

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A248 252



DTIC
ELECTE
APR 07 1992
S D D

THESIS

PATH TRACKING USING SIMPLE PLANAR CURVES

by

LT Richard James Abresch

March 1992

Thesis Advisor:

Yutaka Kanayama

Approved for public release; distribution is unlimited.

92-08903



92 4 06 16 4

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY		3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4 PERFORMING ORGANIZATION REPORT NUMBER(S)		5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS	
		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) PATH TRACKING USING SIMPLE PLANAR CURVES (U)			
12. PERSONAL AUTHOR(S) Abresch, Richard James			
13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM 03/89 TO 03/92	14 DATE OF REPORT (Year, Month, Day) March 1992	15 PAGE COUNT 103
16. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
		Path Planning, Obstacle Avoidance, Autonomous Vehicle Motion	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This thesis presents a method of controlling an autonomous vehicle's motion in a two dimensional environment. Its' purpose is to expand the functionality of a vehicle's motion by complementing a point to point path planning scheme with a path to path scheme. The method introduced in this paper will use the vehicle's position and the desired reference path to calculate the necessary curvature to effect movement onto the desired reference path. The reference path will be a simple planar curve, such as, a circle or line. After successful testing of an operating algorithm, the method shall be incorporated into a robot's software system. This path tracking method will lay the groundwork for a dynamic obstacle avoidance system for a mobile robot.			
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS		21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Yutaka Kanayama		22b. TELEPHONE (Include Area Code) (408) 646-2095	22c. OFFICE SYMBOL CS/Ka

Approved for public release; distribution is unlimited

***PATH TRACKING
USING SIMPLE PLANAR CURVES***

by
Richard James Abresch
Lieutenant, USN
B.S., United States Naval Academy, 1985


Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

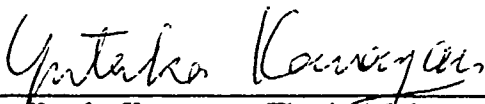
NAVAL POSTGRADUATE SCHOOL
March 1992

Author:

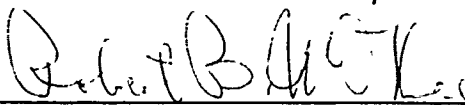


Richard James Abresch

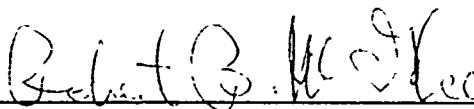
Approved By:



Yutaka Kanayama, Thesis Advisor



Robert B. McGhee, Second Reader



***Robert B. McGhee, Chairman,
Department of Computer Science***

ABSTRACT

This thesis presents a method of controlling an autonomous vehicle's motion in a two dimensional environment. Its' purpose is to expand the functionality of a vehicle's motion by complementing a point to point path planning scheme with a path to path scheme. The method introduced in this paper will use the vehicle's position and the desired path to calculate the necessary curvature to effect movement onto the desired reference path. The reference path will be a simple planar curve, such as, a circle or line. After successful testing of an operating algorithm, the method shall be incorporated into a robot's software system. This path tracking method will lay the groundwork for a dynamic obstacle avoidance system for a mobile robot.

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	BACKGROUND	1
B.	THESIS ORGANIZATION	2
II.	PROBLEM STATEMENT	3
III.	METHOD	5
A.	PATH CONTROL BY CURVATURE	5
B.	COEFFICIENTS BY APPROXIMATION	7
C.	EXPERIMENTAL RESULTS	13
D.	SUMMARY	14
IV.	DETAILED ALGORITHMS	20
A.	CLOSEST DISTANCE	20
1.	Point to Line	20
2.	Point to Circle	21
a.	Positive Curvature	21
b.	Negative Curvature	25
3.	General Distance Equation	25
B.	THE IMAGE	26
1.	Lines	26
2.	Circles	28
C.	EVALUATE NEW CONFIGURATION	30
D.	SUMMARY	31
V.	TRANSITIONING	32
A.	POSSIBLE METHODS	32
1.	Minimum Distance	32
2.	Dynamic Transitioning	33
B.	A SUITABLE SCHEME	36
1.	Intersection Point	37
a.	Line to Line	37
b.	Line to Circle/ Circle to Line	39
2.	Transitioning Distance	42
3.	Deriving A Transitioning Function	44
a.	Relationship Between TD and Turn Angle	44
b.	Relationship Between TD and S_0	47
C.	SUMMARY	53
VI.	IMPLEMENTATION	57
A.	OVERVIEW OF SYSTEM	57
1.	MML System	57

