

1987

# Software design and theory for a mobile robot /

Mona Ahmad Elgayar  
*Lehigh University*

Follow this and additional works at: <https://preserve.lehigh.edu/etd>



Part of the [Electrical and Computer Engineering Commons](#)

---

## Recommended Citation

Elgayar, Mona Ahmad, "Software design and theory for a mobile robot /" (1987). *Theses and Dissertations*. 4746.  
<https://preserve.lehigh.edu/etd/4746>

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact [preserve@lehigh.edu](mailto:preserve@lehigh.edu).

SOFTWARE DESIGN AND THEORY  
FOR A  
MOBILE ROBOT

by

Mona Ahmad Elgayar

A Thesis

Presented to the Graduate Committee

of Lehigh University

in candidacy for the degree of

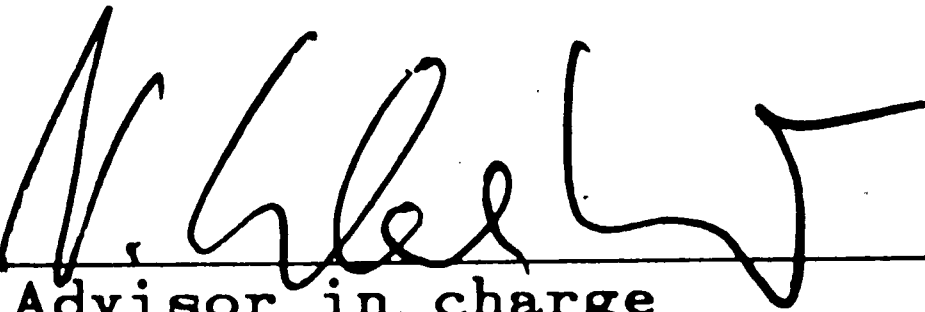
Master of Science in Computer Science

Lehigh University

1987

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

date: May 14, 1987

  
\_\_\_\_\_  
Advisor in charge

  
\_\_\_\_\_  
CSEE Department Chairperson

## Acknowledgements

I would like to give my special thanks to Professor Nikolai Eberhardt who was responsible for the design of the vehicle and the mathematical theory behind the navigation of the system. It has been a great pleasure and experience working with him, I have learned a great deal while working with him. I would also like to express my thanks to Professor Meghanad Wagh who taught me a great deal in programming the vehicle, and helping me out whenever I needed help. In addition, I would like to thank the most wonderful and closest people to me, my father, mother, brother, sister, and Waleed for the support they gave me during my stay at Lehigh University.

# Table of Contents

|  |           |
|--|-----------|
| <b>1. Introduction</b>   | <b>3</b>  |
| <b>2. Optical Navigation System</b>                            | <b>8</b>  |
| 2.1 Beacon Identification                                      | 12        |
| 2.1.1 Identification Via Angle Range                           | 12        |
| 2.1.2 Distance Thresholding                                    | 16        |
| 2.1.3 Identification Via Distance Range                        | 18        |
| 2.1.4 Final Checking   | 20        |
| 2.2 Beacon Selection   | 20        |
| 2.2.1 Beacon Selection Algorithm                               | 23        |
| 2.3 The Mathematical Theory For Obtaining The Vehicle Position | 25        |
| 2.3.1 Vehicle Position   | 25        |
| 2.3.2 Vehicle Bearing  | 33        |
| 2.4 The Optical Navigation System Software                     | 35        |
| 2.4.1 The goniometer procedure                                 | 35        |
| 2.4.2 Identify beacons Procedure                               | 36        |
| 2.4.3 The Expected Angle Procedure                             | 38        |
| 2.4.4 Compute Vehicle Position procedure                       | 39        |
| 2.4.5 Calculate vehicle points procedure                       | 41        |
| 2.4.6 Find Center Procedure                                    | 42        |
| 2.4.7 Find Vehicle Coordinates Procedure                       | 42        |
| 2.4.8 Beacon Selection Procedure                               | 44        |
| 2.4.9 Shuffpoints Procedure                                    | 44        |
| <b>3. The Ground Navigation System</b>                         | <b>46</b> |
| 3.1 The Ground Navigation System                               | 47        |
| 3.2 The Mathematical Theory For Ground Navigation System       | 47        |
| 3.3 The Ground Navigation Software                             | 53        |
| <b>4. The Drive Routine</b>                                    | <b>57</b> |
| 4.1 The Mathematical Theory of The Drive Routine               | 58        |
| 4.1.1 Computation of Parameters                                | 58        |
| 4.1.2 The Computation of Double Arcs And Radius                | 60        |
| 4.2 The Computation of the Steering Angle                      | 69        |
| 4.3 The Drive Routine Software                                 | 69        |
| 4.3.1 Compute Parameters and Angles Procedure                  | 73        |
| 4.3.2 One Arc and Straight Line ONEC1 Procedure                | 74        |
| 4.3.3 A Straight Line and One Arc ONEC2 Procedure              | 74        |
| 4.3.4 Double arc TWOC1 Procedure                               | 76        |
| 4.3.5 Double arc TWOC2 Procedure                               | 76        |
| 4.3.6 Double arc TWOC3 Procedure                               | 78        |
| <b>5. The Overall System Design</b>                            | <b>79</b> |
| 5.1 The Full Travel  | 79        |
| 5.2 The Simplified Travel                                      | 82        |

|  |           |
|--|-----------|
| <b>6. Conclusions</b>                      | <b>85</b> |
| <b>References</b>                          | <b>88</b> |
| <b>Appendix A. The Goniometer Software</b> | <b>89</b> |
| <b>Appendix B. The Drive Routine</b>       | <b>99</b> |

## List of Figures

|                     |  |    |
|---------------------|--|----|
| <b>Figure 1-1:</b>  | "Cyclopion" the Test Bed Vehicle                               | 4  |
| <b>Figure 1-2:</b>  | The Software Design of The Complete System                     | 6  |
| <b>Figure 2-1:</b>  | The Optical Goniometer.  | 9  |
| <b>Figure 2-2:</b>  | The Overall Software Design for the Optical Navigation System. | 13 |
| <b>Figure 2-3:</b>  | The Expected Angle $\psi_i$ .                                  | 15 |
| <b>Figure 2-4:</b>  | The Result of the Identification Process.                      | 17 |
| <b>Figure 2-5:</b>  | The Computation of Distance.                                   | 19 |
| <b>Figure 2-6:</b>  | Distance Error.  | 21 |
| <b>Figure 2-7:</b>  | The Beacon Selection Algorithm.                                | 24 |
| <b>Figure 2-8:</b>  | The Three Circles A, B and C.                                  | 26 |
| <b>Figure 2-9:</b>  | The Calculation of the Center of a Circle.                     | 28 |
| <b>Figure 2-10:</b> | Computation of the Vehicle's Point.                            | 31 |
| <b>Figure 2-11:</b> | Vehicle Bearing.   | 34 |
| <b>Figure 2-12:</b> | Identify Beacons Procedure                                     | 37 |
| <b>Figure 2-13:</b> | Expected Angle Procedure                                       | 38 |
| <b>Figure 2-14:</b> | Compute Vehicle Position Procedure                             | 40 |
| <b>Figure 2-15:</b> | Compute Vehicle Points Procedure                               | 41 |
| <b>Figure 2-16:</b> | Find Vehicle Coordinates Procedure                             | 43 |
| <b>Figure 2-17:</b> | Shuffpoints Procedure  | 45 |
| <b>Figure 3-1:</b>  | The geometry of the vehicle                                    | 48 |
| <b>Figure 3-2:</b>  | The minimum number of $n$ counts                               | 51 |
| <b>Figure 3-3:</b>  | The geometry of the vehicle                                    | 52 |
| <b>Figure 3-4:</b>  | $\Delta l$ Signal service routine                              | 56 |
| <b>Figure 4-1:</b>  | Case (1) of Double Arc   | 61 |
| <b>Figure 4-2:</b>  | Case (2) of Double Arc   | 65 |
| <b>Figure 4-3:</b>  | Case (3) of Double Arc   | 66 |
| <b>Figure 4-4:</b>  | Case (4) of Double Arc   | 67 |
| <b>Figure 4-5:</b>  | Case (5) of Double Arc   | 68 |
| <b>Figure 4-6:</b>  | The Drive Routines   | 71 |
| <b>Figure 4-7:</b>  | One circle and straight line procedure                         | 75 |
| <b>Figure 4-8:</b>  | Double Arcs Procedure  | 77 |
| <b>Figure 5-1:</b>  | The Full Travel  | 80 |
| <b>Figure 5-2:</b>  | The Simplified Travel  | 83 |

# List of Tables

**Table 6-1:** The Result of A Typical Vehicle Run

87



## Abstract

This thesis presents the mathematical theory and the software design for an autonomous guided vehicle. The mathematical theory and algorithms enable the autonomous vehicle to determine its location accurately in a known world. The vehicle has two navigation systems, 1) the optical navigation system is based on a goniometer that emits an infrared beam. This is reflected from retroreflectors (beacons) scattered in the constant environment (factory floor). Angular displacement of the vehicles heading relative to each beacon is recorded by the rotating goniometer, and the actual position of the vehicle and bearing is then computed using trigonometry. 2) The ground navigation system consists of a gyroscope to measure the vehicles bearing, and the wheel encoder (odometer) to measure the distance traveled by the front wheel. The bearing and the distance traveled is used to determine the vehicles position frequently. The two navigation systems complement each other, since the ground navigation system can compute the vehicles position more frequently but less accurately, whereas the optical navigation system produces more accurate results but can compute the vehicles position only every 0.5 seconds

The system software is written in PLM/86 for the 8086 micro processor and operates in conjunction with a 8087 math processor, except for the ground navigator program, which is written in 8086 assembly language. This thesis describes the algorithm and software used to compute the vehicle position and bearing via the optical systems and the ground navigation system. It also includes the drive routine that drives the vehicle from the current point to the next point along the prescribed path. Finally, the overall systems and their interactions are described.

# Chapter 1

## Introduction

The correct determination of the location of an autonomous vehicle is of great importance to manufacturing and material handling systems. In particular, robots that navigate and perform more autonomous roles are becoming more and more necessary in today's and tomorrow's factories. Path planning, obstacle avoidance, and replanning are necessary for an autonomous robot to achieve a goal. The more autonomous the robot becomes the more it will need to rely on sensors to ascertain its environment. One important task in the overall perception of the environment is the determination of its position. Previous researchers employed various methods for robot self-location such as computer vision, artificial markers, and an angular optical scanning sensor to detect signals from three or four transmitters placed in strategic locations in the area of navigation [1].

This thesis describes the mathematical theory and the software design that will enable an autonomous guided vehicle AGV to determine its  $x-y$  position and bearing within a known world. Figure 1-1 Shows the "Cyclopion" the test bed vehicle for this project. The vehicle is equipped with two navigation systems: an "Optical" navigation system that consists of a goniometer that emits an infrared laser beam towards retroreflectors (beacons) that are scattered in the plant environment, and measures the angles under which the individual beacons are sighted. By determining the angles at which three beacons are sighted, the on board computer can calculate the position of the vehicle accurately. The vehicle uses another navigation system called the "Ground" navigation system.

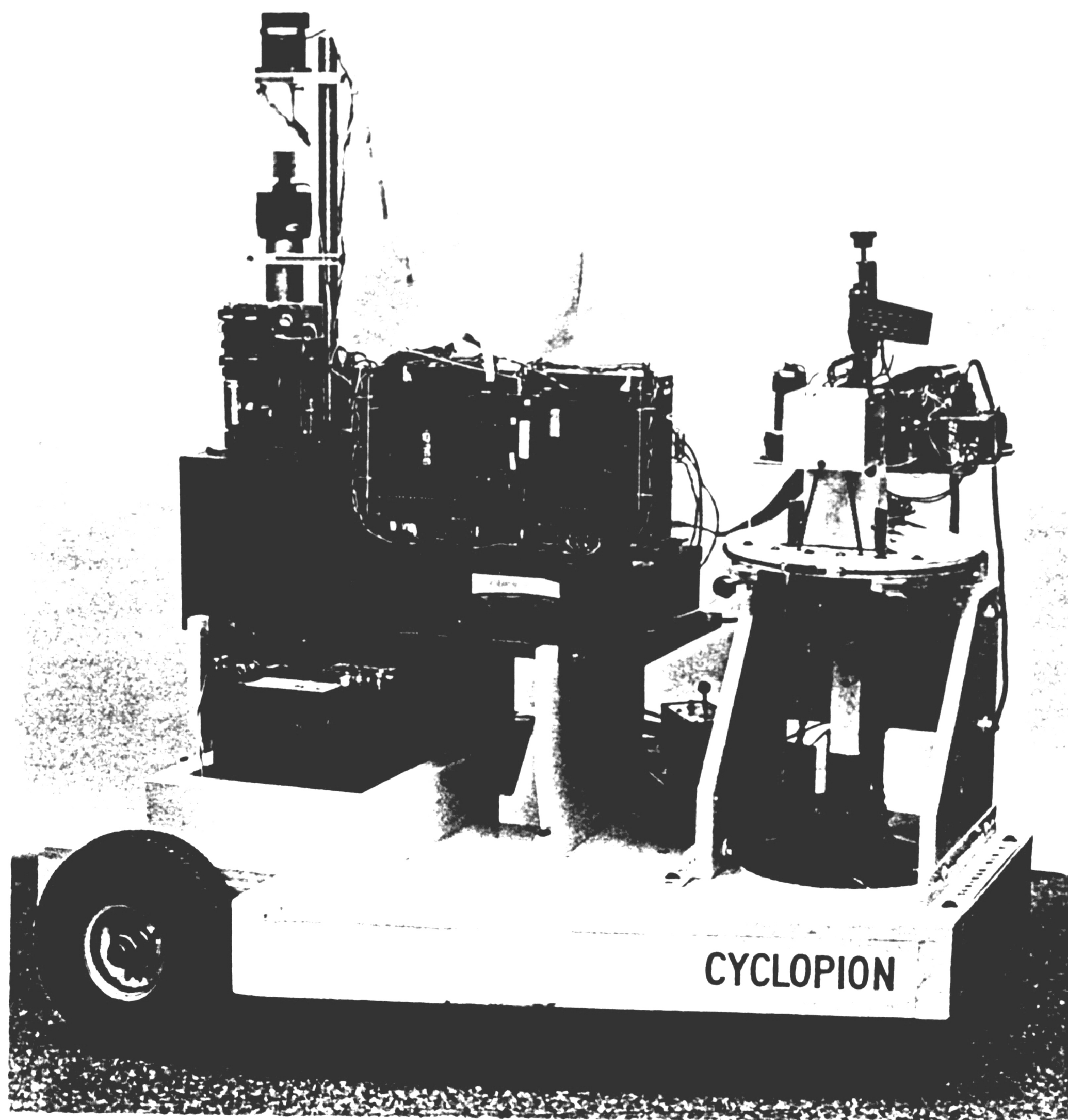


Figure 1-1: "Cyclopion" the Test Bed Vehicle

The ground navigation system consists of a wheel encoder ( odometer ) that measures the travel of the front wheel, and a gyroscope that measures the vehicle bearing. With this data the on-board computer can compute the vehicle position and bearing less accurately. Therefore, the two navigation systems support each other to allow the vehicle to travel along its prescribed path as accurate as possible. The optical supplies the ground navigator with an accurate vehicle position every 0.5 seconds. The ground navigation computes the vehicle position every 0.005 seconds. The thesis describes the derivation of the equations used for guiding this vehicle and describes the algorithms that are used to determine the flow and the logic of each procedure . The program written for the algorithms were run on an 8086 micro processor in conjunction with an Intel 8087 numeric coprocessor.

Figure 1-2 on page 6 shows a block diagram of the overall functional diagram of the entire system. A summary of each procedure is given below:

- **The Goniometer Procedure.** Reads the beacon angles from the specified ports, the beacon identification process is executed to determine the right beacons, and to omit anonymous objects. The beacons selection process is then followed to select the best three beacons that are needed for triangulation. Finally, it computes the vehicle position and bearing.
- **The Ground Navigation Procedure.** With the readings of the wheel encoder ( odometer ) and the reading of the gyroscope, a set of equations are obtained to compute the vehicle position for every 0.005 seconds.
- **The Drive Routine.** Requests position of vehicle and bearing when needed and computes the double arcs and steering angle of the vehicle for each arc. This is then translated to the drive routine as a command.

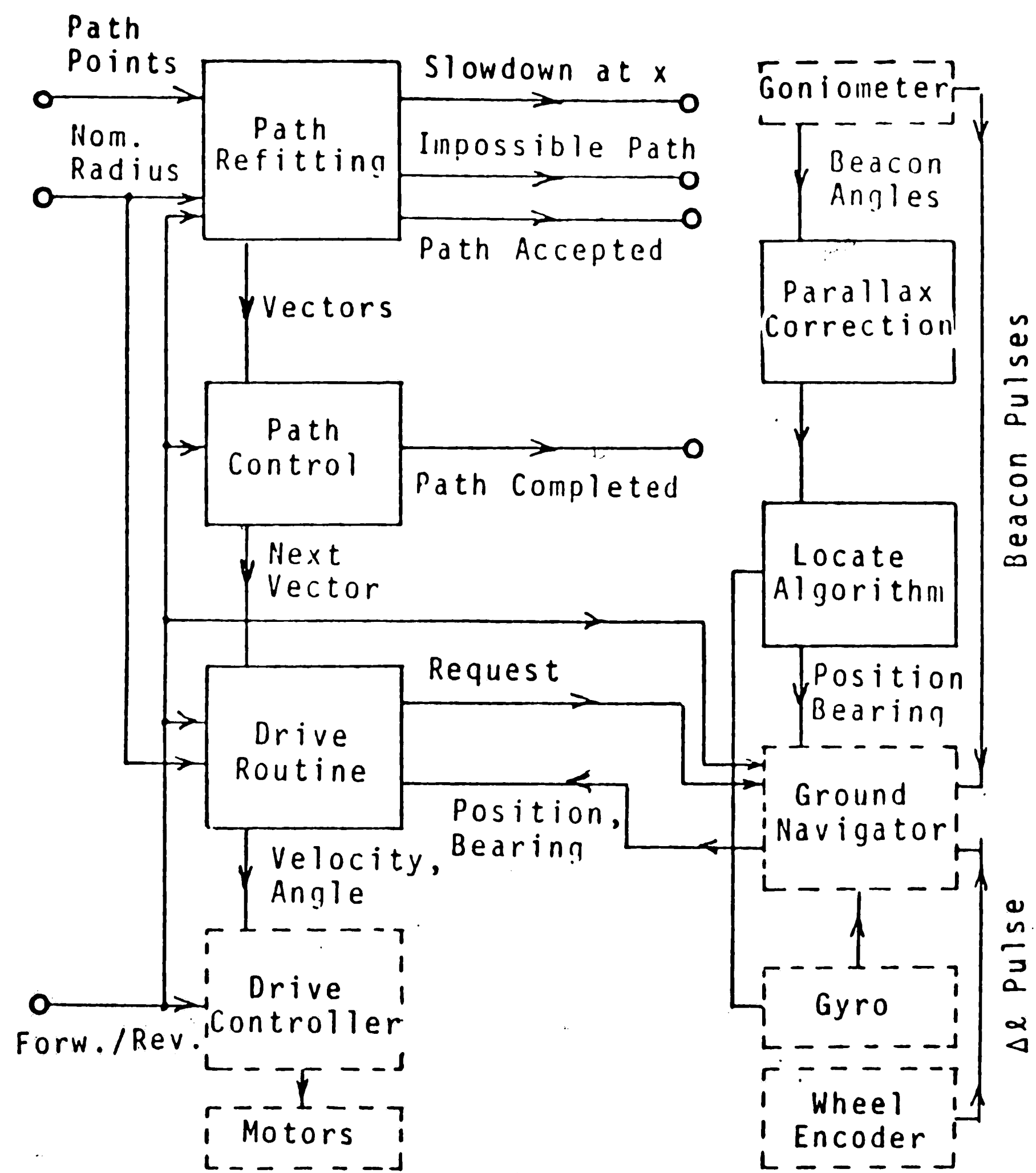


Figure 1-2: The Software Design of The Complete System

- **The Overall System.** Describes the sequence of events for computing the vehicle position and bearing, and when the double arcs are to be computed. The full design of the overall system and the simplified design that has been implemented successfully by the vehicle are also described.

The software described in this thesis is designed for the Cyclopion navigation system, but the theory and the concepts can be very well applied to any autonomous vehicle of this kind. Other approaches to non wire guided vehicles can be found in [1] and [2].

## Chapter 2

# Optical Navigation System

The optical system consists of a goniometer, a device designed to emit a horizontally sweeping infrared laser beam. This beam detects retroreflecting surfaces, called beacons and measures the angles under which the individual beacons are sighted.

The goniometer is mounted on the vehicle at the midpoint of the rear axle as shown in Figure 1-1 on page 4. Figure 2-1, page 9, depicts the device in more detail. It consists of an Amprex Corporation CQL16 diod laser called a collimator pen. This pen produces a well collimated beam that is about five millimeters in diameter and is directed upwards so as to strike a front surface mirror. The mirror is situated at a  $45^\circ$  angle to the beam so that it reflects the laser beam in a direction  $90^\circ$  to its incidence. This mirror is connected to a synchronous motor which rotates the mirror at two revolutions per second. Mounted on top of the motor is a Hewlett Packard HEDS-6000 series shaft encoder that produces one thousand pulses and an index pulse for each revolution of the mirror. The index pulse is used as a zero reference angle for the goniometer and also initiates readout and computation of vehicle position and bearing.

Whenever the laser beam strikes a retroreflecting tape (a beacon) the beam is reflected back onto itself. Since the reflected beam is larger in diameter than the incidence beam, it is focused on arrival by a mirror lens into the photodetector. The resulting output is a pulse which indicates the sighted

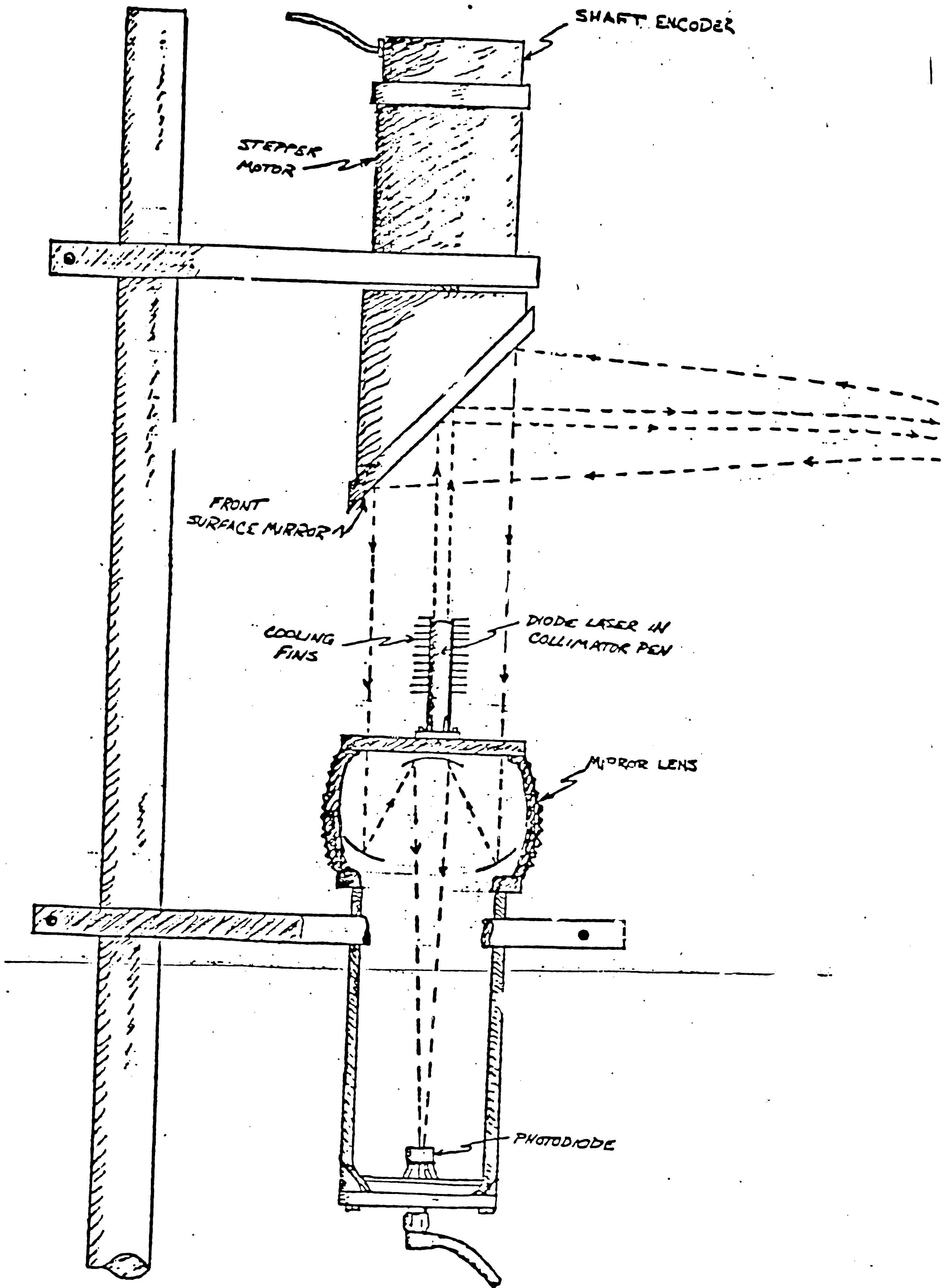


Figure 2-1: The Optical Goniometer.



beacon angle and the angular width of the beacon. A description of the manner in which the angle is measured follows.

Once the encoder produces an index pulse, an internal counter starts counting the number of pulses generated by a clock synchronized with the mirror rotation. Every time a beacon is sighted, the current count is stored in RAM. The angles are read through the I/O ports and are scaled since they are produced in binary numbers ranging from zero to 50,000, where the 50,000 represents 360°. In addition to storing the beacon sighted angle, the system also stores the beacon width. The actual beacons are all of the same width, but their widths appear smaller to the goniometer as the distance from goniometer to beacon increases. The apparent width is measured by the duration of the received optical pulse. At present, the system stores up to fifteen beacon angles and widths per revolution. Also part of the optical system is an electronic circuitry and computer interface board.

Calculations of position may be performed only once every half second. This is one reason why the navigation of the vehicle cannot depend on the optical system alone. Another navigation system, the ground navigation, almost continuously keeps track of the vehicle's position and bearing using inputs from an odometer and a gyroscope. This system, however, constantly accumulates errors generated by the ground navigation system and these have to be corrected by the optical navigation system every time a position fix becomes available.

The computer software of the optical navigation system is designed to compute the vehicle position and bearing once the beacon has been sighted. The index pulse activates the goniometer program. Once an index pulse is generated and beacon angles and widths have been read into the buffer through I/O ports, an interrupt is generated masking all interrupt service routines excepting the goniometer routine. Also, once the beacons angles have been read in, the system prohibits further reading in of beacon angles and widths during the time of computation.

Once they have been read in, the beacons angles are calibrated and stored in an array of dimension  $M$ , where  $M$  is the number of beacons sighted. With an approximate vehicle position and bearing, and a map of beacon coordinates and expected angles as input, the goniometer procedure goes through the following steps in the order stated, to produce an accurate vehicle position and bearing as output:

1. It identifies the beacon, determining if the angles read in represent valid beacons, since the goniometer can encounter other reflecting objects like mirror or glass. The algorithm is described in Section 2.1, page 12.
2. It selects the most desirable set of beacons from the set of identified beacons. This should not be less than three beacons, the minimum numbers needed for a position fix.
3. Finally, it computes the vehicle's position and bearing using the triangulation routine as described in Section 2.3, page 25.

The index pulse can not be deactivated by the software but can be ignored whenever not needed. Once the accurate vehicle position is calculated, the ground navigation system utilizes the computed vehicle position to correct

the vehicle's position and bearing. The correct vehicle position will be used as input to the drive routine, thus driving the vehicle towards the next programmed point.

Shown on page 13, Figure 2-2 describes the overall design for the software used in the optical system. If the vehicle fails to find enough beacons to calculate its position after several attempts, the ground navigation system will stop the vehicle. The distance the vehicle will be permitted to travel without optical position fixes depends on the accuracy of the ground navigation system. Such a distance has yet to be determined.

## 2.1 Beacon Identification

Beacon identification involves matching a beacon sighted through the goniometer against the beacons stored in the map—if they match then that beacon has been identified. Therefore, the beacon identification process intelligently takes care of invalid objects such as mirrors and glass windows scattered in the factory or the lab. Since they are objects that do not retroreflect, such occurrences are rare, yet possible. The identification program takes care of ambiguous objects by rejecting them from the set of sighted beacons. Varying degrees of refinement for identifying the correct beacons can be applied to such vehicles and what follows is a description of these refinements.

