Table X.

σ	safety cost function value Γ	smoothness cost function value Σ
5	48.9584	0.18851
10	69.2488	0.01303
20	74.3775	0.00126
40	77.5919	0.00018

Table X. Relation between smoothness and safety cost function values for motion planning (V)

The example in Figure 120 shows the result when a vehicle is browsing randomly in the free space. The vehicle tracks the following path class where its initial configuration $q_0 = ((90, 120), \pi/2, 0)$ and the safety clearance $d_0 = 80$.

$$\begin{aligned} \pi &= [B_0/B_5]\ [B_4/B_5]\ [B_4/B_2]\ [B_4/B_1]\ [B_4/B_0]\ [B_5/B_0]\ [B_5/B_3] \\ &\quad [B_5/B_2]\ [B_4/B_2]\ [B_4/B_1]\ [B_4/B_0]\ [B_5/B_0]\ [B_5/B_3] \end{aligned}$$

The effect of using distinct values of smoothness with $\sigma = 5, 10, 20$, and 40 is shown in Table XI.

σ	safety cost function value Γ	smoothness cost function value Σ
5	63.0592	0.18534
10	72.8446	0.01213
20	84.4241	0.00061
40	91.6753	0.00015

Table XI. Relation between smoothness and safety cost function values for motion planning (VI)

The local motion planning algorithm was also implemented on Yamabico after being successfully developed on a simulator (see Chapter VIII).

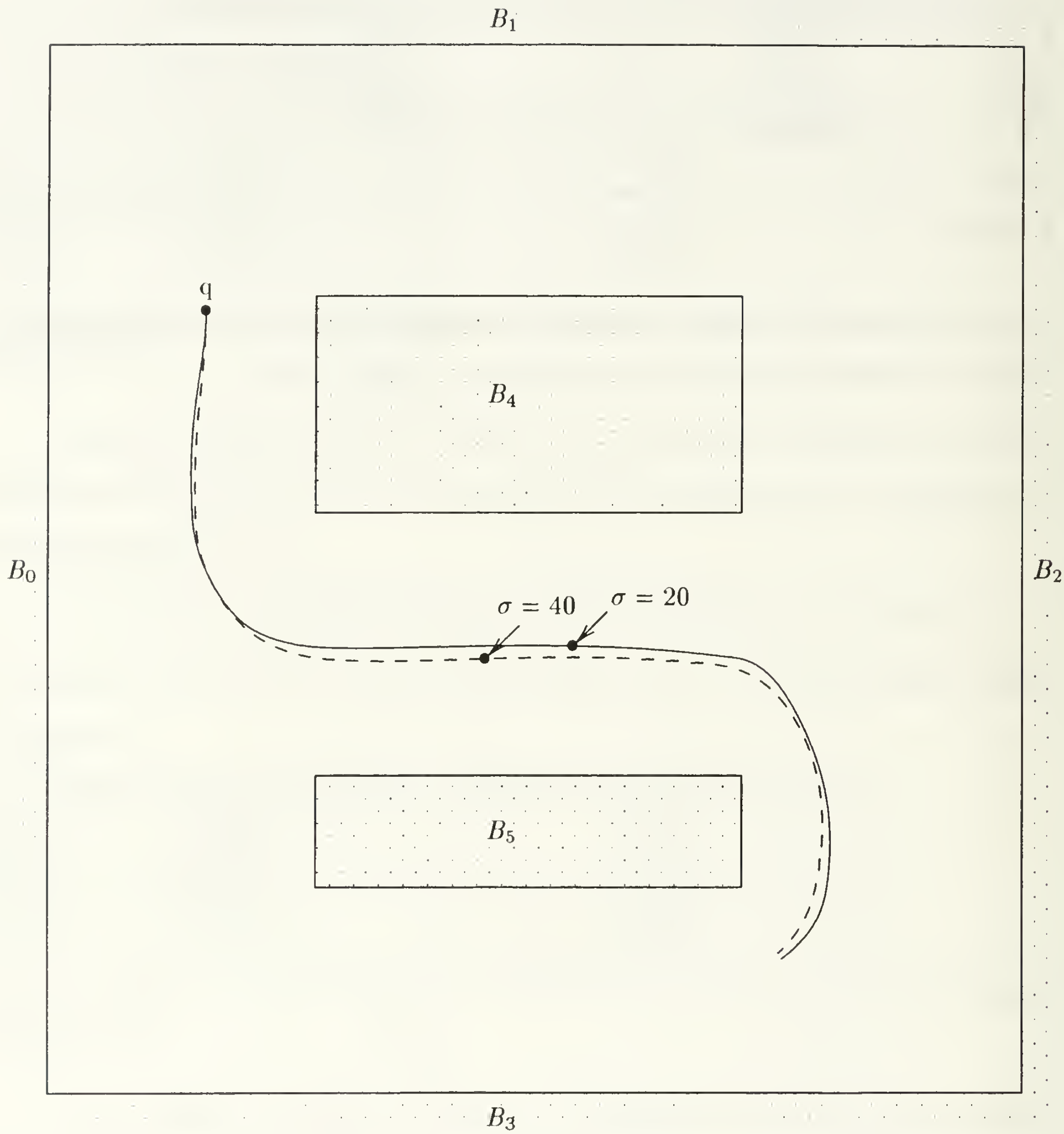


Figure 115: Motion planning and execution result (I)

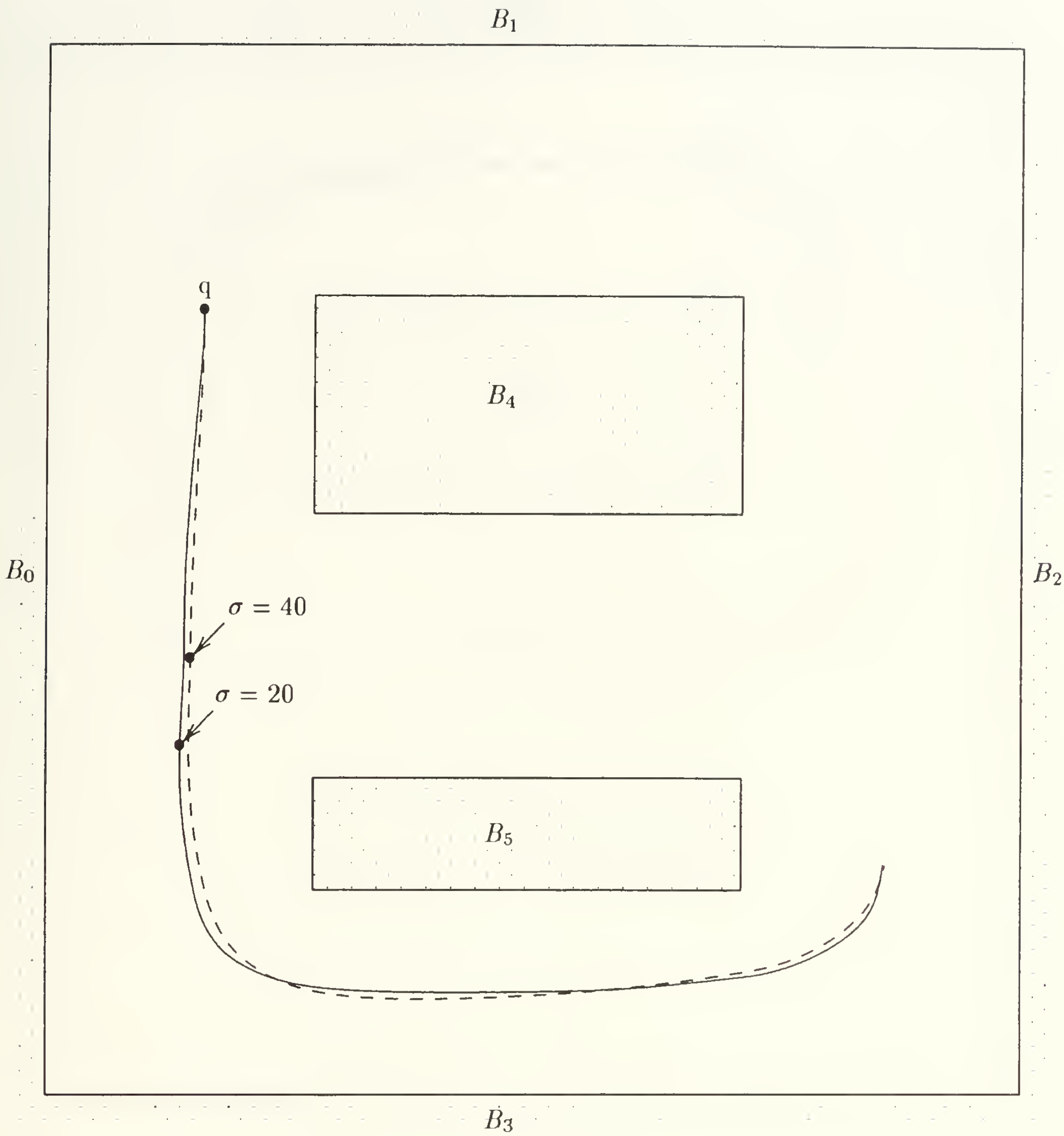


Figure 116: Motion planning and execution result (II)

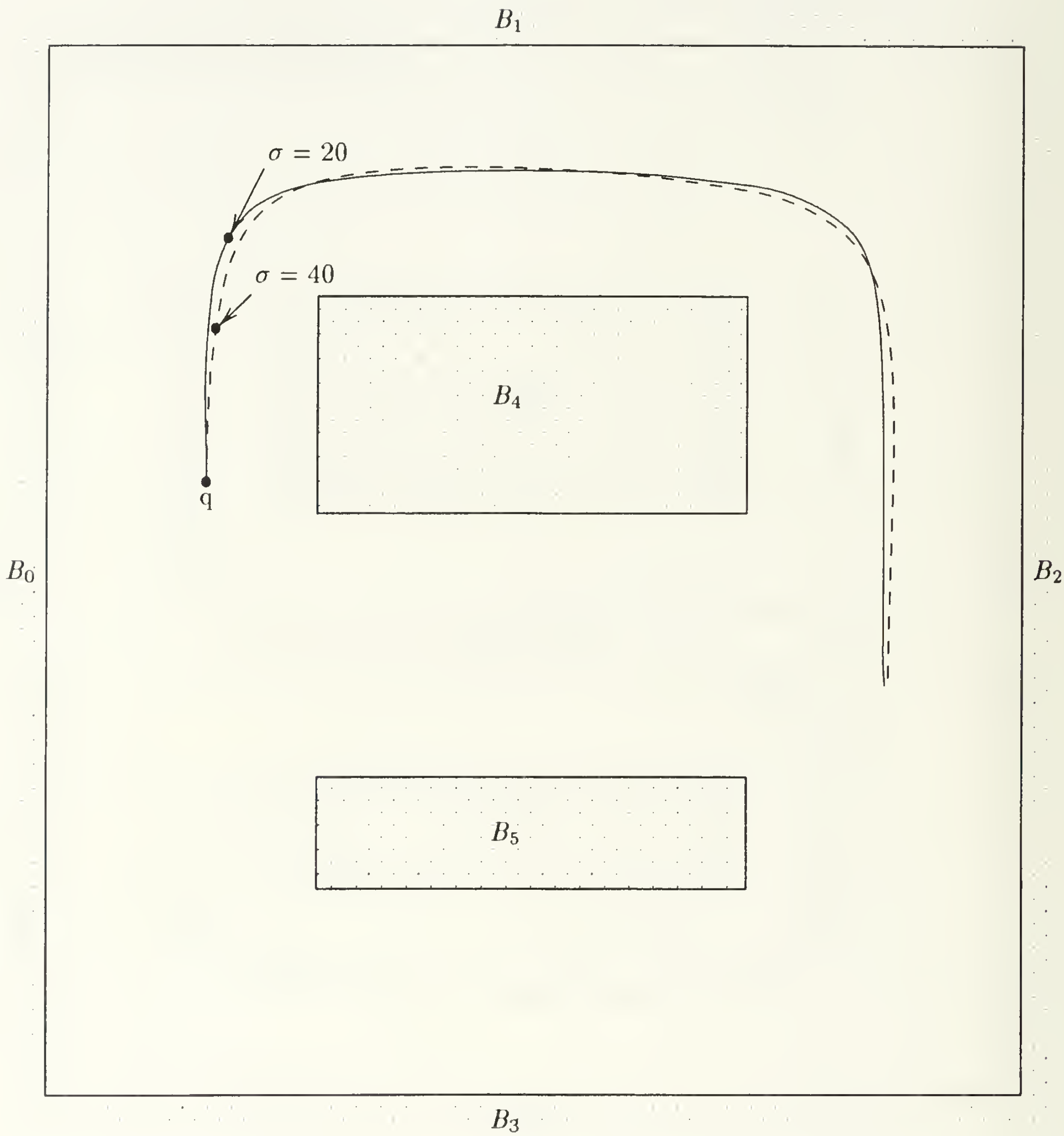


Figure 117: Motion planning and execution result (III)

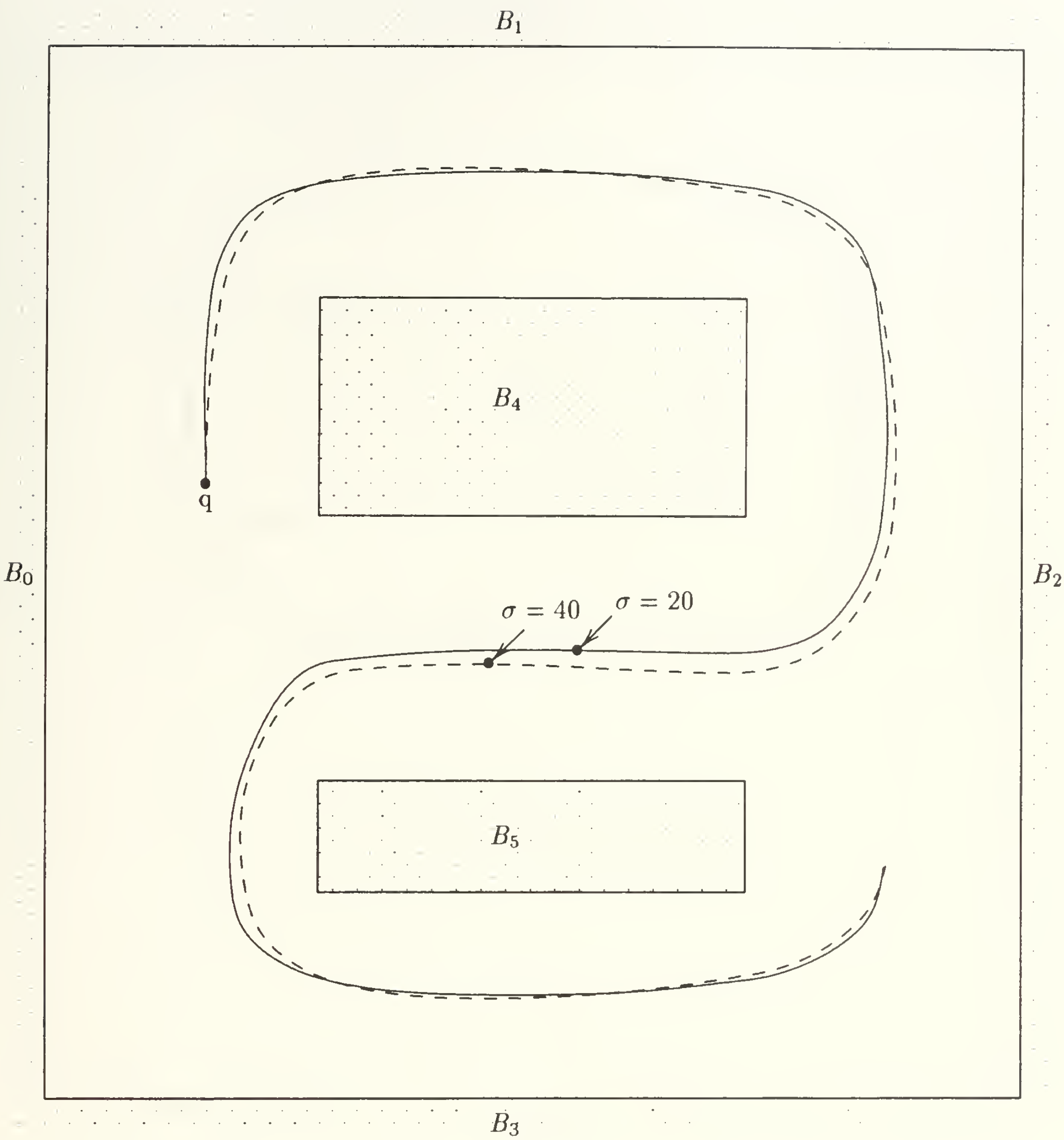


Figure 118: Motion planning and execution result (IV)

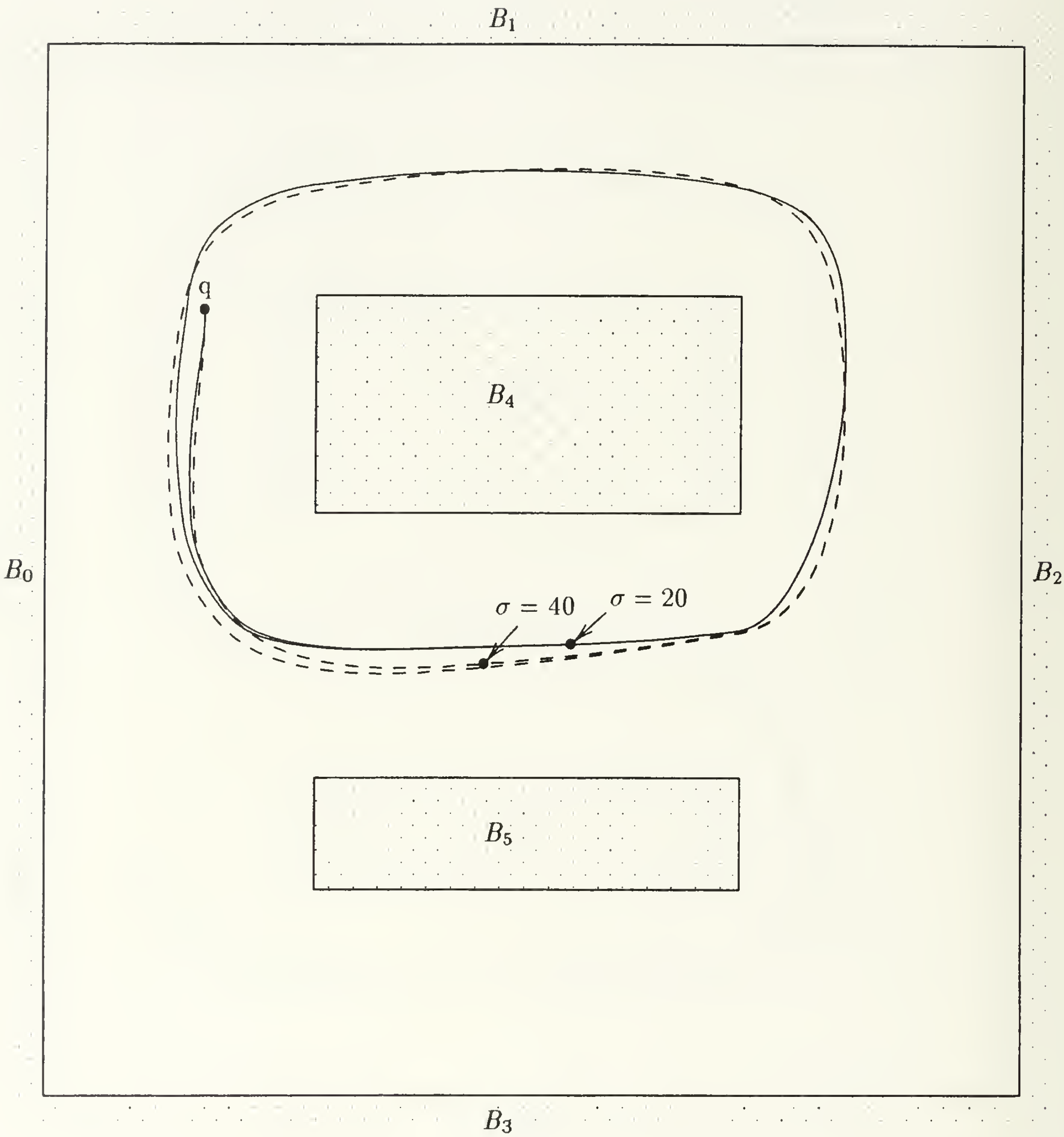


Figure 119: Motion planning and execution result (V)