### 2.1.1 Identification Via Angle Range

Vehicle position and bearing is only approximately known, therefore a given beacon will not appear exactly in the expected direction  $\Psi_i$  as computed from the beacon map, but in a vicinity of  $\Psi_i \pm \delta$ . A threshold  $\Delta\psi$  can be

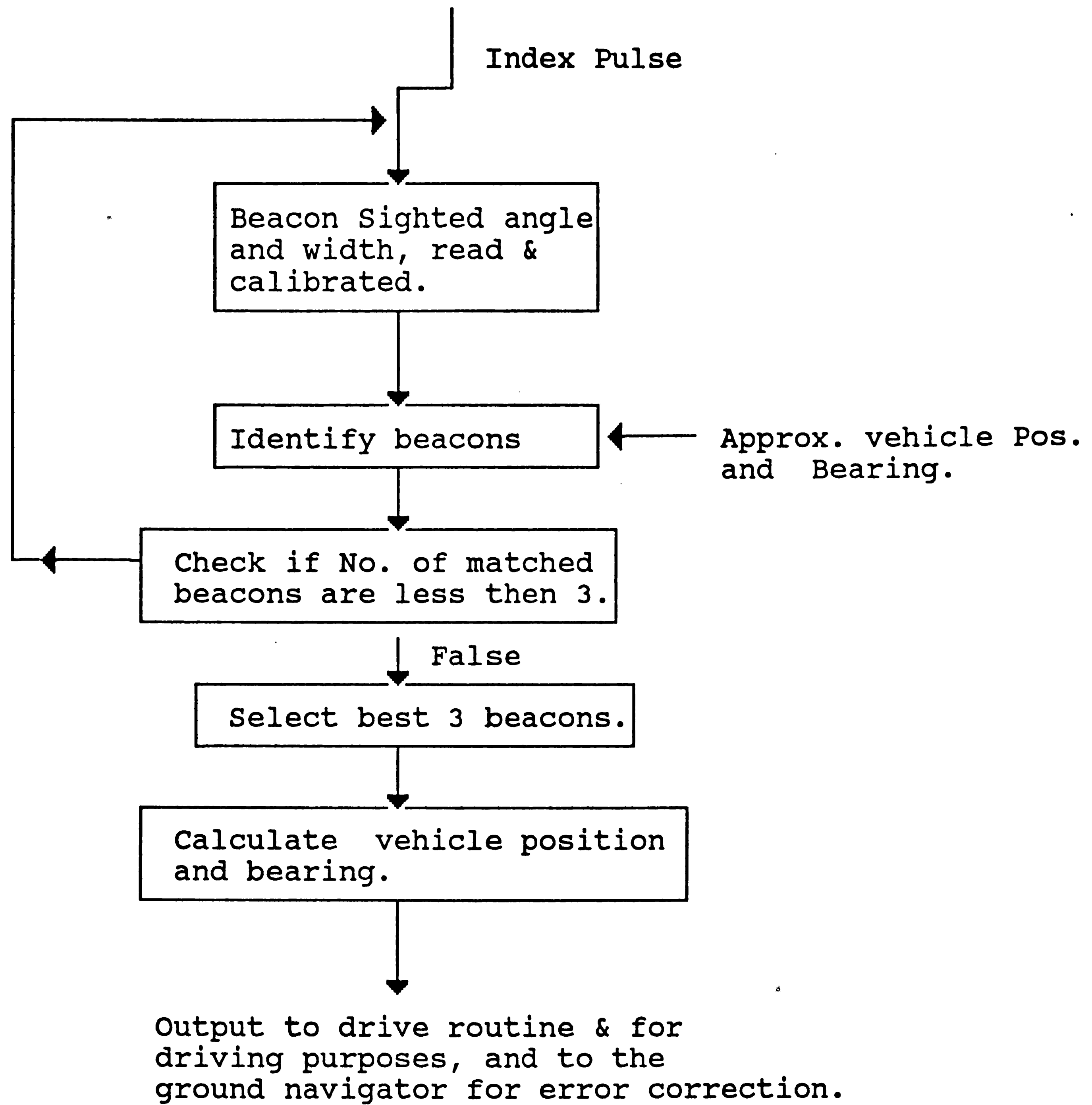


Figure 2-2: The Overall Software Design for the Optical Navigation System.

selected such that if the sighted angle falls within the interval  $\psi_i \pm \Delta\psi$ , the beacon is considered to be identified as the  $i^{\text{th}}$  beacon. Otherwise, the  $i^{\text{th}}$  beacon is rejected. Rejection also occurs if two or more beacons are sighted within one angle range, or if adjacent ranges overlap. In these cases beacons  $i$  and  $i+1$  could be accidentally interchanged and therefore must be rejected.

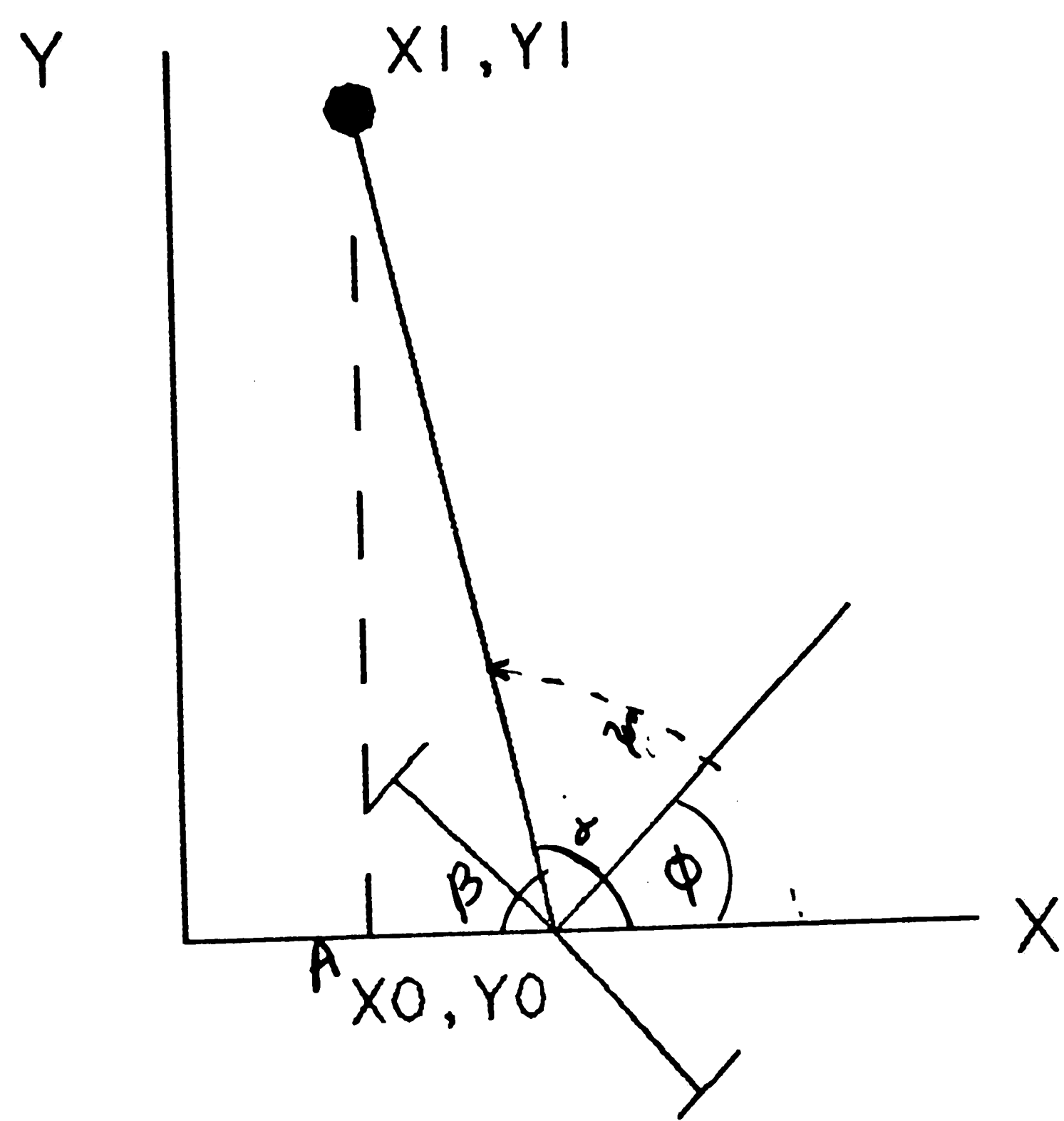
Figure 2-3, page 15, describes the computation of expected angles.  $(X_0, Y_0)$  is the vehicle position,  $(X_i, Y_i)$  are the beacon coordinates,  $\phi$  is the bearing and  $\psi_i$  is the expected angle. At this point  $\alpha$  can be computed from the triangle  $[A, (X_i, Y_i), (X_0, Y_0)]$ . Namely

$$\beta = \tan^{-1} \left[ \frac{Y_i - Y_0}{X_i - X_0} \right], \text{ and,}$$

$$\alpha = 180^\circ - \beta.$$

Therefore,  $\psi_i$  is  $(\alpha - \phi)$ . In this algorithm several cases have to be distinguished due to the modulo  $360^\circ$  nature of angles and the multivaluedness of  $\tan^{-1}$ .

The idea behind this process is to compute the expected beacon angle  $\Psi_k$ —for  $k = 1 \dots L$ , where  $L$  is the number of beacons in the map—using the approximate vehicle position and the  $X$  and  $Y$  coordinates of beacons stored in the map. The expected angles are subtracted from each of the sighted angles  $\Theta_i$ , ascertaining if the difference is less than a certain threshold  $\Delta\psi$ . If so then the  $k^{\text{th}}$  beacon has been identified, unless a second match has been obtained in the same angular range. In this case the two beacons will be rejected. If a beacon has matched then it is added to that set of matched beacons. This process has been successfully applied to the vehicle, but for safer and more reliable results other refinements will be discussed in the following sections.



THE EXPECTED ANGLE

Figure 2-3: The Expected Angle  $\psi_i$ .

In order to identify the right beacons, let us assume that the inputs to the "Identify\_beacon" procedure are as follows:

- $X_a, Y_a$  is the approximate coordinate of the vehicle position as given by the ground navigator;
- $\Phi$  is the approximate vehicle bearing supplied by the gyroscope;
- $X_k, Y_k$  are the beacon coordinates stored in a map, where  $k = 1 \dots L$ , and  $L$  is the number of beacons stored in the map;
- $\Theta_i$  are the sighted beacon angles, where  $i = 1 \dots M$  and  $M$  is the number of beacons sighted.  $\Delta\psi$  is the selected threshold.

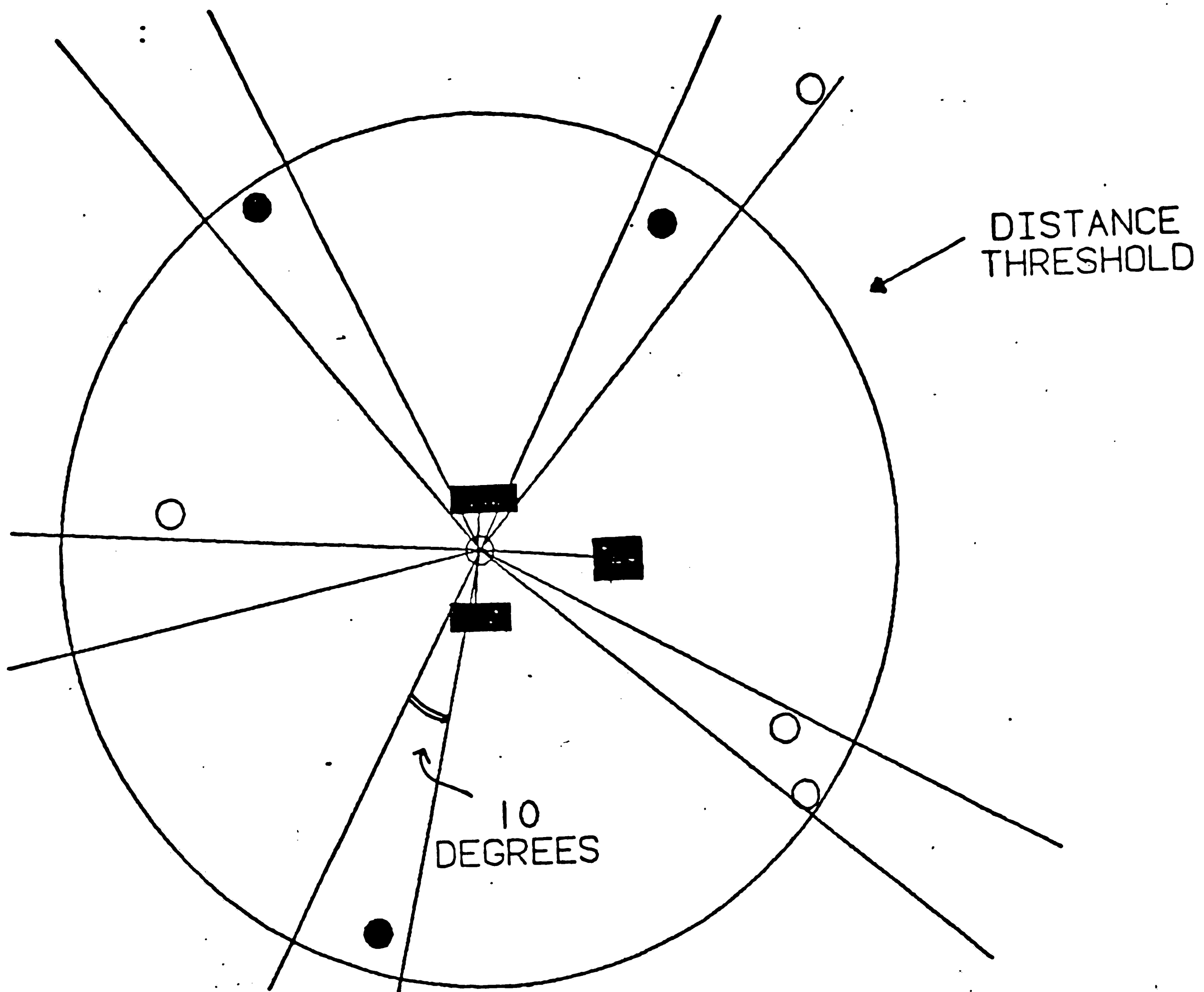
The following algorithm is applied to identify the right beacons:

1. If  $M < 3$ , then stop.
2. Compute the expected beacon angle  $\Psi_k$  from  $X_a, Y_a$  and  $X_k, Y_k$  for each  $k$ , where  $k = 1 \dots L$ .
3. Starting with  $i = 1 \dots M$  for a sighting  $\Theta_i$ , compute  $|\Psi_k - \Theta_i| = \Delta_k$  for  $k = 1 \dots L$ .
4. If  $\Delta_k < \Delta\psi$ , then the  $k^{\text{th}}$  beacon has been identified, unless a second match occurs for a different  $k$ . If no positive identification occurs, go to the next  $i$  in Step 3. If there is a match, record  $X_k, Y_k$  and  $\Theta_i$  in a set that contains the matched beacons. Repeat Step 3 until  $i = M$ . If the number of matched beacons is less than three, increase the threshold by a computed constant, and repeat Step 3 until enough beacons have been identified.

Figure 2-4, page 17, illustrates the results of the identification process.

### 2.1.2 Distance Thresholding

This is a first order improvement upon the previous approach. It may happen that in a given floorplan and beacon map too many beacons are sighted, for instance, exceeding the maximum number of fifteen; or, so much crowding



5. Figure 2-4: The Result of the Identification Process.



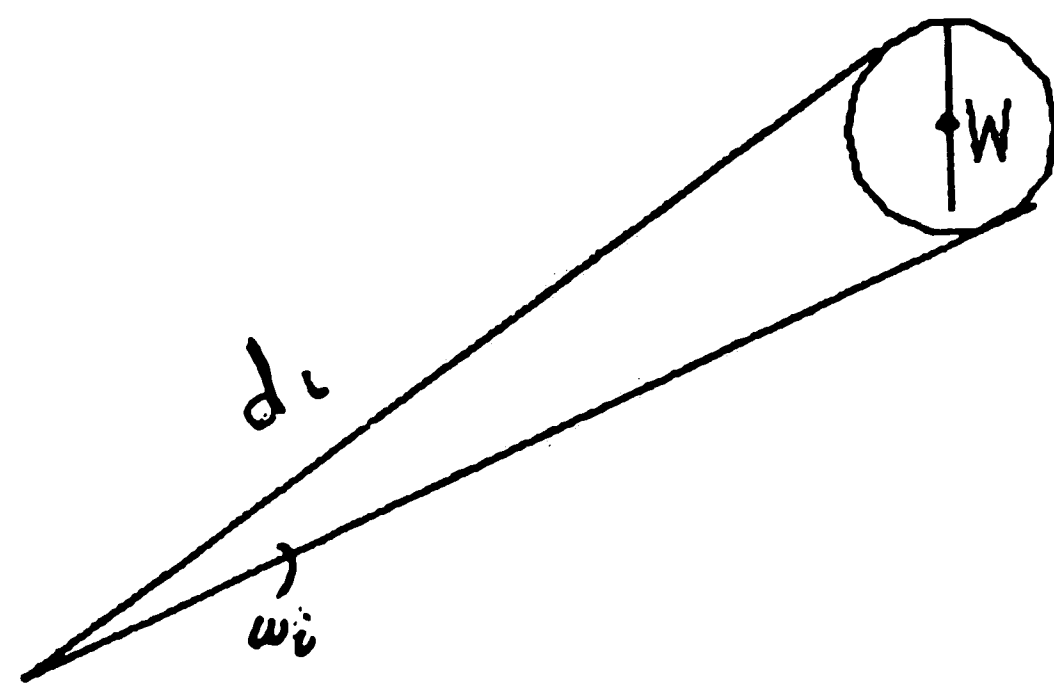
exists that too often more than one sighting occurs in a given angle range plus or minus  $\alpha$ . In this case one can use the measured apparent beacon width in order to eliminate sightings that indicate a beacon beyond a certain threshold,  $T$ , in distance.

The process involves computing  $d_i$ , the sighted beacon distance. Figure 2-5 on page 19 represents the geometry of the beacons made into cylindrical posts. The distance  $d_i$  is inversely proportional to the angle  $w_i$ . Therefore,  $d_i = W \tan^{-1} w_i$ . Once the distance  $d_i$  is computed for  $i = 1 \dots M$ , where  $M$  is the number of beacons sighted, the distance is checked to see if it falls within the chosen threshold  $T$ . If it does fall within, then that beacon is accepted, otherwise it is rejected. Furthermore, if not enough beacons were identified (for example, less than three beacons) one can increase the threshold by a certain constant, repeating the previous process until enough beacons have been identified.

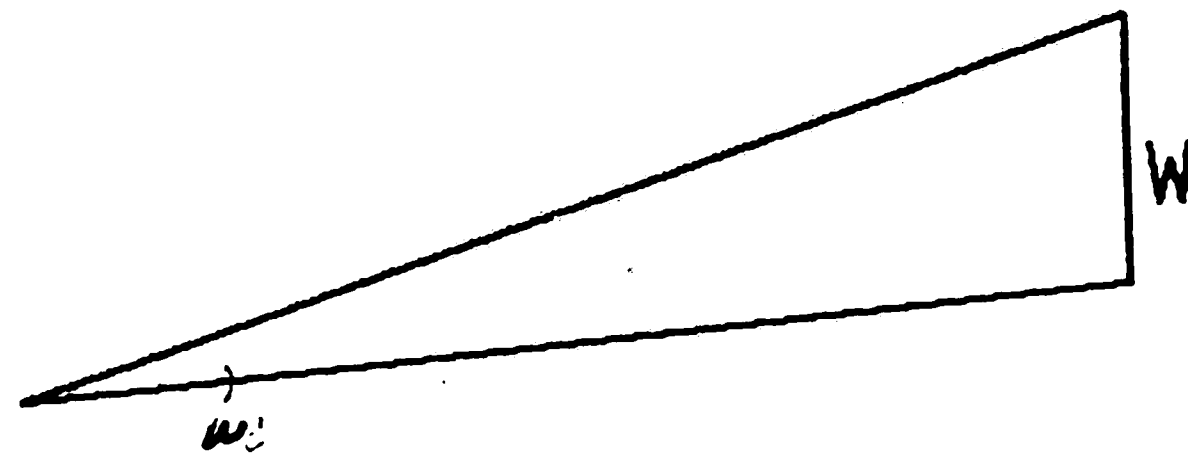
This process will also help in preventing misidentification due to spurious reflections from objects other than beacons, for such reflections have the tendency to be narrow in angle. In addition if two or more beacons fall within the same angular width after distance thresholding then the vehicle should discard those beacons.

### 2.1.3 Identification Via Distance Range

Further safety against misidentification can be incorporated if the measured distance  $d_i$ , as computed above in Section 2.1.2, is compared to the distance  $D_i$ , as computed from the beacon map. Then, if  $|D_i - d_i| > \Delta d$ , the sighting should



CYLINDRICAL BEACON



FLAT BEACON

Figure 2-5: The Computation of Distance.

be rejected.  $\Delta d$  can not be made too small, since the distance measurement via the beacon width is not very accurate. Nevertheless, this procedure reduces the occurrence of further misidentification.

#### 2.1.4 Final Checking

After the computation of position and bearing other checks of the results's validity may be made:

- If in making computations more than one triplet of beacons is used, where each gives an independent position fix, the results should be all within the same vicinity. If a large variance exists, one or more beacons may have been misidentified.
- If only one triplet is used and the computed position or bearing has significantly deviated from the estimated values, misidentification quite likely occurred.

### 2.2 Beacon Selection

Once the beacons have been correctly identified, the beacon selection process is applied to select the most feasible set of triplets for use in calculating the vehicle's position and bearing. One approach is to find all sets of triplets using permutation, calculating the vehicle position from each set of triplets, then averaging the results. This approach can generate very accurate results, but one disadvantage is that the approach will generate a great amount of computer time. And, reducing computation time is a goal within a real time system.

A different approach would be to select one best set of triplets that will give the most accurate results. Through theorizing we intend prove that the best set of beacon triplets is the one in which the spacing of the three adjacent beacons is closest to  $90^\circ$  each. To prove that, let us consider the angle  $\alpha$  be-

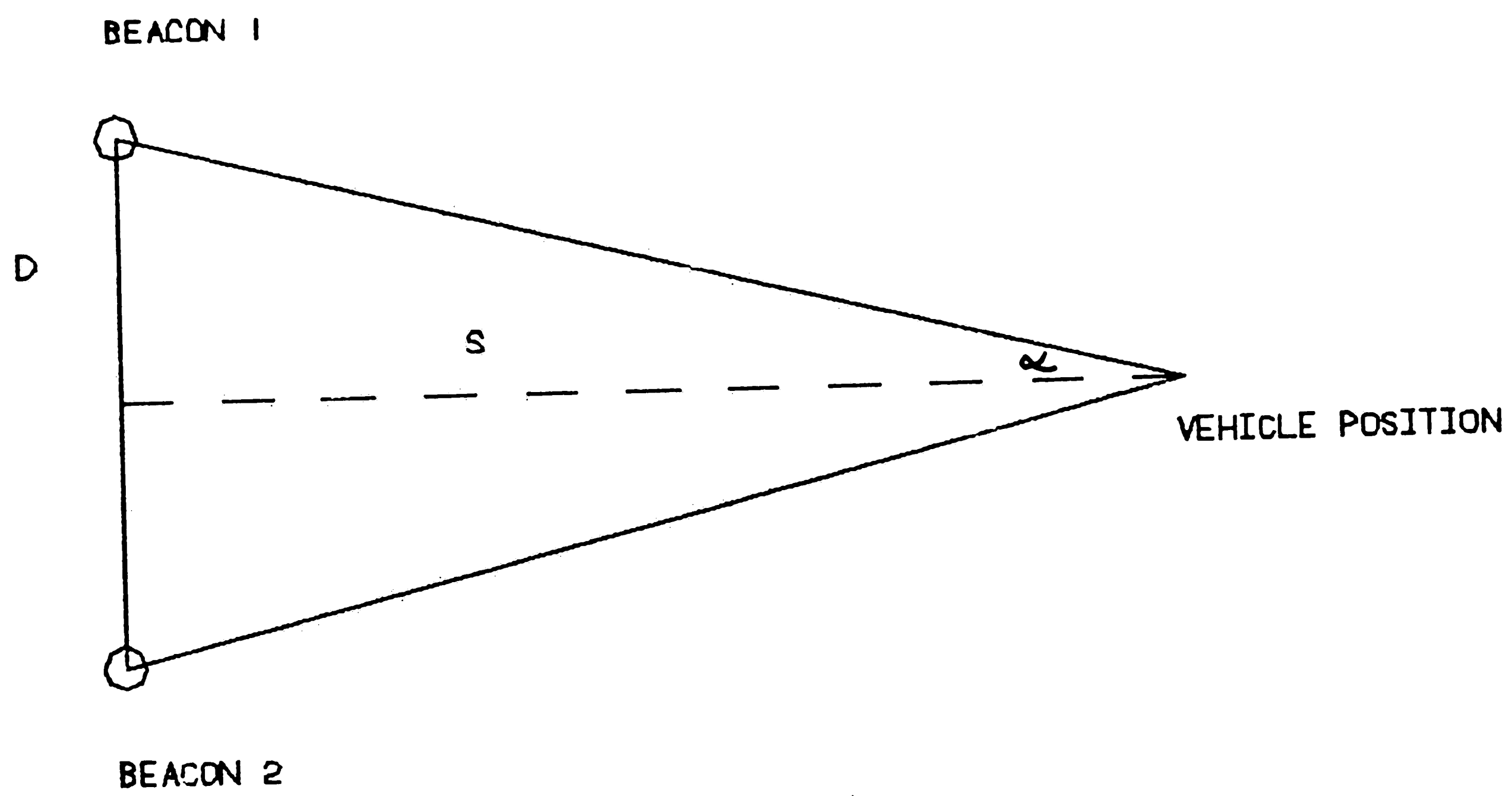


Figure 2-6: Distance Error.

tween beacons one and two as shown on page 21 in Figure 2-6.  $D$  is half the distance between the two beacons, while  $s$  is the distance between the midpoint between beacons one and two and the vehicle position. From this figure one obtains:

$$\frac{D}{s} = \tan \alpha; \text{ and, then}$$

$$s = D \cot \alpha.$$

Taking the derivative of this equation produces

$$\frac{ds}{d\alpha} = -D (\cot \alpha + 1).$$

A distance error  $\Delta s$  may now be stated in terms of an angle error,  $\Delta \alpha$ :

$$\Delta s = -D (\cot \alpha + 1) \Delta \alpha, \text{ and}$$

$$\Delta s = -D \left[ \frac{1}{\sin^2 \alpha} \right] \Delta \alpha.$$

$\Delta s$  will increase when  $\alpha$  decreases, implying that with a larger angle  $\alpha$  the vehicle estimation will be more accurate. The minimum error will occur with an angle  $\alpha$  of  $90^\circ$ .

Obviously, the above geometry represents a special case, however, it serves as an illustration. A general error analysis is difficult to generate since the error will depend on too many parameters, making a reasonable general discussion evasive.

### 2.2.1 Beacon Selection Algorithm

As flowcharted in Figure 2-7, page 24, the beacon selection algorithm is used to select the three most feasible beacons which are then used to calculate the vehicle position. Let us assume that the sighted angles of the matched beacons are  $\Theta_i$ , where  $i = 1 \dots N$ , and  $N$  is the number of beacons matched.

1. If  $N = 3$ , then those are the beacons used to calculate the vehicle position.
2. If  $N > 3$ , then find the difference between two angles of adjacent beacons for every beacon in the set of matched beacons. For example:

$$\text{Dif}[i] = \Theta_{i+1} - \Theta_i, \quad i = 1 \dots N-1, \quad \text{and,}$$

$$\text{Dif}[N] = \Theta_1 - \Theta_N.$$

This is the pair that includes a zero degree in between. If the difference between two angles is less than zero, then the angle difference should be complemented by adding an angle of  $360^\circ$  to it.

3. Find the minimum of the angle differences where

$$\text{Min} = \text{Dif}[k], \quad \text{for some } k, \text{ such that } 1 \leq k \leq N.$$

Next, begin omitting unwanted beacons from the set of matched beacons using the following algorithm:

- a. First case: If  $1 < k < N$ .
  - i. If  $\text{Dif}[k-1] < \text{Dif}[k+1]$ , omit beacon  $k$  by setting  $\text{Dif}[k-1] = \text{Dif}[k-1] + \text{Dif}[k]$ .
  - ii. If  $\text{Dif}[k-1] > \text{Dif}[k+1]$ , omit beacon  $k+1$  by setting  $\text{Dif}[k] = \text{Dif}[k+1] + \text{Dif}[k]$ .
- b. Second case: If  $k = 1$ .
  - i. If  $\text{Dif}[N] < \text{Dif}[k+1]$ , omit beacon  $k$  from the set of matched beacons and set  $\text{Dif}[N] = \text{Dif}[N] + \text{Dif}[k]$ .

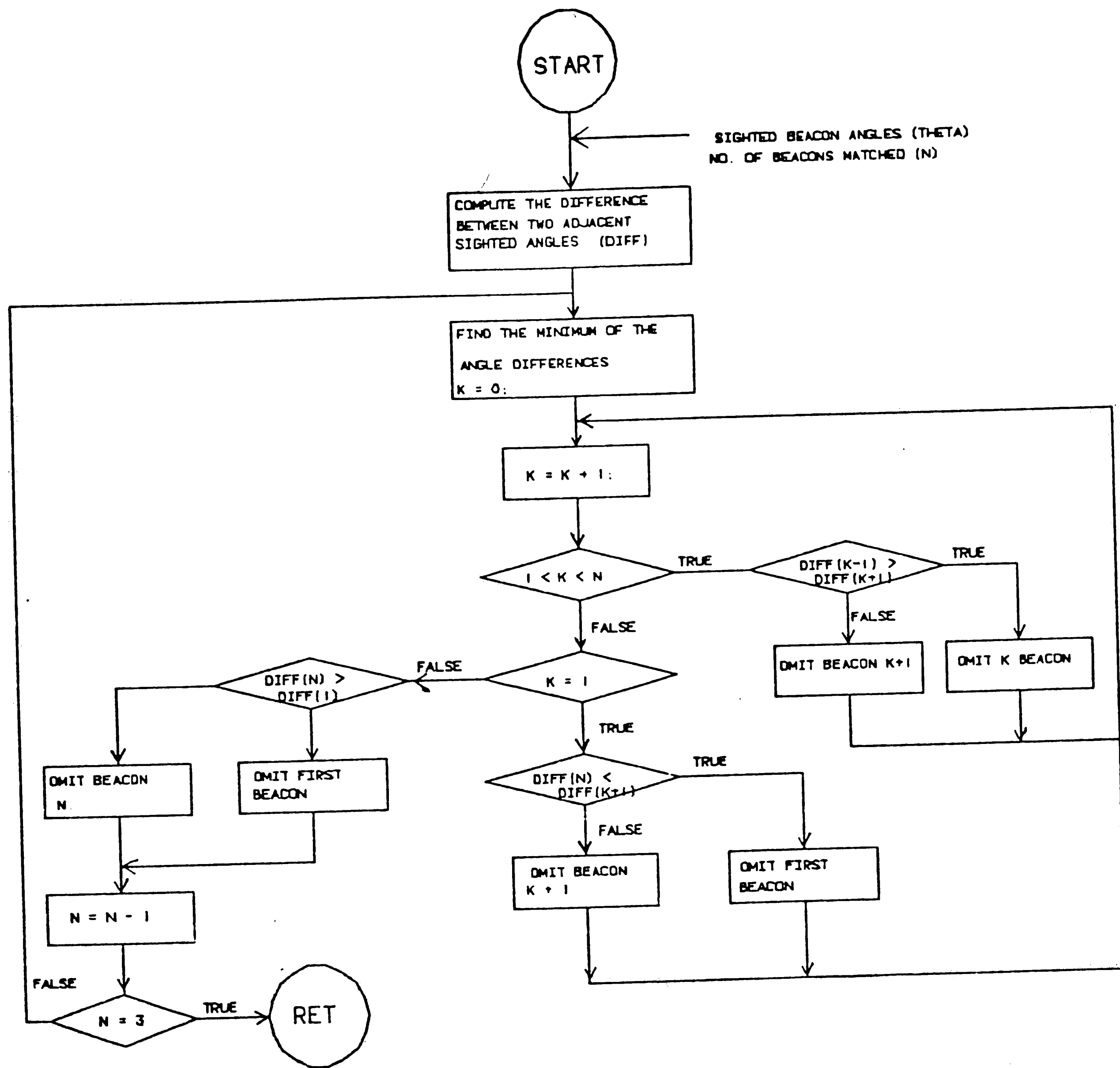


Figure 2-7: The Beacon Selection Algorithm.

ii. If  $\text{Dif}[N] > \text{Dif}[k+1]$ , omit beacon  $k+1$  from the set of matched beacons and set  $\text{Dif}[k] = \text{Dif}[k+1] + \text{Dif}[k]$ .

c. Third case: If  $k = N$ .

i. If  $\text{Dif}[N-1] > \text{Dif}[1]$ , omit beacon 1 from the set of matched beacons and set  $\text{Dif}[N-1] = \text{Dif}[N] + \text{Dif}[1]$ .

ii. If  $\text{Dif}[N-1] < \text{Dif}[1]$ , omit beacon  $N$  from the set of matched beacons and set  $\text{Dif}[N-1] = \text{Dif}[N-1] + \text{Dif}[N]$ .