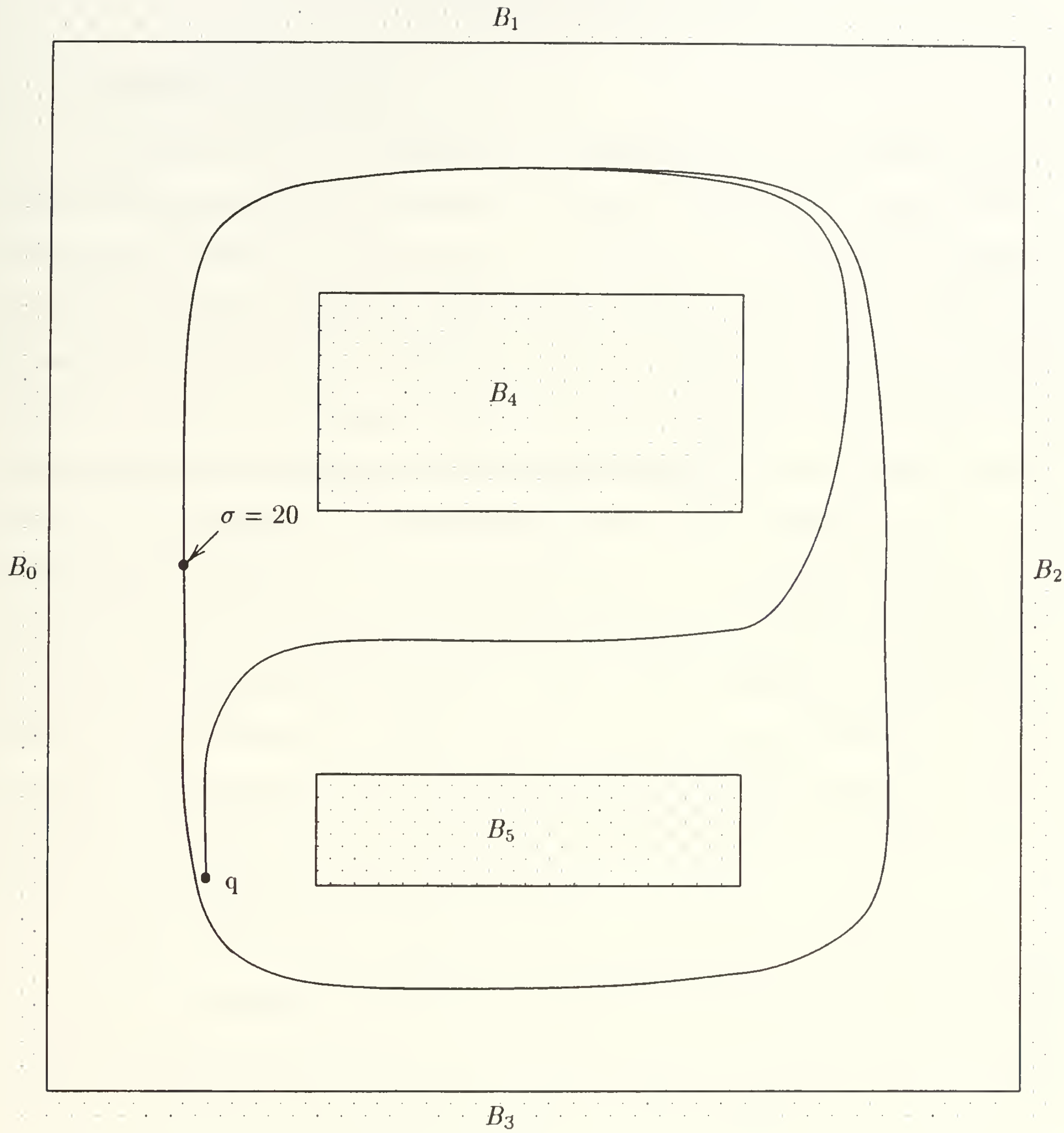


Figure 120: Motion planning and execution result (VI)

VII. SELF LOCALIZATION USING MODEL-SONAR FEATURE CORRESPONDENCE

A. INTRODUCTION

A mobile robot can be assisted in its navigation tasks by providing it with *a priori* knowledge about the environment in which it will navigate, usually called a *world model* or a *map.* One of the issues to be addressed in using a stored model as an aid in mobile robot navigation is that of estimating the position and orientation of the robot with respect to the model. Once the robot accurately estimates its location within the model, other navigation tasks can be performed. Most mobile robots are equipped with wheel encoders that can estimate the robot's relative position at every instant. A key capability of an autonomous mobile robot operating in an indoor environment is *localization*, i.e. determination of its current position and orientation. The usual method for position estimation of a wheeled autonomous mobile robot is *odometry* or *dead reckoning.* However, due to wheel slippage and quantization effects, these estimates of the robot's position contain errors. These errors accrue and can grow limitlessly as the robot moves, causing the position estimate to become increasingly uncertain. So, most mobile robots use additional forms of sensing, such as sonar to aid the position estimation process.

In order to effectively use the stored world model of the environment and the sensor data, it is necessary to establish correspondence between the sensory obsevations and the model information. To deal with this problem, the robot should observe its surroundings and recognize landmarks with its external sensors.

We assume that the vehicle

1. has a geometric model of the static portions of an indoor world,
2. possesses the dead-reckoning capability,
3. executes model-based navigation through these two capabilities, and

4. has sonic sensors.

This chapter introduces an algorithm for self localization. The method used here is based on the two dimensional transformation and least squares linear fitting algorithm [36, 40]. The theory of two dimensional transformation groups [4, 24, 39] is a powerful tool to deal with the positional error evaluation. It is used to calculate the robot's position and motion in a two dimensional region. Feature extraction from sensory data is a basis for model–based navigation of mobile robots. This computationally efficient method allows to correct localization error in real-time. Two dimensional transformation and least square fitting are not a new concept, but using them makes self localization more amenable to human understanding.

B. GOAL AND FEATURES OF SELF LOCALIZATION METHOD

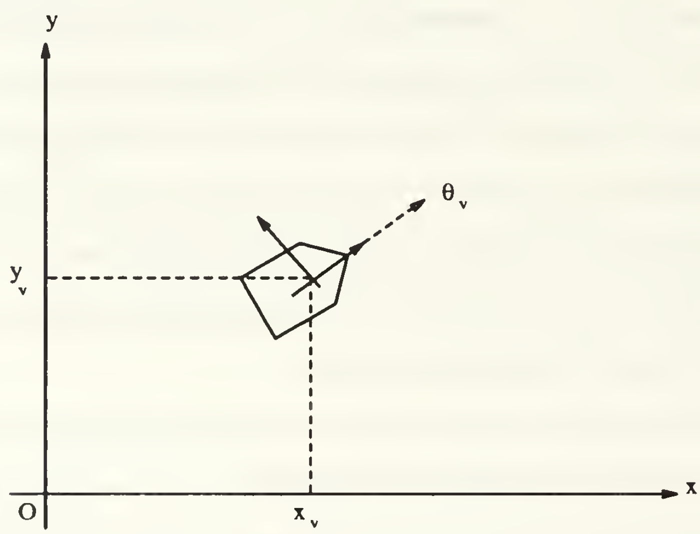


Figure 121. Positioning of rigid body robot as configuration

A rigid-body robot has three degrees of freedom in its positioning: its position p_v (corresponding to x_v and y_v) and heading θ_v (we call the position-heading pair *configuration*) (Figure 121). A useful vehicle must have dead reckoning ability to maintain the current vehicle configuration using its wheels' incremental motions.

However, errors in the configuration obtained by dead-reckoning accumulate over time. It is known that the uncertainty in the position p_v is represented by an ellipse. Our goal is

1. to find a robust algorithm for the vehicle to continually eliminate its positional uncertainty so that the uncertainty ellipse and the directional uncertainty will be reset to a point using the geometrical model of the world and sonars in real time, and

2. to implement this algorithm using the autonomous self-contained mobile vehicle Yamabico-11 for testing and evaluation.

The proposed algorithm and the implementation method have the following features:

1. They use a two-dimensional abstract geometric model of the indoor environment.

2. They use ultrasonic sensors and least squares fitting algorithm to sense the transformations of immobile known edges in the environment.

3. They match a sensed edge transformation *landmark* against the corresponding edge transformation in the model.

4. Odometry correction is done whenever a side-locking sonar scans a known object at an angle nearly normal to its surface. Since this event takes place relatively frequently in a normal indoor environment, the vehicle's location error does not increase indefinitely. Thus, the vehicle's safe motion and correct sensor data interpretation are guaranteed.

5. In the implementation of this algorithm on Yamabico-11, the localization correction task is superimposed in real-time on the current vehicle's main mission. No extra motion or extra time is needed.

6. This algorithm for odometry correction is vehicle-independent.

Through this method, the robot can minimize its positional uncertainty, can make safe and reliable motions, and can perform useful tasks in a partially-known world. Thus, self-localization is actually an essential component of model-based navigation for indoor applications.

C. TWO DIMENSIONAL TRANSFORMATION

In the field of robot manipulators, three-dimensional homogeneous transformation algebra has widely been used in analysis and design [58, 53]. Likewise, we need a framework for analyzing motions of two-dimensional rigid bodies. One obvious method is the two-dimensional version of the homogeneous transformations. This approach has, however, one drawback: the orientation of a rigid body is not explicitly represented. Since placement in a place is simpler than that in a space, there might exist a simpler and more efficient algebra for this purpose.

Two dimensional transformation groups [36] have the same advantage as three-dimensional homogeneous transformations, i.e., translation and rotation are described in a single mathematical structure. The major differences between two-dimensional transformation groups and three-dimensional homogeneous transformations include

1. The vehicle orientation is explicity represented and a transformation in this system keeps the full orientation information beyond the range of $[-\pi, \pi]$.
2. The composition function and inverse function are the only two functions needed to solve all problems related to two-dimensional discrete motion analysis problems.
3. It does not have a point of singularity, one of the drawbacks of the homogeneous transformations. As a result, the inverse function is defined for any transformation.

The analysis of localization errors described in Section D would not be possible without this theory.

1. Definitions

Let $\mathcal{R}$ denote the set of all real numbers.

Definition: A *transformation*, q, is defined by

$$q \equiv \begin{pmatrix} x \\ y \\ \theta \end{pmatrix},$$

where $x, y, \theta \in \mathcal{R}$.

The set of all transformations is denoted by $\mathcal{T}$. For example, $(3, 1, \pi/3)^T \in \mathcal{T}$. Obviously, a transformation q is interpreted as a two dimensional coordinate transformation from the global Cartesian coordinate system $\mathcal{F}_0$ to another coordinate system $\mathcal{F}$.

Definition: The transformation group $\langle \mathcal{T}, \circ \rangle$ consists of the set $\mathcal{T}$ of transformations, where

$$\mathcal{T} = \{(x, y, \theta)^T | x, y, \theta \in \mathcal{R}\}$$

and the binary operator (*composition function*), $\circ$, is defined as follows:

Let $q_1 = (x_1, y_1, \theta_1)^T$, $q_2 = (x_2, y_2, \theta_2)^T \in \langle \mathcal{T}, \circ \rangle$, then

$$q_1 \circ q_2 \equiv \begin{pmatrix} x_1 + x_2 \cos\theta_1 - y_2 \sin\theta_1 \\ y_1 + x_2 \sin\theta_1 + y_2 \cos\theta_1 \\ \theta_1 + \theta_2 \end{pmatrix}.$$

The interpretation of $q_1 \circ q_2$ in the domain of two-dimensional coordinate transformations is the *composition* of the coordinate transformations q_1 and q_2.

Definition: The inverse q^{-1} of a given transformation $q = (x, y, \theta)^T$ is defined as:

$$q^{-1} = \begin{pmatrix} -x\cos\theta - y\sin\theta \\ x\sin\theta - y\cos\theta \\ -\theta \end{pmatrix}.$$

For more details, see [4, 24, 36]

D. LINEAR FEATURE EXTRACTION

1. Calculation of Global Sonar Return

We consider an autonomous mobile vehicle on which a *reference transformation* is defined. The reference transformation is a point with orientation attached on vehicle's body. The current transformation,

$$q_c = \begin{pmatrix} x_c \\ y_c \\ \theta_c \end{pmatrix},$$

describes the robot's current position and orientation in the global frame in terms of the reference transformation. This transformation q_c also defines the local robot coordinate system. Furthermore, we assume a sensor is mounted on the vehicle and its local positioning is described in the local vehicle coordinate system. For instance, if a sensor is mounted at the reference transformation, its transformation is (0, 0, 0,). The transformation,

$$q_{s0} = \begin{pmatrix} x_{s0} \\ y_{s0} \\ \theta_{s0} \end{pmatrix},$$

describes the sensor's position and orientation in the local coordinate system. This sensor's transformation q_s in the global coordinate system is the composite transformation of q_c and q_{s0}, i.e.,

$$q_s = q_c \circ q_{s0}. \qquad \text{(VII.1)}$$

Therefore, if the robot moves, the current transformation q_c changes, and hence, so does the sensor's transformation q_s by Eq. VII.1. If the combination of the robot's transformation q_c and the local transformation q_{s0} of the sensor is appropriate, the ray scans objects in the vehicle's environment to give a set of points of Eq. VII.1. Thus a simple range sensor can obtain an envelope of objects in the robot's environment. This operation is called *scanning.* A scan is not attainable without sensor (vehicle) motions.

For example, let the robot's configuration in the global coordinate system be $q_c = (80, 40, \pi/4)^T$, and the sonar's configuration on the vehicle be $q_{s0} = (0, -20.5, -\pi/2)^T$. The sonar's configuration in the global coordinate (Figure 122), q_s, is:

$$q_s = \begin{pmatrix} 80 \\ 40 \\ \pi/4 \end{pmatrix} \circ \begin{pmatrix} 0 \\ -20.5 \\ -\pi/2 \end{pmatrix} = \begin{pmatrix} 94.5 \\ 25.5 \\ -\pi/4 \end{pmatrix}$$

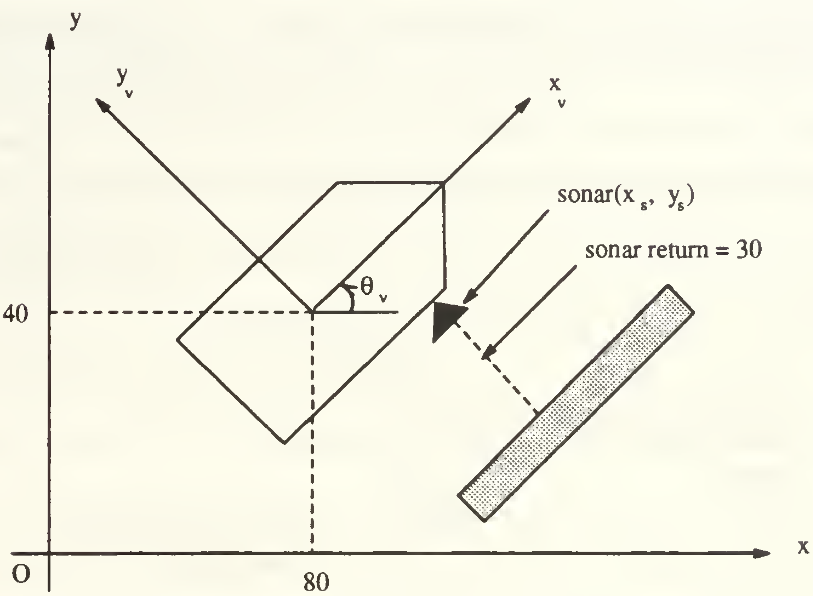


Figure 122. Sonar configuration in global coordinate

There might be an argument that if there are multiple sensors on a robot, multiple range data can be obtained at one time which could also describe the envelope of obstacles. Although this is theoretically correct, the quality of data is not as good as that of data through a single sensor, because it is practically impossible to adjust multiple sensors to have the same sensitivity in amplitude and orientation. One of the most important elements in this method is in that the same sensor is used for a sequence of positional data. This data set is used for the least squares fit algorithm given in subsection 2.

Although a scan is used in combination with various types of motions, two types of scanning, *translational scanning* and *rotational scanning*, are most common. Translational scanning is a mode of scanning in which the vehicle makes forward motion using a side range finder to scan lateral objects. In rotational scanning, the vehicle rotates about its center using a sensor to scan objects radially.

2. Generalized Least Squares Linear Fitting

In addition to simple range and point position data, we desire more abstract features of objects, especially linear features, from a set of positional data [22, 40]. This is accomplished in reverse fashion, i.e. we presume the data we are receiving belongs to such a set and continuously modify a descriptive line segment to a best fit of the data using a least squares fitting algorithm. This line segment continues to grow until the incoming data or certain measures of the line segment indicate that the line segment should be ended and a new one started.

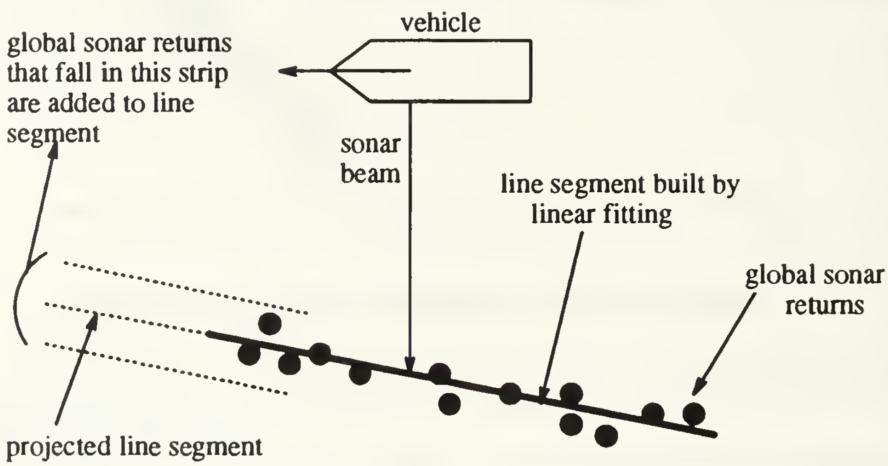


Figure 123. Least square linear fitting procedure

We want to extract a linear feature from a set of points obtained by a scan. We will use a least-squares linear fitting method. In "APPENDIX. LEAST SQUARES LINEAR FITTING", we review some definitions about the least squares fit method [28]. Linear fitting of global sonar data for a given sonar is performed in order to extract line segments representing the sonar reflecting surface in robot's world space.

The linear fitting algorithm examines each individual global sonar return (this data set is obtained by Eq. VII.1), and determines if it can be fitted to the current line segment. When ten or more points fall onto a straight line (with a user's selected tolerance), the linear fitting algorithm builds a line segment for a particular sonar. Linear fitting continues as long as sonar returns fall onto the line segment under construction. Linear fitting is terminated when one global sonar return fails to fall onto the projected line segment being constructed (Figure 123).

E. PRINCIPLES OF REDUCING UNCERTAINTY

The operational conditions in this context are

1. the vehicle knows its estimated configuration through dead reckoning,
2. the vehicle knows the geometrical relation of the world and the proximity information related,
3. the vehicle knows the local configuration of every sonar, and hence, knows, knows its global configuration, and
4. we have actual data from sensors, whose characteristics are known.

Therefore, if the vehicle's dead-reckoning is correct, we can consistently interpret the sensor data. However, if there is any error in the vehicle dead-reckoning, some inconsistency in the sensor data interpretation will be recognized. By comparing the information pieces (2), (3), and (4), we will be able to evaluate the error of dead-reckoning and can reduce the uncertainty. This is the basic principle of self-localization.

Typically, we consider three situations where the positional uncertainty can be reduced.

1. A sonar obtains a range value against a wall at an approximately right angle or against a concave corner. In this case, we have "one degree of constrants," and the vehicle's x coordinate, y coordinate, or a linear combination of both can be corrected. By this process, the uncertainty ellipse of positions becomes a line segment. We generally cannot reduce uncertainty in the vehicle heading by this information.

2. If the robot moves along a wall, its side sonar scans the wall at a right angle. In this case, by applying a linear fitting algorithm (see Figure 123), the robot obtains a line segment, which contains "two degrees of constraints." Therefore, the vehicle's x and θ, for instance, can be corrected. Through this operation, the uncertainty ellipse becomes a line segment and the uncertainty in the vehicle heading becomes one point.