Decrement the number of matched beacons by 1. If  $N = 3$ , then stop, otherwise go to Step 3.

This algorithm has been very successful in choosing the proper beacons.

## 2.3 The Mathematical Theory For Obtaining The Vehicle Position

After appropriate beacons are identified and selected, the vehicle position and bearing may be computed. To compute the vehicle position  $X_0, Y_0$ , let the cartesian coordinates of the three selected beacons be denoted by  $(X_1, Y_1)$ ,  $(X_2, Y_2)$ ,  $(X_3, Y_3)$ , and let the angle between beacons 1 and 2 be  $\alpha_1$ , the angle between beacons 2 and 3 be  $\alpha_2$ , and the angle between beacons 1 and 3 be  $\alpha_3$ .

### 2.3.1 Vehicle Position

Let us assume a circle,  $A$ , is drawn through the points of beacons 1 and 2, and the as yet unknown vehicle position  $X_0, Y_0$ . Circle  $B$  is drawn through the points of beacons 2 and 3 and  $X_0, Y_0$ . Circle  $C$  is drawn through the points of beacons 1 and 3 and  $X_0, Y_0$  as shown in Figure 2-8, page 26.

Each of the circles is determined through a theorem of geometry stating that from each point on a circle, say  $A$ , the distance between beacons 1 and 2 appears under the same angle. This angle is  $\alpha_1$  and serves to find the center



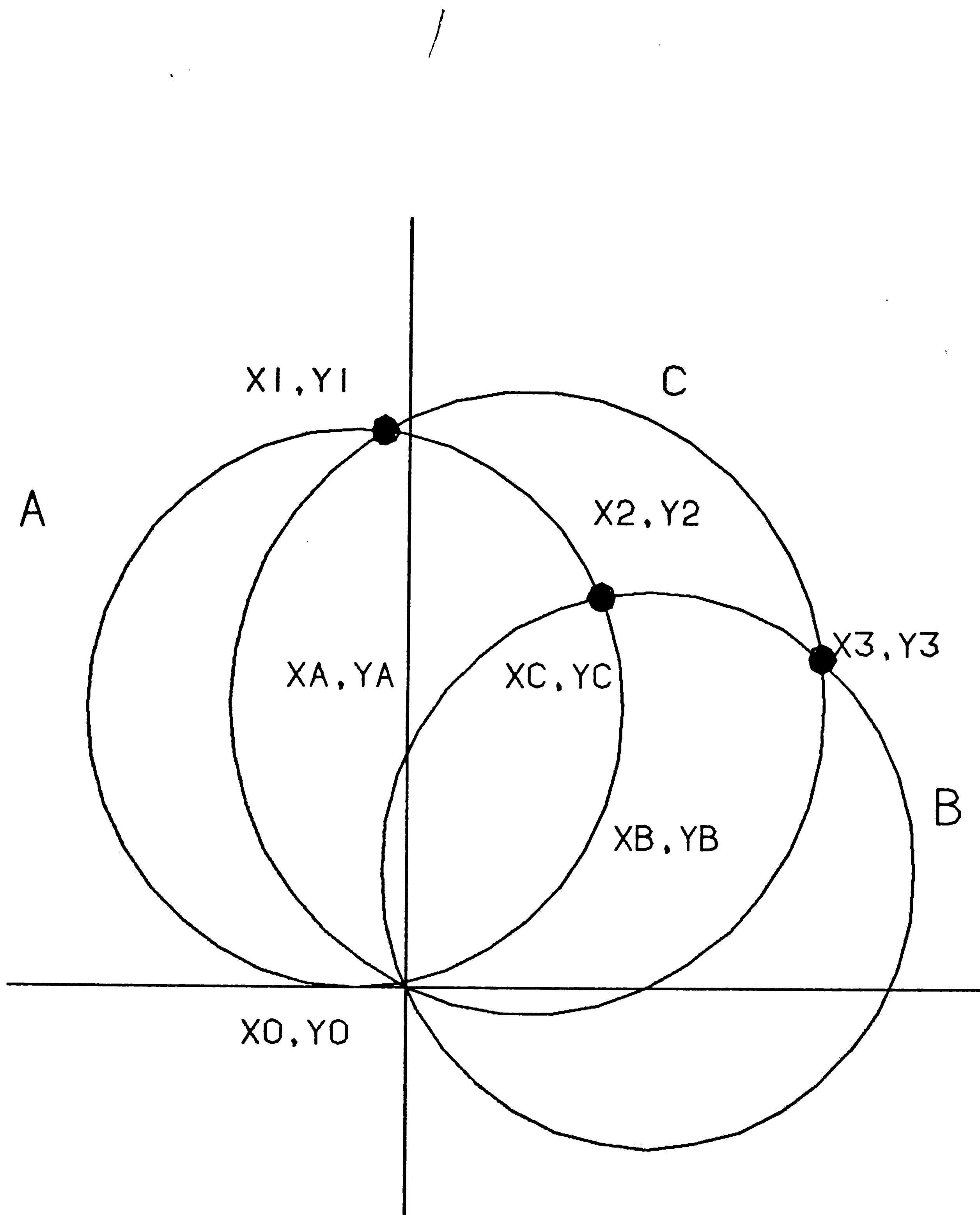


Figure 2-8: The Three Circles A, B and C.

of circle *A*. Intersections between the circles provide the vehicle coordinates. The centers of circles *A*, *B* and *C* are calculated as shown in the following paragraphs.

As in Figure 2-9, Circle *A* will be an example for the calculating the center of a circle given the coordinates of the beacons that fall on the same circle. Let line *a* be the line between beacons 1 and 2, line *b* be the line drawn through the center of the circle  $(X_c, Y_c)$ , and let  $X_m, Y_m$  be the middle of line *a* where

$$X_m = \frac{X_1 + X_2}{2}, \quad Y_m = \frac{Y_1 + Y_2}{2}.$$

Let *l* be half the distance between beacons 1 and 2, where

$$l = \frac{1}{2} \sqrt{[X_2 - X_1]^2 + [Y_2 - Y_1]^2}. \quad (2.1)$$

The slope of line *a* is

$$m_1 = \frac{Y_2 - Y_1}{X_2 - X_1}, \quad (2.2)$$

and, since line *a* is perpendicular to line *b*, the slope of line *b* is

$$m_2 = \frac{-1}{m_1}, \quad \text{or} \\ m_2 = \frac{X_2 - X_1}{Y_1 - Y_2}. \quad (2.3)$$

The equation of line *b* through  $(X_c, Y_c)$  is

$$Y_c = Y_m + \frac{X_1 - X_2}{Y_2 - Y_1} (X_c - X_m). \quad (2.4)$$

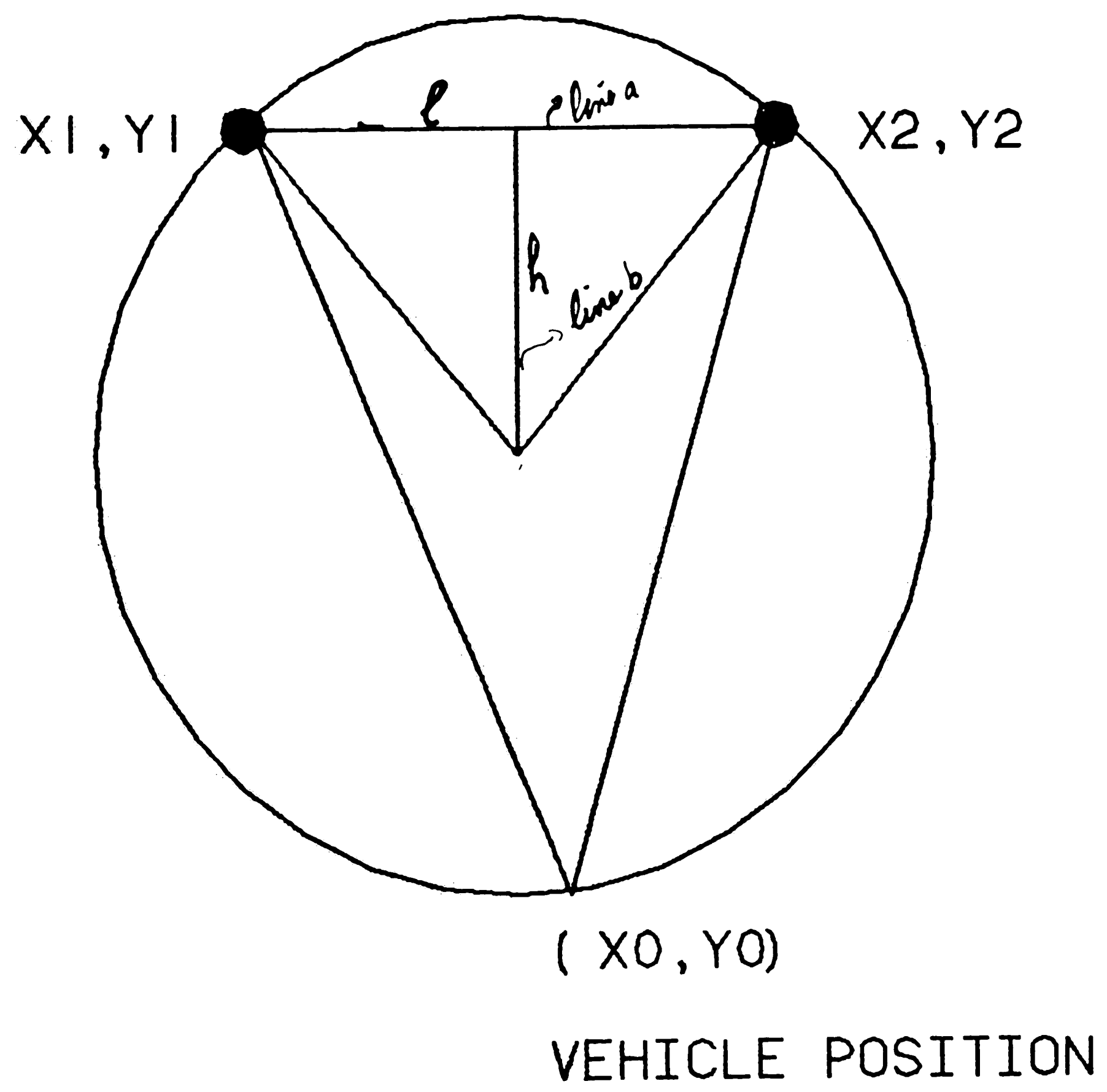


Figure 2-9: The Calculation of the Center of a Circle.

And, referring to Figure 2-9,

$$h = l \cot \alpha_1 = \sqrt{[X_c - X_m]^2 + [Y_c - Y_m]^2}. \quad (2.5)$$

From Equation 2.5 the following is generated:

$$(X_c - X_m)^2 = (l \cot \alpha)^2 - (Y_c - Y_m)^2. \quad (2.6)$$

Substituting  $(X_c - X_m)$  into Equation 2.4 results in

$$Y_c = Y_m \pm (X_1 - X_2) \left[ \frac{\cot \alpha_1}{2} \right].$$

From Equation 2.6 we can generate the following:

$$(Y_c - Y_m)^2 = (l \cot \alpha_1)^2 - (X_c - X_m)^2. \quad (2.7)$$

Then, substitute  $(Y_c - Y_m)$  in Equation 2.4 so that

$$X_c = X_m \pm (Y_1 - Y_2) \left[ \frac{\cot \alpha_1}{2} \right].$$

And, having substituted the value of  $X_m$ , the center of circle A is

$$X_A = \frac{X_1 + X_2}{2} + (Y_1 - Y_2) \left[ \frac{\cot \alpha_1}{2} \right], \quad Y_A = \frac{Y_1 + Y_2}{2} - (X_1 - X_2) \left[ \frac{\cot \alpha_1}{2} \right].$$

Similarly, the center of circle B is

$$X_B = \frac{X_2 + X_3}{2} + (Y_2 - Y_3) \left[ \frac{\cot \alpha_2}{2} \right], \quad Y_B = \frac{Y_2 + Y_3}{2} - (X_2 - X_3) \left[ \frac{\cot \alpha_2}{2} \right];$$

and the center of circle C is

$$X_C = \frac{X_1 + X_3}{2} + (Y_1 - Y_3) \left[ \frac{\cot \alpha_3}{2} \right], \quad Y_C = \frac{Y_1 + Y_3}{2} - (X_1 - X_3) \left[ \frac{\cot \alpha_3}{2} \right].$$

Having found the centers of the circles, one next calculates the intersection points in the manner shown in Figure 2-10, page 31. Intersections are computed between every pair of unique circles and each intersection point serves as the computed vehicle position. The average of the three intersection points provides an accurate vehicle position.

Let us take the two circles  $A$  and  $B$ , where  $X_{01}, Y_{01}$  and  $X_{02}, Y_{02}$  are the center of the circles respectively—see Figure 2-10, page 31. And, letting the common beacon coordinates between the two circles be  $X_c, Y_c$ , and  $X_m, Y_m$  be the midpoint between the vehicle position and the point  $X_c, Y_c$ , then the slope of line  $a$  is

$$m = \frac{Y_{02} - Y_{01}}{X_{02} - X_{01}};$$

and, the slope of line  $b$  is

$$m' = \frac{-1}{m} = \frac{X_{02} - X_{01}}{Y_{01} - Y_{02}}.$$

Thus, the equation of line  $a$  through points  $(X_m, Y_m)$  and  $(X_{01}, Y_{01})$  is

$$Y_m - Y_{01} = (X_m - X_{01}) m,$$

and the equation of line  $b$  through  $(X_m, Y_m)$  and the common point  $(X_c, Y_c)$  is

$$Y_m - Y_c = (X_m - X_c) m'. \quad (2.8)$$

With these two equations one calculates the intersection of lines  $a$  and  $b$ :

$$(X_m - X_{01}) m + Y_{01} = (X_m - X_c) m' + Y_c;$$

$$(X_m - X_{01}) m = (X_m - X_c) m' + Y_c - Y_{01}.$$

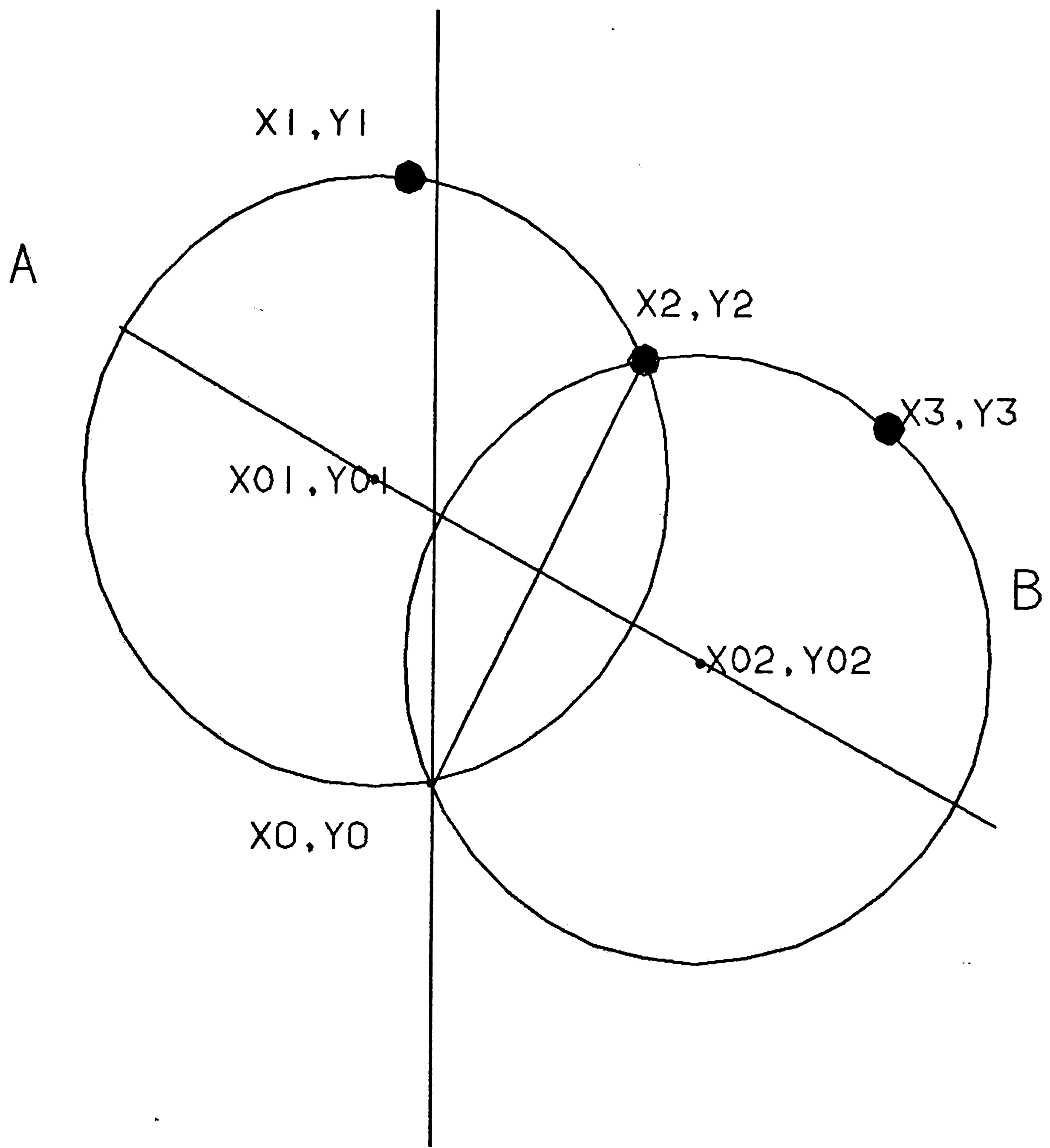


Figure 2-10: Computation of the Vehicle's Point.

Thus,

$$X_m = \frac{Y_c - Y_{01} - X_c m' + X_{01} m}{m - m'}. \quad (2.9)$$

As shown in Figure 2-10, one may find the vehicle position  $(X_0, Y_0)$ , since

$$X_m = \frac{X_0 + X_c}{2}.$$

As a result,

$$X_0 = 2X_m - X_c. \quad (2.10)$$

And, one substitutes the values of  $(X_m, Y_m)$  in Equation 2.10 so that the computed vehicle position is

$$X_0 = 2 \left[ \frac{Y_c - Y_{01} - Y_c m' + X_{01} m}{m - m'} \right] - X_c.$$

Similarly,

$$Y_m = \frac{Y_0 + Y_c}{2},$$

and as a result,

$$Y_0 = 2Y_m - Y_c. \quad (2.11)$$

By substituting the value of  $Y_m$  from Equation 2.8 into the resulting  $Y$  coordinate of the vehicle's position in Equation 2.11 one gets

$$Y_0 = 2(X_m - X_c)m' + Y_c.$$

Assuming that the vehicle positions obtained using circles  $A$  and  $B$ ,  $B$  and  $C$ , and  $A$  and  $C$  are  $(X_{ab}, Y_{ab}), (X_{bc}, Y_{bc}), (X_{ac}, Y_{ac})$ , respectively, then the averaged vehicle position will be

$$X_0 = \frac{X_{ab} + X_{bc} + X_{ac}}{3}, \quad Y_0 = \frac{Y_{ab} + Y_{bc} + Y_{ac}}{3}.$$

Problems occur if all three beacons and the vehicle lie on the same circle, then it will be impossible to calculate the vehicle position. Near such a situation, calculations will produce large errors, and thus safeguards have to be built into the program to identify such an event.

### 2.3.2 Vehicle Bearing

Figure 2-11, page 34, shows the vehicle bearing  $\phi$ , the beacon sighted angle  $\theta$ , and the angle  $\gamma$ , as referred to the  $x$ -axis. Given both the beacon coordinate  $(X_1, Y_1)$  and the exact vehicle coordinates  $(X_0, Y_0)$  from the previous section one may compute the following:

$$\tan \gamma = \frac{Y_1 - Y_0}{X_1 - X_0}; \text{ and thus}$$

$$\gamma = \tan^{-1} \frac{Y_1 - Y_0}{X_1 - X_0}.$$

Since  $\theta$  is known and  $\gamma$  is computed, the vehicle bearing is  $\phi = \gamma - \theta$ .



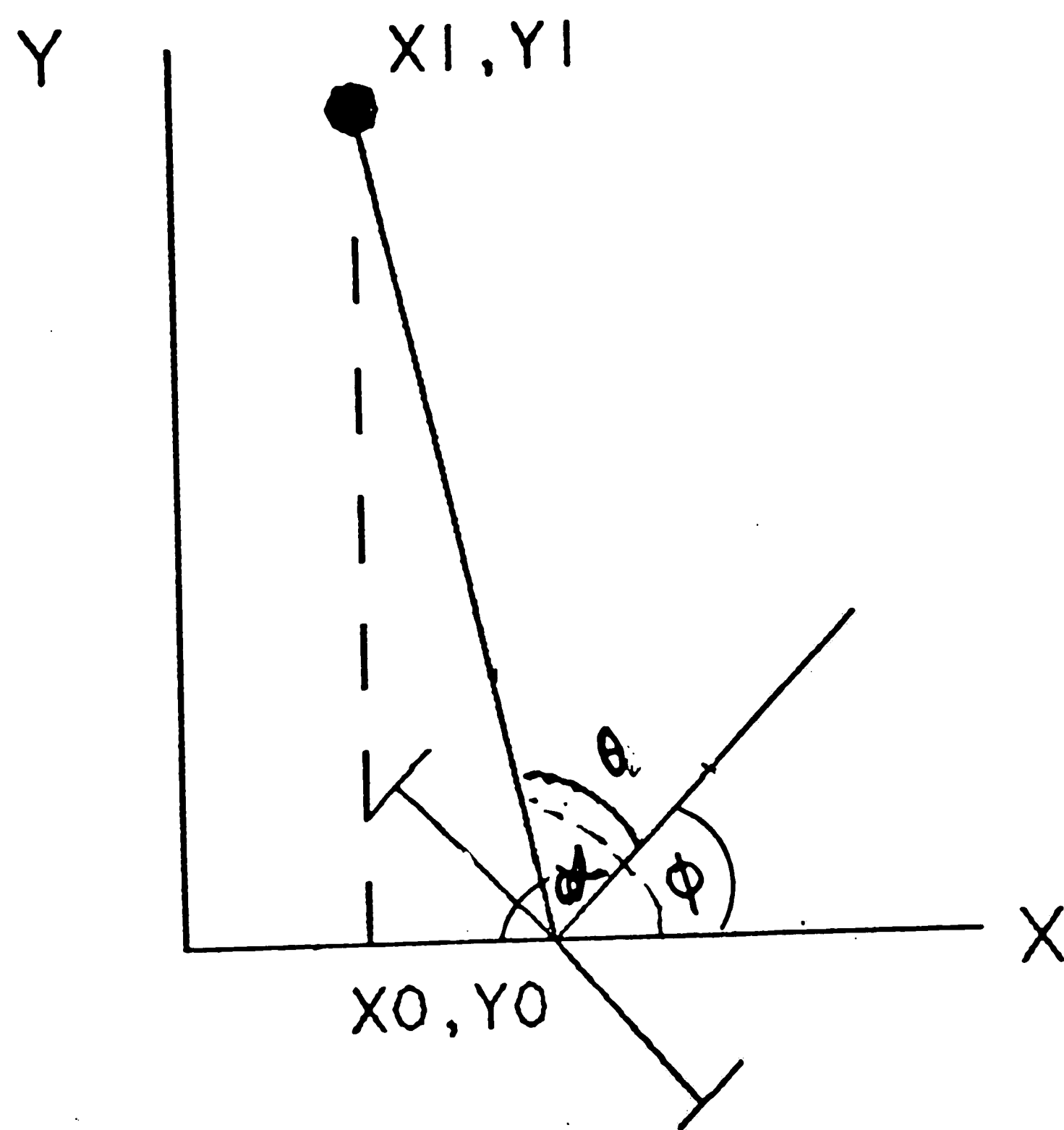


Figure 2-11: Vehicle Bearing.

## 2.4 The Optical Navigation System Software

This section describes the subroutines written for the optical system. Each coming section will describe each subroutine what it does and, and what are the inputs outputs for that particular routine a flow chart will be associated with each routine to explain the algorithms used. See Appendix A for the code programs used for the Cyclopion.

### 2.4.1 The gonimeter procedure

This procedure goes through the following steps

1. Checks if the vehicle is in reverse or forward mode, to determine the vehicle starting point and bearing.
2. Call "Identify\_beacons" procedure to identify the right beacons.
3. Then calls the procedure "Compute\_Vehicle\_Pos" to compute the accurate vehicle position.
4. Finally, computes the vehicle bearing.

#### Inputs:

- |                 |   |
|-----------------|---|
| 1. THETA(I)     | :Sighted beacon angles $\theta_i$ ,<br>For $i = 1 \dots N$ , where $N$ is<br>the Number of sighted beacons. |
| 2. (XP, YP)     | :The Beacons coordinate map   |
| 3. (APRX, APRY) | :The approximate vehicle position   |
| 4. BEA          | :The prescribed bearing   |

#### Outputs:

- |                           |                                |
|---------------------------|--------------------------------|
| 1. (VEH_POS_X, VEH_POS_Y) | :The accurate vehicle position |
| 2. BEARING                | :The computed Bearing          |

Figure 2-2 on page 13 describes the goniometer procedure.

### 2.4.2 Identify beacons Procedure

This procedure identifies the correct beacons using via angular range algorithm described in section 2.1 on page 12. Figure 2-12 on page 37 represents this procedure.

1. Computes the expected beacon angle by calling the "Expected\_angle" routine
2. Identifies the right beacons using the method described in section 2.1.
3. Once a beacons has been identified the corresponding beacon coordinate and angles are stored in a array of matched beacons.

#### Inputs:

1. THETA(I) :Sighted beacon angles  $\theta_i$   
for  $i = 1...N$ , where  $N$   
is the number of sighted beacons.
2. (XP, YP) . :The Beacon coordinates.

#### Outputs:

1. (XPOINTS, YPOINTS) :An array of matched beacon  
coordinates
2. THEATS. :The matched beacon sightings

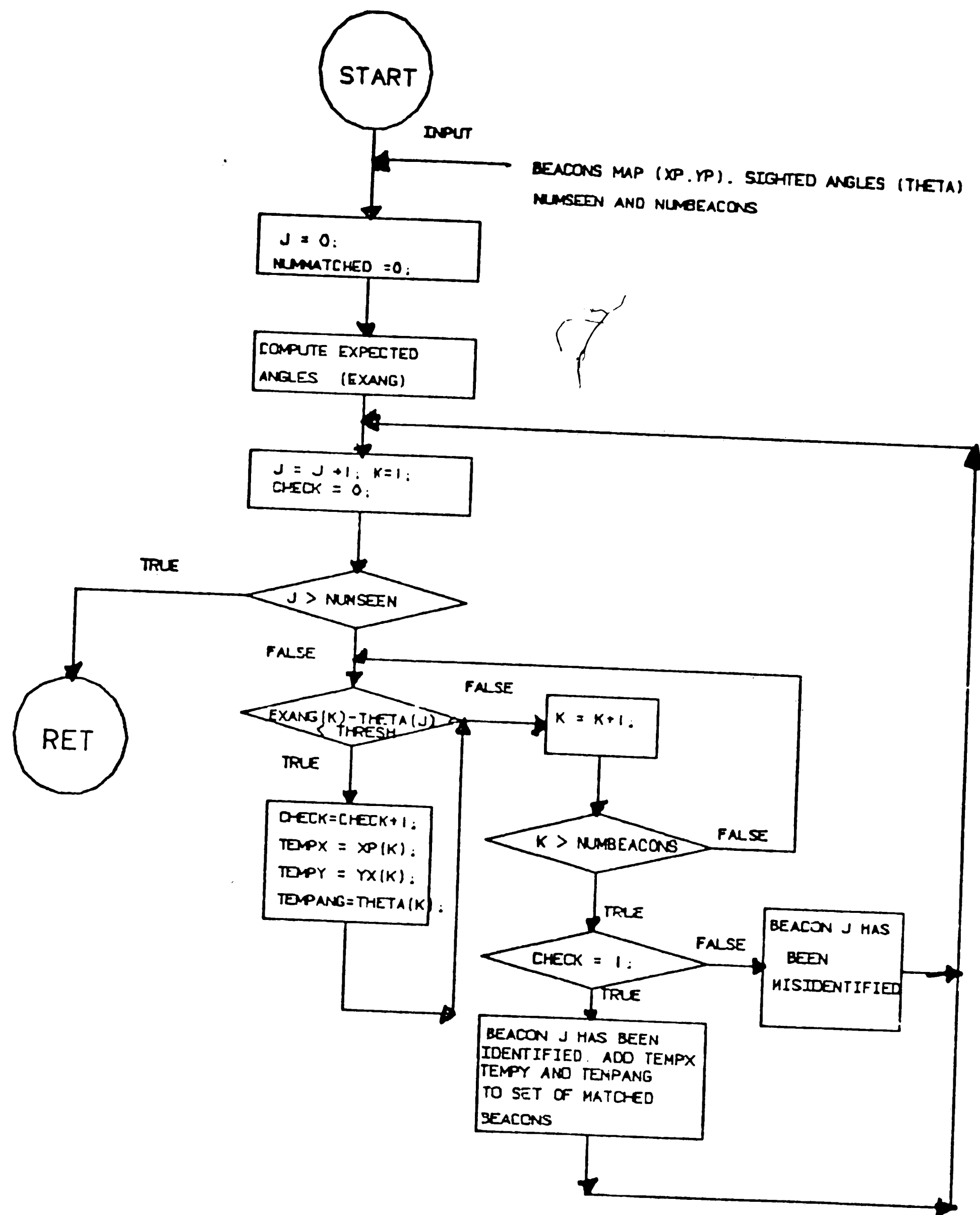


Figure 2-12: Identify Beacons Procedure

### 2.4.3 The Expected Angle Procedure

The procedure computes the expected beacon angles of all beacons in the map relative to the vehicle position. Figure 2-13 on page 38 display the flow of the computation, of the expected angle.