3. If the wall ends in the previous situation, we obtain a line segment with an endpoint (see Figure 123). That information contains the full "three degrees of constraints," and we can make a correction of the whole vehicle configuration. Through this operation, the uncertainty ellipse becomes a point and the uncertainty in the vehicle heading becomes one point.

It is crucial in this method that these operations (1), (2), or (3) are frequently executed so that the dead-reckoning error is always kept small and the robot never misses correct matching between a feature obtained by a sonar and one in the geometric model. Also, in order to make this self-localization possible, the linear fitting process must be done on the robot's on-board software system in real-time.

F. SELF LOCALIZATION ALGORITHM

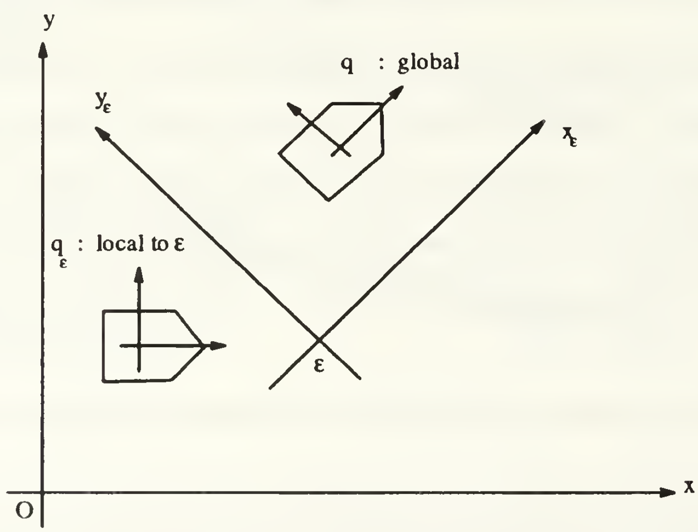


Figure 124. Robot's localization error (I)

Using a two dimensional transformation and linear fitting method, we are now in a position to formulate an algorithm for estimating the position of a robot vehicle.

Let q_v be the vehicle's actual (true) configuration and q_ϵ its estimated configuration by localization. If there is no localization error, $q_\epsilon = q_v$. Otherwise, there is a difference between where the vehicle "thinks" it is (q_ϵ) and where the vehicle "really" is (q_v) (Figure 124). In order to deal with the relation between the two configurations, We propose to define an *error configuration* ϵ such that

$$\epsilon \circ q_v = q_\epsilon \tag{VII.2}$$

i.e., this robot believed its world is ϵ, which is different from the real (global) coordinate system. If q_v and q_ϵ are determined, then the error configuration can be calculated by

$$\epsilon = q_\epsilon \circ q_v^{-1}.$$

For example, if $q_v = (100, 0, 0)^T$ and $q_\epsilon = (101, 0, 0)^T$, then

$$\epsilon = q_\epsilon \circ q_v^{-1} = \begin{pmatrix} 100 \\ 0 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 101 \\ 0 \\ 0 \end{pmatrix}^{1} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}.$$

Note that, $q_\epsilon = (1, 0, 0)^T$ is correct if it is interpreted as a local configuration in ϵ.

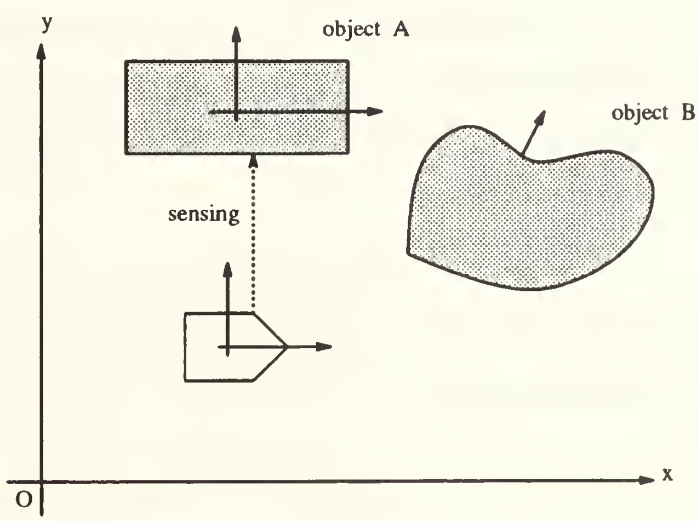


Figure 125. Object configurations

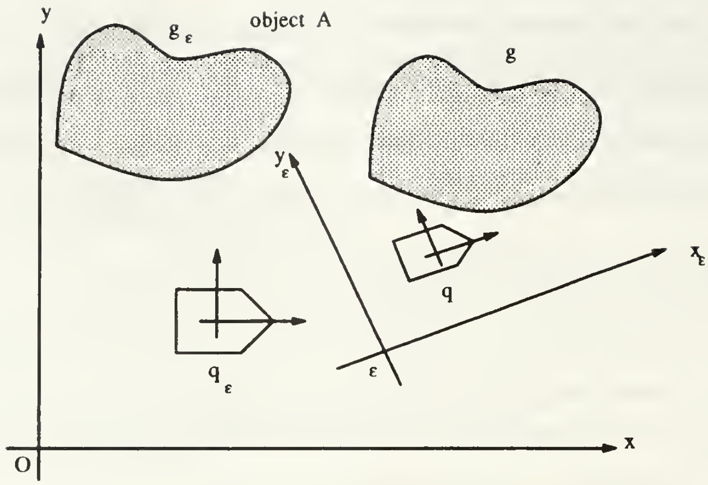


Figure 126. Robot's localization error (II)

The positioning of not only a vehicle but also that of any object in the environment may be described by a configuration. Associated with each object is its local coordinate system; its configuration in this world is described using this local frame of reference (Figure 125). We assume there is an object B_1 whose actual configuration is g (Figure 126). Assume that a sensor, mounted on the vehicle, senses the configuration on an object in the environment. The sensor's capability is assumed to be ideal. That is, the vehicle is able to sense the relative configuration of an object with respect to its own local configuration q_ϵ with an infinite precision. Let g_ϵ be the configuration sensed by the vehicle. Therefore, g_ϵ may be superimposed with the error contained in the localization vehicle configuration q_ϵ. Therefore, the relation between g and g_ϵ is

$$\epsilon \circ g = g_\epsilon. \tag{VII.3}$$

Since the error configurations ϵ in Eqs. VII.2 and VII.3, are the same, we can find the actual vehicle's configuration q_v by

$$\begin{aligned} q &= \epsilon^{-1} \circ q_\epsilon \\ &= (g_\epsilon \circ g^{-1})^{-1} \circ q_\epsilon \\ &= g \circ g_\epsilon^{-1} \circ q_\epsilon \end{aligned} \tag{VII.4}$$

assuming q_ϵ, g and g_ϵ are known (g is given as the knowledge of the world for the robot).

Eq. VII.4 gives a formal way to evaluate the actual configuration q_v of the vehicle using a model and sensors, where

1. q_v is the vehicle's actual configuration, which is unknown,
2. g is the actual configuration of an object in the environment, which is obtained from an environment model,
3. q_ϵ is the localization configuration, which is known but contains an error ϵ, and
4. g_ϵ is the observed configuration of the object, which is also known and may have some error because this observation is made by the ideal sensor on board, using localization configuration q_ϵ as a point of reference.

Next Subsections 1 and 2 show how to evaluate the actual configuration of an object g and the observed configuration of the object g_ϵ.

For example, if $q_\epsilon = (1,1,\pi/2)^T, g_\epsilon = (1,2,\pi/2)^T$, and $g = (2,4,0)^T$, then

$$\epsilon = g_\epsilon \circ g^{-1} = \begin{pmatrix} 1 \\ 2 \\ \frac{\pi}{2} \end{pmatrix} \circ \begin{pmatrix} 2 \\ 4 \\ 0 \end{pmatrix}^{-1} = \begin{pmatrix} 5 \\ 0 \\ \frac{\pi}{2} \end{pmatrix}, \textit{ and}$$

$$q_v = \epsilon^{-1} \circ q_\epsilon = \begin{pmatrix} 5 \\ 0 \\ \pi/2 \end{pmatrix}^{-1} \circ \begin{pmatrix} 1 \\ 1 \\ \pi/2 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 0 \end{pmatrix}.$$

To validate the self-localization algorithm, we implemented the algorithm on the autonomous mobile vehicle Yamabico-11 (see Chapter VIII).

1. Position Information of Natural Landmarks

When we project a three dimensional world onto a two dimensional plane, a vertical plane is projected to a straight edge. There are numerous edges in an environment as a part of a wall or a part of furniture. We consider some of those edges as landmarks for navigational purposes.

Let e be an edge with endpoints p_1 and p_2. We can define a configuration $g_e \equiv (p_1, \theta_e)$ with it. The orientation θ_e is equal to the orientation from p_1 to p_2. Thus we can obtain the actual configuration $g = g_e$ in Eq. VII.4 for an edge e.

2. Position Estimation of Natural Landmarks by Sonar and Odometry

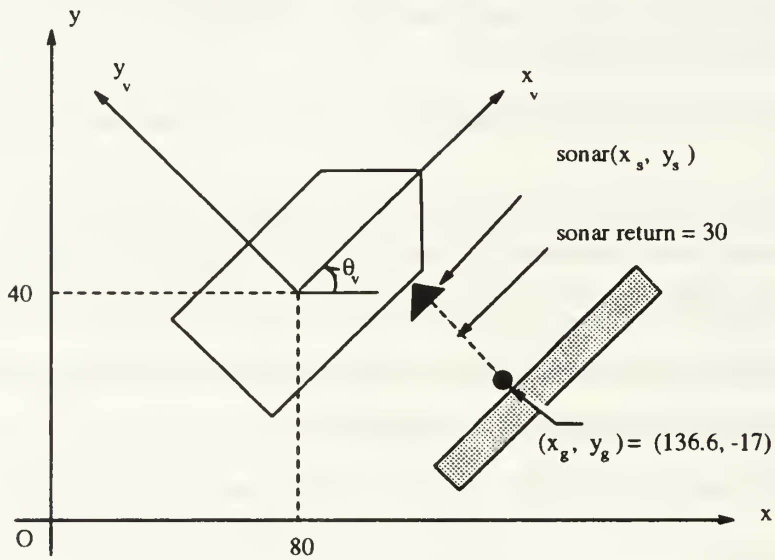


Figure 127. Global position of sonar return

Obtaining the configuration g_e for the edge e using a sonar is accomplished as follows. We propose *translational scanning* including the general least square linear fitting algorithm for obtaining the observed configuration g_e for the edge e using a sonar (see Subsection 2 of Section D).

First, during a vehicle's translational motion, assume a sonar obtains a range value d by a sonar whose instantaneous configuration is $q_{s0} \equiv (x_s, y_s, \theta_s)^T$ (see Figure 122). The sonar's configuration in the global coordinate, Eq. VII.1, is a composition of the vehicle odometry configuration q_e and the sonar local configuration q_{so} in robot-local coordinates. In this context, the sonar configuration includes odometry error. An estimate of the position of a point p on an object that generated a sonar return in the global coordinate system is

$$p = (x_s + d\cos\theta_s, y_s + d\sin\theta_s)$$

For example, if the sonar return is 30 cm and the sonar's configuration in the global coordinate is $q_s = (94.5, 25.5, -\pi/4)^T$, the global position (x_g, y_g) of sonar return (see Figure 127) is given by

$$x_g = 94.5 + 30 * \cos(-\pi/4) = 136.8$$

$$y_g = 25.5 + 30 * \sin(-\pi/4) = -17$$

By knowing where each sonar is on the vehicle (see Table XVI in Chapter IX) and knowing the vehicle's position, we can consistently determine the object's location relative to the robot's world.

The second step is to calculate the moments up to the second order at each new incoming value. With these moments, the equation of the line $L = (r, \alpha)$ (where α and r are the orientation and length of a normal against L from the origin $(0, 0)$) with the least squares fit and the best estimates of the endpoints of L can be obtained (See "APPENDIX. LEAST SQUARES LINEAR FITTING").

The final important step is to determine if the new incoming point should be included in the group of points representing a line.

When one session of the linear fitting process ends, this process returns a pair of endpoints (p_1, p_2) as a result. Obtaining the observed object configuration g_ϵ is done in the same manner as described in previous Subsection 2.

3. Odometry Correction

Assume a situation in which the vehicle knows its actual configuration q_v and the vehicle is moving. When the landmarks are located in the environment and the robot can detect a landmark, the observed segment configuration g_ϵ is obtained. If there is a difference between the observed segment configuration g_ϵ and the actual landmark edge configuration g (see Figure 128), the robot can correct its estimated

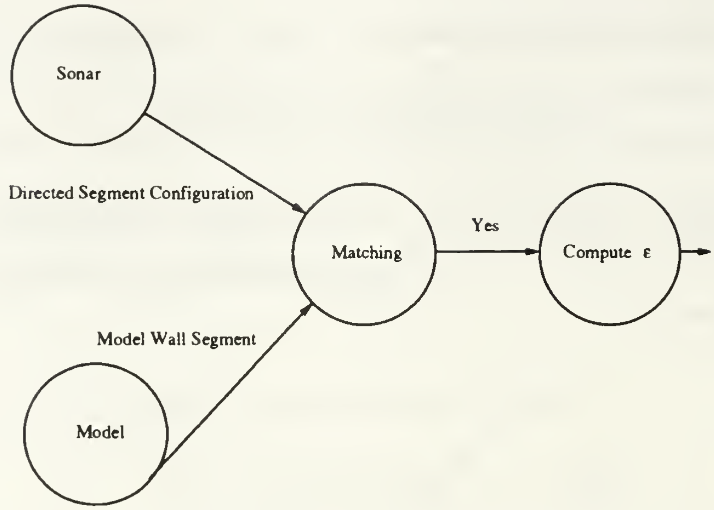


Figure 128. Matching algorithm

position before the error accumilates to be large. For example, in Figure 129, the vehicle believes it is at q_ϵ, which is on the specified directed path π. Actually, though, the vehicle is at q_v and was going to move on a wrong trajectory. Odometry correction is made by simply substituting the odometry configuration with q_v. This causes the odometry configuration to be the true one, and therefore, lets the control algorithm recognizes the non zero distance between the vehicle's configuration and the directed path π. This control algorithm then pulls the vehicle back on track (Figure 129) [38].

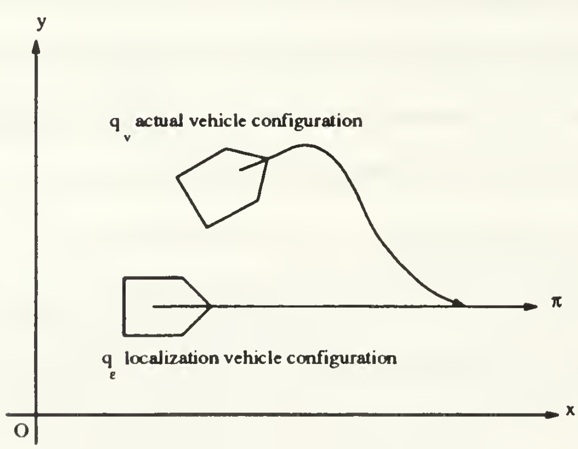


Figure 129. Real-time localization correction

VIII. IMPLEMENTATION OF LOCAL MOTION PLANNING AND SELF LOCALIZATION ALGORITHMS

This chapter describes how to implement the local motion planning algorithm. The chapter will cover each of these in the following sequence. First, the data structures used to represent the world are presented. Second, The experimental results conducted by Yamabico-11 using the MML-11 software system of polygon tracking and local motion planning algorithms will be presented. Third, experimental results of application of self-localization algorithm on an autonomous mobile robot system Yamabico-11 using sonars and natural landmarks will be discussed.

A. GEOMETRIC MODEL OF A ROBOT'S WORLD

This section describes the data structures used to represent the world and the path classes. We propose to represent the robot's world by specifying the vertices of the polygonal holes. Each hole, then, becomes an ordered list of vertices such that traversing the list corresponds to traversing the hole's boundary with the free space on the right. In other words, vertices of *ccw* holes (polygons) are ordered counter-clockwise, while vertices of *cw* holes are ordered clockwise. Since information is commonly needed about a vertex's neighbors, the specific data structure used for implementation must be able to efficiently identify its next and previous vertices. Storing the vertices in a doubly linked list is one alternative.

1. World Model Data Structure

The data structures required include a world structure used to hold information concerning the polygons that make up the world, a subpolygon table to define the subpolygons.

The *world*, illustrated in Figure 130, is represented as a linked list of polygons, where each *polygon* is a double linked list of its vertices. Access to the world is gained

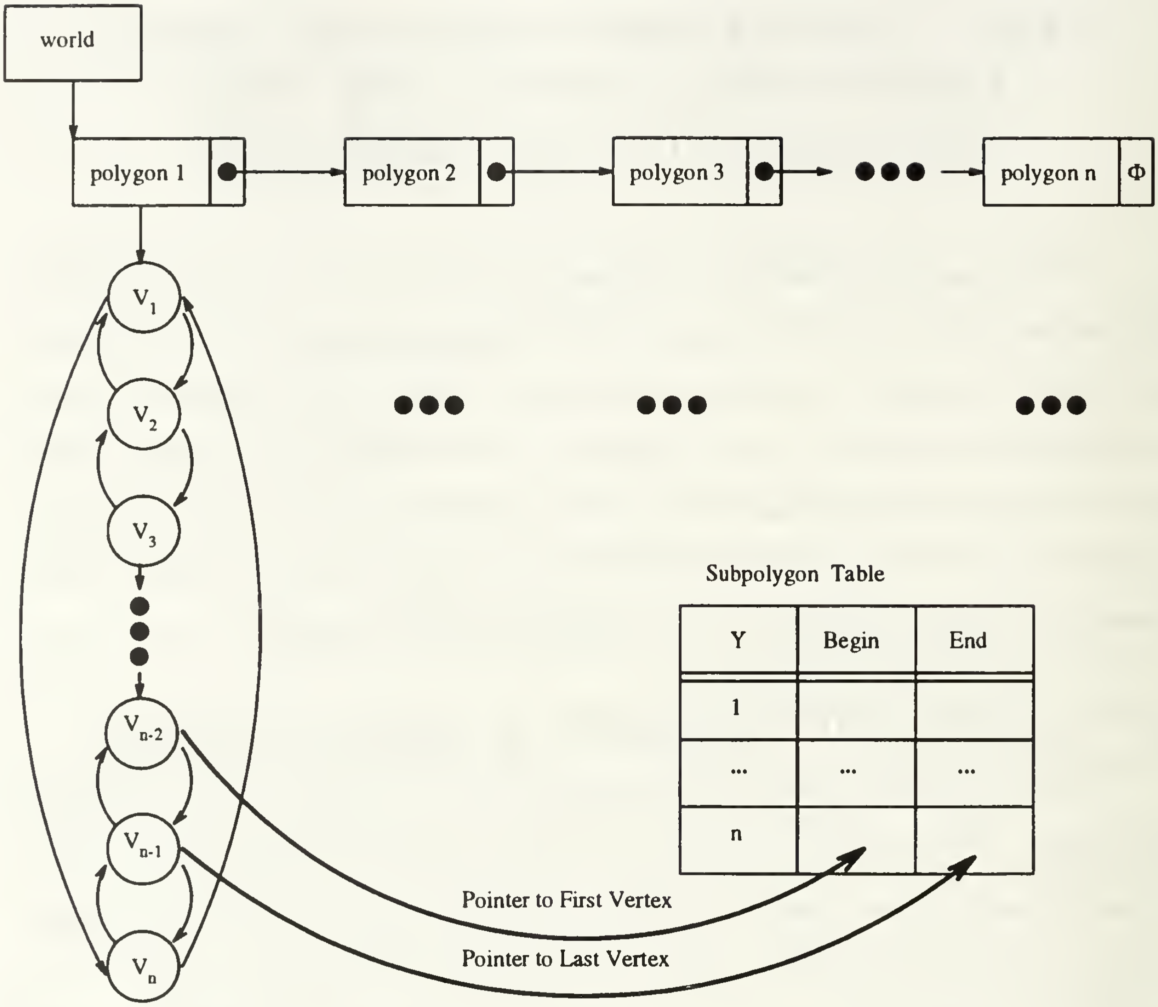


Figure 130. Representation of world data structure

through a pointer to one of the polygons on the list. As the vertices are read, the subpolygons of each polygon are created. The *vertex* structure contains the identity of the vertex, the coordinates of each vertex, and whether or not the vertex is a convex vertex.

The *Subpolygon Table* provides a means of finding all vertices which are contained in a given subpolygon. This data structure is an array which holds a pointer to the first and last vertex in the subpolygon (see Figure 130). Given that the identity of the subpolygon is known, it is used to find the image on the subpolygon. If a subpolygon is convex, then the first and last vertex are identical.

2. Path Class Data Structure

For a path f in a world $\mathcal{W}$, the "path class", π, is represented by a directed v-edges sequence Ξ. This data structure is an array of structures containing a left and right subpolygon identification (see Table XII).

Left Subpolygon	Right Subpolygon
Υ_i	Υ_j
...	...
Υ_m	Υ_n

Table XII. Representation of path class data structure

3. Image Data Structure

An *image structure* contains the identity of the feature type (i.e., edge or vertex) which contains the image point, pointer to vertex v_i (in vertex type, vertex v_i is one of the vertices of B but in edge type, the image lies on edge $\overline{v_i\varphi(v_i)}$), the orientation from a point p to an image, and the closest distance from a point p to its image (see Table XIII). Following each motion cycle of the vehicle, image is updated.

Image Structure
Object Type Containing Image (Vertex or Edge)
Pointer to a vertex v
Orientation
Closest Distance

Table XIII. Representation of image data structure

In Table XIII, **Object Type** is integer type which indicates image type. The type of orientation and closed distance from a point p to its image are double.

B. POLYGON TRACKING EXPERIMENTAL RESULTS

The polygon tracking algorithms described in Chapter V have been implemented in MML-11 (see Chapter IX), and tested on experimental robot Yamabico-11. The results show that the algorithms are practical for the robot motion planning and motion control.

In Figures 131, the vehicle is supposed to track a *ccw* polygon with *ccw* direction, where its initial configuration $q_0 = ((63, 450), -\pi/2, 0)$, the safety clearance $d_0 = 80$, the speed $v = 30cm/sec$, and the value of smoothness, $\sigma = 20$.

The example in Figure 132 shows the result of the trajectory if the polygon is not rectlinear. This means that our algorithm is sufficiently general for arbitrary polygons. The vehicle is supposed to track a *ccw* polygon with *ccw* direction, where its initial configuration $q_0 = ((90, 450), -\pi/2, 0)$, the safety clearance $d_0 = 80$, the speed $v = 30cm/sec$, and the value of smoothness, $\sigma = 20$.

C. LOCAL MOTION PLANNING EXPERIMENTAL RESULTS

Most of the motion planning algorithms described in this dissertation have been implemented in MML-11 (see Chapter IX), and tested on Yamabico-11. As above, the results show that the tested algorithms are applicable to the robot motion planning and motion control. The example in Figure 133 shows the result of different trajectories for the following path class.

$$\pi = [B_4/B_0][B_4/B_5][B_2/B5[B_3/B_5].$$

The initial configuration is $q_0 = ((63, 450), -\pi/2, 0)$, the safety clearance is $d_0 = 80$, the speed is $v = 30cm/sec$, and the value of smoothness is $\sigma = 20, 30$.

In Figure 134, the vehicle's initial configuration is $q_0 = ((63, 450), -\pi/2, 0)$, the safety clearance is $d_0 = 80$, the speed is $v = 30cm/sec$, the value of smoothnessis $\sigma = 20$ and the path class is

$$\pi = [B_4/B_0][B_4/B_5][B_4/B_2][B_4/B_1][B_4/B_0][B_4/B_5].$$

The example in Figure 135 shows the result when a vehicle is browsing randomly in the free space. The vehicle tracks the following path class where its initial configuration is $q_0 = ((90, 120), \pi/2, 0)$, the safety clearance is $d_0 = 80$, and the speed is $v = 30cm/sec$. The path class is

$$\begin{aligned} \pi &= [B_0/B_5]\ [B_4/B_5]\ [B_4/B_2]\ [B_4/B_1]\ [B_4/B_0]\ [B_5/B_0]\ [B_5/B_3] \\ &\quad [B_5/B_2]\ [B_4/B_2]\ [B_4/B_1]\ [B_4/B_0]\ [B_5/B_0]\ [B_5/B_3]. \end{aligned}$$

D. SELF LOCALIZATION EXPERIMENTAL RESULTS

To validate the self localization algorithm (see Section F in Chapter VII), we implemented the algorithm on the autonomous mobile vehicle Yamabico-11. The set of odometry-correction-related functions were incorporated into the MML function library (see Chapter IX).

In the following subsection, we explain one experiment to verify the fundamental correctness of the algorithm.

1. Single Landmark Experiment

In this experiment, a single racetrack path with a single landmark was used. Yamabico moves repeatedly around this racetrack path which is composed of three separate path elements. Yamabico is programmed to make an odometry correction once per lap using a single landmark. In each lap of this racetrack path execution, the odometry correction is performed and the error configuration ϵ is recorded. The resulting robot motion after applying odometry correction code is shown in Figure 136. Table XIV shows the raw experimental data obtained for the robot traveling ten laps at 30 cm/sec. Notice that the results show the error configurations for each lap are small and nearly equal. This provides evidence that Yamabico's motion control and localization functions are precise and that the self localization algorithm is working as desired.

Lap	x (cm)	y (cm)	θ (radians)	θ (degree)
1	2.80471	0.24929	-0.00024	-0.01376
2	0.69485	0.42072	-0.00286	-0.16395
3	1.00984	0.42923	-0.00137	-0.07897
4	0.13315	0.29099	-0.00244	-0.14047
5	-0.89826	0.46305	-0.00444	-0.25449
6	0.58927	0.49313	-0.00075	-0.04326
7	-0.05586	0.10672	-0.00190	-0.10898
8	0.46601	0.36223	-0.00084	-0.04867
9	0.21211	0.95825	-0.00917	-0.05254
10	0.28372	0.19450	-0.00070	-0.04016

Table XIV. Odometry error correction (30 cm/sec)

The average of the error configuration over ten laps at speed of 30 cm/sec is shown in Table XV.

Δx (cm)	Δy (cm)	$\Delta\theta$ (radians)	$\Delta\theta$ (degree)
0.52395	0.39659	-0.00247	-0.09451

Table XV. Average odometry error correction (30 cm/sec)

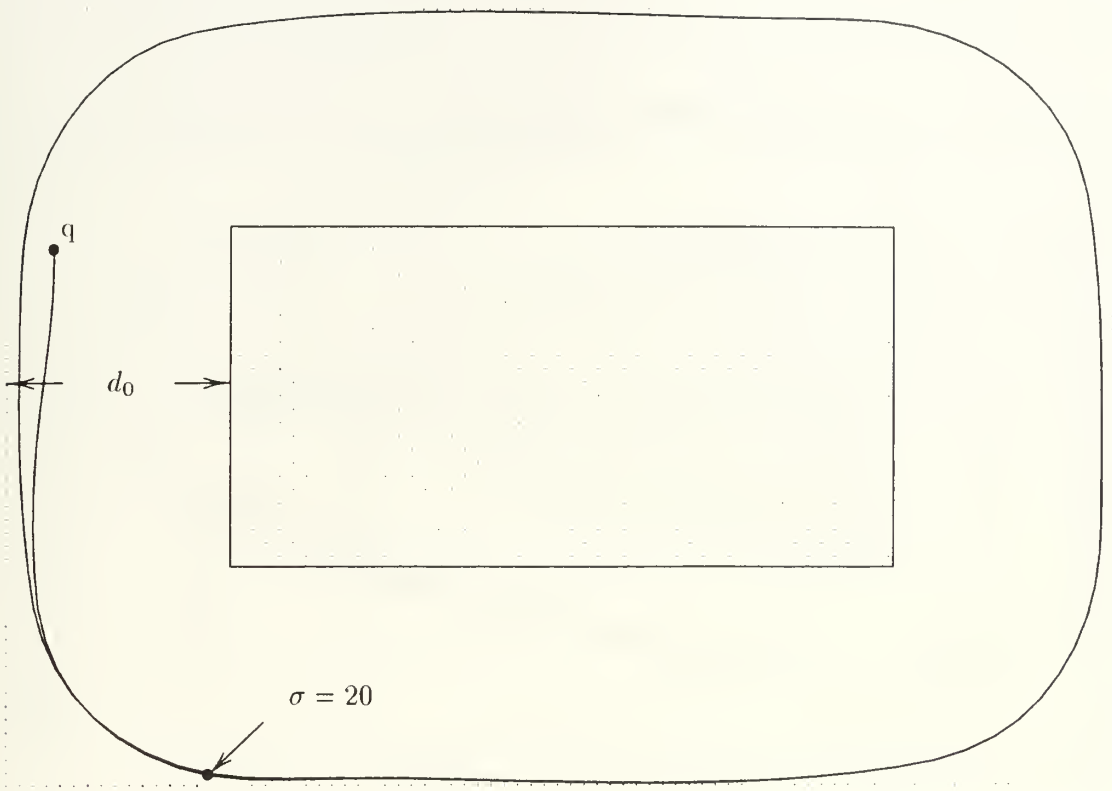


Figure 131: Yamabico-11 polygon tracking and execution result (I)

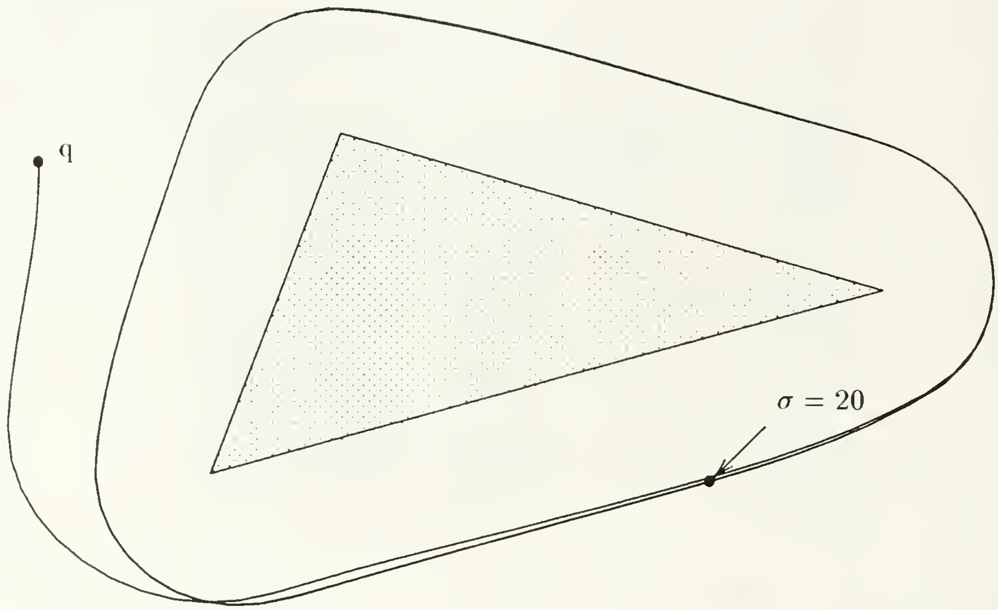


Figure 132: Yamabico-11 polygon tracking and execution result (II)

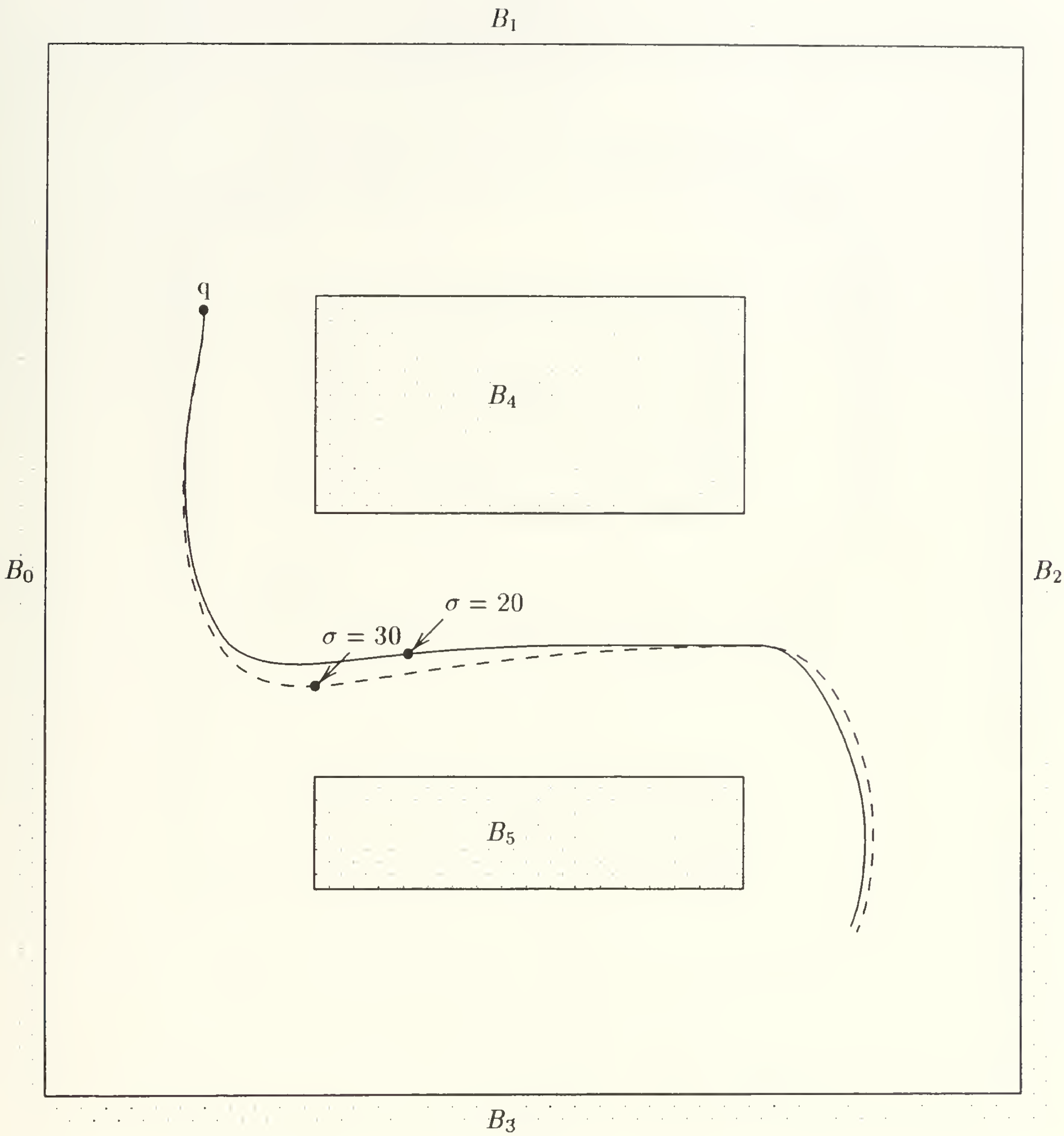


Figure 133: Yamabico-11 local motion planning and execution results (I)

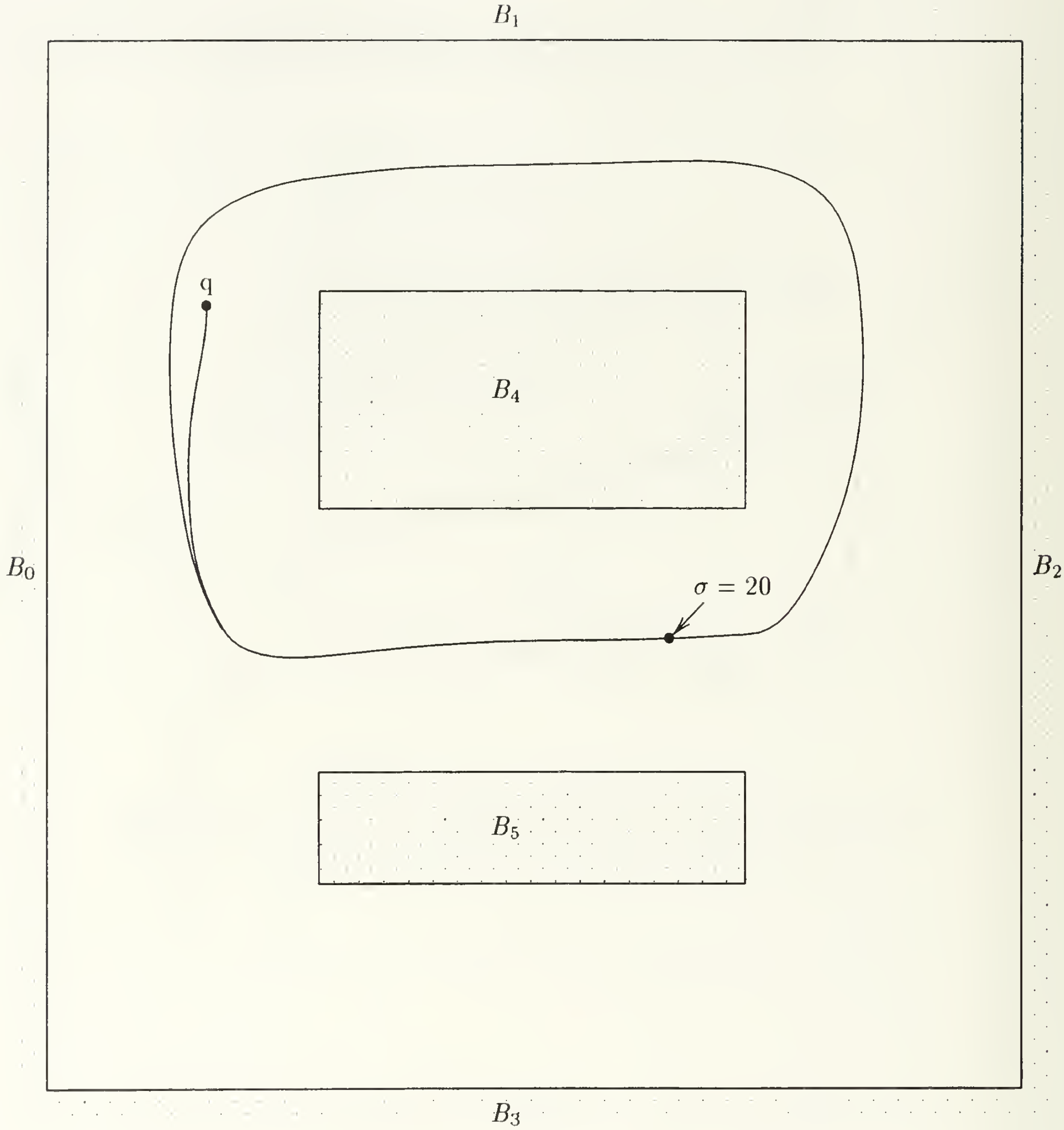


Figure 134: Yamabico-11 local motion planning and execution result (II)