#### Inputs:

1. (APRX, APRY). :The approximate vehicle positions.
2. BEA. :The prescribed bearing.
3. (XP, YP). :The actual coordinates of the beacon

#### Outputs:

1. EXANG(I) :The expected beacon angle  $\psi_i$ .

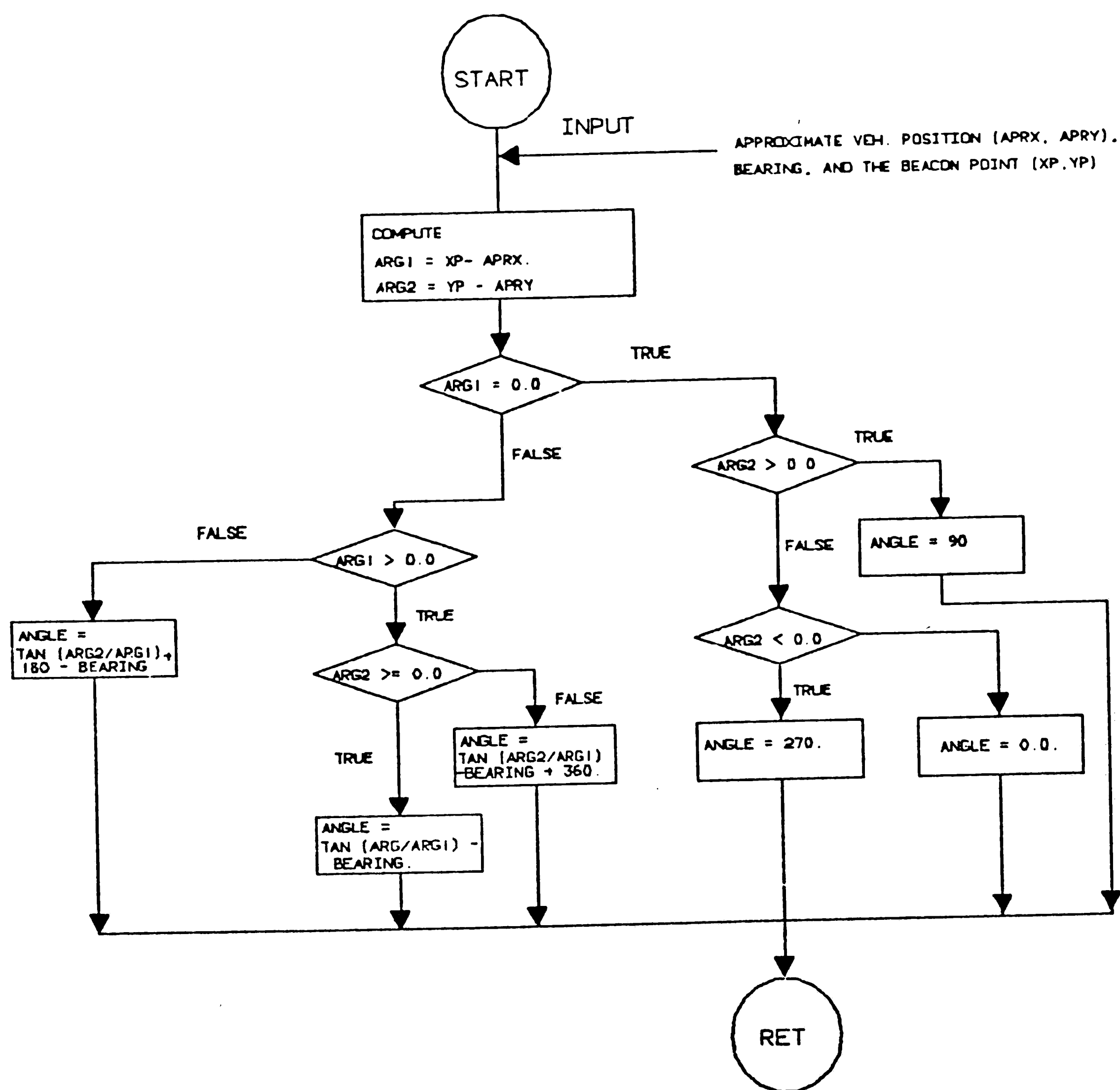


Figure 2-13: Expected Angle Procedure

#### 2.4.4 Compute Vehicle Position procedure

The Following steps describe the flow of this procedure

1. Check if the number of identified beacons are equal to three, if so compute the vehicle position, using those beacons.
2. If not, the "Triplet procedure is called to select the most feasible triplet's using the beacons selection algorithm described in section 2.2.
2. Procedure "Suffpoints" is called to shuffle the beacon points if needed.
3. Then the Procedure "Calcpoints" is called to determine the accurate vehicle position.

##### Inputs:

1. (XPOINTS, YPOINTS) :An array of matched beacon coordinates.
2. THEATS. :The matched beacon sightings.
3. NUMMATCHED. :The number of matched beacons.

##### Outputs:

1. (VEH\_POS\_X, VEH\_POS\_Y) :The accurate vehicle position .

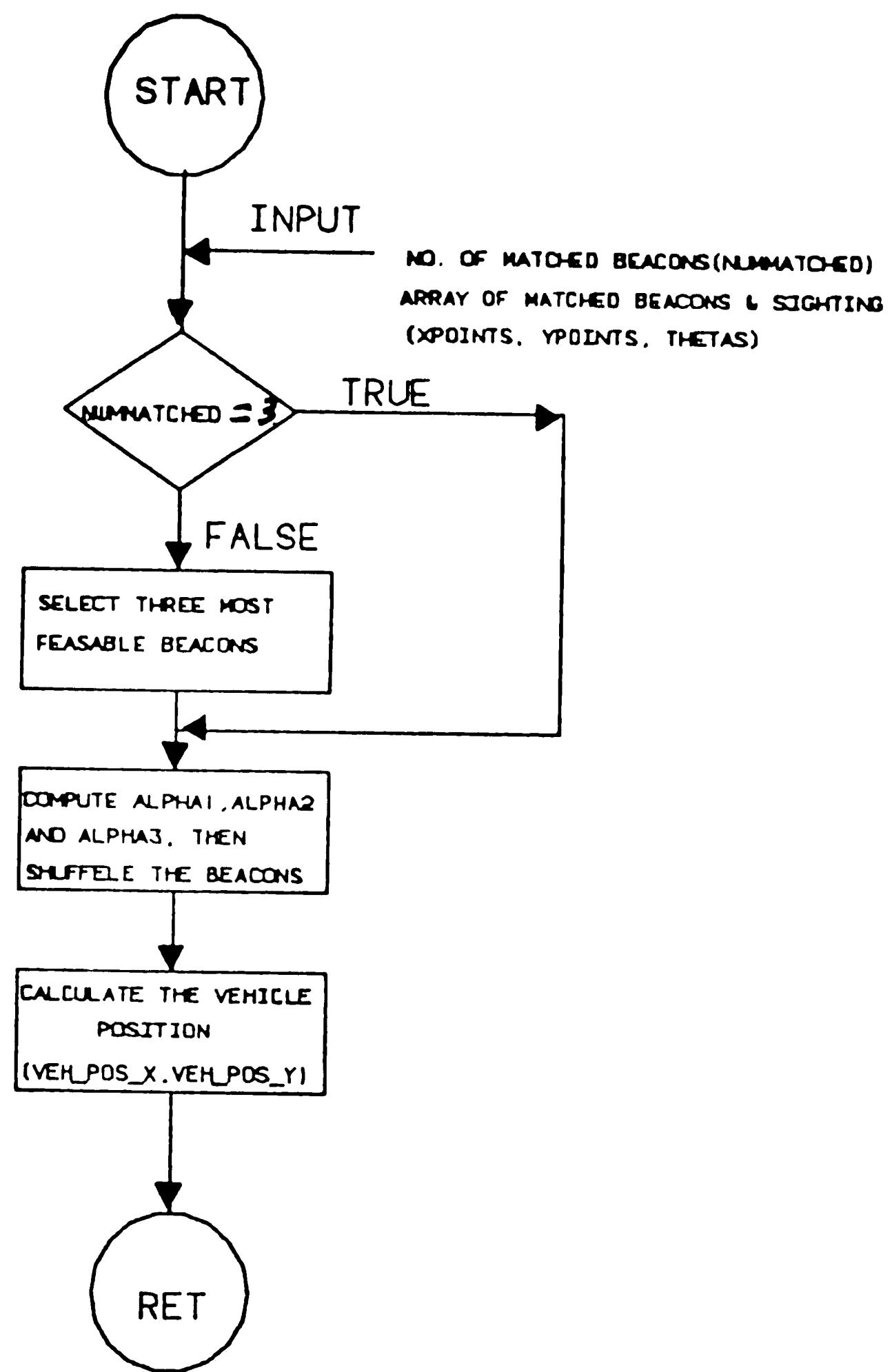


Figure 2-14: Compute Vehicle Position Procedure

### 2.4.5 Calculate vehicle points procedure

The procedure, first, computes the the center of circle, A, Then the center of circle, B, finally the center of circle C. The next step, it computes the coordinates of the vehicle using circles A and B, A and C, and B and C. then averages the three vehicle points to produce the accurate vehicle position.

#### Inputs:

1.  $(X_1, Y_1)$ ,  $(X_2, Y_2)$  and  $(X_3, Y_3)$  :The coordinates of the three selected beacons

#### Outputs:

1.  $(VEH\_POS\_X, VEH\_POS\_Y)$  :The accurate vehicle position .

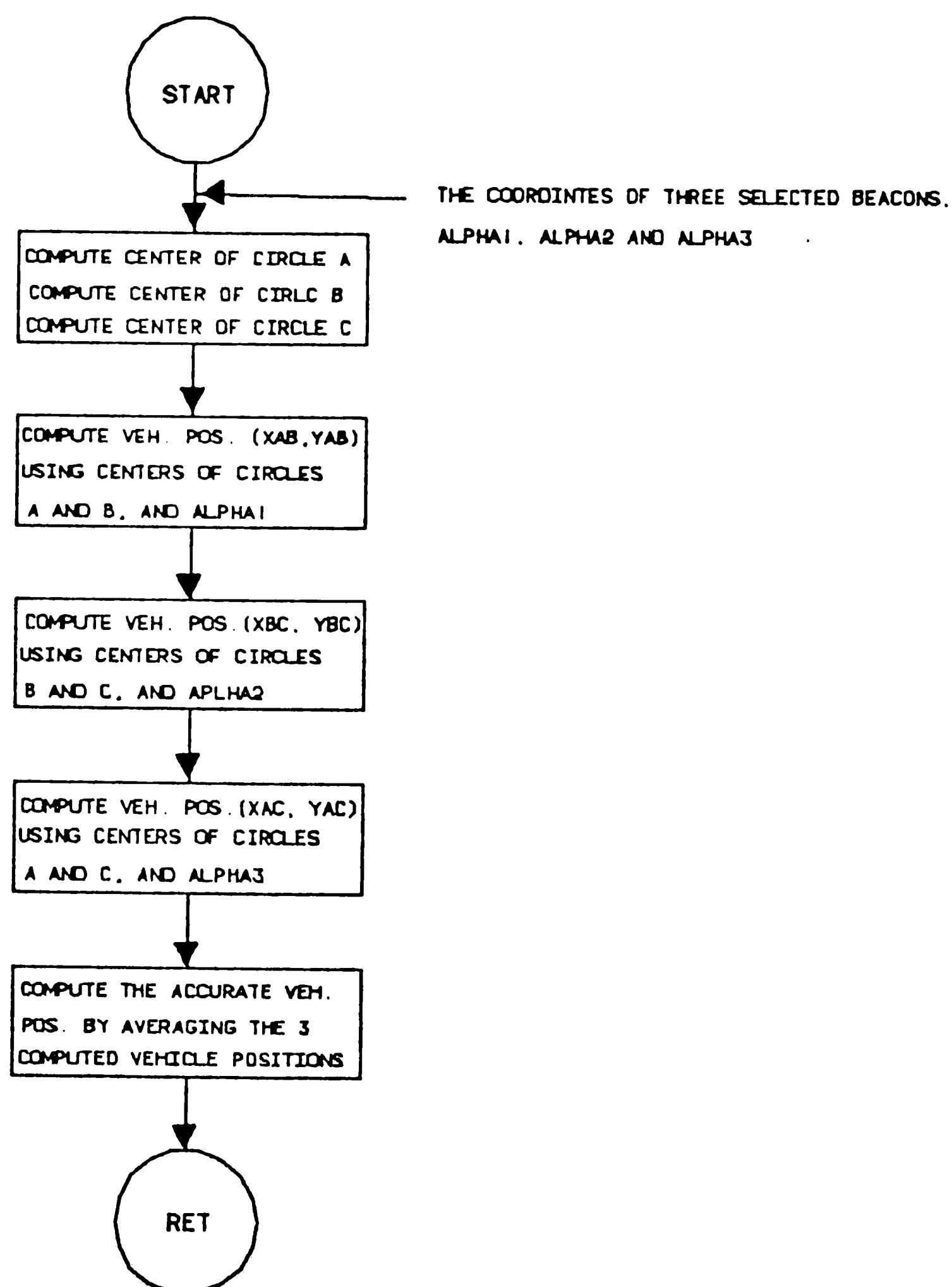


Figure 2-15: Compute Vehicle Points Procedure



#### 2.4.6 Find Center Procedure

The procedure computes the center of a given circle, section 2.3 on page 256. describe the algorithms of that procedure

##### Inputs:

1. (PX1, PY1), (PX2, PY2). :The coordinates of the two beacons that pass through the circle.
2. ALPHA :The angle  $\alpha$  between the two beacons.

##### Outputs:

1. (CENTERX, CENTERY) :The center of the circle

#### 2.4.7 Find Vehicle Coordinates Procedure

The procedure computes the vehicle coordinates.

##### Inputs:

1. (CX1, CY1), (CX2, CY2) :The center coordinates of the two circles
2. (XC, YC) :The common beacon coordinates that passes through the two circles.

##### Outputs:

1. (XCORD, YCORD). :The computed vehicle coordinates

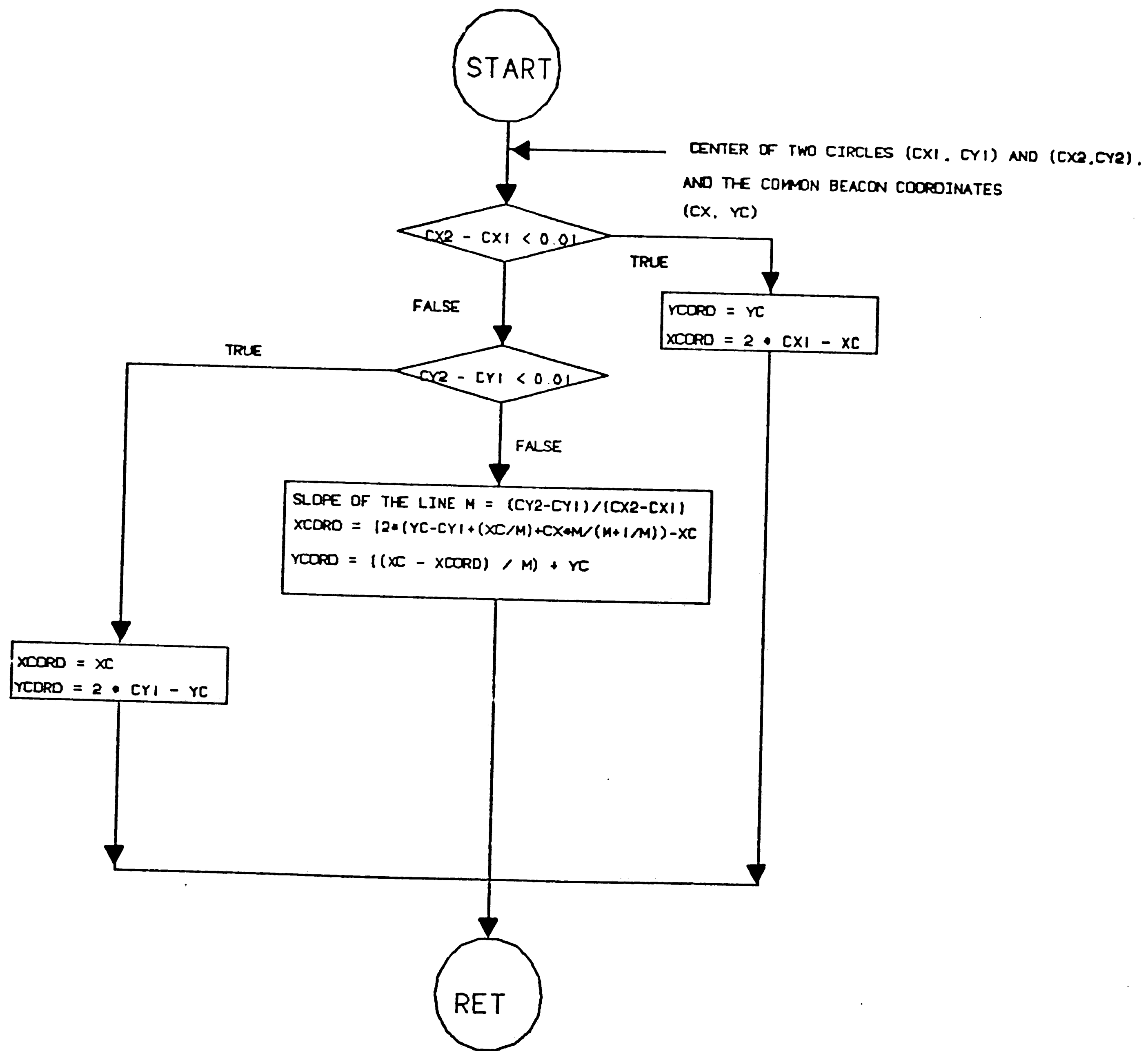


Figure 2-16: Find Vehicle Coordinates Procedure

### 2.4.8 Beacon Selection Procedure

This procedure selects the right sets of beacons to compute, the vehicle position, the algorithm was describes in section 2.2 on page 20

#### inputs:

1. (XPOINTS, YPOINTS). :An array of matched beacons coordinates.
2. THETAS :The matched beacon sightings

#### Outputs:

1.  $(X_1, Y_1)$ ,  $(X_2, Y_2)$ .  
and  $(X_3, Y_3)$  :The coordinates of the three beacons selected

### 2.4.9 Shuffpoints Procedure

This procedure ALPHA1, APLHA2, AND ALPHA3 and rearranges the numbering of the tree selected beacons according to angle difference between them.

#### inputs:

1.  $(X_1, Y_1)$ ,  $(X_2, Y_2)$   
 $(X_3, Y_3)$  :The coordinates of the three beacons selected
2. THEATS. :The matched beacon sightings

#### Outputs:

1.  $(X_1, Y_1)$ ,  $(X_2, Y_2)$   
 $(X_3, Y_3)$  :The coordinates of the three shuffeled beacons
2. ALPHA1, APLHA2, ALPHA3. :The angles  $\alpha_1, \alpha_2, \alpha_3$

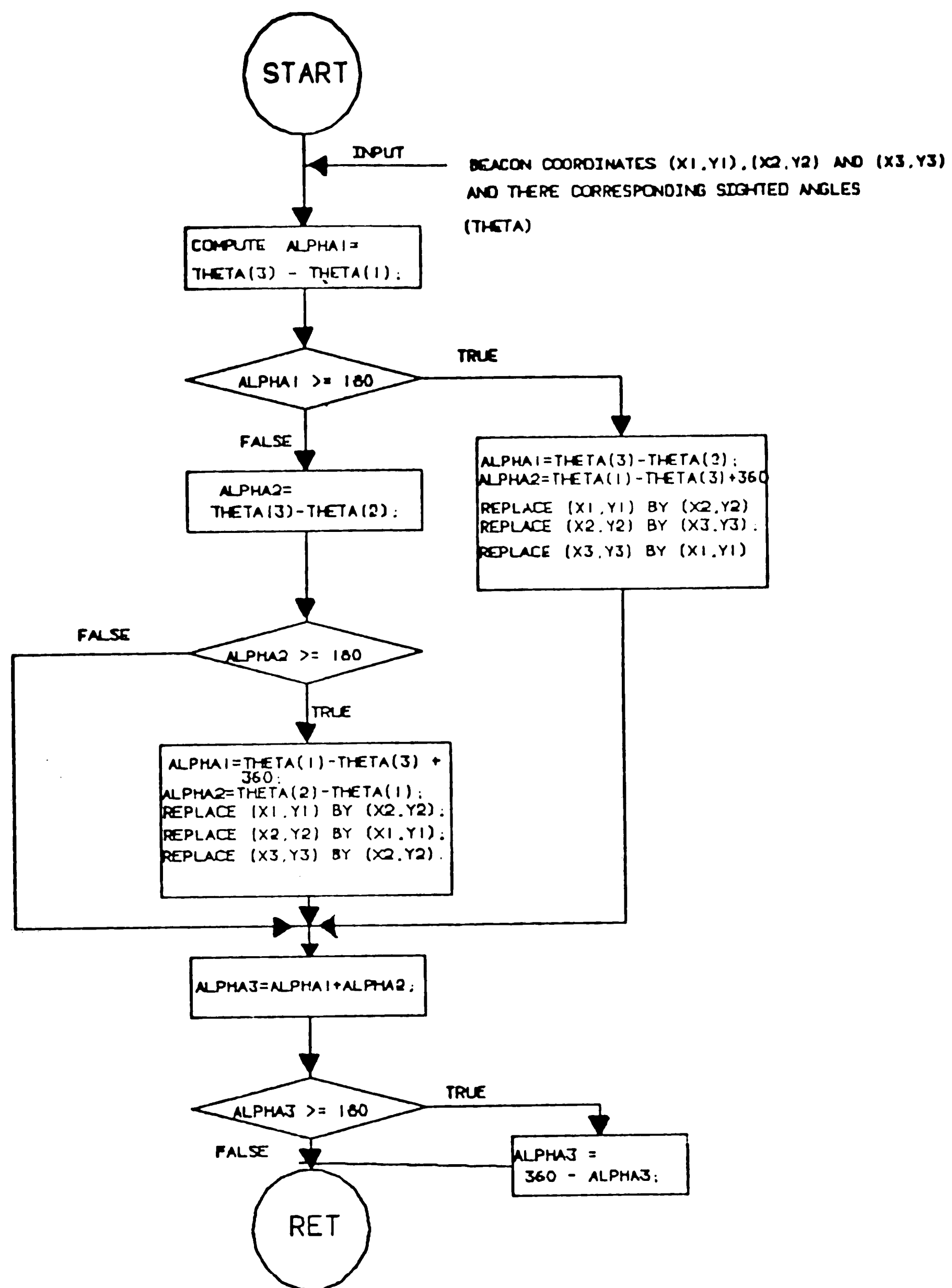


Figure 2-17: Shuffpoints Procedure

## Chapter 3

# The Ground Navigation System

The ground navigation system is the second navigation system for this vehicle, and is needed for three reasons: 1) the vehicle might not be able to calculate its position using the optical system when the beacons are not visible due to dust accumulation or objects in the factory environment, 2) the optical navigation system can determine the vehicle location only once every half second, and therefore the ground navigation will have to be used to locate the vehicle position between fixes. The ground navigation system computes the position every 0.005 of a second, which for all practical purpose is continuous. 3) frequent computation of the vehicle position is needed, the ground navigator board (processor) was introduced to reduce the burden on the 86/85 computer.

The constituents of the ground navigation system are a gyroscope that measures the angle  $\Delta\phi$ , a wheel encoder (odometer) for measuring the distance  $\Delta l$  traveled by the front wheel of the vehicle, and a ground navigation processor that uses the angle  $\Delta\phi$  and  $\Delta l$  to calculate the vehicle position and bearing.

Although, the system allows the vehicle to travel a long distance without relying on the optical system, an error may result in determining the vehicle position and bearing, therefore, the position coordinates of the ground navigation system are updated and corrected using the actual position and bearing calculated by the optical system.

The error originates from two devices, the gyroscope and the wheel encoder odometer. The effect of the earth's rotation causes the gyroscope to drift over time (approximately 1/4 of a degree per minute). In addition, the resolution of the gyroscope is of one quarter of a degree and therefore, the vehicle bearing is measured by units of quarter of a degree. The effect of tire wear or wheel slippage will effect the accuracy of the wheel encoder (odometer) in determining the distance traveled. Only a very small error will be due to the resolution of the wheel encoder which issues a pulse for every 0.01 inches of the front wheel travel.

### 3.1 The Ground Navigation System

As the vehicle travels, at regular distance intervals,  $\Delta l$ , a pulse is generated from the drive board, and the vehicle position is recalculated. Changes in  $X$ - $Y$  coordinates are computed and added to the previous  $X_p$ - $Y_p$  coordinates to provide the current vehicle position. At full speed the positions are recalculated at a rate of once every 0.005 of a second. If an accurate position is determined by the optical system, the ground navigation board corrects the current ground navigator reading.

### 3.2 The Mathematical Theory For Ground Navigation System

The vehicle travel geometry is shown in Figure 3-1 on page 48, where  $\Delta s$  is the distance traveled and  $\Delta\phi$  is the change of angle for  $\phi_1$  to  $\phi_2$ . Vehicle coordinates refer to the point,  $V$ , the center of the rear axle.