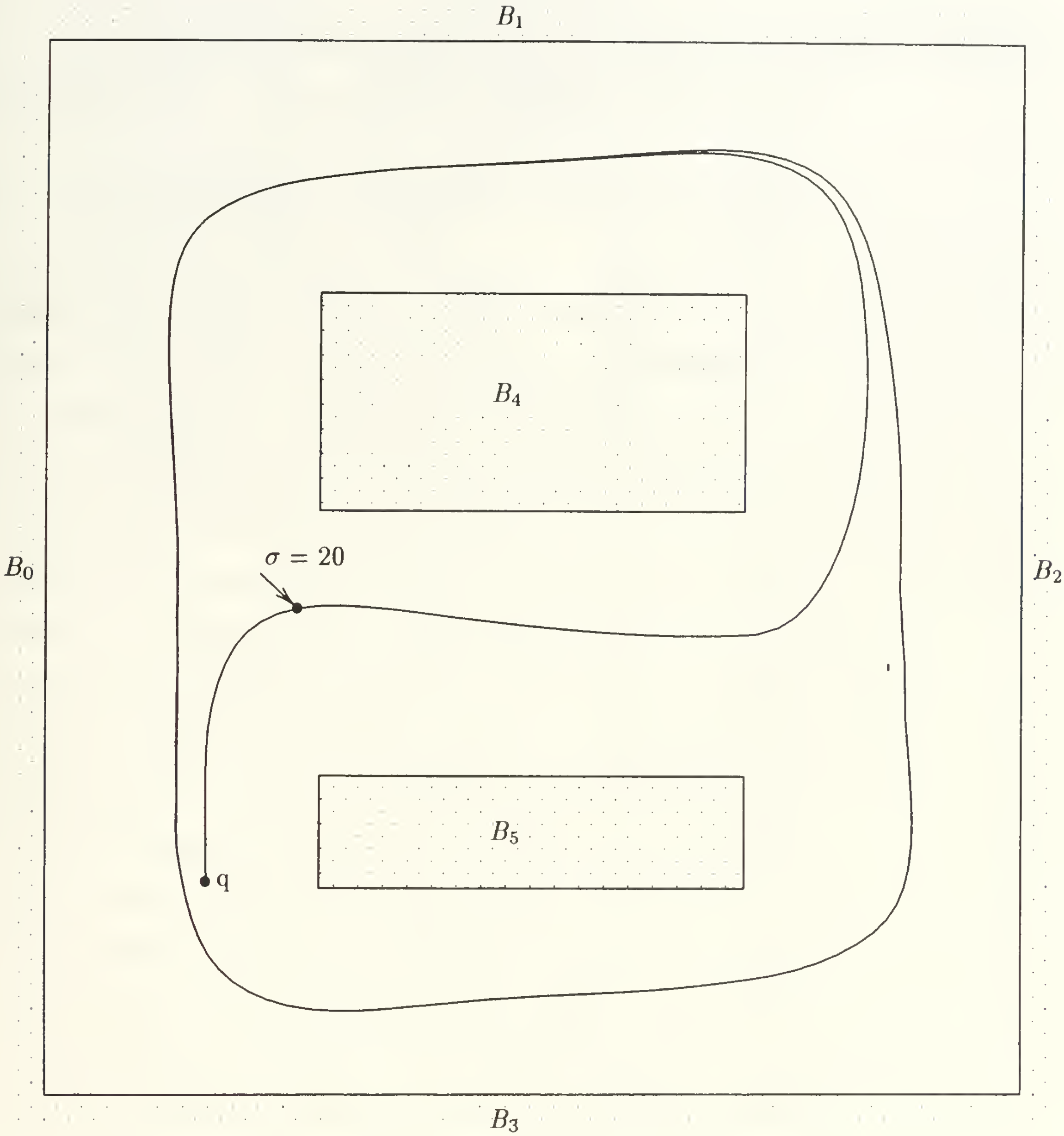


Figure 135: Yamabico-11 local motion planning and execution result (III)

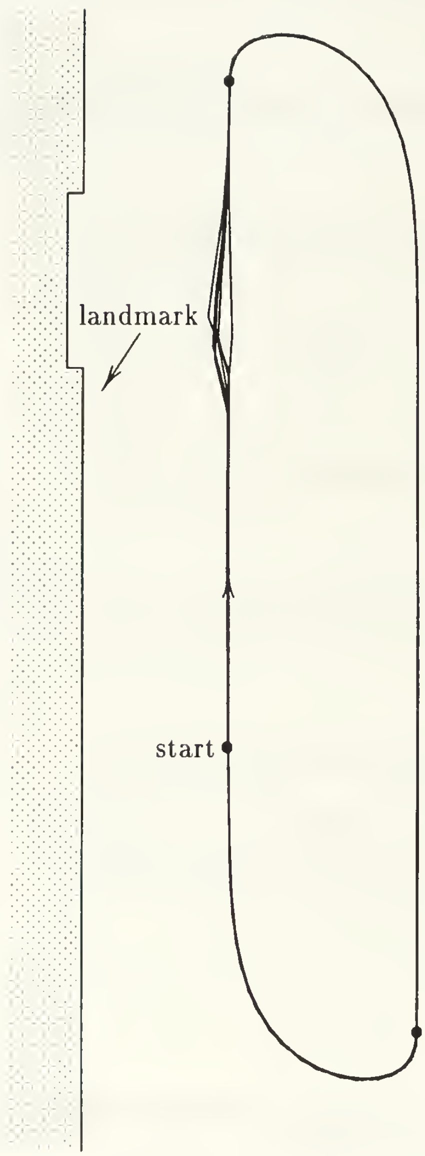


Figure 136: Odometry correction experimental using single landmark

IX. YAMABICO-11 HARDWARE AND SOFTWARE ARCHITECTURE

This Chapter introduces the hardware and software of the robot—Yamabico-11 which was used to test most of our algorithms experimentally.

A. HARDWARE SYSTEM OF YAMABICO-11

Yamabico-11 (see Figure 137) is an experimental, wheeled untethered indoor mobile robot for AI and robotics research. It has been developed at the Naval Postgraduate School (NPS) over the last several years. However, the vehicle is a result of Dr. Yutaka Kanayama's long history of autonomous robotics research at the University of Electro-Communications, the University of Tsukuba, Stanford University, and the University of California at Santa Barbara [38, 41]. Its main CPU board consists of the SPARC microprocessor with a 16 Mbyte RAM storage and is mounted on a VME bus. Besides that, the system includes a dual-axis controller for two motors and two shaft encoders, a tailor-made sonar board, and a serial communication board are also mounted on the VME bus. One lap-top computer is used for a real-time input/output device. The size is 60(W) by 60(L) by 70(H) centimeters. It weighs about 60 Kilograms. A differential drive kinematic architecture is used for the wheel system. Two 35 watt DC motors with shaft encoders are used with 1/24 gear boxes. Twelve 40KHz sonars and one CCD camera are mounted on board. Its power source consists of two 12-volt car batteries. When object code is downloaded from a UNIX system, the vehicle operates as an untethered (self-contained) autonomous robot. The Yamabico-11 hardware architecture is illustrated in Figure 138.

1. IV-SPARC-33 CPU

The Ironics IV-SPARC-33 is a single processor, VMEbus Interface, CPU board. It contains a 25 MHz SPARC Integer Unit, a Floating Point Unit, and a Cache Controller and Memory Management Unit. The card installed in Yamabico has 64 Kbytes

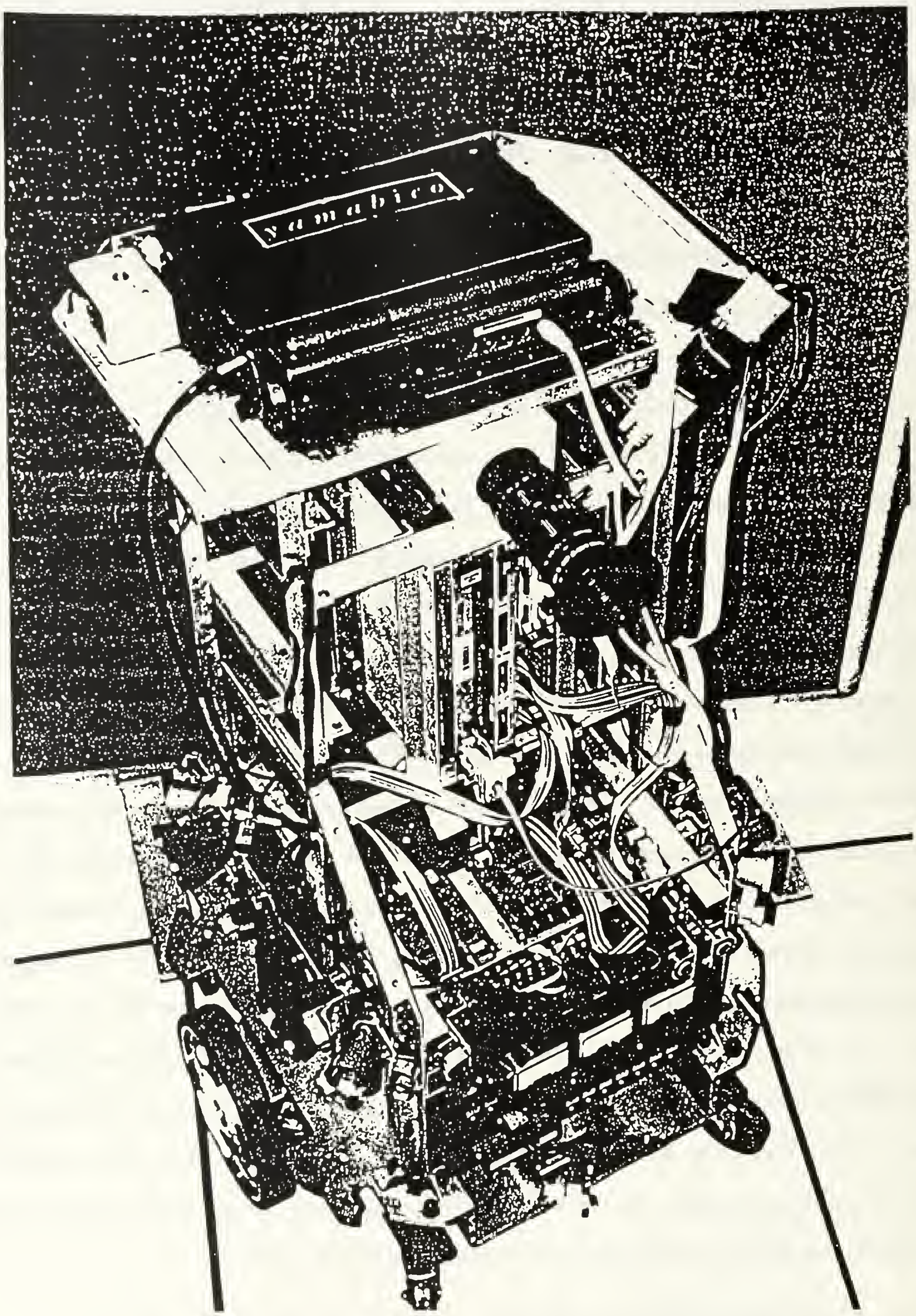


Figure 137. Autonomous mobile robot, Yamabico-11

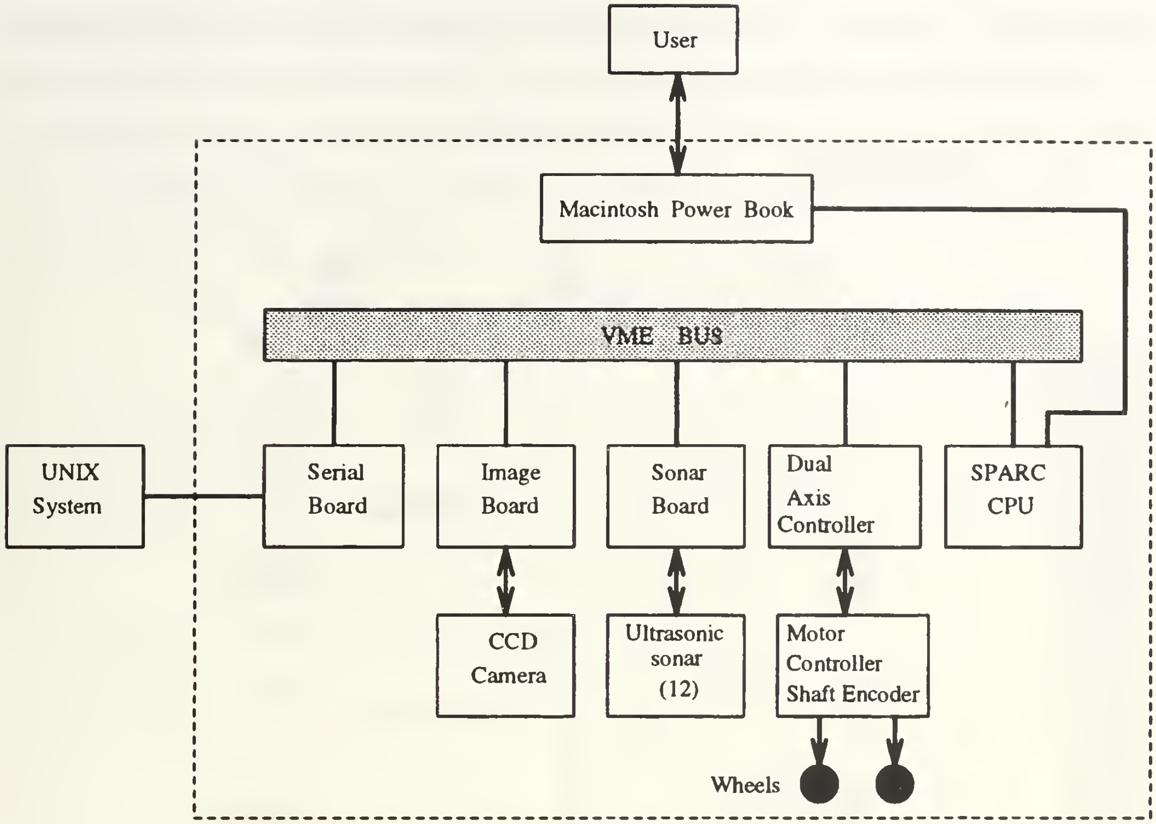


Figure 138. Block diagram of Yamabico-11 hardware architecture

of cache, and 16 Mbytes of 80ns DRAM. It provides two RS-232 serial I/O ports, two programmable timers, and seven user-definable LEDs.

The Ironics SPARC board contains 16 Mbytes of physical memory, yet provides 32 bit addresses (4 GBytes). This 4 GBytes address space is logically divided into several regions. The three most important regions are the Local DRAM, Region 3, and Local I/O (see [34]).

Internal interrupts are those generated on the CPU board. The two most important are the Timer 1 and Timer 2 interrupts. Timer 1 can be set to provide interrupts at 50, 100, or 1000 hz. Currently, MML11 uses Timer 1 to provide the 10ms (100 Hz) motion control interrupt. Timer 2 provides a broader range of interrupts, and is currently unused.

External interrupts are those generated off the CPU board. The most important are from the quad serial boards, and the sonar board, which are handled through the 7 VMEbus Interrupt Request lines.

2. SONARS

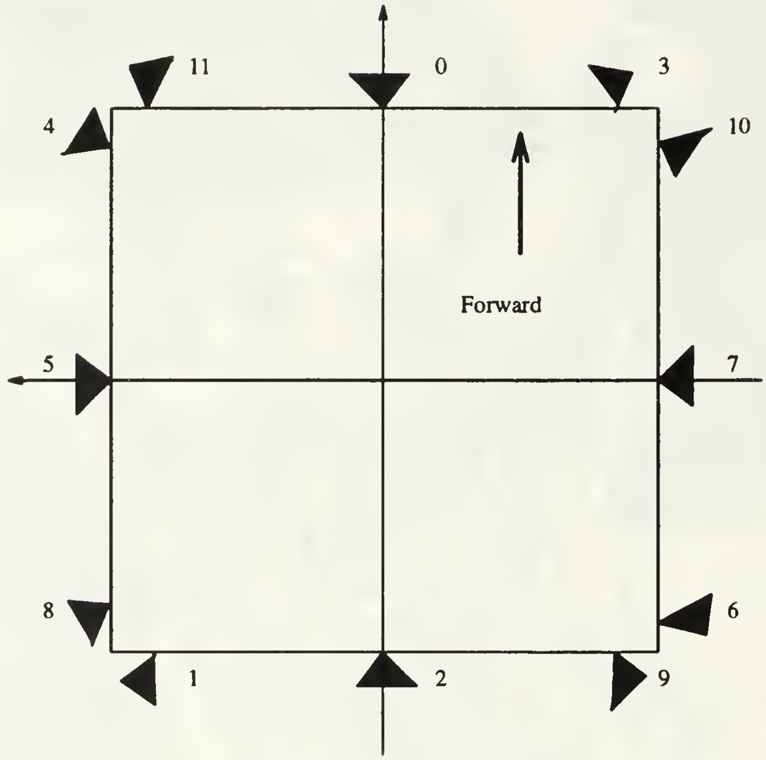


Figure 139. Yamabico-11 ultrasonic sonar configuration

Yamabico's sonar hardware is extremely efficient because a dedicated sonar board with a microprocessor controls the sonar sensors [61]. Yamabico's main central processing unit is interrupted only when data becomes available from the sonar array. The sonar system provides user interface functions that control Yamabico's array of sonar range finders. At any point within a user's program, any of the twelve sonars may be enabled or disabled. This allows the user to operate a given sonar only when necessary for a particular application.

Yamabico employs twelve Nippon Ceramic T40-16/R40-16 ultrasonic sonars, operating at 40 KHz and distributed around the periphery of the robot at 30 degree increments as shown in (Figure 139), approximately 35 *cm* above the floor [52].

Each sensor is actually a pair of transducers, one to transmit the ultrasonic pulse and another to receive the echo. The self-contained sonar system runs on a VME motherboard and interfaces with the Yamabico-11's Central Processing Unit (CPU) via the VME bus. The sonar hardware design gives a range gate of 409 *cm* and a range resolution of 1 *mm* [55].

a. Sonar Grouping

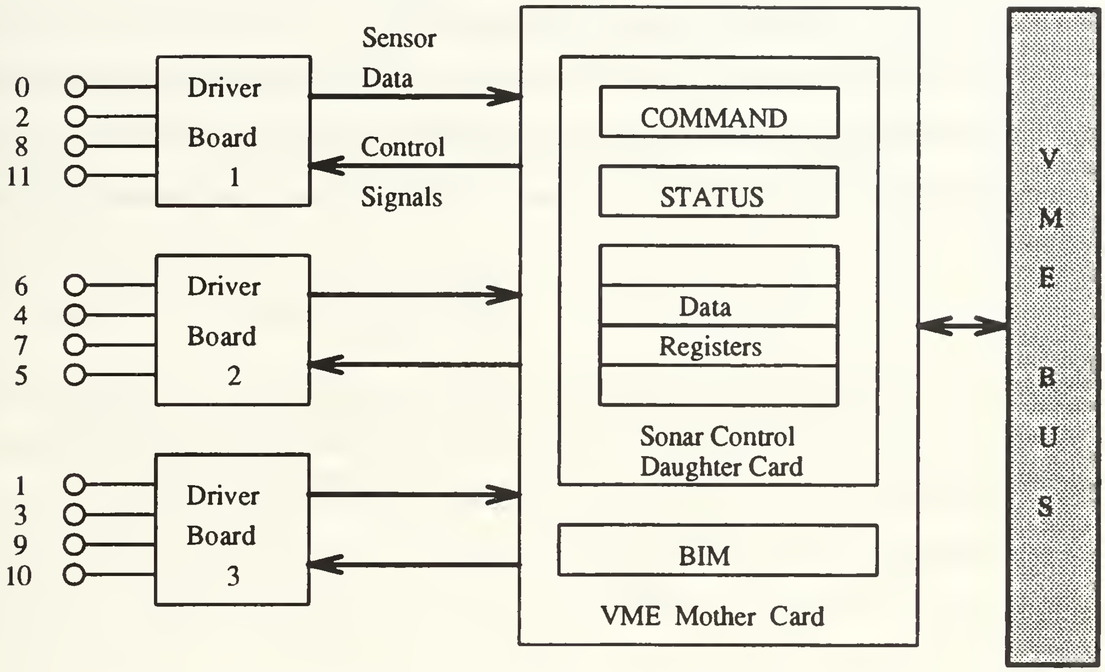


Figure 140. Yamabico-11 sonar hardware architecture

In order to reduce sampling time, the twelve ultrasonic sonars were divided into three logical groups, with four sensors in each group. The sonars of a logical group are all pulsed simultaneously and thus reduce the sampling time by a factor of four as compared to individual firing of the sonars. Group 0 consists of sonars 0, 2, 5 and 7; group 1 of sonars 1, 3, 4, and 6; group 2 of sonars 8, 9, 10 and 11; and group 3 is a virtual group which consists of four permenent test values [61]. The axis of each sonar is oriented at 30 degree angles from its neighbors. Ranging is done on a group basis to prevent mutual interference. Additionally, the sonars are physically grouped in order to distribute the electrical load over the driver boards evenly and thus minimize any electrical transients associated with operation of the sonar (Figure 140). The physical grouping connects sonars 0, 2, 8 and 11 to driver/amplifier board 1; sonars 4, 5, 6 and 7 to board 2; and sonars 1, 3, 9 and 10 to board 3. The reader will note that pairs of sonars from logical groups are assigned to physical groups, for example, sonar 0 and 2 from logical group 0 are assigned to physical group (driver/amplifier board) 1.

b. Sonar Range Calculation

The sonar transducers operate at a constant frequency of 40 KHz. Since Yamabico's programmed maximum range is 409 cm, a sonar pulse width is 1 ms and the speed of sound in air is 340 m/sec, the maximum round trip time can be calculated as follows:

$$round\ trip\ time = \frac{409\ cm}{34000\ cm/sec} \times 2$$

This round trip time is the period during which a valid echo may be received and is referred to as the receive gate. This interval is derived by division of the sonar system's 2 MHz clock to ensure that the receiver is not falsely triggered by a direct path reception from it's adjacent transmitter. We opt to disable the receiver until the transimit pulse is complete. This will have the disadvantage of setting a minimum range equal to half the distance sound would travel in the time of a transmit pulse. The minimum range can be computed as follows:

$$minimum\ range = 34000\ cm/sec \times 1\ msec \times 0.5 = 17\ cm$$

The minimum range lies approximately 9 cm outside the periphery of the robot. In order to allow the measurement of the objects up to the periphery of the robot, the pulse width was decreased to 0.5 $msec$ thus reducing the minimum range to 8.5 cm. However, additional time was needed to accommodate switching and setting within the circuitry; therefore, in actual practice, the minimum range is set by firmware to 9.6 cm [61].

c. Sonar Interrupt Control

The sonar control board is actually a daughtercard which rides on a VME bus mothercard. The mothercard carries address decoders, bus drivers and interrupt control circuitry in the **B**us **I**nterface **M**odule (**BIM**).