$\Delta l$  is the increment of the front wheel travel. This is generally not the distance traveled by the vehicle, except if the vehicle is traveling on a straight line. For each  $\Delta l$ , increments of  $\Delta x$  and  $\Delta y$  are calculated and added to the

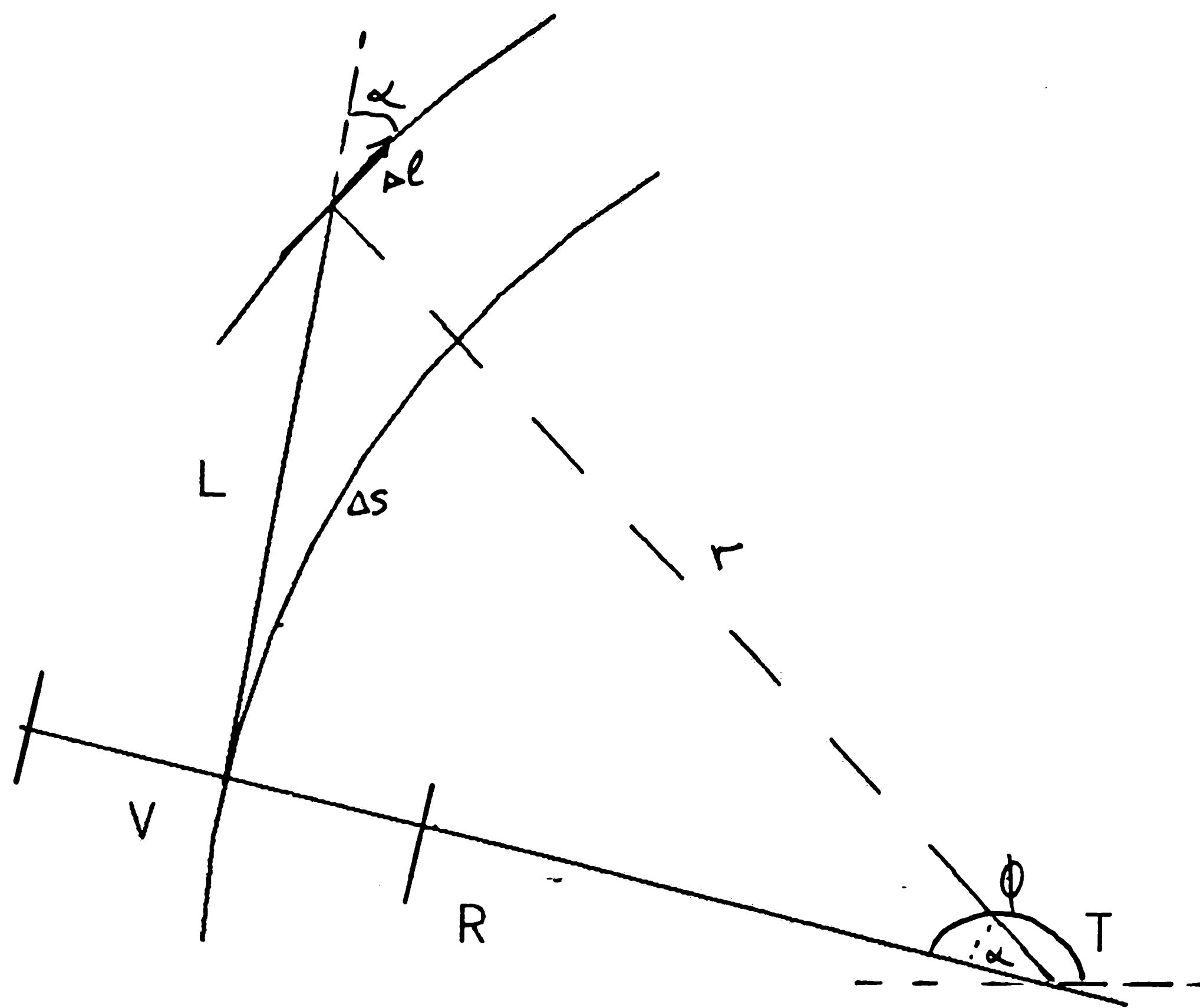


Figure 3-1: The geometry of the vehicle

previous  $X_p - Y_p$  cartesian coordinates of the vehicle position. To provide the current vehicle position the approximate change is determined as follows:

$$\Delta y = \Delta s \sin(\alpha) \quad (3.1)$$

$$\Delta x = \Delta s \cos(\alpha) \quad (3.2)$$

Where,

$$\alpha = \frac{\phi_1 + \phi_2}{2}$$

The arc length

$$\Delta s = R \Delta \phi \quad (3.3)$$

Where  $R$  is the radius of the circle, and  $\Delta \phi$  is the change in angle  $\phi$  (i.e.  $\Delta \phi = \phi_2 - \phi_1$ ). From the geometry,

$$r^2 = L^2 + R^2,$$

where,  $L$  is a constant representing the distance between the front wheel and the rear axle of the vehicle. Hence,

$$R = \sqrt{r^2 - L^2}$$

and from Figure 3-2 on page 51,

$$r = \frac{\Delta l}{\Delta \phi}$$

Substitute  $R$  and  $r$  Equation 3.3

$$\Delta s = \sqrt{\frac{\Delta l^2}{\Delta \phi^2} - L^2}$$



$$\Delta s = \sqrt{\Delta l^2 - L^2 \Delta \phi^2} \quad (3.4)$$

Substituting  $\Delta s$  and  $\alpha$  in equation. 3.1 and Equation 3.2 on page 49, then the increments are:

$$\Delta x = \sqrt{\Delta l^2 - L^2 \Delta \phi^2} \cos \left( \frac{\phi_1 + \phi_2}{2} \right)$$

$$\Delta y = \sqrt{\Delta l^2 - L^2 \Delta \phi^2} \sin \left( \frac{\phi_1 + \phi_2}{2} \right)$$

There are three different situations when calculating the vehicle position:

1. When the steering angle,  $\alpha$ , is  $90^\circ$ , as in Figure 3-2 on page 51 shows, the front wheel will travel:

$$\Delta L = L \Delta \phi = 30 \times 0.25^\circ = 01.308$$

Since  $\Delta l = 0.01$  inches., the count,  $n$ , of the odometer pulses will be

$$\frac{\Delta L}{\Delta l} = 13.08.$$

Hence, the minimum number of odometer counts  $n$  needed to compute  $\Delta s$  of Equation 3.4 on page 50 cannot become imaginary. But due to accidental events, such as startup conditions we have to insure that a count  $n < 14$  will not lead to errors. Therefore, If  $n < 14$ , the vehicle has not changed its position, but the values of  $n$  in this particular situation reflects the steering angle  $\alpha$  of being  $90^\circ$ . In this case the current vehicle position  $(X, Y)$  equals the previous vehicle coordinates  $(X_p, Y_p)$ .

3. As long as there are no changes of  $\Delta \phi$ , reported by the gyroscope, the angle used to calculate the vehicle position will be the previous angle  $\phi$ , (i.e.  $\phi_1$  of the current interval will be  $\phi_2$  of the previous interval). In this case the path is estimated to be continues on a straight line. Therefore,  $\Delta x$  and  $\Delta y$  will be:

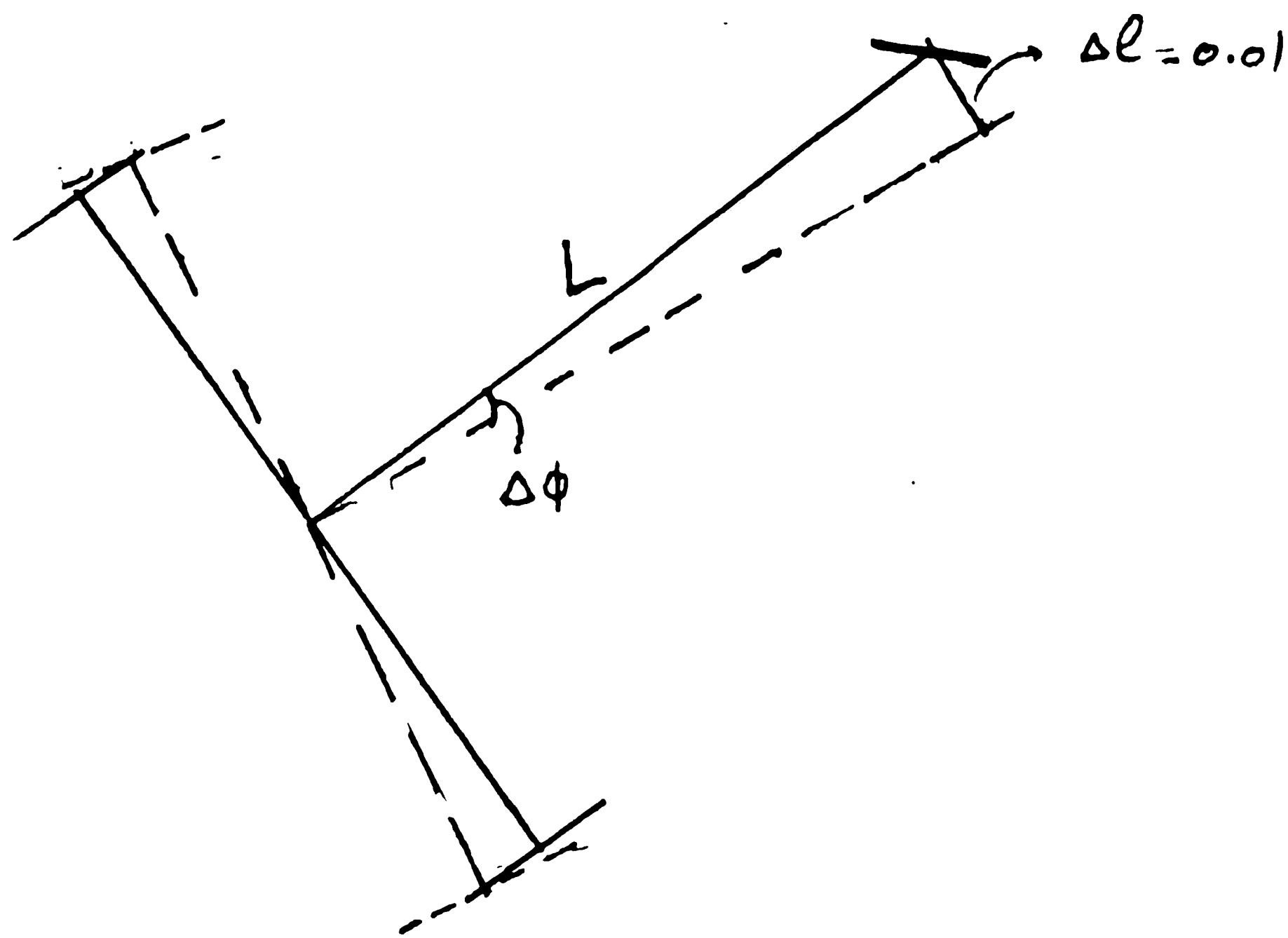


Figure 3-2: The minimum number of  $n$  counts

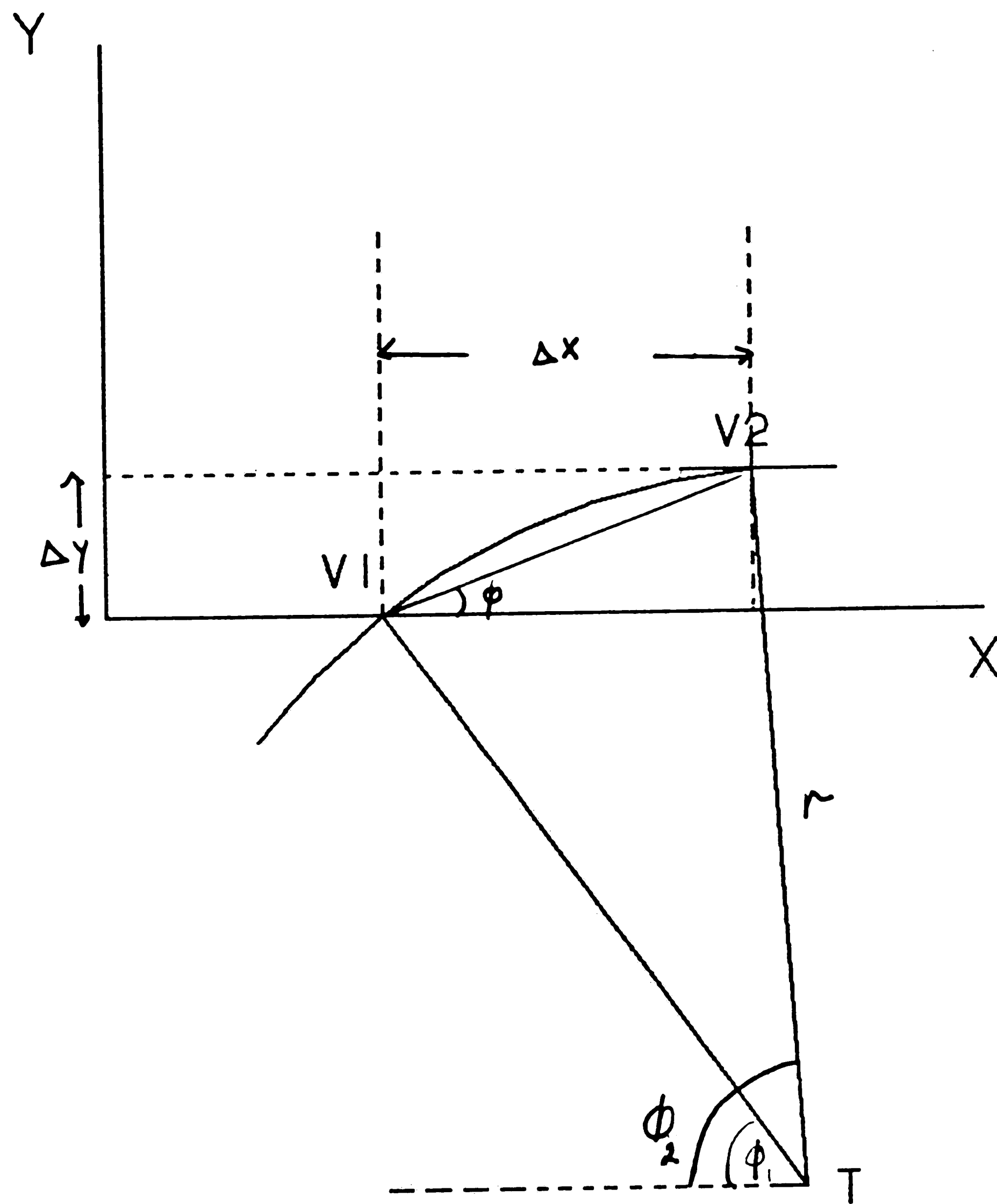
$$\Delta x = \Delta l \sqrt{1 - \left( \frac{L \Delta \phi}{n_p \Delta l} \right)^2} \cos(\phi_1)$$

$$\Delta y = \Delta l \sqrt{1 - \left( \frac{L \Delta \phi}{n_p \Delta l} \right)^2} \sin(\phi_1)$$

Where  $n_p$  is the  $n$  increments of wheel travel  $\Delta l$  during the previous interval. Therefore the computed vehicle position will be:

$$X = X_p + n \Delta x \tag{3.5}$$

$$Y = Y_p + n \Delta y \tag{3.6}$$



3. Figure 3-3: The geometry of the vehicle

where,  $X_p, Y_p$  are the computed vehicle coordinates during the previous change in  $\Delta\phi$ .

4. Finally, if a change in the direction by  $\Delta\phi = 0.25^\circ$  is indicated by the gyroscope, it becomes possible to compute the accurate coordinates from the circular arc in Figure 3-3 on page 52. The increments  $\Delta x$  and  $\Delta y$  are:

$$\Delta x = n \Delta l \sqrt{1 - \left(\frac{L\Delta\phi}{n \Delta l}\right)^2} \cos\left(\frac{\phi_1 + \phi_2}{2}\right)$$

$$\Delta y = n \Delta l \sqrt{1 - \left(\frac{L\Delta\phi}{n \Delta l}\right)^2} \sin\left(\frac{\phi_1 + \phi_2}{2}\right)$$

Then the computed vehicle position is:

$$X = X_p + \Delta x \tag{3.7}$$

$$Y = Y_p + \Delta y \tag{3.8}$$

where  $X_p$  and  $Y_p$  are the previous by computed vehicle position at the end of the previous  $\Delta\phi$  interval.

The **sin** and **cos** values are determined by a look up table stored in the computer. Once the vehicle position is computed using the third situation the previous point  $(X_p, Y_p)$  are set to the current point  $(X, Y)$ , the counter  $n$  is set to zero for the next computing interval, and the angle  $\phi_1$  is set to  $\phi_2$  for the next interval.

### 3.3 The Ground Navigation Software

The software for the ground navigation program is written in assembly language for the 8085 micro processor. The ground navigator program the program has been debugged, but has not yet been implemented by the vehicle.

The program consists of four interrupt procedures. Since the 8085 will function on an interrupt driven basis, all interrupts will be serviced on first come first serve basis, except for the 'Coordinate Request interrupt that is unmaskable[5].

The 'Beacon Sighted' interrupt service routine, is activated by an interrupt that is generated when a beacon is sighted. The routine outputs the current  $X$ - $Y$  coordinates and bearing to the address latched in the RAM and therefore, the coordinates are stored in memory.

The 'Buffer Full' interrupt service routine, is activated by a interrupt indicating that the on board computer has located a set of updated  $X$  and  $Y$  coordinates from the optical navigation system. The routine handles replacing the current  $X$ - $Y$  coordinates by the updated  $X$ - $Y$  coordinates, computed by the optical system.

The 'coordinate Request' interrupt service routine is generated when the on board computer requests the current position coordinates. The service routine, allows the  $X$ - $Y$  coordinates to remain in memory unchanged long enough to allow the on board computer read the data.

The ' $\Delta l$ ' interrupt service routine, is activated every 0.01 inches. of travel by the front wheel. The vehicle position will be calculated using the algorithm described in the previous section. The routine starts by reading the bearing  $\phi$  from ports and checks to see if any changes have occurred, if no change in  $\phi$

has occurred then Equations 3.5 and Equations 3.6 are used, if there was a change, then Equations 3.7 and Equations 3.8 are used. In addition the routine also checks if the number of fixed interval  $n$  is less than 14, in order to avoid computing the square root of a negative number. Therefore, the current vehicle position will equal the previous vehicle position. The  $\cos$  and  $\sin$  of the angle  $\phi$  are obtained from a look up table which will reside in the ROM. Presently this table has increments of  $0.5^\circ$  between points. Only the first quadrant ( $0 - 90^\circ$ ) is stored. Figure 3-4 on page 56 Displays the flow of  $\Delta l$  signal service routine.

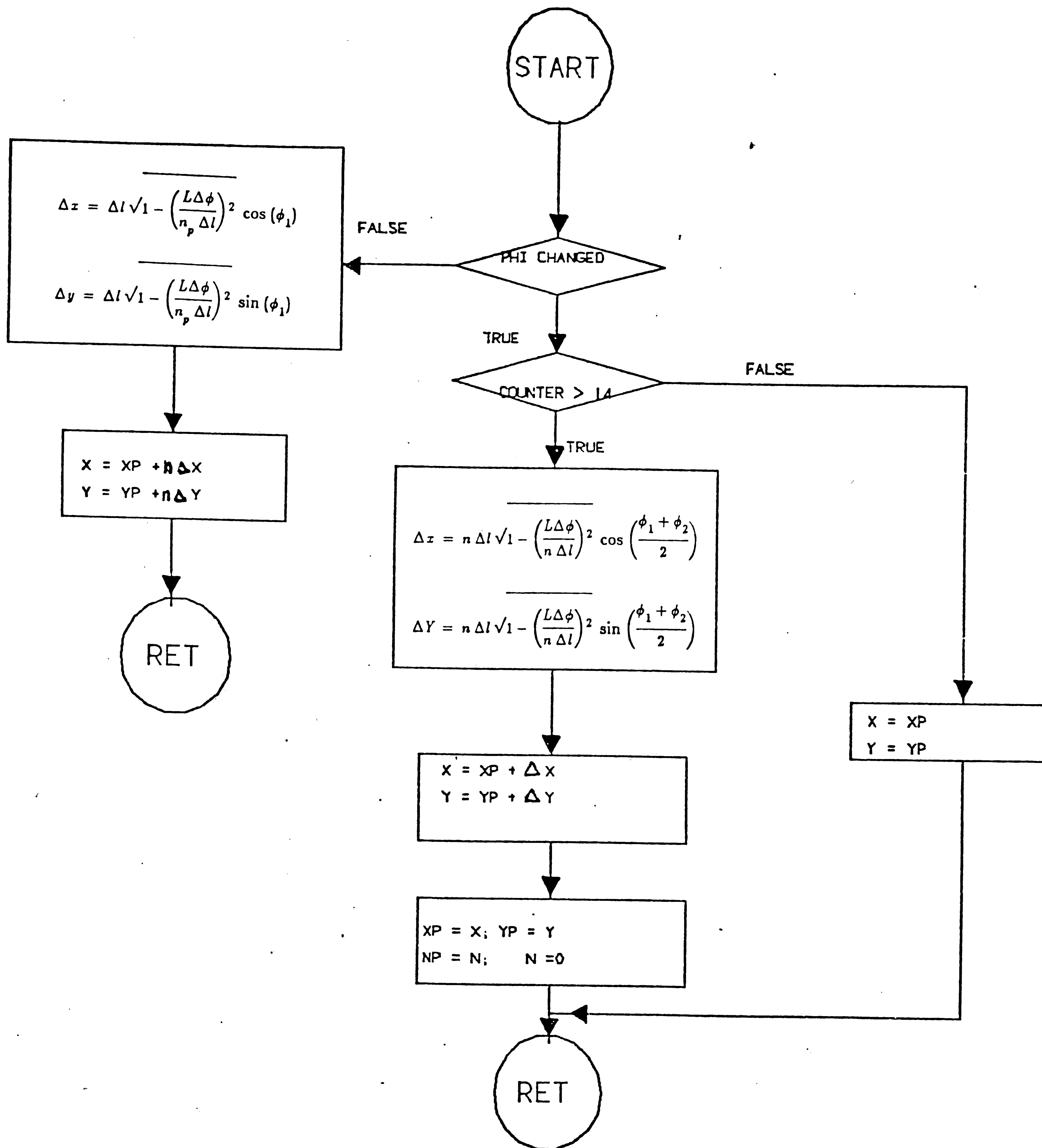


Figure 3-4:  $\Delta l$  Signal service routine

## Chapter 4

# The Drive Routine

The drive routine controls the driving and steering system of the vehicle. It is activated whenever the  $\Delta S$  pulse is generated by the drive board signaling the on-board computer that the present path segment distance is almost completely traversed so that the computer can compute the steering angle, the velocity and the length of the next path segment to travel.

The computation of next path segment has to be finished before the completion of the previous path, in order to guarantee that a drive instruction will always be obtainable in time. The vehicle travels from one point to another within a segment consisting of two circular arcs with two turning radii. The radii are adjusted to be equal in magnitude for smoothness of movement but of opposite signs, where, the sign indicates right or left turns. Since the vehicle travels upon two arcs, the vehicle trajectory is approximated by a sequence of circles. The length of each double arc is of the order of 1.0 meter.

As a result, the following 'Strategy' [3] can be derived which will direct the vehicle towards the ideal path:

- a. At the end of each section, the computed vehicle coordinates and bearing are taken from the ground navigator. Then, the next section is computed in such a way that it will bring the vehicle to the next point on the prescribed path.
- b. Due to the deviation of trajectory, as mentioned above, this point will never be reached, but the true position will again be determined and a new section computed, and so on.



## 4.1 The Mathematical Theory of The Drive Routine

Figure 4-1 on page 61 shows the geometry of the vehicle traveling from point  $(X_1, Y_1)$  to the desired position  $(X_2, Y_2)$ , following two circular arcs  $S_1$  and  $S_2$ . Where the initial vector at point  $(X_1, Y_1)$  is  $\bar{A}$ , and its destination vector at point  $(X_2, Y_2)$  is  $\bar{B}$ . The vector  $\bar{D}$  reflects the straight line between the current vehicle point and the destination point. It turns out that the computation of the two double arcs  $S_1$  and  $S_2$ , and their radii,  $r$ , depends on three parameters. They are: the change of the vehicle bearing  $\beta$ , the angle  $\phi$  between the vectors  $\bar{A}$  and the vector  $\bar{D}$  itself. This results in five different cases of double arc, each having different equations and the mirror image of each. Three of these cases reflect a trajectory of the vehicle traveling along two circular arcs, where as the other two cases reflect a trajectory of the vehicle traveling consecutively along an arc and a straight line. The drive routine computes the following values:

1. The parameters that are needed to select the trajectory.
2. The double arcs lengths  $S_1$  and  $S_2$ , and radius  $r$ .
3. The steering angle  $\alpha_1$  and  $\alpha_2$ .

### 4.1.1 Computation of Parameters

As mentioned previously, some derived parameters are needed to compute the double arcs and the turning radius of the vehicle. The parameter,  $T$ , (translation) is computed to determine whether the target point  $(X_2, Y_2)$  is to the left of the vector  $\bar{A}$  or to the right. Similarly, the rotation,  $C$ , is to determine whether the vector  $\bar{B}$  causes the vehicle to rotate clockwise,  $CW$ , or counter clockwise,  $CCW$ . The following are definitions of the needed parameters:

1. The translation of the vehicle:

$$T = (\bar{A} \times \bar{D})_z$$

If the sign of  $T$  is positive then the vehicle is translated to the left, otherwise it is translated to the right.

2. The parameter  $P$  is computed to determine the range of  $\beta$ , where

$$P = (\bar{A} \cdot \bar{D})$$

If the sign of  $P$  is positive then the range of  $\beta$  is within  $\pm 90^\circ$ , otherwise  $\beta$  is outside of  $\pm 90^\circ$

3. The rotation of the vehicle is determined by the parameter,  $C$ , where

$$C = (\bar{A} \times \bar{B})_z$$

If the sign of  $C$  is positive then the rotation of the vehicle is counter clock wise (CCW), otherwise rotation is clock wise (CW)

4.  $S$  is computed to determine the range of  $\phi$  where:

$$S = (\bar{A} \cdot \bar{B})$$

If the sign of  $S$  is positive then  $\phi$  is within  $\pm 90^\circ$ , otherwise  $\phi$  is outside of  $\pm 90^\circ$ .

5. A number of other desired parameters are needed for the computation of the double arcs. These are:

$$\cos(\phi) = \frac{|S|}{|\bar{A}| |\bar{B}|}$$

$$\sin(\phi) = \frac{|C|}{|\bar{A}| |\bar{B}|}$$

$$d = \frac{P - |T| |S| / |C|}{|\bar{A}|}$$

and,

$$a = \frac{|T|}{|\bar{A}|}; \quad h = \frac{|T| |B|}{|C|}.$$

#### 4.1.2 The Computation of Double Arcs And Radius

Since the computation of the double arc and turning radius depend on the values of  $\phi$  and  $\beta$  and a minimum permissible turning radius. There are five different cases of computing these arcs and their radii. This section will describe each case in detail.

1. **Case (1):** If  $0 < \phi < 2\beta$ . and the rotation of the vehicle and the translation are in the same direction. Then, in order to compute the length of the two arcs  $S_1$  and  $S_2$ , one needs to analyze the geometry of Figure 4-1 on page 61. The parametric equations of circle  $CA$  are obtained as follows:

$$X - X_{01} = -r \cos(\psi_A). \tag{4.1}$$

$$Y - Y_{01} = r \sin(\psi_A). \tag{4.2}$$

Where  $(X_{01}, Y_{01})$  is the center of circle  $CA$  Since  $\psi_A$  at point  $(X_1, Y_1)$  is zero then the parametric equation for circle  $CA$  through point  $(X_1, Y_1)$  must have:

$$X_{01} = r + X_1; \quad Y_{01} = Y_1$$

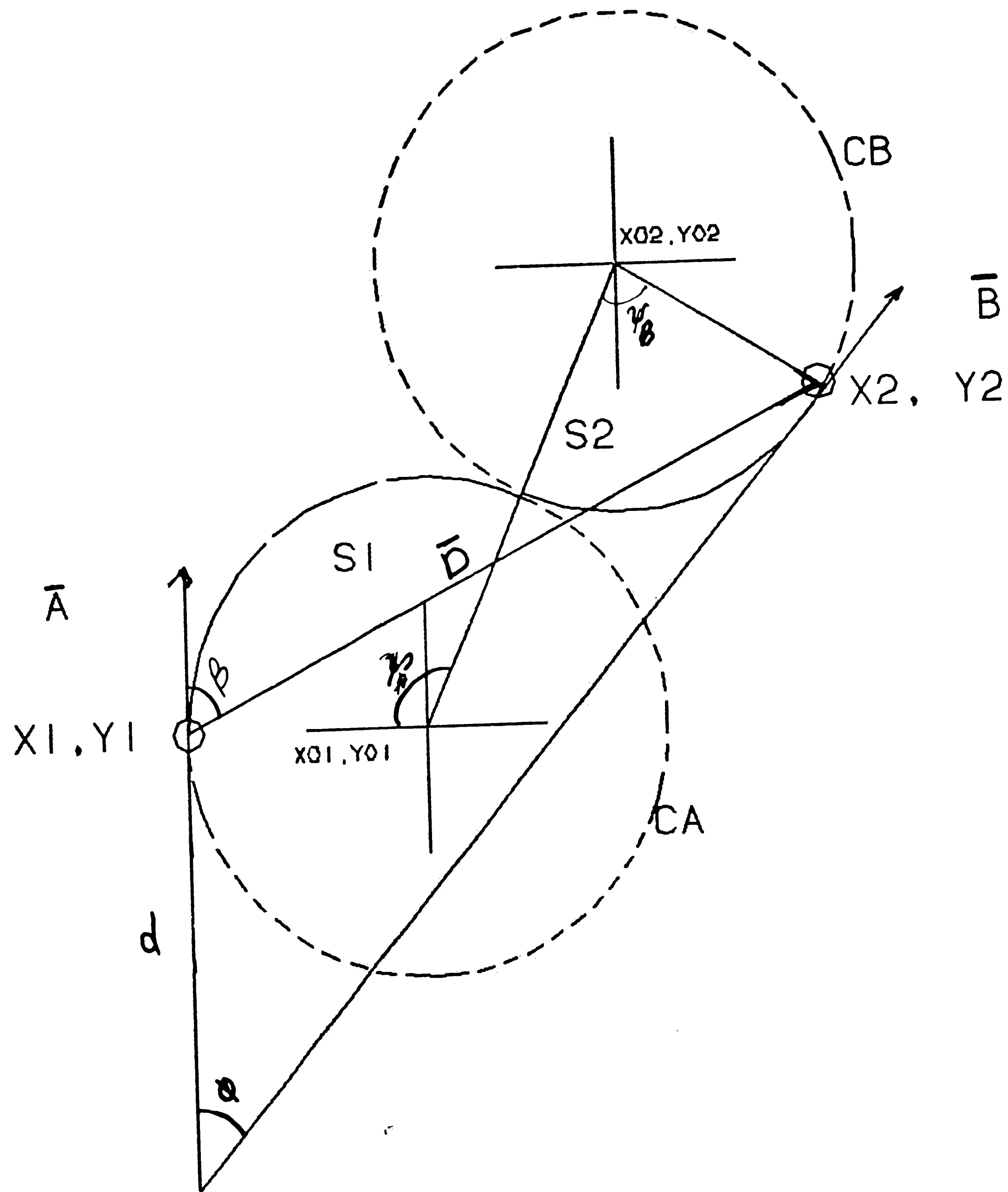


Figure 4-1: Case (1) of Double Arc

By substituting  $X_{01}$  and  $Y_{01}$  into equation 4.1 and 4.2 respectively, the parametric equation of any point on circle  $CA$  will be:

$$X_A = -r \cos(\psi_A) + r + X_1.$$

$$Y_A = r \sin(\psi_A) + Y_1.$$

Similarly, the general parametric equation for circle  $CB$  will be:

$$X - X_{02} = r \cos(\psi_B). \quad (4.3)$$

$$Y - Y_{02} = -r \sin(\psi_B). \quad (4.4)$$

Since  $\psi_B$  at point  $(X_2, Y_2)$  is equal to  $\phi$ , the parametric equations through point  $(X_2, Y_2)$  will have

$$X_{02} = -r \cos(\phi) + X_2. \quad Y_{02} = r \sin(\phi) + Y_2$$

By substituting  $X_{02}$  and  $Y_{02}$  into Equation. 4.3 and 4.4 respectively, the parametric equation of any point on circle  $CB$  will be:

$$X_B = r \cos(\psi_B) - r \cos(\phi) + X_2$$

$$Y_B = -r \sin(\psi_B) + r \sin(\phi) + Y_2.$$

The Two circles  $CA$  and  $CB$  intersect at a point where,

$$\psi_A = \psi_B = \psi.$$

$$X_A = X_B$$

Hence,

(4.5)

$$-r \cos (\psi) + r + X_1 = r \cos (\psi) - r \cos (\phi) + X_2.$$

Similarly,

$$Y_A = Y_B$$

Hence

$$r \sin (\psi) + Y_1 = -r \sin (\psi) + r \sin (\phi) + Y_2 \quad (4.6)$$

By rearranging Equation 4.5, one can obtain the following:

$$-2 r \cos (\psi) + r \cos (\phi) + r = X_2 - X_1.$$

and

$$-2 \cos (\psi) + \cos (\phi) + \frac{(X_2 - X_1)}{r} = \frac{a}{r}. \quad (4.7)$$