When the sonar has completed a ranging cycle an interrupt request is provided to the BIM. The BIM's control register holds information which determines whether an interrupt is to be generated or not, and if so which interrupt level is to be generated. Presuming an interrupt is generated, when the correct acknowledgment returns on the address lines the BIM's vector register provides the vector table entry where the central processor may find the vector to the interrupt handler. The correct interrupt level, the interrupt enable bit and interrupt vector are loaded to the BIM during software initialization.

B. MML-11 SOFTWARE ARCHITECTURE

The **M**odel based **M**obile-robot **L**anguage **MML** is the driving force behind the robot [38, 41]. MML is a portable library of functions written in the **ANSI C** language in the **UNIX** environment. The library supports locomotion functions, sensor functions and other I/O functions. Currently, its eleventh version, called **MML-11**, is under development.

All software routines on the robot were developed and downloaded to the robot via RS232 at a baud rate of 19200. The system consists of a **kernel** (including the MML functions) and a **user program**. Once a user program is downloaded and is triggered to execute, all operations are autonomous.

From the robot control point of view, MML-11 is a programmable software system for mobile robot operation. The main procedure of the system conducts all necessary initializations for both hardware and software. After the initializations are done, a user program is called. Besides the main procedure, MML-11 mainly consists of the motion control subsystem and the sonar control subsystem.

For user application programming convenience, the system provides a set of well-defined functions called **user functions** as the interface between the user and the the system. The user functions are categorized into four modules:

- Operating System Module,
- Motion Planning Module,
- Motion Control Module, and
- Sonar Control Module.

1. System Architecture

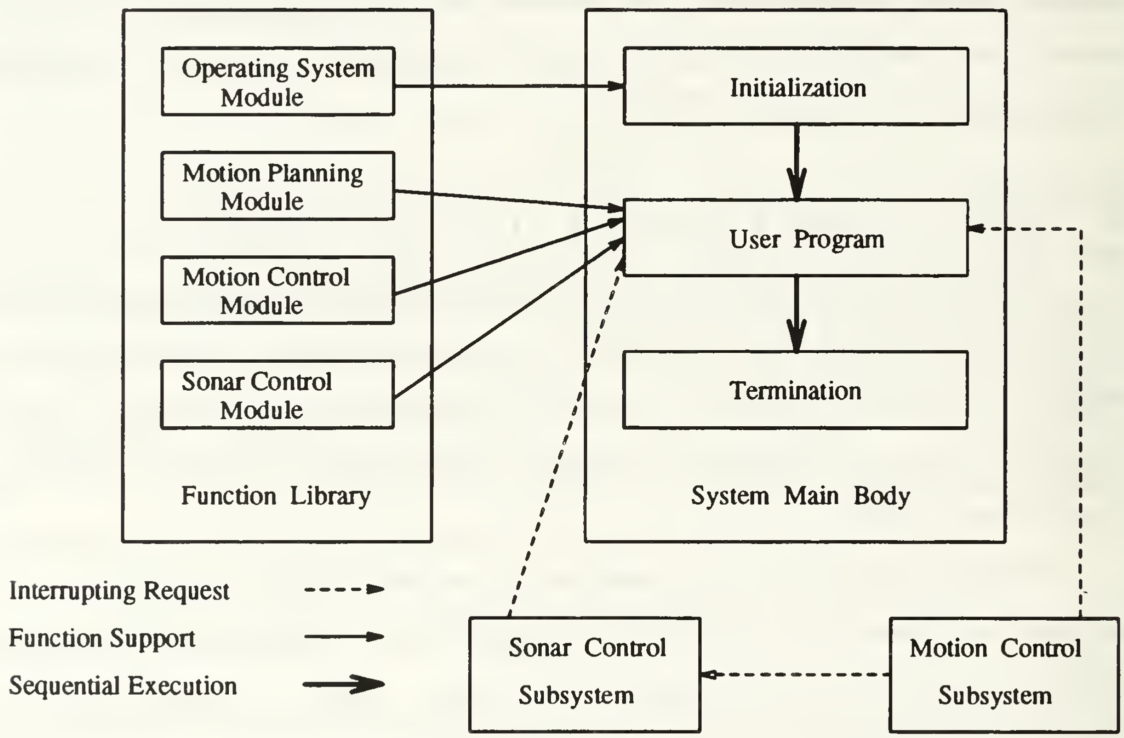


Figure 141. MML-11 software conceptual architecture

This software is developed with a special architecture which incorporates a sequential structure and an interrupt-driven structure. The system initialization and the user application program are executed sequentially in the main procedure of the system. The motion control and sonar control subsystems are periodically called for execution via interrupt requests for the required motion control and/or sonar control operation. The MML-11 software architecture is shown in (Figure 141).

2. Interrupt-driven Subsystems

There are three primary tasks that may be running at any given time. The motion control subsystem is the highest priority task, performing all motion control computations and translating them into low-level wheel controls. It is designed to interrupt other tasks every 10 *msec*. The next highest priority task is the sonar control subsystem, which processes all incoming sonar returns and generates line segments from individual sonar returns from obstacles if required. It issues an interrupt request every 50 *msec*. The lowest level priority, but still a basic, task is the user program. This part of the system feeds both immediate and sequential commands to the motion control subsystem through a command queue. All higher priority tasks interrupt the tasks with lower priorities to gain the CPU control. The design of MML-11 subsystems will be described in the following sections.

3. RealTime Operating System

The Yamabico-11 onboard CPU, IV-SPARC 33, provides no standard operating system functions but a small set of libraries for console I/O. All other operating system primitives, such as interrupt handling, memory management, data formatting and logging must be provided by the MML system.

4. User Program

In this software, the robot's motion is instructed by the user program, which sends commands to the motion control system and/or sonar control system. However, motion planning - and control - specific concepts are hidden from the user. Only those defined as user functions are allowed to be considered by the user program. Sonar data is available to the user in either a raw or processed format via user sonar functions. In "APPENDIX. USER PROGRAM EXAMPLES", we give a sample user program. The MML-11 user function specifications will be described in Section C.

5. MOTION CONTROL ARCHITECTURE

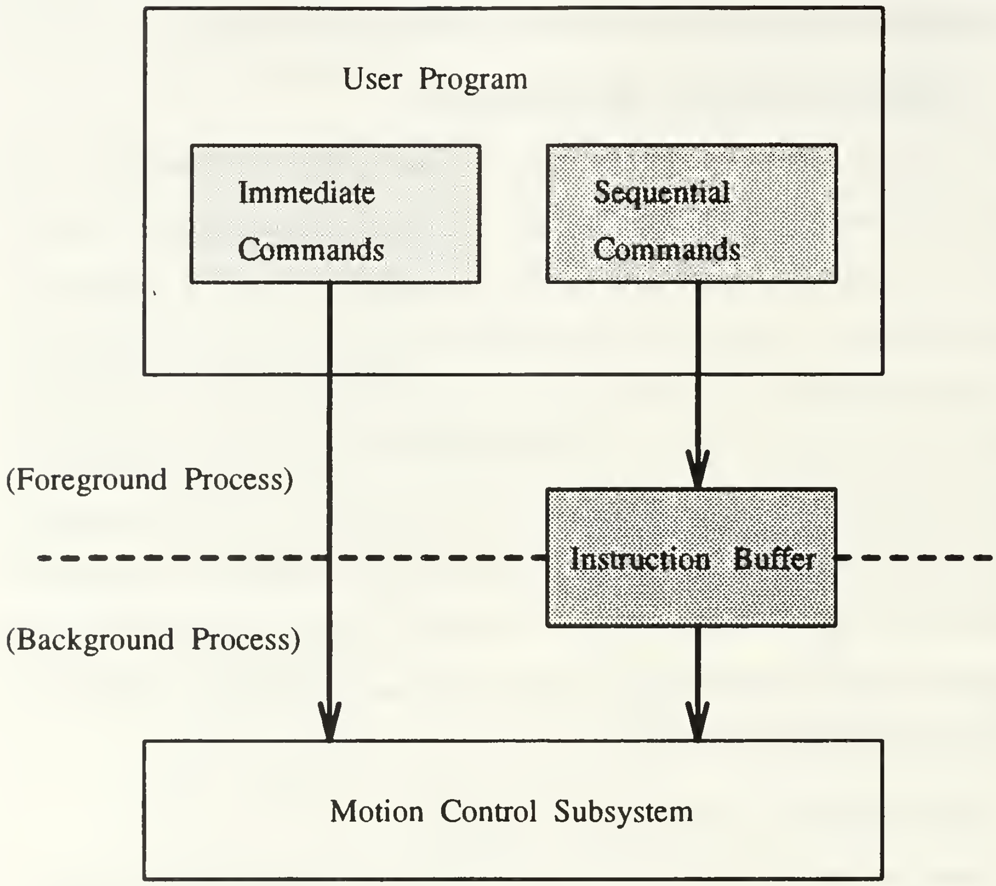


Figure 142. MML-11 motion control software architecture

The motion control must be repeatedly performed in a short period. It is difficult to impose this control in user's program. As we design an interrupt-driven software system, the foreground job and background job concepts are introduced into MML-11 motion control software. In MML-11, the motion control mechanism is designed in such a way that the execution of user program is somewhat separated from motion control. This allows the user being able to program applications by using simple functions. The user program is considered the *foreground process* which sends either immediate or sequential commands to the system. The robot motion control task conducted by motion control subsystem is considered the *background process* which performs motion control to acheive the motion instruction it gains control at a frequency of 10 *msec*. The immediate commands in the user program will be executed immediately, while the sequential commands will be enqueued to a buffer called the *instruction buffer* waiting for execution sequentially. The motion control subsystem fetches an instruction sequentially. When the execution of one instruction is finished, the control subsystem picks and executes another instruction from the buffer until the buffer is empty. The motion control architecture is illustrated in Figure 142.

6. Motion Control Subsystem

Motion control subsystem, named **MotionSysControl**, is the foreground process of the entire system. It is designed to compute all data necessary for motion control by interrupting system main procedure (or user program) every 10 *msec*. When the interrupt request is granted, this subsystem gains the control of CPU. It actually acts as an interrupt service routine.

MotionSysControl performs following computations for the robot motion control in order to accomplish its mission.

- Measure the distance traveled, Δs, in a cycle by the reading robot's left and right shaft encoders.
- Compute the orientation changes, $\Delta\theta$.
- Localize current configuration, q.
- Compute commanded linear and rotational velocity, V_L, V_ω, for next cycle.
- Translate commanded velocity into control signals, PWM, for driving motors.
- Transition point simulation to decide whether to read next instruction.

By reading the robot's left and right shaft encoders, the process can measure the distance traveled. Computations of distance traveled and orientation changes are done in order by a module with outputs Δs and $\Delta\theta$. These data will be used by localization module to compute robot's current configuration. The current configuration q is needed for motion rule module to compute commanded linear and rotational wheel velocities, V_L, V_ω, for next cycle. These velocities are translated in left and right PWMs as signals to drive corresponding motors. The last step in MotionSysControl is to determine whether to start transitioning to the next path. If it decides to transition, the next motion commanded in the instruction buffer will be read and followed.

C. MML-11 LANGUAGE SPECIFICATION

In this section, we describe the design of user functions which will be used as interface between user and MML-11 software. The specifications of functions for motion control, sonar control and geometric calculation are presented. Some of the basic data structures which will be used to describe the functions are presented also. The user functions are categorized into following subsets:

- Geometric functions,
- Motion planning functions,
- Motion control functions, including sequential functions and immediate functions,
- Sonar control functions, and
- Self localization functions.

The geometric functions simply "define" some utility functions for algebraic manipulation of geometric variables. The motion planning functions provide the user with simple interface functions to build a world model and to conduct motion planning when given a specific mission. The motion control functions include sequential functions and immediate functions. The sequential functions define a set of motion control commands that are stored in a buffer when they are used in the user program and are executed sequentially as the robot's background tasks. The immediate functions define the commands which take effect immediately when they are executed in a user's program. The sonar control functions are the functions used to control sonar operation and to obtain sonar data.

1. Data Structures

- **Point**

The **POINT** structure is used to describe a position in a two-dimensional cartesion coordinate system. The structure includes a double X and a double Y.

- **Configuration**

The **CONFIGURATION** is the standard structure for describing location and direction for an object. It consists of **Posit**, with type of POINT, which identifies an objects position in two-dimensional cartesion coordinates. Another element is **Theta** of type double that describe's the object's orientation in relation to the X coordinate. Finally, there is another double called **Kappa** that represents the curvature of an object's path.

- **Path Element**

The **PATH_ELEMENT** data structure is used to describe and store the various types of movements. This data structure consists of **config** which is of type CONFIGURATION. It holds the configuration of the path that the robot is to follow. PATH_ELEMENT also contains **pathType**, which is of type **PATH_TYPE**. A PATH_TYPE is a data structure used to identify the various paths that are available to the robot. It consists of the mode which is of type **MODE** and class which is of type **CLASS**. Type MODE is an enumeration type that gives a name to each path that the robot follows. Presently, the modes that are available include NOMODE, ENDMODE, STOPMODE, PATHMODE, ROTATEMODE, KSPIRALMODE, PCMODE and FOLLOWMODE. Type CLASS, which is also an enumeration type, is used to name and categorize the various path mode types. The list of classes include NOCLASS, LINECLASS, CIRCLECLASS, BLINECLASS, NBLINECLASS, CCWLEFT, CCWRIGHT, CWLEFT and CWRIGHT.

- **Velocity**

The **VELOCITY** structure is used to describe a velocity. The data structure is made up of two doubles that represent the linear and rotational elements of velocity. They are appropriately named Linear and Rotational, respectively, in the VELOCITY structure.

Sonar Number	SonarPosit.X	SonarPosit.Y	SonarTheta
0	0.0	−0.5	0
1	−23.0	13.1	$5\pi/6$
2	−22.6	−1.0	π
3	24.7	−14.6	$-\pi/6$
4	13.4	21.3	$\pi/3$
5	0.0	20.6	$\pi/2$
6	−12.6	−21.3	$-2\pi/3$
7	0.0	−20.5	$-\pi/2$
8	−13.4	21.3	$2\pi/3$
9	−23.5	−14.9	$-5\pi/6$
10	12.1	−21.3	$-\pi/3$
11	25.2	14.1	$\pi/6$
12	0.0	0.0	0
13	1.5708	21.5	1.5708
14	4.7124	21.5	4.7124
15	0.0	0.0	0

Table XVI. Sonar position

- **Sonar Table**

The sonar table **SONARD** contains not only the new range (**d**) of type double and the old range (**d0**) of type double but the robot's position at the time of the range (**posit.X**, **posit.Y** of type POINT and **t**) of type double and the global coordinates corresponding to that range and position (**global.X** and **global.Y**) of type POINT. The sonar table also contains the position of the individual sonar relative to the robot's coordinate system (**SonarPosit** of type POINT, the euclidean distance from robot center to sonar center and **SonarTheta** of type double, the angle from the robot's x-axis to the sonar center) of type double. Table XVI shows where each sonar on the vehicle. The sonar table also contains two flags which guide the operation of the sonar system. These are **fitting**, with type of integer, which indicates linear fitting requests and **update**, with type of integer, which inform the sonar system of the presence of new data in **d**. An array of sixteen of these structures is formed, and is then indexed by sonar number.

- **Segment Descriptors**

The segment structure (**SEGMENT_RES**) contains all the data necessary to completely describe a line segment. This includes an integer to represent the sonar which recorded the segment, the number of data points thus far included in the line segment (**m00**) and real numbers to record the endpoints (**start.X** and **start.Y**, **end.X** and **end.Y**), the angle and length of a normal to the segment from the origin (**alpha** and **r**), the length of the line segment. This structure is arranged in a two dimensional array. One index is the number of the sonar from which the segment is derived; the other index holds an integer (0 through 29). This segment list can hold the 30 most recent segments described by a given sonar. It is presumed that any navigation program will not require more history than these thirty segments; if so, the second index of segment list can be increased.

- **Sonar Data Logs**

The sonar data logs are arrays to which the user program writes data during it's execution. These logs are converted to ASCII strings at the completion of the user program and those strings are in turn transferred to the host when all data are ready to down load. There are three types of data logs: the raw data, the global data and the segment data. For each log type, there is corresponding data file. The filenames created on the host will depend upon the type of logging performed and the sonar number. The tracing frequency is used to specify how many sonar cycles are skipped before data is logged. A value of 1 or less causes the logging to occur with each cycle. The raw data records the range and the robot's position and orientation at the time of the range. The global data records the range and global x and y values for sonar returns. The segment data records line segments in the form of segment descriptors previously described.

2. User Function Specification

1. Geometric Functions

- **Define Configuration**

Synopsis:	CONFIGURATION **defineConfig**($x, y, theta, kappa$)	
Parameters:	double	x;
	double	y;
	double	$theta$;
	double	$kappa$;

Description:

When passed the values that define a configuration ($x, y, theta, kappa$), this function allocates and assigns a configuration. It returns a configuration. The configuration can be used to represent a path which is either a line or a circle. If the configuration is defined with curvature zero, i.e. $\kappa = 0.0$, it specifies a straight line passing through the point (x, y) with orientation θ. If its curvature is greater than zero, i.e. $kappa > 0.0$, the path is a counterclockwise circle. If $kappa < 0.0$, then the path is a clockwise circle. Figure 143 illustrates theses concepts.

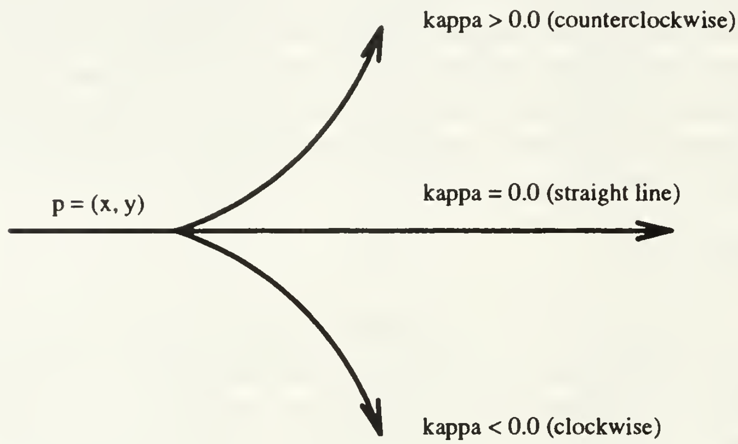


Figure 143. A configuration represents a line or a circle

- **Inverse**

  Synopsis: CONFIGURATION **inverse**(q)
  Parameters: CONFIGURATION q;
  Description:

  The purpose of this function is to calculate the inverse of a given configuration such that: $q * q^{-1} = e$.

- **Compose**

  Synopsis: CONFIGURATION **compose**(q_1, q_2)
  Parameters: CONFIGURATION q_1;
  CONFIGURATION q_2;
  Description:

  The purpose of this function is to calculate the composition of two configurations. Specifically, the function takes parameter q_1 and composes it with parameter q_2 to calculate and return the composed value.