From Equation 4.6, one can obtain the following:

$$2 \sin (\psi) + \sin (\phi) = \frac{(Y_2 - Y_1)}{r} = \frac{a \cot (\phi) + d}{r}.$$

Then,

$$r = \frac{a \cot (\phi) + d}{2 \sin (\psi) + \sin (\phi)}.$$

Substitute  $r$  in Equation 4.7, to solve for  $\psi$ :

$$-2 \cos (\psi) + \cos (\phi) + 1 = \frac{a 2 \sin (\psi) + a \sin (\phi)}{a \cot (\phi) + d}$$

where

$$-2 \cos (\psi) + \cos (\phi) + 1 = \frac{a}{a \cot (\phi) + d} \left( 2 \sin (\psi) + \sin (\phi) \right).$$

and

$$2 \cos (\psi) + \frac{2 a \sin (\psi)}{d + a \cot (\phi)} = \frac{a \sin (\phi)}{d + a \cot (\phi)} + \cos (\phi) + 1.$$

By trigonometric identity

$$\begin{aligned} & 2 \sqrt{1 + \left[ \frac{a}{d + a \cot (\phi)} \right]^2} \cos \left( \psi - \tan^{-1} \left[ \frac{a}{d + a \cot (\phi)} \right] \right) \\ & = \cos (\phi) + \frac{a}{d + a \cot (\phi)} \sin (\phi) + 1. \end{aligned} \quad (4.8)$$

Define

$$K = \frac{a}{d + a \cot (\phi)} = \tan (\beta)$$

Substitute  $K$ , in Equation. 4.8

$$2 \sqrt{1 + K^2} \cos (\psi - \tan^{-1}(K)) = \cos (\phi) + K \sin (\phi) + 1.$$

Then,

$$\psi = \cos^{-1} \left( \frac{\cos (\phi) + K \sin (\phi) + 1}{2 \sqrt{1 + K^2}} \right) + \beta.$$

From Equation 4.7 one can obtain  $r$  where,

$$r = \frac{a}{\cos (\phi) - 2 \cos (\psi) + 1}$$

Since  $\psi$  and  $r$  are known, the equations for the lengths of the double arcs are

$$S_1 = r \psi ; \quad S_2 = r (\psi - \phi).$$

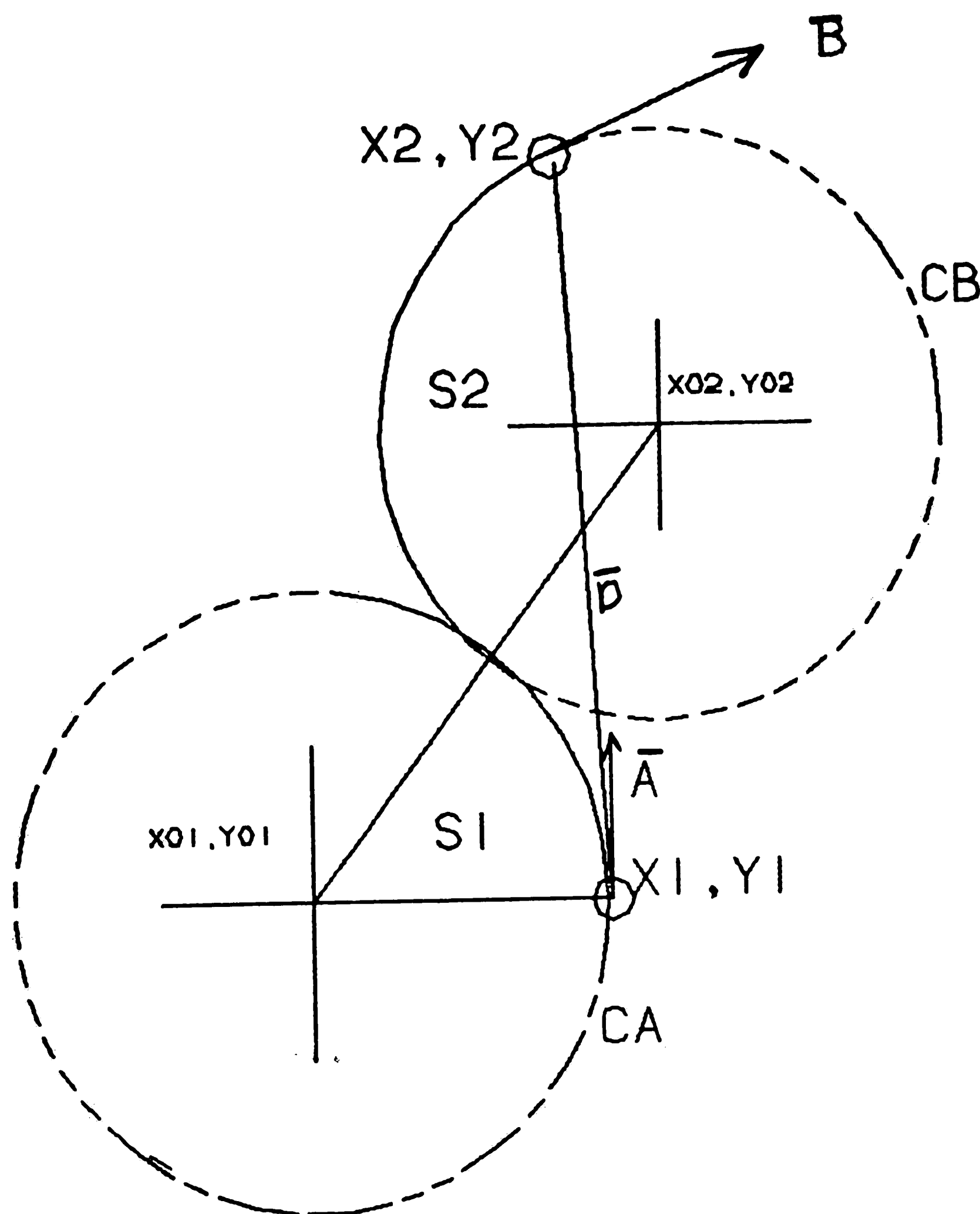


Figure 4-2: Case (2) of Double Arc

2. Case (2): If  $\phi > 2\beta$ , and the rotation and translation of the vehicle are in the same direction. Then by looking at Figure 4-2 on page 65, the following set of equations can be derived similarly to case (1):

$$\psi = \cos^{-1} \left( \frac{K \cos(\phi) + \sin(\phi) + 1}{2\sqrt{1+K^2}} \right) - \beta$$

The radius will be:

$$r = \frac{a}{2 \cos(\psi) - \cos(\phi) - 1}$$

and the equations for the lengths of the double arcs are:

$$S_1 = r\psi; \quad S_2 = r(\psi + \phi).$$



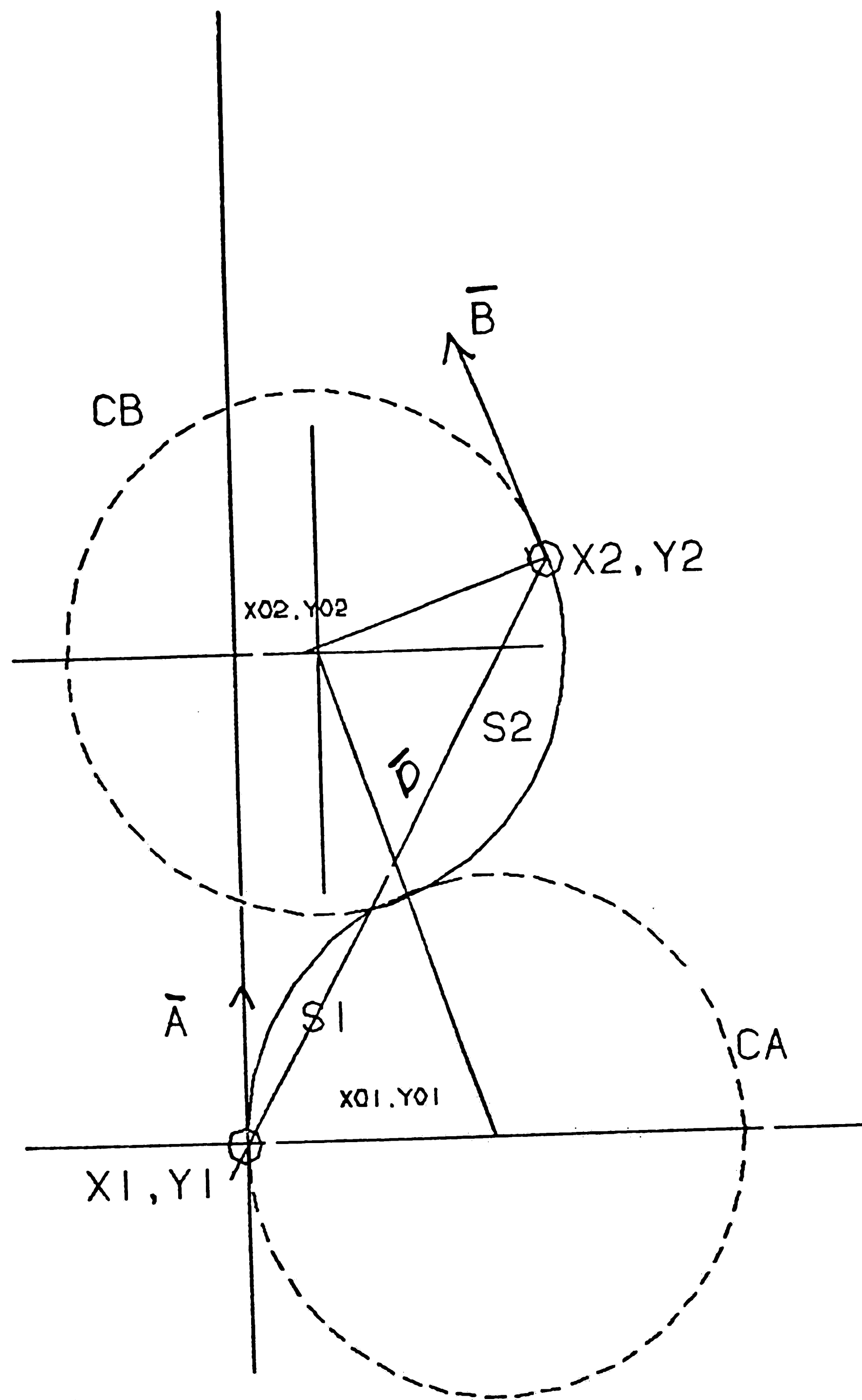


Figure 4-3: Case (3) of Double Arc

3. Case (3): If  $0 < \phi < 90^\circ$ , and the translation and rotation of the vehicle are in opposite direction, then from Figure 4-3 on page 66, one obtains:

$$\psi = \cos^{-1} \left( \frac{K \sin(\phi) - \cos(\phi) - 1}{2\sqrt{1+K^2}} \right) - \beta.$$

The radius will be:

$$r = \frac{a}{2 \cos(\psi) + \cos(\phi) + 1}$$

The equations of the length of the two arcs  $S_1$  and  $S_2$  are :

$$S_1 = r(\pi - \psi); \quad S_2 = r(\pi - \psi + \phi).$$

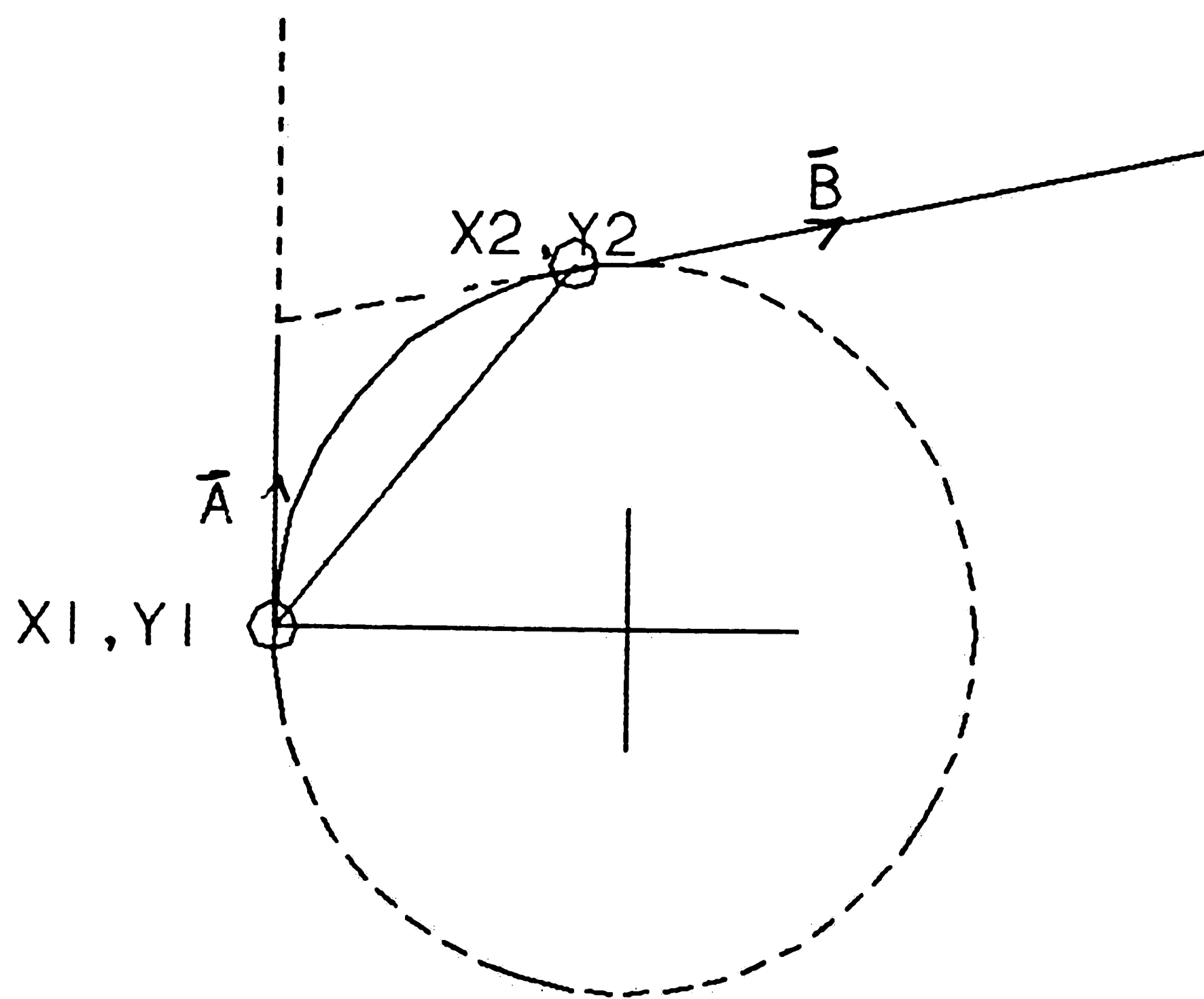


Figure 4-4: Case (4) of Double Arc

4. Case (4): If  $\phi < 2\beta$ , translation and rotation of the vehicle are in the same direction, but the vehicle first travels in an arc then travels on a straight line. Figure 4-4 on page 67 The following are a set of equations that should be used: The turning radius  $r$  will be:

$$r = \frac{d}{\tan(\phi/2)}$$

The equations of the two lengths of arcs  $S_1$  and  $S_2$  are:

$$S_1 = r\phi \quad S_2 = h - d$$

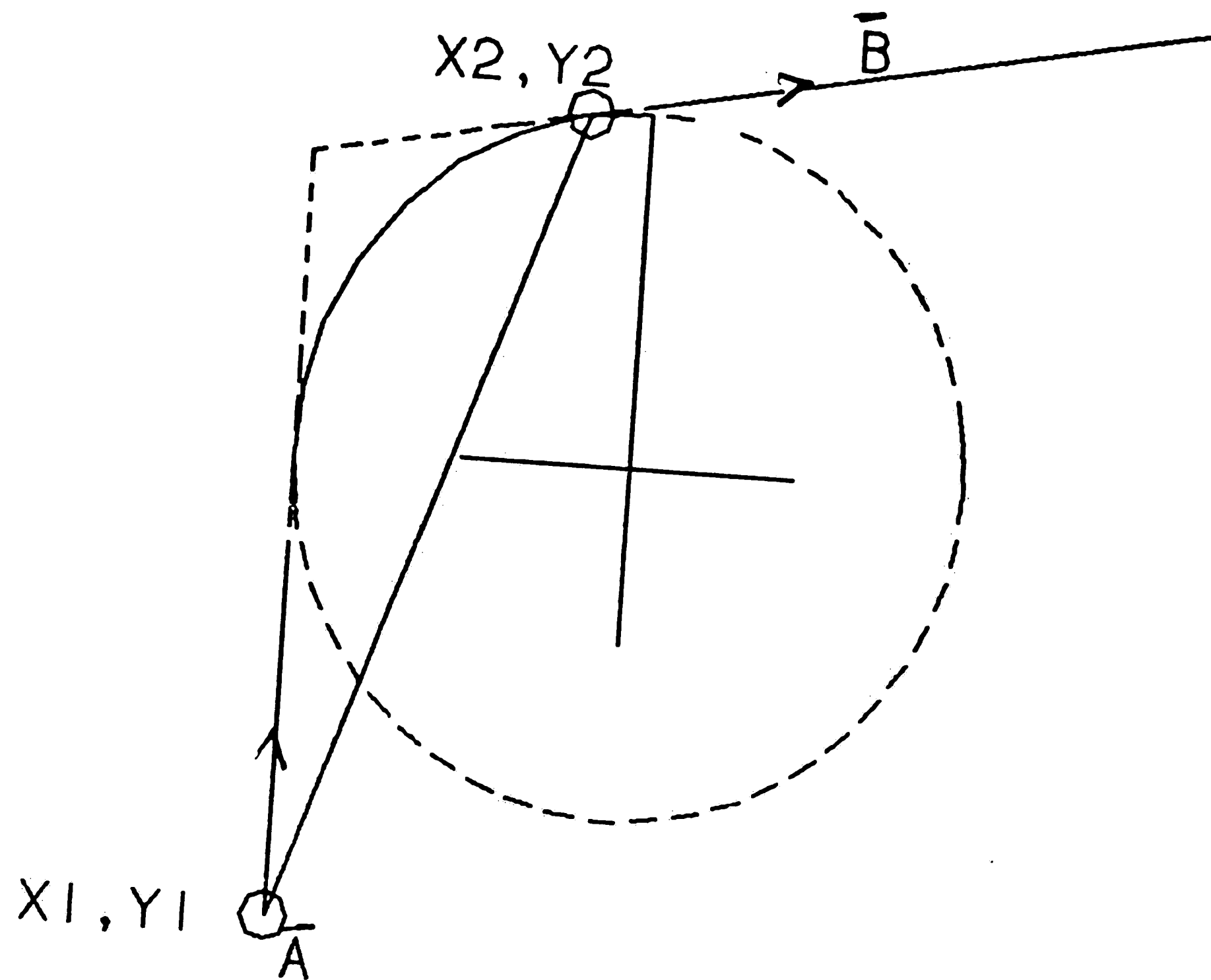


Figure 4-5: Case (5) of Double Arc

5. Case (5): If  $\phi > 2\beta$ , and the translation and rotation of the vehicle are in the same direction. The vehicle first will travel in a straight line, then on an arc Figure 4-5 on page 68. The following set of equations should be used.

The turning radius  $r$  is:

$$r = \frac{h}{\tan(\phi/2)}$$

The equations of the length of the two arcs are:

$$S_1 = d - h \quad S_2 = r\phi$$

## 4.2 The Computation of the Steering Angle

Once  $S_1$  and  $S_2$ , and the turning radius have been computed, the commands to the drive controller are determined by computing the length of the arcs  $l_1$  and  $l_2$  that are covered by the front wheel, and the steering angle  $\alpha$  where,

$$l_1 = 4.4 \sqrt{1 - [L/r]^2} S_1; \quad l_2 = 4.4 \sqrt{1 - [L/r]^2} S_2$$

Where,  $L$ , is the length of the vehicle from the front wheel to the rear axle.  $L$  in this case is 30.0 inches.  $r$ , is the computed turning radius, and the factor (4.4) is used to convert the arc length from units of  $\Delta l$  increments of travel, to inches. The steering angles,  $\alpha_1$  and  $\alpha_2$  are:

$$\alpha_1 = \tan^{-1}(L/r_1). \quad \alpha_2 = \tan^{-1}(L/r_2).$$

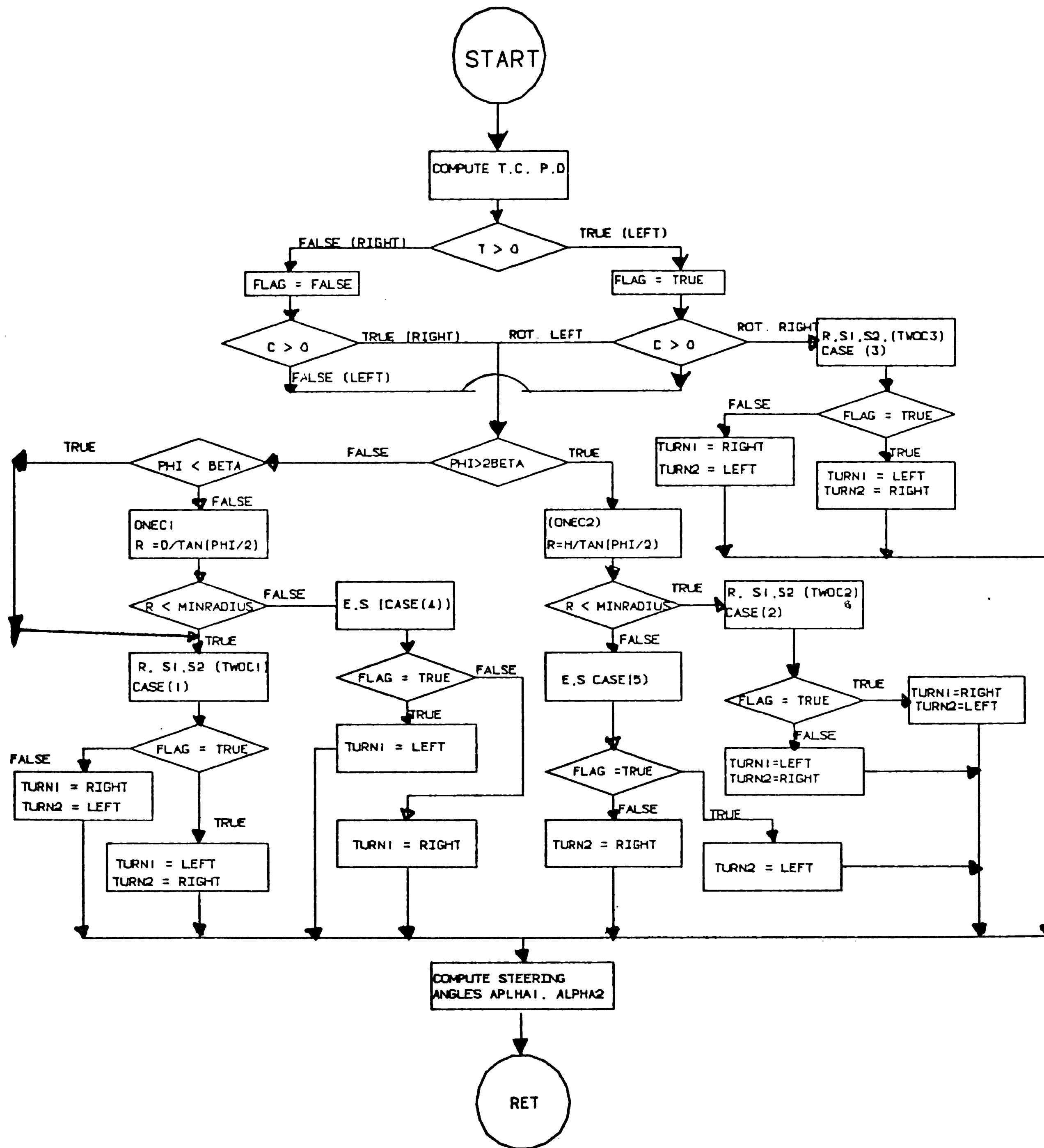
Where  $r_1$  and  $r_2$  are equal in magnitude but of opposite directions. For backward travel, the signs of the all prescribed vectors  $\bar{B}$  and measured vectors  $\bar{A}$  are changed and the left and right turns are reversed. The direction of vectors  $\bar{A}$  and  $\bar{B}$  are used in this routine, therefore one simply can use the prescribed vectors  $\bar{B}$  to compute the velocity of the vehicle at each prescribed point.

## 4.3 The Drive Routine Software

The drive program was written to assist the vehicle in traveling from one point to another using double arcs with a constant turning radius. This program provides the vehicle with intelligence, by allowing it to select the the correct path it should traverse.

Figure 4-6 on page 71, describes the flow of the different procedures to compute the length of the arcs and the turning radius of the vehicle. The procedures are executed after the measured vehicle coordinates are received from the ground navigator and the bearing from the gyroscope. Once the drive routine is activated, it follows a sequence of steps to compute the length of double arcs and turning radius. The steps that it follows are described below:

1. Compute parameters  $T$ ,  $C$ , and  $S$ , and the angles  $\phi$  and  $\beta$ .
2. Check if the sign of the translation,  $T$ , is positive. If so, then the translation of the vehicle is to the left and the flag is set to true, otherwise the translation of the vehicle is to the right and the flag is set to false.
3. Check if the sign of the rotation,  $C$ , is positive. If so, then the rotation of the vehicle is to the left  $CCW$ , otherwise the rotation of the vehicle is to the right,  $CW$ .
4. If the rotation and the translation are of opposite directions, then Case (3) is selected and the following steps are executed:
  - a. Compute the double arcs  $S_1$  and  $S_2$  and the turning radius,  $r$ .
  - b. If the flag is set to true then the vehicle first turns to the left then turns to the right. Otherwise the vehicle first turns to the right then turns to the left.
  - c. Go to step (7).
5. If the rotation and the translation are in the same direction, and  $\phi > 2\beta$  then.
  - a. Compute the turning radius  $r$ .
  - b. If the radius,  $r$ , is less than the permitted radius then Case (2) of double arcs is selected, and the following steps are executed:
    - i. Compute the lengths of the double arcs  $S_1$ ,  $S_2$  and turning radius,  $r$ . If the flag is set to true, then the vehicle first turns to the right and then turns to the left. Otherwise the vehicle first turns to the left then to the right.



ii. Figure 4-6: The Drive Routines

- ii. Go to step (7).
- c. If the turning radius,  $r$ , is greater than the permitted radius the Case (5) of double arcs is selected, and the following steps are executed.:
  - i. Compute the length of the straight line,  $e$ , and the length of the arc.
  - ii. If the flag is set to true, then the vehicle first travels in a straight line then turns to the left. Otherwise the vehicle first travels in a straight line then turns to the right.
  - iii. Go to step (7).
- 6. If the rotation and translation are in the same direction and  $\phi < 2\beta$  then:
  - a. If  $\phi < \beta$  then the go to step (6.b), otherwise the following steps are executed.
    - i. Compute the turning radius,  $r$ .
    - ii. If the computed radius,  $r$ , is less than the permitted radius then go to step (6.b). Otherwise Case (4) of double arcs is selected.
    - iii. compute the length of the first arc,  $e$ , which in this case is a straight line and the length of the second arc.
    - iv. If the flag is set to true then the vehicle first turns to the left then along a straight line. Otherwise the vehicle turns to the right then travels along a straight line.
    - v. Go to step (7).
  - b. Case (1) of double arcs is selected, and the following steps are executed.:
    - i. Compute the length of the double arcs  $S_1$  and  $S_2$ , and the radius  $r$ .
    - ii. If the flag set to true then the vehicle first travels to the left then to the right. Otherwise the vehicle first travels to the right then to the left.

7. Compute Steering angles  $\alpha_1$  and  $\alpha_2$ .

8. Return to main the program.

#### 4.3.1 Compute Parameters and Angles Procedure

This procedure is responsible of computing the parameters such as the rotation and translation of the vehicle and the angles  $\phi$  and  $\beta$ . The program assumes that the vehicle is traveling from the measured point  $(X_1, Y_1)$  to the prescribed point  $(X_2, Y_2)$  the following inputs are needed to compute those parameters:

##### Inputs

1. (VEH\_DIR\_XA, VEH\_DIR\_YA) : The direction at point A.
2. (VEH\_DIR\_XB, VEH\_DIR\_YB) : The direction at point B.
3. (VEH\_POS\_X, VEH\_POS\_Y) : The coordinates of the vehicle point A
4. (POINTS\_X, POINTS\_Y) : The prescribed point B.

##### Outputs

1. d and h
2. PHI : The angle  $\phi$
3. CPHI AND SPHI :  $\cos(\phi)$  and  $\sin(\phi)$
4. BETA : The angle  $\beta$
5. TRANS : The translation of the vehicle
6. C : The Rotation of the vehicle.
7. S : Its value determines the range of  $\phi$
8. P : Its value determines the range of  $\beta$



### 4.3.2 One Arc and Straight Line ONEC1 Procedure

This procedure is executed, whenever  $\phi < 2\beta$  and the translation and rotation of the vehicle are in the same direction. If the computed radius is less than the minimum radius of turning the procedure TWOC1 is called resulting in double arcs computation. Figure 4-7 on page 75 describes the flow of ONEC1 procedure.

#### Inputs

1. d and h
2. PHI : The angle  $\phi$
3. CPHI AND SPHI :  $\cos(\phi)$  and  $\sin(\phi)$
4. BETA : The angle  $\beta$
5. MIN\_RADIUS : The minimum constant radius.

#### Outputs

1. RADIUS1, RADIUS2 : The radius of the arc
2. ARC1, ARC2 : The length of the two arcs ARC1 & ARC2 where ARC2 represents a straight line.

### 4.3.3 A Straight Line and One Arc ONEC2 Procedure

This procedure is executed, whenever  $\phi > 2\beta$ , and the translation and the rotation of the vehicle are in the same directions. In this case the vehicle first travels on a straight line then along an arc resulting in Case (5) of double arcs, unless the computed radius is less than the minimum permitted radius of turning. In this case select Case (2) of double arcs by calling procedure TWOC2. This procedure has the same inputs and outputs of procedure ONEC1.

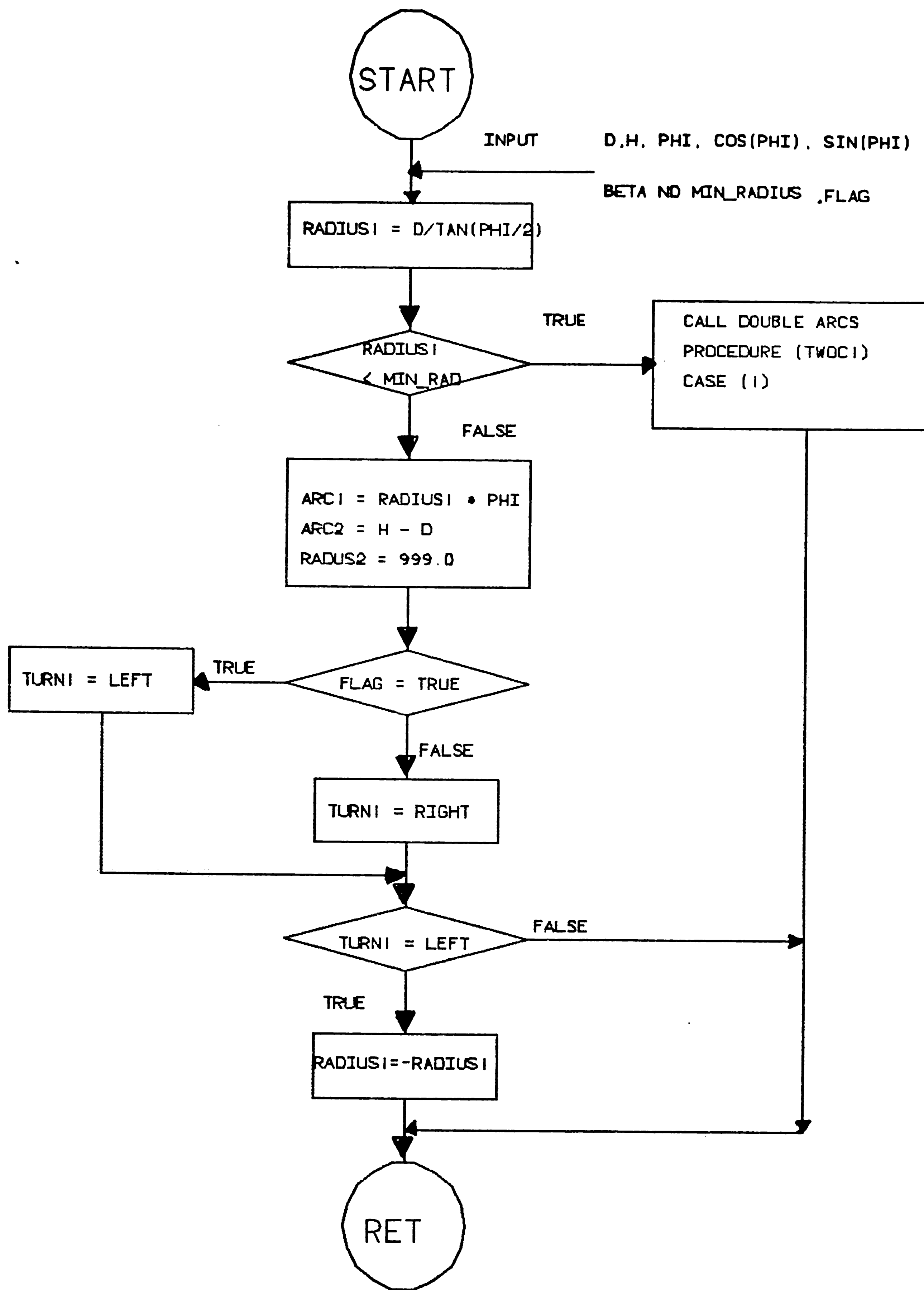


Figure 4-7: One circle and straight line procedure

#### 4.3.4 Double arc TWOC1 Procedure

This procedure is executed under two conditions, 1) if the radius computed in procedure ONEC1 is smaller than the permitted radius or, 2)  $\phi < \beta$ . As a result the vehicle will travel along two circular arcs. Figure 4-8 on page 77 describes that procedure. Below is a description of the inputs and outputs to that procedure.

##### Inputs

1. d, and h
2. PHI : The angle  $\phi$
3. CPHI and SPHI :  $\cos(\phi)$  and  $\sin(\phi)$
4. BETA : The angle  $\beta$
5. MIN\_RADIUS : The minimum constant radius.
6. FLAG : The flag.

##### Outputs

1. RADIUS1, RADIUS2 : The radius of the circular arc.
2. ARC1, ARC2 : The length of the two arcs ARC1 and ARC2.

#### 4.3.5 Double arc TWOC2 Procedure

This procedure is executed if the radius computed in procedure ONEC2 is smaller than the permitted minimum radius. Therefore, the vehicle will travel through two double arcs. The inputs, outputs, and flow chart are similar to procedure TWOC1.

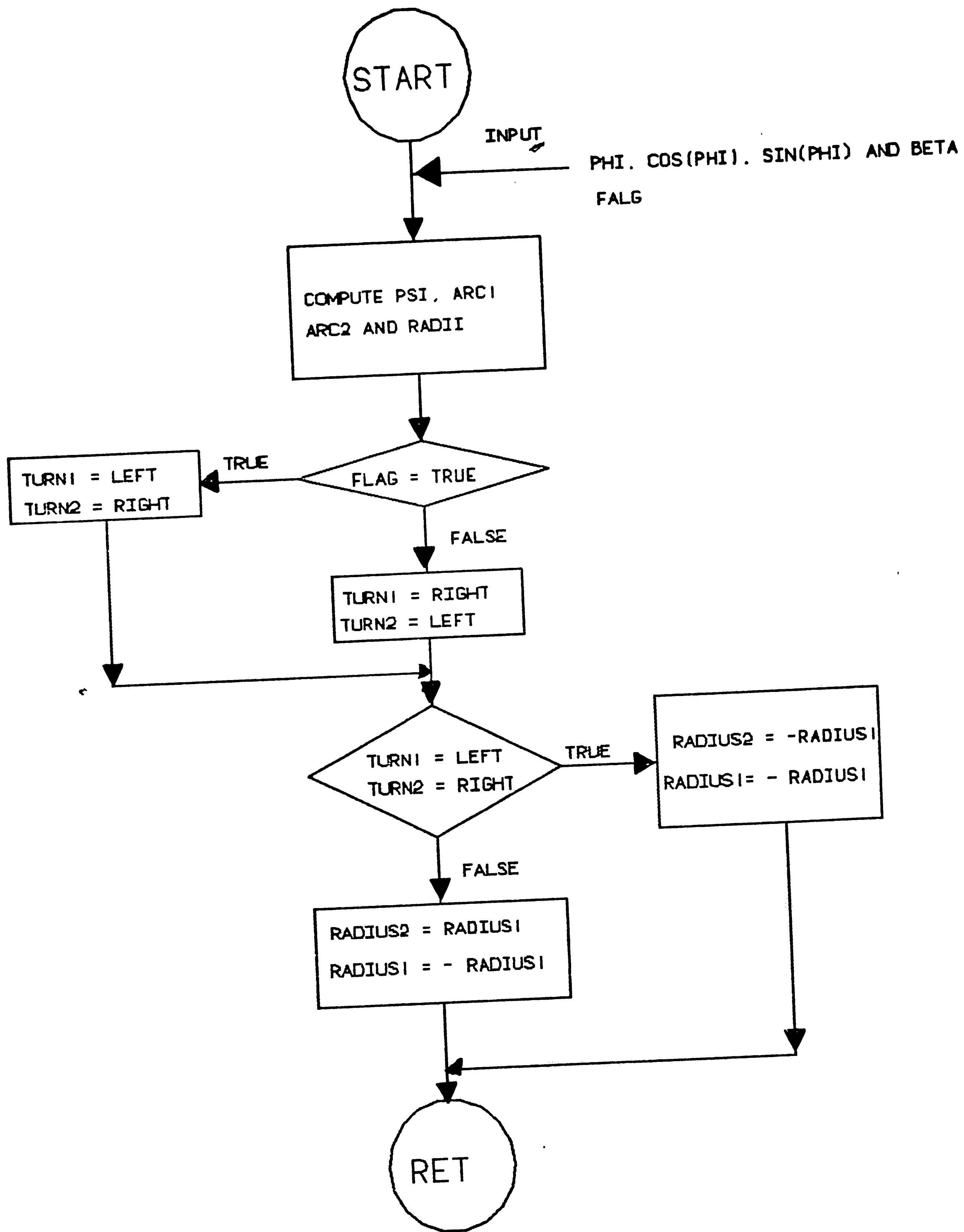


Figure 4-8: Double Arcs Procedure

#### **4.3.6 Double arc TWOC3 Procedure**

This procedure is executed, whenever  $0 > \phi < 90^\circ$ , and the translation and the rotation of the vehicle are of opposite directions. The procedure's inputs, outputs and flow chart are similar to procedures **TWOC1** and **TWOC2**.

## Chapter 5

# The Overall System Design

This chapter describes the overall system design, and the overall timing of events for the vehicle Cyclopiion. At the present time the ground navigation program has not yet been in operation. Nevertheless, this chapter will describe the full travel sequence of events. Including the ground navigation system. Then a simplified method of travel will be described which has been successfully implemented on this test bed vehicle.

### 5.1 The Full Travel

Figure 1-1 on page 4 [3], shows the diagram of the complete system, where one can distinguish two separate units the drive software on the left hand side, and the goniometer and locating vehicle position software to the right. These two units communicate through "the request for position and bearing" and the "position and bearing response". Before the vehicle starts, the path vectors with their speeds are down loaded

to the computer from the base station, to be used by the path refitting routine. The routine also accepts the "nominal radius" which the vehicle will normally use at every corner of the path. The radius is selected big enough to produce a smooth ride and small enough to avoid obstacles. The vehicle accepts the initial steering angle and the direction.

Before the vehicle starts its run, the path fitting routine will compute suitable secondary path vectors, filling in between the primary points in such a

way, that the distance corresponds to the desired double arc length of the drive routine as explained in [4]. The vehicle then issues a ready signal to start computing and traveling along its prescribed path. The sequence of events for the complete system of a vehicle of this kind is shown below:

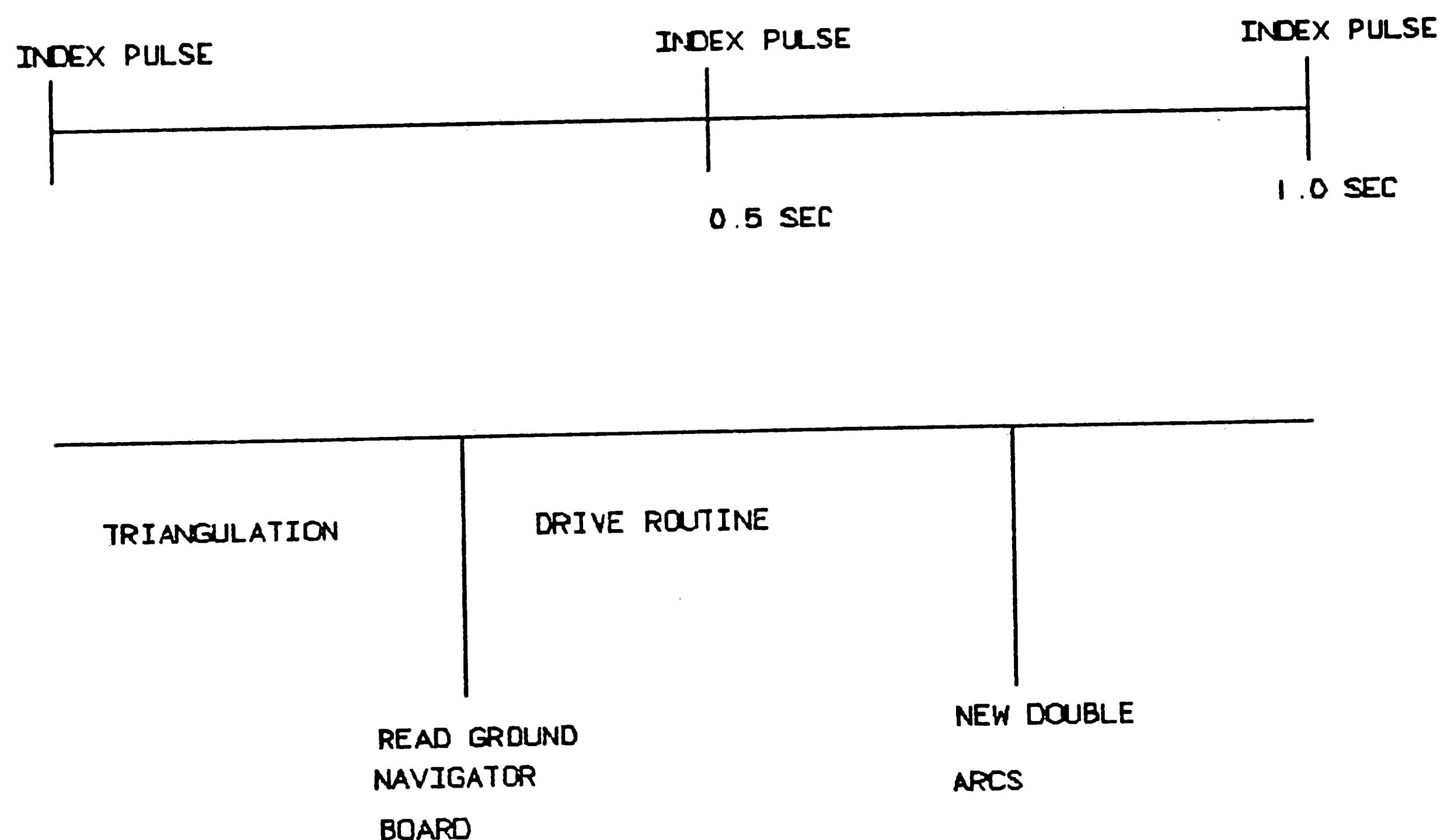


Figure 5-1: The Full Travel

At  $t_0$ , at the second index pulse, the approximate vehicle position and bearing are read from the ground navigator, with the approximate vehicle position and bearing, and the beacon map. The accurate vehicle position and bearing are computed by the goniometer routine using triangulation. The vehicle position computed by the ground navigator is corrected. Then the drive routine requests the vehicle position and bearing from the ground navigation and computes the

double arcs and steering angle which will be completed at  $t_1 + \Delta t_1$ , noting, if the prescribed vector  $\bar{B}$  is the last vector in the prescribed path, then the computer must be aware in order to stop the vehicle once it reaches that point. Once the double arcs have been computed they are sent to the drive board at the time  $t_1 + \Delta t_1$ . At the next index pulse the entire sequence is repeated. Reading last point on the path it will stop and wait for orders.

In more detail, one can summarize the sequence of events for the full drive as follows:

1. In order for the vehicle to start functioning the following inputs are loaded:
  - a. The path vectors with their associated speed.
  - b. The initial vector.
  - c. The selected nominal radius.
  - d. The direction of travel: forward / backward.
  - e. The secondary path points are computed and the ready sign issued by the vehicle if path acceptable.
  - f. First selected target vector and its speed.
  - g. First double arc computed and loaded into drive controller, this starts the run.
2. At  $t_0$ , when the second index pulse is generated, the following steps are executed:
  - a. Read the approximate vehicle position from the ground navigation board.
  - b. Compute the accurate vehicle position and bearing.
3. At time  $t_1$ :



- a. From the deviation between the approximate vehicle position and bearing, and the computed vehicle position, correct the ground navigator state, and set the gyroscope.
  - b. Estimate future vehicle position and bearing at  $t_1 + \Delta t_1$ .
  - c. Determine next target vector  $\bar{B}$ .
  - d. Compute the next double arcs leading from expected position at  $t_1 + \Delta t_1$  to  $\bar{B}$ .
4. At time  $t_1 + \Delta t_1$ :
- a. Load double arcs into drive controller. If the vector  $\bar{B}$  was the last point, go to stopping routine, step(5).
  - b. Wait for a period of  $\Delta t_2$ , to make sure the servo has stabilized.
  - c. At the next index pulse go to 2a.
5. The stopping Routine has the following steps:
- a. Reduce the speed of the vehicle at end of the first arc segment.
  - b. At the end of the second arc, stop the vehicle.

## 5.2 The Simplified Travel

Due to the uncompleted ground navigator, a simplified design was introduced to test the performance of the optical navigation system, and the drive routine. The Figure 5-2 on page 83 describes the sequence of events of the simplified version: The procedures that are activated by the index pulse generated by the goniometer, and the  $\Delta S$  pulse generated by the drive board are interrupt driven procedures. By software the program ignores the first index pulse indicating a revolution of the goniometer. At the end of the second index pulse all the interrupt procedures are masked except for the goniometer proce-

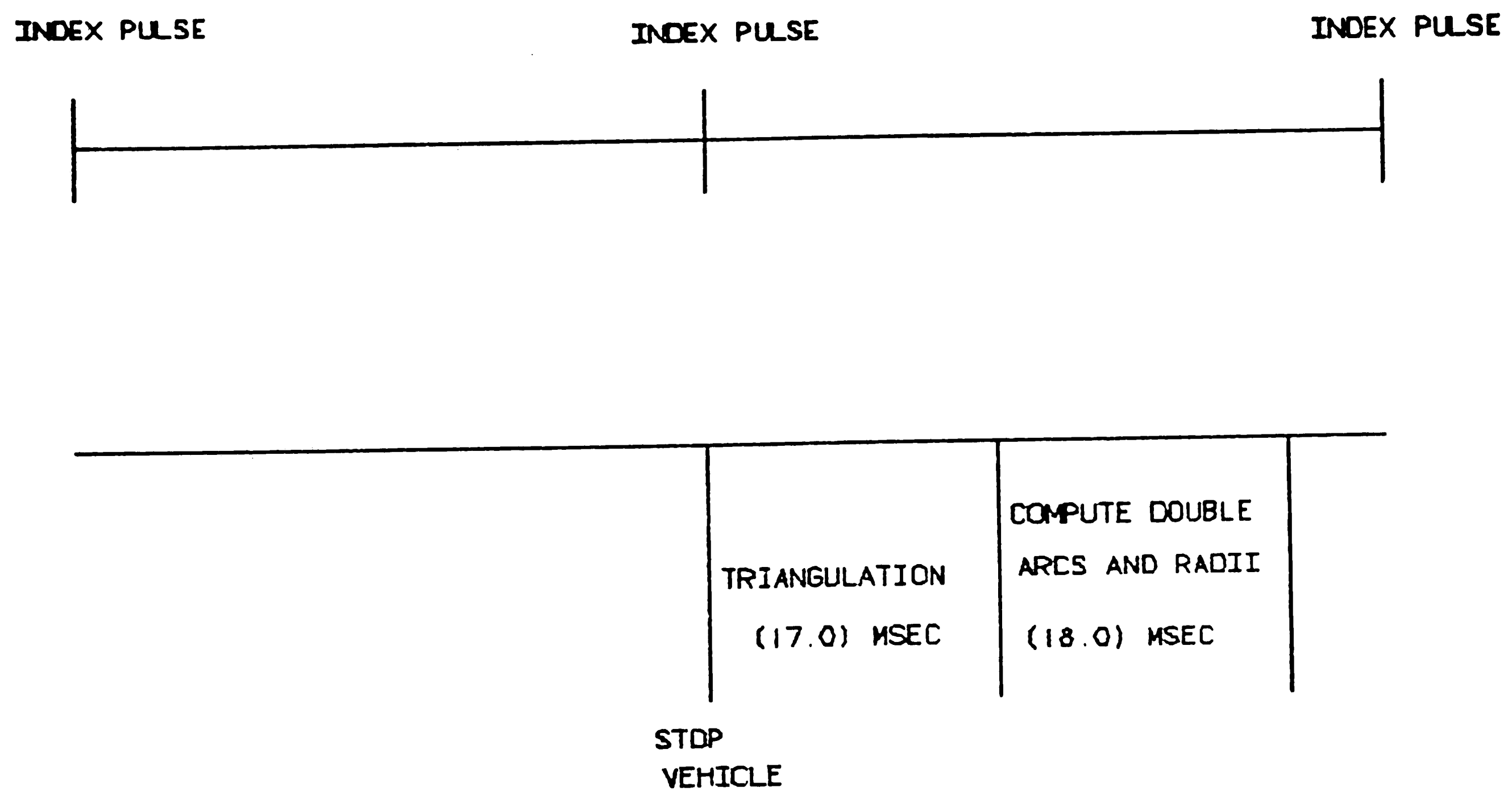


Figure 5-2: The Simplified Travel

dure that reads the beacon sighting from the ports. Once that is done the index pulse is disabled to prevent any other readings from the goniometer while computing the vehicle position and bearing, which is computed in a period of 17.0 msec. Once the computation of the vehicle position and bearing is completed, the length of the double arcs and steering angle is calculated, the length of the double arcs and steering angles is calculated within a period of 18.0 msec. Once they are completed they are down loaded to the drive controller. This starts the drive motor. The vehicle then moves towards the next point. At the end of the first circular arc a  $\Delta S$  pulse is generated by the drive board, signaling the completion of the first arc. No action is taken. At the end of the second arc a

second  $\Delta S$  pulse is generated signalling the completion of second arc. Now zeros speed is loaded into the drive board. This stops the drive motor. At the second index pulse, the cycle is repeated all over again.

The program does not handle the situation, where the second circular arc may be too small to generate a second  $\Delta S$  pulse, which can obscure the sequence of events. We overcome this problem by checking to see if the length of the second arcs is less than a minimum length that will be needed to generate a clear  $\Delta S$  pulse.

One can describe the simplified sequence of events as follows:

1. Stop the vehicle in order to start computation after the second index and  $\Delta S$  pulses.
2. Enable the routine that handles the reading of the goniometer.
3. Read beacon sighted from the specified ports.
4. Disable the index pulse in order to prevent any further reading from the goniometer during computation of the vehicle position.
5. Compute the vehicle position and bearing
6. Compute the length of the double arcs and steering angle.
7. Down load the length of the arcs, steering a angle and the speed of the vehicle to the drive board. This starts the drive motor
8. At the second  $\Delta S$  pulse generated by the drive board Signaling the completion of the second double arc, load zero velocity.
9. Select next target vector  $\bar{B}$ . If the vehicle has reached the last point in the prescribed path, stop. Otherwise go to step (2).

## Chapter 6

# Conclusions

The test bed vehicle Cyclopiion described in this thesis, is a demonstration of an optically based automated guided vehicle. As in any experiment, there are observations and suggestions to be added for future work.

As mentioned previously the procedures that were used for the vehicle software are interrupt driven procedures. During experimentation, the vehicle sometimes acted unpredictable due to noise on the bus lines. One might overcome this problem by using "State Table Control Systems", which allows the vehicle to act by matching the state in a state table and firing the next action according to a matching condition state without having to rely on any interrupts coming from the vehicle's circuitry. The National Bureau of Standards (NBS) have adopted "State Table Control Systems" for the under water vehicles very successfully [8]. Careful attention should be given to quieting the circuitry in a finalized design.

In the beacon identification process, the vehicle currently uses one beacons identification algorithm. In the future one should consider using all the identification methods mentioned in this thesis. For that and other reasons one should use a faster computer. The present clock rate is only 4 MHZ.

In addition, further research should be done in selecting the best three sets of beacons. A simulation program was written using C-Terp interfaced with HALO graphics for experimenting on different beacon selection algorithms. This package will be used for further beacon selection experimentation.

↵

Finally, due to accumulation of error in the vehicle position by the ground navigator one would like to determine the maximum accumulated error allowed, to keep the vehicle on the its correct path.

For the typical results of vehicle runs, please refer to Table 6-1 on page 87. Note that with careful adjustment of the steering system much better accuracy has been achieved.

| PRESCRIBED |       |       | MEASURED |        |        | ACTUAL |       |
|------------|-------|-------|----------|--------|--------|--------|-------|
| X          | Y     | Phi   | X        | Y      | Phi    | X      | Y     |
| 36.0       | 142.0 | 270.0 | 34.71    | 144.12 | 270.0  | 36.00  | 142.0 |
| 42.0       | 108.0 | 300.0 | 45.68    | 106.88 | 294.0  | 40.50  | 107.5 |
| 36.0       | 60.0  | 270.0 | 30.80    | 61.50  | 264.9  | 31.00  | 62.50 |
| 48.0       | 36.0  | 315.0 | 44.95    | 31.45  | 308.40 | 46.00  | 33.50 |
| 72.0       | 30.0  | 0.0   | 77.82    | 29.96  | 358.56 | 77.50  | 27.0  |
| 108.0      | 30.0  | 0.0   | 108.34   | 29.45  | 356.04 | 107.25 | 28.5  |
| 144.0      | 30.0  | 0.0   | 141.15   | 27.35  | 356.6  | 141.50 | 28.5  |

Table 6-1: The Result of A Typical Vehicle Run

## References

- [1] Rafic Bazzi, Repe Siy., *Angular Optical Scanning for Autonomous Vehicle Self Location in Constarint Environment*, Oakland University, center for Robotics and Advanced Automation intelligent systems and Machines, conference, Rochester, MI, APR 23-24,85, P. 92(5).
- [2] Rathbone R. Rodin., Robert A. Valley, Jr, and Peter J. Kindlman, "Beacon-Referenced Dead Reckoning: A Versatile Guidance System"; *Robotics Engineering*, V8, P. 11(6), December 1986.
- [3] Eberhardt N. and Wagh, M., "Cyclopion, An Autonomous Guided Vehicle for Factory Use"; *Proceedings of SPIE; Applications of Artificial Intelligence III*, Vol. 635, P. 536, April 1986.
- [4] Singh, S., "Path Planning and Navigation for a Mobil Robot"; A Thesis Presented to the Graduate Comittee of Lehigh University, June 1985.
- [6] Marshall, C., "Hardware Design for an Intelligent Vehicle"; A Thesis Presented to the Graduate Comittee of Lehigh University, June 1986.
- [7] Shaub-Wen Ma., "Active Optical Detection Of Retroreflectors"; A Thesis to be Presented to the Graduate Committee of Lehigh University, 1987.
- [8] Barbera j. Anthony, M.L. Fitzgerald, James S. Albus, Leonard S. Haynes. "RCS: The NBS Real-Time Control System", *Proceedings of Robots 8 conference and Exposition*, Detroit, Michigan, June 6, 1984.

**Appendix A**  
**The Goniometer Software**



\*\*\*\*\* DIST \*\*\*\*\*

The procedure computes the distance between beacon points  
X1, Y1) and (X2, Y2).

\*\*\*\*\*/

```
DIST: PROCEDURE(X1, Y1, X2, Y2) REAL;  
  DECLARE (X1, Y1, X2, Y2) REAL;  
  RETURN(mqerY2X((X2-X1)*(X2-X1) + ((Y2-Y1)*(Y2-Y1)), 0.5));  
END DIST;
```

\*\*\*\*\* CROSS \*\*\*\*\*

The procedure computes the cross product of two vectors.

\*\*\*\*\*/

```
CROSS: PROCEDURE(X1, Y1, X2, Y2) REAL;  
  DECLARE(X1, Y1, X2, Y2) REAL;  
  RETURN((X1*Y2) - (Y1*X2));  
END CROSS;
```

/\*\*\*\*\*\* DOT \*\*\*\*\*

The procedure computes the DOT product of two vectors.

\*\*\*\*\*/

```
DOT: PROCEDURE(X1, Y1, X2, Y2) REAL;  
  DECLARE (X1, Y1, X2, Y2) REAL;  
  RETURN((X1*X2) + (Y1*Y2));  
END DOT;
```

/\*\*\*\*\*\* MAG \*\*\*\*\*

The procedure computes the MAGNITUDE of a vector.