- **Circular Arc**

  Synopsis: CONFIGURATION **CircleArc**($l, alpha$)
  Parameters: double l;
  double $alpha$;
  Description:

Given a tangential orientation $alpha$ and the arc length l in a curve, this function computes its configuration in the local coordinate system. In the case of motion control, length would actually be Δs and $alpha$ would be $\Delta\theta$. The function can be called to determine the configuration after an incremental move in the local coordinate system of the original configuration.

- **Euclidean Distance**

Synopsis:	double **euDis**($p1, p2$)	
Parameters:	POINT	$p1$;
	POINT	$p2$;
Description:		

This function computes the Euclidean distance between two given points.

- **Normalize**

Synopsis:	double **norm**($theta$)	
Parameters:	double	$theta$;
Description:		

This function returns a normalized angle in the range $[-\pi, \pi]$.

2. **Motion Planning Functions**

- **Create World Model**

Synopsis:	void **createPolyModel**()
Description:	

This function builds a world of polygons. It will generate the set of data which is needed in planning robot's motion.

- **Image**

Synopsis:	Image **convexImage**($p, B, direction$)	
Parameters:	POINT	p;
	int	B;
	int	$direction$;
Description:		

This function finds the image of a given point p in free space on a polygon B. The parameter *direction* indicates the direction *ccw* or *cw*. The output of this function is structure containing the identity of the feature type (edge or vertex) which contains the image point, pointer to vertex v_i, the orientation from a point p to an image, and the closest distance from a point p to its image(see Table XIII in Chapter VIII).

- **Polygon Tracking**

  Synopsis: void **polygonTracking**()
  Description:

  The purpose of this function is to indicate the direction of tracking a polygon (*ccw* or *cw*). This function sets the value of the current path element in motion control to the path element passed in as a parameter.

- **Polygon Planning**

  Synopsis: VELOCITY **FollowRule**(*actual*, *commanded*)
  Parameters: VELOCITY *actual*;
  VELOCITY *commanded*;
  Description:

  This function returns the robot's linear and rotational velocities to follow a polygon in *ccw* or *cw* direction.

- **Motion Tracking**

  Synopsis: void **motionTracking**()
  Description:

  The purpose of this function is to set the value of the current path element in motion control to the path element passed in as a parameter.

- **Local Motion Planning**

  Synopsis: VELOCITY **LocalMPRule**(*actual*, *commanded*)
  Parameters: VELOCITY *actual*;
  VELOCITY *commanded*;
  Description:

  This function generates the motion instructions along the path. Those instructions will be taken to drive the robot until it stops.

3. **Motion Control Sequential Functions**
The sequential functions define a set of motion control commands which are stored in a buffer that acts as an interface between user and robot. When the user program is being executed, commands of this type included in the user program do not take effect immediately instead they are loaded in buffer as motion instructions. The motion control system reads the instructions from the top of the buffer sequentially and controls the robot's motion accordingly. The specifications of those functions are listed below.

- **Tracking a line**

Synopsis: void **line**(q)
Parameters: CONFIGURATION q;
Description:

The function defines a command that orders the robot to follow the line or circle specified by the configuration q. If the robot's last configuration before the command is executed is not on the track of the line specified, the robot uses the steering function to transfer to the line with a smooth motion. Figure 144 illustrates robot's behavior when executing **line**(q) with a straight line q.

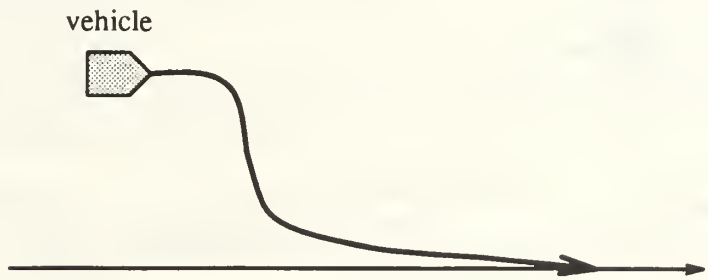


Figure 144. The line tracking function

- **Tracking the Line form its Back and Stopping**

Synopsis: void **bline**(q)
Parameters: CONFIGURATION q;
Description:

This function defines a command that orders the robot to track the line specified by the configuration q from its back. If the robot's image is on the back half of the line, the robot tracks the line as function **line()**x and stops when its image reaches the configuration. If the robot's image falls on the forward part of the line initially, the robot would not move (see Figure 145).

- **Tracking the Line form its Back and no Stopping**

Synopsis: void **nbline**(q)
Parameters: CONFIGURATION q;
Description:

This function is similar to the backward line function, **bline()**, except the vehicle does not stop at the configuration q. The vehicle may transition to another path element after reaching the configuration q if another path element command follows. To stop the vehicle, the **stop()** function must follow it (see Figure 146).

Figure 145. The backward line tracking with stopping function

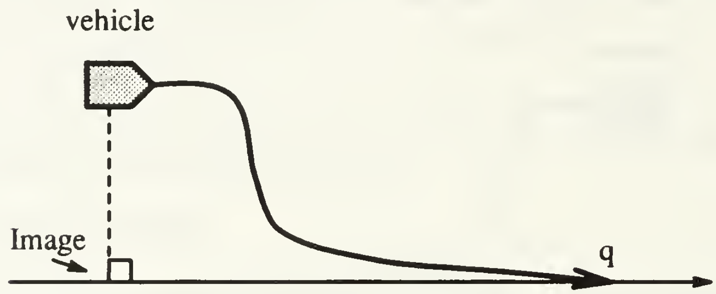


Figure 146. The backward line tracking with no stopping function

- **Set Robot's Configuration**

  Synopsis: void **setRobotConfig**(q)
  Parameters: CONFIGURATION q;
  Description:

  This function sets robot's configuration to a given configuration q.

4. **Motion Control Immediate Functions**

- **Set Path Element**

  Synopsis: void **setPathElement**($path$)
  Parameters: PATH_ELEMENT $path$;
  Description:

  This function sets the value of the current path element in motion control to the path element passed in as a parameter.

- **Set Robot's Configuration Immediately**

  Synopsis: void **setRobotConfigImm**(q)
  Parameters: CONFIGURATION q;
  Description:

This function sets robot's configuration to a given configuration q immediately.

- **Get Path Element**

  Synopsis: PATH_ELEMENT **getPathElement**()
  Description:

  This function retrieves the current path element in motion control module.

- **Set Robot's Linear Speed Immediately**

  Synopsis: void **setLinVelImm**(*speed*)
  Parameters: double *speed*;
  Description:

  This function sets the robot's linear velocity immediately.

- **Set Sigma Immediately**

  Synopsis: void **setSigmaImm**(*sigma*)
  Parameters: double *sigma*;
  Description:

  This function sets the robot's sigma which control the sharpness of its trajectory when the robot is turning.

- **Set Total Distance Traveled Immediately**

  Synopsis: void **setTotalDistanceImm**(*distance*)
  Parameters: double *distance*;
  Description:

  This function sets the total distance travelled by the robot to the value passed as a parameter.

- **Get Total Distance Traveled Immediately**

  Synopsis: void **getTotalDistanceImm**()
  Description:

  This function returns the total distance travelled by the robot.

- **Stop Immediately**

  Synopsis: void **stopImm**()
  Description:

  This function stops the robot immediately with the current acceleration rate until the speed reaches 0.

- **Logging Motion Data**

Synopsis:	void **Motionlog**($Filename, Frequency, BufferSize$)	
Parameters:	char	$Filename$;
	int	$Frequency$;
	int	$BufferSize$;
Description:		

  This function prepares the tracing system to log motion data. Tracing is automatically turned on after this function is called. The $Filename$ specifies a file name that will be used to store data when the data is uploaded to the host. $Frequency$ specifies how many motion cycles are skipped before data is logged.

5. **Sonar Control Functions**

- **Enable Sonar**

Synopsis:	void **EnableSonar**($SonarNumber$)	
Parameters:	int	$SonarNumber$;
Description:		

  This function enables sonar with $SonarNumber$. More precisely, it enables the sonar group that contains $SonarNumber$, which causes all the sonars in that group to echo-range and write data to the data registers on the sonar control board.

- **Disable Sonar**

Synopsis:	void **DisableSonar**($SonarNumber$)	
Parameters:	int	$SonarNumber$;
Description:		

  This function removes $SonarNumber$ from the enabled_sonars list. If $SonarNumber$ is the only enabled sonar from it's group, then the group is disabled as well and will stop echo ranging. This has benefit of shortening the ping interval for other groups that remain enabled.

- **Get Sonar Returns**

Synopsis:	double **Sonar**($SonarNumber$)	
Parameters:	int	$SonarNumber$;
Description:		

  This function returns the distance (cm) sensed by $SonarNumber$ ultrasonic sensor. If no echo is received, an INFINITY (999999.0) is returned. If the distance is less than 10 cm, then a 0 is returned.

- **Calculate Global**

Synopsis: void **CalculateGlobal**(*SonarNumber*)
Parameters: int *SonarNumber*;
Description:

This function calculates the global x and y coordinates for the range value and robot configuration in the sonar table. The results are stored in the sonar table.

- **Enable Linear Fitting**

Synopsis: void **EnableLinearFitting**(*SonarNumber*)
Parameters: int *SonarNumber*;
Description:

This function causes the background system to gather data points from *SonarNumber* and form them into line segments.

- **Disable Linear Fitting**

Synopsis: void **DisableLinearFitting**(*SonarNumber*)
Parameters: int *SonarNumber*;
Description:

This function causes sonar system to cease forming line segments.

- **Logging Sonar Data**

Synopsis: void **SonarLog**(*Freq*, *BSize*, *SonarNumber*, *LogType*)
Parameters: int *Freq*;
int *BSize*;
int *SonarNumber*;
int *LogType*;
Description:

This function prepares the tracing system to log sonar data. The tracing *Freq* specifies how many sonar cycles are skipped before data is logged. A value of 1 or less causes the logging to occur each cycle. The *BSize* specifies how many bytes of storage to allocate to save the data. If a value of 0 is specified, a default size is used. The *SonarNumber* specifies the sonar you wish to log. The *LogType* specifies the type of logging performed. There are three types.

  - **SONAR_RAW** logs only new sonar data.
  - **SONAR_GLOBAL** logs global sonar data.
  - **SONAR_SEGMENT** logs segment data.
  - **SONAR_ALL** logs all three types of data.

Tracing is automatically turned on after this call. The filenames created on the host will be depend on the type of logging performed and the sonar number. For example, if logging were initiated using:

SonarLog(0, 0, 3, SONAR_SEGMENT)

then the filenemes **SEGMENT.3** will be created on the host.

6. **Self Localization Functions**

- **Wait Segment**

  Synopsis: void **WaitSegment**(*SonarNumber*)
  Parameters: int *SonarNumber*;
  Description:

  This function is busy waiting until the line segment being built is completed.

- **Get Segment Configuration**

  Synopsis: CONFIGURATION **GetSegmentConfig**()
  Description:

  This function returns the observed configuration of the object after applied the linear fitting algorithm.

- **Match**

  Synopsis: int **Match**(*qsegment*, *qmodel*)
  Parameters: CONFIGURATION *qsegment*;
  CONFIGURATION *qmodel*;
  Description:

  This function compares between observed segment *qsegment* and model wall segment *qmodel*.

- **Odometry Correction**

  Synopsis: void **CorrectOdometryError**(*qsegment*, *qmodel*)
  Parameters: CONFIGURATION *qsegment*;
  CONFIGURATION *qmodel*;
  Description:

  This function corrects the vehicle's odometry error if there is a difference between where the vehicle thinks it is and where the vehicle really is.

X. CONCLUSIONS

This dissertation addressed new motion planning and real time localization methods using proximity under the structure of a layered planning approach. This approach divides the planning task into global path planning and local motion planning. Three major contributions to the field of robotics were made from the reseach conducted in this dissertation. The first is the development of the theory of homotopic decompositions which solves the problem of homotopic class representation using a Voronoi diagram. A homotopic decomposition captures the topology of the world in terms of homotopy classes. A global path planner was able to deliver a plan representing a distinct homotopy class making it available for the local motion planning, which is responsible for executing the global path plan. Second, the safe local motion planning algorithm is the first steering function algorithm to provide a theoritical and a practical solution to safe motion planning problem, a great step in promoting motion planning in the real world. The effectiveness of the method of using the left and right polygons was confirmed. The problem making a smooth motion when the vehicle gets close to an intersection of two distinct boundaries was solved. A striking advantage of this method is that this is effective in more dynamic environments. This method may be useful even in unknown worlds as well, because the images on the polygons can be taken by sensors instead of through information extraction from the model. Third, a transparent method of robust real-time positional-uncertainty elimination (self localization) was described. The problem of gradual error accumulation when the robot moves long distances was solved. This method is a simple application of group theory that requires very little computational overhead.

Another contribution was The description of a geometrical algorithm for finding images in real-time for safe motion planning.

The algorithms targeted for Yamabico-11 were first developed on a simulator then successfully transported to the real robot.

XI. FUTURE RESEARCH

This chapter presents a few topics for future research in the several areas related to the topics covered herein.

Configuration-to-configuration motion planning is a most difficult planning problem. It must be addressed in final parking maneuvers. There is clearly a need to solve the final motion planning problem [47, 9].

The path planner uses the geometrical constraints of the environment and kinematic and dynamic constraints of the robot to provide the global reference path plan. This layer optimizes the cost function of the mission using the known part of the environment. In a partially-known static environment, this optimal path will be achieved only if there is no interaction of the robot with the unknown portion of the environment, a highly unlikely event. Nonetheless, the global path will serve to guide the actions of the local planner when faced with unforeseen obstacles. However, a well defined theory exactly describing how to avoid the previously unknown but recently detected obstacle still requires much work [40, 6, 1, 5, 7, 8, 62].

It is impossible to absolutely guarantee collision avoidance in a dynamic environment. Moreover, it is almost pointless to specify optimal trajectories in a dynamic environment, since the data become obsolete with time. As the information becomes older, it becomes less reliable. Systems which build detailed reconstructions of the environment from sensor data suffer from delays due to information processing times. Therefore, the representation of the known and recently discovered environment features must be made efficiently available to modules that have short reaction time requirements. The representation is vital in integrating higher-level plan objectives with local behavior decision processes and in minimizing the loss of information when unforeseen obstacles arise. There seems to be no single algorithm to handle all possible cases in a dynamic environment. Consequently, the use of multiple algorithms, multiple sensors, and multiple responses seems to provide the most likely chance of successfully achieving a goal. Future research is needed to determine what information is relevant to achieve a goal and what details of the information are necessary to utilize sensors and actuators effectively? In a dynamic environment, path plans should serve as an aid to the selection of appropriate motion, rather than constraints upon that selection in many of the cases [21].

The large repertoire of behaviors and strategies used by the local motion planner may require a variety of sensing capabilities. A vision processing system would also aid in obstacle avoidance maneuvers at a distance beyond the current range of the ultrasonic sensors.

APPENDIX A. NORMALIZING ANGLES

Generally, testing whether an angle between two directions is positive or negative gives us an idea on the relation between the two directions. However, in some situations, a simple subtraction operation does not work. For example, if $\theta_1 = \frac{3\pi}{4}$ and $\theta_2 = \frac{-3\pi}{4}$, the angle α between them becomes

$$\alpha = \theta_2 - \theta_1 = -\frac{3\pi}{4} - \frac{3\pi}{4} = -\frac{3\pi}{2}$$

However, this angle is naturally considered as a $\frac{\pi}{2}$ left turn rather than a $\frac{3\pi}{2}$ right turn. To handle this situation, we use the *normalization function* Φ: $\mathcal{R} \rightarrow [-\pi, \pi]$. For instance,

$$\Phi\left(\frac{\pi}{2}\right) = \Phi\left(\frac{5\pi}{2}\right) = \Phi\left(\frac{-3\pi}{2}\right) = \frac{\pi}{2}$$

and

$$\Phi(\pi) = \Phi(-\pi) = \pi$$

Definition: The normalization function Φ is formally defined by the following conditions:

1. For any angle $\alpha \in \mathcal{R}$,
$$-\pi \leq \Phi(\alpha) \leq \pi$$
2. For any angle $\alpha \in \mathcal{R}$,
$$\alpha = \Phi(\alpha) \ \ mod \ \ 2\pi$$

The normalization function Φ: $\mathcal{R} \rightarrow [-\pi, \pi]$ can be defined using a recursive definition:

$$\Phi(\alpha) = \begin{cases} \Phi(\alpha - 2\pi) & \text{if } \alpha > \pi \\ \Phi(\alpha + 2\pi) & \text{if } \alpha < -\pi \\ \alpha & \text{otherwise} \end{cases}$$

APPENDIX B. LEAST SQUARES LINEAR FITTING

Let

$$R = \{p_1, \cdots, p_n\} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

be a set of n points, We obtain the moments m_{jk} of R with $0 \leq j, k \leq 2; j + k \leq 2$.

$$m_{jk} = \sum_{i=1}^{n} x^j y^k$$

Notice that $m_{00} = n$. The centroid C is given by

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \equiv (\mu_x, \mu_y)$$

The secondary moments around the centroid are given by

$$M_{20} \equiv \sum_{i=1}^{n}(x_i - \mu_x)^2 = m_{20} - \frac{m_{10}^2}{m_{00}}$$

$$M_{11} \equiv \sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y) = m_{11} - \frac{m_{10}m_{01}}{m_{00}}$$

$$M_{02} \equiv \sum_{i=1}^{n}(y_i - \mu_y)^2 = m_{02} - \frac{m_{01}^2}{m_{00}}$$

We adopt the parametric representation of a line with constants r and α. If a point $p = (x, y)$ satisfies an equation

$$x \cos\alpha + y \sin\alpha = r, \tag{B.1}$$

then the point p is on a line L whose normal has an orientation α and whose distance from the origin is r (Figure 147). This representation has a striking advantage as opposed to the usual method of using a formula $y = f(x)$, because the former method has no difficulty in expressing lines that are perpendicular to the X axis. Note that two axes X and Y are symmetric in the plane. The signed distance (or residual) δ_i from point $p_i = (x_i,\ y_i)$ to the line $L = (r, \alpha)$ is

$$\delta_i = x_i \cos\alpha + y_i \sin\alpha - r. \tag{B.2}$$

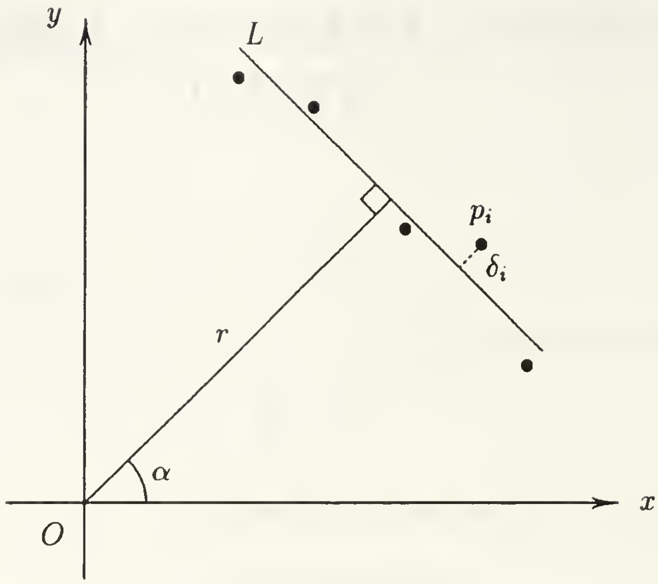


Figure 147. Fitted line

Therefore, the sum of the squares of all the residuals is

$$S = \sum_{i=1}^{n}\Big((x_i \cos\alpha + y_i \sin\alpha) - r\Big)^2$$

Since the line which best fits the set of points is supposed to minimize S, the optimum line (r, α) must satisfy

$$\frac{\partial S}{\partial r} = \frac{\partial S}{\partial \alpha} = 0$$

Thus,

$$\begin{aligned}
\frac{\partial S}{\partial r} &= -2\sum_{i=1}^{n}\Big((x_i \cos\alpha + y_i \sin\alpha) - r\Big) \\
&= 2\left(r\left(\sum_{i=1}^{n} 1\right) - \left(\sum_{i=1}^{n} x_i\right)\cos\alpha - \left(\sum_{i=1}^{n} y_i\right)\sin\alpha\right) \\
&= 2(r\, m_{00} - m_{10}\cos\alpha - m_{01}\sin\alpha) = 0
\end{aligned}$$

and

$$r = \frac{m_{10}}{m_{00}}\cos\alpha + \frac{m_{01}}{m_{00}}\sin\alpha = \mu_x \cos\alpha + \mu_y \sin\alpha \tag{B.3}$$

where r may be negative. Substituting r in Eq. B.1 by Eq. B.3, we obtain

$$\frac{\partial S}{\partial \alpha} = 2\sum_{i=1}^{n}\Big((x_i - \mu_x)\cos\alpha + (y_i - \mu_y)\sin\alpha\Big)\Big(-(x_i - \mu_x)\sin\alpha + (y_i - \mu_y)\cos\alpha\Big)$$

$$\begin{aligned}
&= 2\sum_{i=1}^{n}\left((y_i-\mu_y)^2-(x_i-\mu_x)^2\right)\sin\alpha\cos\alpha \\
&\quad +2\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)(\cos^2\alpha-\sin^2\alpha) \\
&= (M_{02}-M_{20})\sin 2\alpha+2M_{11}\cos 2\alpha=0
\end{aligned}$$

Therefore

$$2\alpha = \mathrm{atan2}(-2M_{11},\ M_{02}-M_{20}) \tag{B.4}$$

Note that, by Eq. B.4, $2\alpha \in [-\pi, \pi]$, and then $\alpha \in [-\pi/2, \pi/2]$. Eqs. B.3 and B.4 are the solutions to the least squares problem.

Now, we do some pre-filterring of the data in order to remove points from the data stream which are clearly not colinear with the existing points of set R. When a new input $p = (x,\ y)$ is given to this algorithm, we can compute how far it is located from the previously obtained line L (Eq. B.2). The distance is

$$\delta = x\cos\alpha + y\sin\alpha - r.$$

If $|\delta|$ is greater than a given threshold value, we finish the line-fitting task to complete the line segment and to start a new segment with this last point.

Since the residual δ_i of a point $p_i = (x_i, y_i)$ is

$$\delta_i = x_i\cos\alpha + y_i\sin\alpha - r,$$

the projection, p_i' of the point p_i onto the major axis is

$$p_i' = (x_i - \delta_i\cos\alpha,\ y_i - \delta_i\sin\alpha).$$

We will use p_1' and p_n' as estimates of the endpoints of the line segment L obtained from the set of data points R (Figure 148).

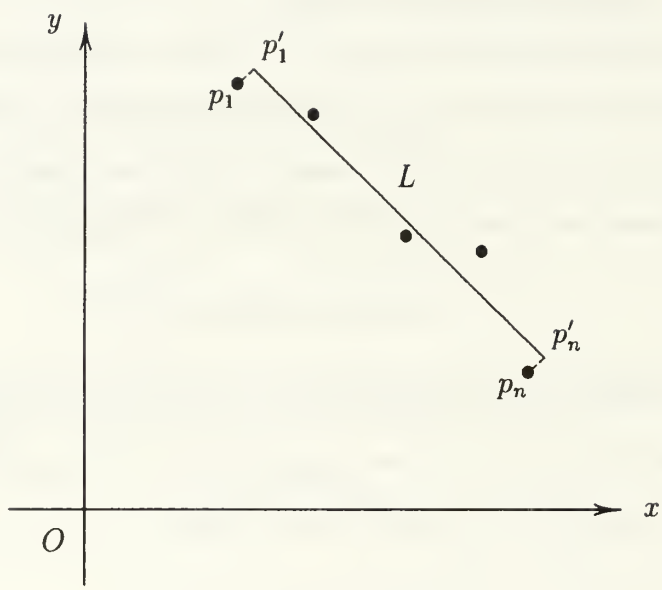


Figure 148. End points

APPENDIX C. USER PROGRAM EXAMPLES

```
/********************************************************
  Function   : user()
  Purpose    : For Model Based Motion Planning Demo.
  Parameters: void
  Returns    : void
  Comments  :  Aug. 20, 1996  Mahmoud Wahdan
*********************************************************/

#include "user.h"
#define FREQUENCY 50

void user1();
void user2();
void user3();
void user4();

void user()
{
  int selection;

  printf("\n Enter 1 for racetrack without localization correction.");
  printf("\n Enter 2 for racetrack with localization correction");
  printf("\12 Enter 3 for POLYGON TRACKING");
  printf("\12 Enter 4 for LOCAL MOTION");

  printf("\n\n The choice is: ");

  selection = GetInt();

  switch (selection)
  {
    case 1:
      user1();
        break;
    case 2:
        user2();
        break;
    case 3:
        user3();
```

```
        break;
    case 4:
        user4();
        break;
    default:
        break;

  }
}

/*******************************************************
  Function   : user1()
  Purpose    : racetrack without localization correction
  Parameters: void
  Returns    : void
  Comments   : Aug. 20, 1996  Mahmoud Wahdan
*******************************************************/

void user1()
{
  CONFIGURATION start;
  CONFIGURATION reference_path;
  CONFIGURATION delta1, delta2, delta3;

  int laps;
  int lap_count = 0;

  start   = defineConfig(77.0, 512.0, HPI, 0.0);
  delta1  = defineConfig(225.0, 0.0, 0.0, 0.0);
  delta2  = defineConfig(-325.0, -100.0, -PI, 0.0);
  delta3  = defineConfig(-100.0, -100.0, -PI, 0.0);

  reference_path  = start;

  setLinVelImm(35.0);
  setSigmaImm(30.0);

  setRobotConfigImm(start);

  printf("\n Enter desired number of laps. ");
  laps=GetInt();
```

```
  while (lap_count < laps)
  {
        reference_path = compose(&reference_path, &delta1);
        nbline(reference_path);

        reference_path = compose(&reference_path, &delta2);
        nbline(reference_path);

        reference_path = compose(&reference_path, &delta3);

        if(lap_count == (laps-1))

            bline(reference_path);

        else

            nbline(reference_path);

++lap_count;
   }

 }

/*******************************************************
  Function  : user2()
  Purpose   : racetrack with localization correction
  Parameters: void
  Returns   : void
  Comments  : Aug. 20, 1996  Mahmoud Wahdan
*******************************************************/

void user2()
{
  CONFIGURATION start;
  CONFIGURATION reference_path;
  CONFIGURATION delta1, delta2, delta3;
  CONFIGURATION qsegment;
  CONFIGURATION qmodel;

  int laps;
  int lap_count = 0;
  int match_seg;
```

```
start   = defineConfig(77.0, 512.0, HPI, 0.0);
delta1  = defineConfig(225.0, 0.0, 0.0, 0.0);
delta2  = defineConfig(-325.0, -100.0, -PI, 0.0);
delta3  = defineConfig(-100.0, -100.0, -PI, 0.0);
qmodel  = defineConfig(0.0, 612.14, -HPI, 0.0);

setLinVelImm(30.0);
setSigmaImm(30.0);

reference_path  = start;

setRobotConfigImm(start);

MotionLog(NULL,FREQUENCY,0);
EnableSonar(S270);
EnableLinearFitting(S270);

printf("\n Enter desired number of laps. ");
laps=GetInt();

while (lap_count < laps)
{

      reference_path = compose(&reference_path, &delta1);
      nbline(reference_path);

      while(1)
{
          WaitSegment(S270);
          qsegment = GetSegmentConfig();
          match_seg = Match(qsegment, qmodel);
          printf("\n match_seq = %d", match_seg);
          if (match_seg == -1)
          break;
      }

      printf("\n qmodel.Posit.X = %f",qmodel.Posit.X);
      printf("\n qmodel.Posit.Y = %f",qmodel.Posit.Y);
      printf("\n qmodel.Theta = %f",qmodel.Theta*RAD);
      printf("\n qsegment.Posit.X = %f",qsegment.Posit.X);
```

```
        printf("\n qsegment.Posit.Y = %f",qsegment.Posit.Y);
        printf("\n qsegment.Theta = %f",qsegment.Theta*RAD);

        CorrectOdometryError(qsegment, qmodel);

        reference_path = compose(&reference_path, &delta2);
        nbline(reference_path);

        reference_path = compose(&reference_path, &delta3);

        if(lap_count == (laps-1))

            bline(reference_path);

        else

            nbline(reference_path);


++lap_count;

  }

  waitMotionEnd();
  DisableLinearFitting(S270);
}

/*****************************************************
  Function   : user3()
  Purpose    : polygon tracking
  Parameters: void
  Returns    : void
  Comments   : Aug. 20, 1996  Mahmoud Wahdan
*****************************************************/

void user3()
{
  double sigma, speed,clearance ;
  CONFIGURATION  q;

  createPolyModel();
```

```
  printf("\nInput desired speed: ");
  speed = GetReal();
  setLinVelImm(speed);

  printf("\nInput desired clearance: ");
  clearance = GetReal();
  setClearanceImm(clearance);

  printf("\nInput desired smoothness: ");
  sigma = GetReal();
  setSigmaImm(sigma);

  MotionLog(NULL,Frequency,0);

  q = defineConfig(90.0, 450.0, -HPI, 0.0);

  setRobotConfigImm(q);

  polygonTracking();

}

/*****************************************************
  Function   : user4()
  Purpose    : For Polygon Tracking motion
  Parameters: void
  Returns    : void
  Comments   :  Aug. 20, 1996  Mahmoud Wahdan
*****************************************************/

void user4()
{
  double sigma, speed,clearance ;
  CONFIGURATION  q;

  PATH_ELEMENT    path;

  createPolyModel();

  printf("\nInput desired speed: ");
  speed = GetReal();
  setLinVelImm(speed);
```

```
  printf("\nInput desired clearance: ");
  clearance = GetReal();
  setClearanceImm(clearance);

  printf("\nInput desired smoothness: ");
  sigma = GetReal();
  setSigmaImm(sigma);

  MotionLog(NULL,FREQUENCY,0);

  q = defineConfig(90.0, 450.0, -HPI, 0.0);

  setRobotConfigImm(q);

  motionTracking();

}
```

LIST OF REFERENCES

[1] R. C. Arkin. The impact of cybernetics on the design of a mobile robot system: A case study. *IEEE Transactions on System, Man, and Cybernetics*, 20:1245–1257, 1990.

[2] J. Barraquand and J. C. Latombe. On non-holonomic mobile robots and optimal maneuvering. *Proc. of the 4th IEEE International Symposition on Intelligent Control, Albany, NY*, 1989.

[3] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *Internat. J. Robot. Res.*, 10:2628–649, 1991.

[4] N. Bloch. *Abstract Algebra with Applications.* Prentice Hall, 1987.

[5] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, Vol. RA-2 N 1*, 1986.

[6] C. E. Buckley. The application of continuum methods to path planning. *Ph.D. Dissertation, Stanford University, CA*, August 1985.

[7] J. Budenske and M. Gina. Achieving goals through interaction with sensors and actuators. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 903–908, 1992.

[8] J. Budenske and M. Gina. Why is it so difficult for a robot to pass through a doorway. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3124–3129, 1994.

[9] C. L. Chuang. Layered safe motion planning for autonomous vehicles. *Ph.D. Dissertation, Naval Postgraduate School, Monterey, California*, September 1995.

[10] J. Connell. Sss: A hybrid architecture applied to robot navigation. *Proc. IEEE Conf. Robotics and Automation*, pages 2719–2725, 1992.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[12] I. Cox. Blanche - an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7:193–204, 1991.

[13] F. H. Croom. *Basic Concepts of Algebraic Topology.* Springer-Verlag, 1978.

[14] J. L. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. *Proc. of IEEE International Conf. on Robotics and Automation, Scottsdale, Arizona*, pages 674–680, May 1989.

[15] F. Dierks. *Freie Navigation Autonomer Fahrzeuge, In: P. Levi, T. Bräunl, eds., Autonome Mobile Systeme 1994.* Springer-Verlag, Berlin, 1994.

[16] M. Drumheller. Mobile robot localization using sonar. *Tech. Report A.I.Memo 826, MIT, AI Laboratory, Cambridge, MA*, 1985.

[17] R. Hollier (ed.). *Automated Guided Vehicle Systems.* Springer-Verlag, Berlin, 1987.

[18] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation, RA-3(3)*, pages 674–680, 1987.

[19] J. Evans, B. Krishnamurthy, B. Barrows, and T. Skewis. Handling real-world motion planning: A hospital transport robot. *IEEE Control Systems Magazine*, 12:15–20, 1992.

[20] H. Everett and E. Stitz. *Survey of Collision Avoidance and Ranging Sensors for Mobile Robots.* Naval Command, Control and Ocean Surveillance Center, Technicle Note 1194, Update 1, December 1992.

[21] R. J. Firby. Adaptive execution in complex dynamic worlds. *Ph.D. Dissertation, Yala University, CT*, May 1989.

[22] C. Floyd, Y. Kanayama, and C. Magrino. Underwater obstacle recognition using a low-resolution sonar. *Proc. Seventh International Symposium on Unmanned Untethered Submersible Technology*, September 1991.

[23] T. Fraichard and C. Laugier. Path-velocity decomposition revisited and applied to dynamics trajectory planning. *IEEE int. Conf. on Robotics and Automation*, pages 40–45, 1993.

[24] J. B. Fraleigh. *A First Course in Abtract Algebra.* Addison-Wesley Pub., 1993.

[25] T. W. Gamelin and R. E. Greene. *Introduction to Topology.* Saunders College Pub., 1983.

[26] B. Gray. *Homotopy Theory: An Introduction to Algebraic Topology.* The Academic Press, 1975.

[27] V. Guillemin and A. Pollack. *Differential Topology.* Prentic Hall, 1974.

[28] I. Guttman and S. S. Wilkes. *Introduction to Engineering Statistics.* John Wiley x Sons, Inc., New York, 1965.

[29] A. Holenstein, M. Muller, and E. Badreddin. Mobile robot localization in structured environment cluttered with obstacles. *IEEE Conf. of Robotics and Automation*, pages 2576–2582, May 1992.

[30] J. Horn and G. Schmidt. Continuous localization for long-range indoor navigation of mobile robots. *IEEE International Conf. On Robotics and Automation*, pages 387–394, 1995.

[31] C. Hsiung. *A First Course in Differential Geometry.* John Wiley and Sons, 1981.

[32] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys, Surv., 24(3)*, pages 3219–291, 1992.

[33] Y. K. Hwang, P. C. Chen, A. A. Maciejewski, and D. D. Neidigk. A global motion planner for curve-tracing robots. *IEEE int. Conf. on Robotics and Automation*, pages 2–7, 1994.

[34] Inc. Ironics. *IV-SPARC-25A/33A VMEbus Single Board Super Computer and MultiProcessing Engine—User's Manual.* Ironics, Inc., New York, 1992.

[35] Y. Kanayama. Two dimensional wheeled vehicle kinematics. *IEEE Int. Conf on Robotics and Automation, in San Diego, California*, pages 3079–3084, May 1994.

[36] Y. Kanayama. Introduction to theoretical robotics. *Lecture Notes of the Advanced Robotics Course, Department of Computer Science, Naval Postgraduate School*, 1996.

[37] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. *IEEE int. Conf. on Robotics and Automation*, pages 1265–1270, 1989.

[38] Y. Kanayama, K. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. *IEEE int. Conf. on Robotics and Automation*, pages 1315–1317, 1988.

[39] Y. Kanayama, D. L. MacPherson, and G. W. Krahn. Two dimensional transformations and its application to vehicle motion control and analysis. *Proc. of International Conf. on Robotics and Automation, in Atlanta, Georgia*, pages 13–18, May 1993.

[40] Y. Kanayama and T. Noguchi. Spatial learning by an autonomous mobile robot with ultrasonic sensors. *University of California Santa Barbara Dept. of Comp. Sci. Technical Report TRCS89-06*, February 1989.

[41] Y. Kanayama and M. Onishi. Locomotion functions for a mobile robot language, mml. *IEEE int. Conf. on Robotics and Automation*, pages 1110–1115, 1991.

[42] O. Khatib. Teal-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90–98, September 1986.

[43] J. R. Kirkwood. *An Introduction to Analysis.* PWS-KENT Pub. Company, 1989.

[44] R. Klein. *Concrete and Abstract Voronoi Diagrams, Lecture Notes in Computer Science.* Springer-Verlag, 1987.

[45] T. M. Knasel. Mobile robotics - state of the art review. *International Journal of Robotics Research*, 1, 1986.

[46] A. Kosaka and A. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *Proc. 1992 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Raleigh, North Carolina*, pages 2177–2186, 1992.

[47] J. G. Kovalchik. *Layered Motion Planning for Autonomous Mobile Robots using Free Space Decomposition and Steering Functions.* Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, 1995.

[48] D. J. Kriegman, E. Triendl, and T.O. Binford. Stereo vision and navigation in building for mobile robots. *IEEE Trans. on Robotics and Automation, 5(6)*, December 1989.

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

[50] L. Laumond. Feasible trajectory for mobile robots with kinematic and environment constraints. *Proc. of the International Conf. on Intelligent Autonomous Systems, Amsterdam, The Netherlands*, pages 346–354, 1986.

[51] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7:376–382, 1991.

[52] J. T. Lochner. *Analysis and Improvement of an Ultrasonic Sonar System on an Autonomous Mobile Robot.* Master Thesis, Naval Postgraduate School, Monterey, California, 1994.

[53] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transaction on Computers*, 32:108–119, 1983.

[54] T. Lozano-Perez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22:165–175, 1979.

[55] D. L. Macpherson. *Automated Cartography by an Autonomous Mobile Robot using Ultrasonic Range Finders.* Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, 1993.

[56] L. Matthies and S. A. Shafer. Error modeling in stereo navigation. *IEEE Journal of Robotics and Automation, RA-3(3)*, pages 239–1.548, 1987.

[57] H. M. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* Stanford AI Lab Memo AIM-340, 1980.

[58] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* The MIT Press, 1984.

[59] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[60] K. A. Ross. *Elementary Analysis: The Theory of Calculus.* Springer-Verlag, 1980.

[61] S. R. Sherfey. *A Mobile Robot Sonar System.* Master Thesis, Naval Postgraduate School, Monterey, California, 1991.

[62] T. Skewis and V. Lumelsky. Experiments with a mobile robot operating in a cluttered unknown environment. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1482–1482, May 1992.

[63] M. Spivak. *A Comprehensive Introduction to Differential Geometry, Vol 1 and 2.* Publish or Perish, Inc., Berkeley, CA, 1979.

[64] S. H. Suh and K. G. Shin. A variational dynamic programming approach to robot-path planning with a distance-safety criterion. *IEEE Journal of Robotics and Automation*, 4:334–349, 1988.

[65] F. Vacherand. Fast local path planner in certainty grid. *IEEE int. Conf. on Robotics and Automation*, pages 2132–2137, 1994.

[66] Y. Watanabe and S. Yuta. Position estimation of mobile robots with internal and external sensors using uncertainty evolution technique. *Proc. of IEEE International Conf. on Robotics and Automation,Scottsdale, Arizona*, pages 2011–2016, May 1990.

[67] D. Zwillinger. *Standard Mathematical Tables and Formulae.* CRC Press, 1996.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Road., Ste 0922 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3.	Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5101	1
4.	Professor Yutaka Kanayama, Code CS/Ka Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5101	2
5.	Professor C. Thomas Wu, Code CS/Wq Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5101	1
6.	Professor Cynthia Irvine, Code CS/Ir Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5101	1
7.	Professor Craig Rasmussen, Code MA/Ra Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5101	1
8.	Professor Fariba Fahroo, Code MA/Ff Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5101	1

9. Professor Xiaoping Yun, Code ECE/Yu 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5101

10. Professor Chirs Frenzen, Code MA/Fr 1
Department of Mathematics
Naval Postgraduate School
Monterey, CA 93943-5101

11. Professor Harold Fredricksen, Code MA/Fs 1
Department of Mathematics
Naval Postgraduate School
Monterey, CA 93943-5101

12. LTC. Nabil Khalil, Code ECE/Ph 3
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5101

13. Maj. Khaled Morsy, Code CS/Ph 3
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5101

14. Maj. Ashraf Mamdouh, Code ECE/Ph 3
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5101

15. Egyptian Military Attache 2
2308 Tracy Place NW
Washington, DC 20008

16. Egyptian Armament Authority - Training Department 2
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-0051

17. Military Technical College (Egypt) 2
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-0051

18. Military Research Center (Egypt) 2
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-0051

19. COL. Mahmoud Wahdan (Egypt) 3
5 El-Shrif Street//Roxy Cario//Egypt