\*\*\*\*\*/

```
MAG: PROCEDURE(X, Y) REAL;  
  DECLARE (X, Y) REAL;  
  RETURN(mqerY2X((X*X + Y*Y), 0.5));  
END MAG;
```

```

/***** SHUFFPOINTS *****/
The procedure computes the angles, alpha1, alpha2, alpha3,
and exchanges the beacons coordinates if necessary
*****/

```

```

SHUFFPOINTS: PROCEDURE(ANG1, ANG2, ANG3);
DECLARE (ANG1, ANG2, ANG3) REAL;
DECLARE (TY, TX) REAL;

ANGLE1 = ANG2 - ANG1;
IF ANGLE1 >= 180.0
THEN DO;
    ANGLE1 = ANG3 - ANG2;
    ANGLE2 = ANG1 - ANG3 + 360.0;
    TX = X1;
    TY = Y1;
    X1 = X2;
    Y1 = Y2;
    X2 = X3;
    Y2 = Y3;
    X3 = TX;
    Y3 = TY;
END;
ELSE DO;
    ANGLE2 = ANG3 - ANG2;
    IF ANGLE2 >= 180.0
    THEN DO;
        ANGLE1 = ANG1 - ANG3 + 360.0;
        ANGLE2 = ANG2 - ANG1;
        TX = X1;
        TY = Y1;
        X1 = X3;
        Y1 = Y3;
        X3 = X2;
        Y3 = Y2;
        X2 = TX;
        Y2 = TY;
    END;
END;
ANGLE3 = ANGLE2 + ANGLE1;
END SHUFFPOINTS;

```

```
/****** EXPECTED_ANGLE*****
```

The procedure calculates the expected angle.

```
*****/
```

```
EXPECTED_ANGLE: PROCEDURE(POSX, POSY) REAL;
```

```
    DECLARE(POSX, POSY) REAL;  
    DECLARE(ARG1, ARG2, ANGLE, BEAR) REAL;
```

```
    BEAR = BEARING * (PI / 180.0);
```

```
    ARG1 = POSX - APRX;
```

```
    ARG2 = POSY - APY;
```

```
    IF ARG1 = 0.0
```

```
        THEN DO;
```

```
            IF ARG2 > 0.0
```

```
                THEN ANGLE = PI / 2.0;
```

```
            ELSE IF ARG2 < 0.0
```

```
                THEN ANGLE = 1.5 * PI;
```

```
            ELSE IF ARG2 = 0.0
```

```
                THEN ANGLE = 0.0;
```

```
        END;
```

```
    ELSE DO;
```

```
        IF ARG1 > 0.0 THEN DO;
```

```
            IF ARG2 >= 0.0 THEN
```

```
                ANGLE = mqrATN(ARG2 / ARG1) - BEAR;
```

```
            ELSE
```

```
                ANGLE = mqrATN(ARG2 / ARG1) - BEAR +  
                    (2.0 * PI);
```

```
        END;
```

```
    ELSE
```

```
        ANGLE = PI + mqrATN(ARG2 / ARG1) - BEAR;
```

```
    END;
```

```
    IF ANGLE < 0.0
```

```
        THEN
```

```
            RETURN((2.0 * PI + ANGLE) * 180.0 / PI);
```

```
    ELSE
```

```
        RETURN(ANGLE * (180.0/PI));
```

```
END EXPECTED_ANGLE;
```

```
/****** COMPUTE_VEH_POSITION *****
```

The procedure does the following steps:

- 1- Checks if the number of identified beacons are equal to three if so, it use those beacon for traingulation by going to step3.
- 2- Otherwise it selects the three best beacons by calling procedure "TRIPLET".
- 3- Then it calls procedure shuffpoints.
- 4- Call procedure "CALACPOINTS" to determine the vehicle position.

```
*****/
```

```

COMPUTE_VEH_POSITION: PROCEDURE(N, LIMIT);
  DECLARE N INTEGER;
  DECLARE(ANG1, ANG2, ANG3) REAL;

  IF N = 3 THEN DO;
    X1 = XPOINTS(1);      Y1 = YPOINTS(1);
    X2 = XPOINTS(2);      Y2 = YPOINTS(2);
    X3 = XPOINTS(3);      Y3 = YPOINTS(3);
    CALL SHUFFPOINTS(THETAS(1), THETAS(2), THETAS(3));
    CALL CALCPOINTS(ANGLE1, ANGLE2);
  END;
  ELSE DO;
    CALL TRIPLET(N, LIMIT);
    X1 = XX(1);      X2 = XX(2);      X3 = XX(3);
    Y1 = YY(1);      Y2 = YY(2);      Y3 = YY(3);
    ANG1 = AG(1);    ANG2 = AG(2);    ANG3 = AG(3);
    CALL SHUFFPOINTS(ANG1, ANG2, ANG3);
    CALL CALCPOINTS(ANGLE1, ANGLE2);
  END;
END COMPUTE_VEH_POSITION;

```

```

/***** TRIPLET *****/
The procedure finds the a set of triplet beacon points to determine
the coordinates of the vehicle.
/*****/

```

```

TRIPLET: PROCEDURE(N, LIMIT);
  DECLARE(N, MATCH, I, J, K, LL) INTEGER;
  DECLARE(ANG1, ANG2, ANG3, LIMIT, MIN) REAL;
  DECLARE(XX, YY, AG, DIFF) (20) REAL;

  MATCH = 0;
  DO I = 1 TO N;
    XX(I) = XPOINTS(I);
    YY(I) = YPOINTS(I);
    AG(I) = THETAS(I);
  END;
  DO I = 1 TO N;
    DIFF(I) = AG(I+1) - AG(I);
    IF I = N THEN DO;
      DIFF(I) = AG(1) - AG(I);
      IF DIFF(I) < 0.0 THEN
        DIFF(I) = DIFF(I) + 360.0;
    END;
  END;
  DO LL = 1 TO N;
    I = 1;
    MIN = DIFF(I);
    J = I;
    DO I = 2 TO N;
      IF DIFF(I) < MIN THEN DO;
        MIN = DIFF(I);
        J = I;
      END;
    END;
  END;
  DO I = 1 TO N;
    IF I = J THEN DO;
      IF (J > 1) AND (J < N) THEN DO;
        IF DIFF(J-1) < DIFF(J+1) THEN DO;
          DIFF(J-1) = DIFF(J-1) + DIFF(J);
          DO K = I TO N-1;
            DIFF(K) = DIFF(K+1);
            XX(K) = XX(K+1);
            YY(K) = YY(K+1);
            AG(K) = AG(K+1);
          END;
        END;
      END;
    END;
  END;

```

```

        END;
    END;
ELSE DO;
    DIFF(J) = DIFF(J) + DIFF(J+ 1);
    DO K = I + 1 TO N-1 ;
        DIFF(K) = DIFF(K+1);
        XX(K) = XX(K+1);
        YY(K) = YY(K+1);
        AG(K) = AG(K+1);
    END;
END;
I = N;

END;
ELSE
IF J = 1 THEN DO;
IF DIFF(N) > DIFF(J + 1) THEN DO;
    DIFF(J) = DIFF(J) + DIFF(J + 1);
    DO K = 2 TO N-1;
        DIFF(K) = DIFF(K+1);
        XX(K) = XX(K+1);
        YY(K) = YY(K+1);
        AG(K) = AG(K+1);
    END;
END;
ELSE DO;
    DIFF(N) = DIFF(J) + DIFF(N);
    DO K = 1 TO N-1;
        DIFF(K) = DIFF(K+1);
        XX(K) = XX(K+1);
        YY(K) = YY(K+1);
        AG(K) = AG(K+1);
    END;
END;
I = N;
END;
ELSE
IF J = N THEN DO;
    IF DIFF(N-1) > DIFF(1) THEN DO;
        DIFF(N-1) = DIFF(1) + DIFF(N);
        DO K = 1 TO N-1;
            DIFF(K) = DIFF(K+1);
            XX(K) = XX(K+1);
            YY(K) = YY(K+1);
            AG(K) = AG(K+1);
        END;
    END;
ELSE
    DIFF(J-1) = DIFF(J-1) + DIFF(I);
END;
I = N;
END;
N = N-1;
IF N = 3 THEN
    LL = N;
END;
END;
END;
END;
END TRIPLET;

```

```
/****** EXPECTED_GAMA *****/
```

```
The procedure calculates the expected GAMA.
```

```
*****/
```

```
EXPECTED_GAMA: PROCEDURE(POSX, POSY) REAL;  
  DECLARE(POSX, POSY) REAL;  
  DECLARE(ARG1, ARG2, ANGLE, BEAR) REAL;  
  
  BEAR = BEARING * (PI / 180.0);  
  ARG1 = POSX - APRX;  
  ARG2 = POSY - APRY;  
  IF ARG1 = 0.0  
    THEN DO;  
      IF ARG2 > 0.0  
        THEN ANGLE = PI / 2.0;  
        ELSE IF ARG2 < 0.0  
          THEN ANGLE = 1.5 * PI;  
          ELSE IF ARG2 = 0.0  
            THEN ANGLE = 0.0;  
      END;  
    ELSE DO;  
      IF ARG1 > 0.0 THEN DO;  
        IF ARG2 >= 0.0 THEN  
          ANGLE = mqrATN(ARG2 / ARG1);  
        ELSE  
          ANGLE = mqrATN(ARG2 / ARG1) +  
            (2.0 * PI);  
        END;  
      ELSE  
        ANGLE = PI + mqrATN(ARG2 / ARG1);  
      END;  
    IF ANGLE < 0.0  
      THEN  
        RETURN((2.0 * PI + ANGLE) * 180.0 / PI);  
    ELSE  
      RETURN(ANGLE * (180.0/PI));  
  END EXPECTED_GAMA;
```

```
/****** FINDCORD *****/
```

```
The procedure finds the coordinates of the position of the  
vehicle.
```

```
*****/
```

```
FINDCORD: PROCEDURE(CX1, CX2, CY1, CY2, XC, YC);  
  DECLARE(CX1, CX2, CY1, CY2, XC, YC) REAL;  
  DECLARE(R1, R2, M) REAL;  
  
  R1 = DIST(CX1, CY1, X1, Y1);  
  R2 = R1 * R1;  
  IF R2 < 5.0 THEN DO;  
    XCORD = 0.0;  
    YCORD = 0.0;  
  END;  
  ELSE IF ABS(CX2 - CX1) < 0.01
```

```

THEN DO;
  YCORD = YC;
  XCORD = 2.0 * CX1 - XC;
END;
ELSE IF ABS(CY2 - CY1) < 0.01
THEN DO;
  YCORD = 2.0 * CY1 - YC;
  XCORD = XC;
END;
ELSE DO;
  M = (CY2 - CY1) / (CX2 - CX1);
  XCORD = (2.0 * (YC - CY1 + XC * (1.0/M)) + CX1
    * M) / (M + (1.0 / M)) - XC;
  YCORD = ((XC - XCORD) * (1.0/M)) + YC;
END;

```

END FINDCORD;

/\*\*\*\*\*\* FINDCENTER \*\*\*\*\*\*/

The procedure finds the center of a circle.  
 /\*\*\*\*\*\*/

```

FINDCENTER: PROCEDURE(PX1, PY1, PX2, PY2, ANGLE);
  DECLARE(PX1, PY1, PX2, PY2, ANGLE) REAL;
  DECLARE(MIDX, MIDY, U, LEG1, SLOPE) REAL;

```

```

  MIDX = (PX1 + PX2) / 2.0;
  MIDY = (PY1 + PY2) / 2.0;
  U = mperCOS(DTOR * ANGLE) / mperSIN(DTOR * ANGLE);
  CENTERX = MIDX + (PY1 - PY2) * (U / 2.0);
  CENTERY = MIDY - (PX1 - PX2) * (U / 2.0);

```

END FINDCENTER;

/\*\*\*\*\*\* CALCPOINTS \*\*\*\*\*\*/

The procedure goes through the following steps:

1- computes the center of the 3 circles.

2- for every two circles, it computes the vehicle position  
 and bearing, resulting in 3 vehicle coordinates.  
 /\*\*\*\*\*\*/

```

CALCPOINTS: PROCEDURE(ANGLE1, ANGLE2);
  DECLARE(ANGLE1, ANGLE2) REAL;
  DECLARE(CX1, CX2, CX3, CY1, CY2, CY3) REAL;

```

```

  /* Compute the center of circle A using the coordinates
  of beacons 1 & 2 */

```

```

  CALL FINDCENTER(X1, Y1, X2, Y2, ANGLE1);
  CX1 = CENTERX;
  CY1 = CENTERY;

```

```

  /* Compute the center of circle B using the coordinates
  of beacons 2 & 3 */

```

```

  CALL FINDCENTER(X2, Y2, X3, Y3, ANGLE2);
  CX2 = CENTERX;
  CY2 = CENTERY;

```

```

  /* Compute the center of circle c using the coordinates
  of beacons 1 & 3 */

```

```

IF ANGLE3 > 180.0 THEN DO;
    ANGLE3 = 360.0 - ANGLE3;
    CALL FINDCENTER(X3,Y3,X1,Y1,ANGLE3);
END;
ELSE
    CALL FINDCENTER(X1,Y1,X3,Y3,ANGLE3);
CX3 = CENTERX;
CY3 = CENTERY;

/**** Find the vehicle coordinates using circle A & B ****/
CALL FINDCORD(CX1,CX2,CY1,CY2,X2,Y2);
XCORD1 = XCORD;
YCORD1 = YCORD;

/**** Find the vehicle coordinates using circle A & C ****/
CALL FINDCORD(CX1,CX3,CY1,CY3,X1,Y1);
XCORD2 = XCORD;
YCORD2 = YCORD;

/**** Find the vehicle coordinates using circle B & C ****/
CALL FINDCORD(CX2,CX3,CY2,CY3,X3,Y3);
XCORD3 = XCORD;
YCORD3 = YCORD;

VEH_POS_X = (XCORD1 + XCORD2 + XCORD3) / 3.0;
VEH_POS_Y = (YCORD1 + YCORD2 + YCORD3) / 3.0;
END CALCPOINTS;

```

```

/***** IDENTIFY_BEACON *****/
The procedure identifies beacon within a certain angle and distance
limit
*****/

```

```

IDENTIFY_BEACON: PROCEDURE(NUMB,NUMBS);
    DECLARE(NUMB,NUMBS,I,CHECK ) INTEGER;
    DECLARE(X,Y,ANG) REAL;
    DECLARE EXANG(20) REAL;

    NUMMATCHED = 0;
    DO I = 1 TO NUMB;
        EXANG(I) = EXPECTED_ANGLE(XP(I),YP(I));
    DO I = 1 TO NUMBS;
        CHECK = 0;
        DO K = 1 TO NUMB;
            IF ABS(EXANG(K) - THETA(I)) <= THRESH1 THEN DO;
                CHECK = CHECK + 1;
                X = XP(K);
                Y = YP(K);
                ANG = THETA(I);
            END;
        END;
        IF CHECK = 1 THEN DO;
            NUMMATCHED = NUMMATCHED + 1;
            XPOINTS(NUMMATCHED) = X;
            YPOINTS(NUMMATCHED) = Y;
            THETAS(NUMMATCHED) = ANG;
        END;
    END;
END IDENTIFY_BEACON;

```



```

GONIOMETER: PROCEDURE;
COUNTER1 = 0.0;
RTOD = 180.0 /PI;
DTOR = PI /180.0;

$INCLUDE(:F1:BEARING.DAT)      /* File that contains the bearing */
/* at each point                */
$INCLUDE(:F1:MAP.DAT)         /* Beacon coordinates map        */
$INCLUDE(:F1:PATH.DAT)       /* Prescribed points              */
NUMBEACONS = 9;
IF REV = 1 THEN DO;
    BEARING = BEA(II + 1);
    APRX = POINTSX(II + 1);
    APRY = POINTSY(II + 1);
END;
IF REV = 0 THEN DO;
    BEARING = BEA(II - 1);
    APRX = POINTSX(II - 1);
    APRY = POINTSY(II - 1);
END;
CALL IDENTIFY_BEACON(NUMBEACONS, NUMSEEN);
CALL COMPUTE_VEH_POSITION(NUMMATCHED, 120.0);
APRX = VEH_POS_X;
APRY = VEH_POS_Y;
SUMSI = 0.0;

GAMA = EXPECTED_GAMA(XPOINTS(1), YPOINTS(1));
BEARING = (GAMA - THETAS(1)) * (PI/180.0);
IF BEARING < 0.0 THEN
    BEARING = BEARING + (2.0 * PI);

END GONIOMETER;

```

Appendix B  
The Drive Routine

~S.

```

/***** COMPUTE_PARAMETERS *****/

* The procedure computes the parameters needed for
* for double arc computations. It also computes the
* angles Phi and Beta
*****/

COMPUTE_PARAMETERS: PROCEDURE(PX1, PY1, PX2, PY2, DX1, DY1, DX2, DY2);

    DECLARE (PX1, PY1, PX2, PY2, DX1, DY1, DX2, DY2) REAL;
    DECLARE (C, S, P, PP, ALPHA) REAL;

    FAIL = FALSE;
    TRANS = CROSS(DX1, DY1, PX2-PX1, PY2-PY1); /* T = VECTOR A X VECTOR D
                                                D*/

    IF TRANS <= 0.0
    THEN TRANSLATION = RIGHT;
    ELSE TRANSLATION = LEFT;
    P = DOT(DX1, DY1, PX2-PX1, PY2-PY1);
    IF P < 0.0
    THEN FAIL = TRUE; /* Beta out of range */
    ALPHA = ABS(TRANS) / (MAG(DX1, DY1) * MAG(PX2-PX1, PY2-PY1));
    IF TRANS = 0.0 THEN TRANS = 0.01;
    BIGA = ABS(TRANS) / P;
    BETA = mgerATN(BIGA); /* ARCTAN OF A */
    C = CROSS(DX1, DY1, DX2, DY2);
    IF C < 0.0
    THEN ROTATION = RIGHT;
    ELSE ROTATION = LEFT;
    S = DOT(DX1, DY1, DX2, DY2);
    IF ABS(S) < THRESH
    THEN S = 0.0;
    IF S < 0.0
    THEN FAIL = TRUE; /* PHI OUT OF RANGE */
    IF FAIL = FALSE THEN DO;
        CPHI = S / (MAG(DX1, DY1) * MAG(DX2, DY2));
        SPHI = ABS(C) / (MAG(DX1, DY1) * MAG(DX2, DY2));
        IF S = 0.0
        THEN PHI = PI/2.0;
        ELSE PHI = mgerATN(ABS(C)/S);
        A = ABS(TRANS) / MAG(DX1, DY1);
        IF C <> 0.0 THEN
            DD = (P - ABS(TRANS) * S / ABS(C)) / MAG(DX1, DY1);
        IF SPHI <> 0.0 THEN
            H = A / SPHI;
    END;
END COMPUTE_PARAMETERS;

```

```

/***** TWOC1 *****/
* The procedure computes Case (1) of double arcs *
-----

```

```

TWOC1: PROCEDURE(PHI, CPHI, SPHI, A, bigA, BETA, FLAG1);
  DECLARE (PHI, CPHI, SPHI, A, bigA, BETA) REAL;
  DECLARE (FLAG1) BYTE;
  DECLARE (PSI, ALPHA) REAL;

  ALPHA = (CPHI + bigA*SPHI + 1.0) /
           (2.0 * mqrY2X(1.0 + (bigA*bigA), 0.5));
  IF ALPHA = 0.0 THEN
    PSI = PI / 2.0;
  ELSE
    PSI = mqrACS(ALPHA) + BETA;
    RADIUS1 = A / (CPHI - 2.0 * mqrCOS(PSI) + 1.0);
    RADIUS2 = RADIUS1;
    ARC1 = RADIUS1 * PSI;
    ARC2 = RADIUS1 * (PSI - PHI);
    IF FLAG1 = TRUE THEN DO;
      TURN1 = LEFT;
      TURN2 = RIGHT;
      RADIUS1 = -RADIUS1;
    END;
  ELSE DO;
    TURN1 = RIGHT;
    TURN2 = LEFT;
    RADIUS2 = -RADIUS1;
  END;
END TWOC1;

```

```

/***** TWOC2 *****/
* The procedure computes Case (2) of double arcs *
-----

```

```

TWOC2: PROCEDURE(PHI, CPHI, SPHI, A, BIGA, BETA, FLAG1);

  DECLARE (PHI, CPHI, SPHI, A, BIGA, BETA) REAL;
  DECLARE (FLAG1) BYTE;
  DECLARE (PSI, ALPHA) REAL;

  ALPHA = (CPHI + BIGA*SPHI+1.0) /
           (2.0* mqrY2X(1.0 +(BIGA*BIGA), 0.5));
  IF ALPHA = 0.0 THEN
    PSI = PI/2.0;
  ELSE
    PSI = mqrACS(ALPHA) - BETA;
    RADIUS1 = A / (2.0* mqrCOS(PSI) - CPHI - 1.0);
    RADIUS2 = RADIUS1;
    ARC1 = RADIUS1 * PSI;
    ARC2 = RADIUS1 * (PSI + PHI);
    IF FLAG1 = FALSE THEN DO;
      TURN1 = LEFT;
      TURN2 = RIGHT;
      RADIUS1 = -RADIUS1;
    END;
  ELSE DO;
    TURN1 = RIGHT;
    TURN2 = LEFT;
    RADIUS2 = - RADIUS1;
  END;
END TWOC2;

```

```

/***** TWOC3 *****/
* The procedure computes Case (3) of double arcs *
*****/

```

```

TWOC3: PROCEDURE(PHI, CPHI, SPHI, A, BIGA, BETA, FLAG1);
DECLARE (PHI, SPHI, CPHI, A, BIGA, BETA) REAL;
DECLARE (PSI, ALPHA) REAL;
DECLARE (FLAG1) BYTE;

ALPHA = (-1.0*CPHI+BIGA*SPHI-1.0)/
        (2.0*mqerY2X(1.0+(BIGA*BIGA),0.5));
IF ALPHA = 0.0 THEN
  PSI = PI/2.0;
ELSE
  PSI = mqerACS(ALPHA) - BETA;
RADIUS1 = A/(2.0* mqerCOS(PSI) + CPHI + 1.0);
RADIUS2 = RADIUS1;
ARC1 = RADIUS1 * (PI - PSI);
ARC2 = RADIUS1 * (PI + PHI - PSI);
IF FLAG1 = TRUE THEN DO;
  TURN1 = LEFT;
  TURN2 = RIGHT;
  RADIUS1 = -RADIUS1;
END;
ELSE DO;
  TURN1 = RIGHT;
  TURN2 = LEFT;
  RADIUS2 = - RADIUS1;
END;

END TWOC3;

```

```

/***** ONEC1 *****/
* The procedure computes Case (4) of double arcs *
* and The anle PHI < 2 Beta *
*****/

```

```

ONEC1: PROCEDURE(MIN_R, A, D, H, PHI, FL, CP, SP, BA, BETA);
DECLARE (MIN_R, A, D, H, PHI, SP, CP, BA, BETA) REAL;
DECLARE TPHI2 REAL;
DECLARE (FL) BYTE;

TPHI2 = mqerTAN(PHI/2.0);
IF TPHI2 = 0.0 THEN
  RADIUS1 = MAX_NUM;
ELSE
  RADIUS1 = D / TPHI2;
IF RADIUS1 < MIN_RADIUS THEN
  CALL TWOC1(PHI, CP, SP, A, BA, BETA, FL);
ELSE DO;
  ARC1 = RADIUS1 * PHI;
  ARC2 = H - D;
  RADIUS2 = MAX_NUM;
  IF FL = TRUE THEN DO;
    TURN1 = LEFT;
    RADIUS1 = -RADIUS1;
  END;
  ELSE
    TURN1 = RIGHT;
    TURN2 = STRAIGHT;
  END;
END;

END ONEC1;

```

```

/***** ONEC2 *****/
* The procedure computes Case (5) of double arcs
* and The anle PHI > 2 Beta
-----/

```

```

ONEC2: PROCEDURE(MIN_R, A, D, H, PHI, FL, CP, SP, BA, BETA);
DECLARE (MIN_R, A, D, PHI, CP, SP, BA, BETA) REAL;
DECLARE (FL) BYTE;
DECLARE (TPHI2, H) REAL;

```

```

TPHI2 = mgerTAN(PHI/2.0);
IF TPHI2 = 0.0 THEN
  RADIUS2 = MAX_NUM;
ELSE
  RADIUS2 = H/TPHI2;
IF RADIUS2 < MIN_RADIUS THEN
  CALL TWOC2(PHI, CP, SP, A, BA, BETA, FL);
ELSE DO;
  ARC1 = D - H;
  RADIUS1 = MAX_NUM;
  TURN1 = STRAIGHT;
  ARC2 = RADIUS2* PHI;
  IF FL = TRUE THEN DO;
    TURN2 = LEFT;
    RADIUS2 = -RADIUS2;
  END;
  ELSE
    TURN2 = RIGHT;
END;

```

```

END ONEC2;

```

```

/***** DRIVE *****/
* The procedure does the following steps:
*
* 1- Call the procedure COMPUTE_PARAMETERS .
* 2- It select the correct case of double arcs.
* 3- Computes the steering angles
-----/

```

```

DRIVE: PROCEDURE(MIN_RAD, PX1, PY1, PX2, PY2, DX1, DY1, DX2, DY2);
DECLARE (MIN_RAD, PX1, PY1, PX2, PY2, DX1, DY1, DX2, DY2) REAL;
DECLARE (RTOD) REAL;

```

```

FLAG = FALSE;
RTOD = 180.0 / PI;
CALL COMPUTE_PARAMETERS(PX1, PY1, PX2, PY2, DX1, DY1, DX2, DY2);
IF PHI > PI/2.0 THEN
  PHI = PI/2.0;
IF (PHI < THRESH) AND (BETA < THRESH) THEN DO;
  RADIUS1 = MAX_NUM;
  ARC1 = MAG(PX2-PX1, PY2-PY1)/2.0;
  ARC2 = ARC1;
  TURN1 = STRAIGHT;
  TURN2 = STRAIGHT;
END;
ELSE DO;
  IF NOT FAIL THEN DO;
    IF TRANSLATION = LEFT THEN
      FLAG = TRUE;
    ELSE
      FLAG = FALSE;
    IF ((TRANSLATION = LEFT) AND (ROTATION = RIGHT)) OR
      ((TRANSLATION = RIGHT) AND (ROTATION = LEFT)) THEN
      CALL TWOC3(PHI, CPHI, SPHI, A, BIGA, BETA, FLAG);
  END;

```

```

ELSE DO;

```

```

      IF ABS(PHI) < ABS(2.0 * BETA) THEN
        CALL ONEC1(MIN_RAD, A, DD, H, PHI, FLAG, CPHI, SPHI,
                  BIGA, BETA);
      ELSE
        CALL ONEC2(MIN_RAD, A, DD, H, PHI, FLAG, CPHI,
                  , SPHI, BIGA, BETA);
      END;
    END;

  END;

  END;
  IF REV = 1 THEN DO;
    RADIUS1 = -RADIUS1;
    RADIUS2 = -RADIUS2;
  END;

  IF RADIUS1 = ABS(MAX_NUM) THEN
    STEERING_ANGLE1 = 128;
  ELSE
    STEERING_ANGLE1 = FIX(81.49 *mgerATN(30.0/RADIUS1) + 128.0);
  IF RADIUS2 = ABS(MAX_NUM) THEN
    STEERING_ANGLE2 = 128;
  ELSE
    STEERING_ANGLE2 = FIX(81.49 *mgerATN(30.0/RADIUS2) + 128.0);
    /** 256/ PI = 81.49 **/
  ARC_LENGTH1 =FIX(4.4*ARC1*mgerY2X(1.0+(900.0/(RADIUS1*RADIUS1)),0.5));
  ARC_LENGTH2 =FIX(4.4*ARC2*mgerY2X(1.0+(900.0/(RADIUS2*RADIUS2)),0.5));

  END DRIVE;

  /***** OUTPUT_TO_PORTS *****/
  * The procedure outputs the length of the double arcs
  * , steering angles, and speed to the drive board.
  *****/

  OUTPUT_TO_PORTS : PROCEDURE;

  OUTPUT(PORTC) = 002H;
  OUTPUT(C8255) = 080H;

  OUTPUT(PORTA) = LOW(UNSIGN(STEERING_ANGLE1)) XOR 0FFH;
  /** OUTPUT STEERING ANGLE ***/
  OUTPUT(PORTB) = 40H;
  OUTPUT(PORTB) = 00H;

  OUTPUT(PORTA) = LOW(UNSIGN(ARC_LENGTH1)) XOR 0FFH;
  /** OUTPUT ARC LENGTH ***/
  OUTPUT(PORTB) = 20H;
  OUTPUT(PORTB) = 00H;

  OUTPUT(PORTA) = LOW(UNSIGN(STEERING_ANGLE2)) XOR 0FFH;
  /** OUTPUT STEERING ANGLE ***/
  OUTPUT(PORTB) = 50H;
  OUTPUT(PORTB) = 00H;

  OUTPUT(PORTA) = LOW(UNSIGN(ARC_LENGTH2)) XOR 0FFH;
  /** OUTPUT ARC LENGTH ***/
  OUTPUT(PORTB) = 30H;
  OUTPUT(PORTB) = 00H;

  D.BUFF(2) = 28H; /** SPEED VARIABLE MUST BE DECLARED 40 ***/
  OUTPUT(PORTA) = D.BUFF(2) XOR 0FFH;
  OUTPUT(PORTB) = 60H;

```

```

OUTPUT(C8255) = 90H;
OUTPUT(PORTC) = 82H;

END OUTPUT_TO_PORTS;

/* begin drr */
DRR: PROCEDURE;
NUMPOINTS = 7;
FF2 = 1;          /* Set the flag for the delts s pulse */

/* Cpmute the direction of the vehicle at the distenation point */
DIR_X(II)      = 40.0 * mgerCOS(BEA(II) * PI/180.0);
DIR_Y(II)      = 40.0 * mgerSIN(BEA(II) * PI/180.0);

MIN_RADIUS = 10.0;
PRED_DIR = CALCANGLE(PRED_DX, PRED_DY);

/* Cpmute the direction of the vehicle at the current point */
VEH_DI_X = 40.0 * mgerCOS(BEARING);
VEH_DIR_Y = 40.0 * mgerSIN(BEARING);
IF REV = 0 THEN DO;

    SP1          = FIX(MAG(DIR_X(II), DIR_Y(II)));
    SPEED        = LOW(UNSIGN(SP1));

    CALL DRIVE(MIN_RADIUS, VEH_POS_X, VEH_POS_Y, POINTSX(II), POINTSY(II),
              VEH_DIR_X, VEH_DIR_Y, DIR_X(II), DIR_Y(II));
    CALL OUTPUT_TO_PORTS;

END;
ELSE DO;

    SP1          = FIX(MAG(VEH_DIR_X, VEH_DIR_Y));
    SPEED        = LOW(UNSIGN(SP1));
    CALL DRIVE(MIN_RADIUS, VEH_POS_X, VEH_POS_Y, POINTSX(II), POINTSY(II),
              -VEH_DIR_X, -VEH_DIR_Y, -DIR_X(II), -DIR_Y(II));
    CALL OUTPUT_TO_PORTS;

END;
END DRR;

```



## Biography

The author, *Mona Ahmad Elgayar*, was born on November 05, 1956 to Mr. Ahamad Elgayar and Mrs. Wedad Abdel Aziz Nada. Her undergraduate work was done at Kuwait University where she received a *B.S* in Computer Science in 1978. She, then worked at Kuwait Institute for Scientific Research for five years as a computer programmer, where she developed software for different applications, among them software for an industrial project and helped in designing a database system for *OAPEC* (Organization of Arabic Petroleum Countries). Her graduate work was completed at Lehigh University where she was involved in research to develop the software for an optically based, automated guided vehicle navigation system. She will receive a *M.S.* from Lehigh University in Computer Science in 1987.