B.	PATH TRACKING SYSTEM	58
1.	User Program	58
2.	Path Descriptor	58
3.	Executor	59
C.	SUMMARY	60
VII.	SUMMARY AND CONCLUSIONS	61
A.	CONTRIBUTIONS OF RESEARCH	61
B.	FUTURE RESEARCH	62
	APPENDIX A	64
	APPENDIX B	66
	APPENDIX C	78
	APPENDIX D	90
	LIST OF REFERENCES	93
	BIBLIOGRAPHY	94
	INITIAL DISTRIBUTION LIST	96

LIST OF FIGURES

Vehicle Pulling Away From the Curb	3
Vehicle Merging Onto X-axis.....	7
Simulator Output for Various P_i	11
Simulator Output for Various Values of S_0	12
Output Comparison for $S_0 = 1.0$	16
Output Comparison for $S_0 = 0.5$	17
Output Comparison for $S_0 = 0.25$	18
Output Comparison for $S_0 = 0.125$	19
Distance Between P_i and P_0	22
Distance Between P_i and P_{ref}	22
Positive Curvature Case	23
Negative Curvature Case	23
Calculating P_{image} for a Line	27
Vehicle Overshoot of a 170 Degree Turn	34
Premature Convergence of a 10 Degree Turn.....	35
Intersection of Two Lines	38
Calculating P_{inter}	38
Calculate P_{inter} Between a Line and Circle	41
Rotation of a Transition Problem.....	43
Experimental Minimum Transition Distance	45
TD as Calculated by Table 1 and Equation (5.15).....	48
Relationship Between TD and S_0	49
Plot of the Ratios of TD and S_0	50
Transition From Inner to Outer Circle	54
Transition From Outer to Inner Circle	55
Transitions Involved in Obstacle Avoidance	56
A 15 Degree Turn	67
A 30 Degree Turn	68
A 45 Degree Turn	69
A 60 Degree Turn	70
A 75 Degree Turn	71
A 90 Degree Turn	72
A 105 Degree Turn	73
A 120 Degree Turn	74
A 135 Degree Turn	75
A 150 Degree Turn	76
A 165 Degree Turn	77

I. INTRODUCTION

A. BACKGROUND

Presently, the bulk of research conducted in the area of path planning for an autonomous vehicle deals with sub-dividing the vehicles desired motion into multiple independent paths. The most prominent approach has been to describe the desired motion through a sequence of configurations. These configurations define the vehicles x , y coordinate position coupled with the vehicle's orientation [Ref. 1]. This method of path planning can be best described as a point to point control scheme. As such, a path planning algorithm calculates the path between adjacent configurations to find the independent legs of the vehicle's motion. This scheme reflects a wide spectrum of motion found within the world, and it enjoys many advantages, such as; its' simplicity, wide applicability, and ease of manipulation. However, there are circumstances in which a point to point scheme will not fully replicate the desires of the user.

An example can best be illustrated by reflecting on the motion of a car. Assume a car parked alongside the curb desires to pullout into traffic, proceed down the road for two blocks, and then make a right turn at the light. In this example there are no predetermined points which describe the motion, rather there exist the concept of traffic lanes which define and restrict the motion. Although a sequence of points could define a similar range of motion, the point to point scheme would require the user to have prior knowledge of transition points. In addition, the point to point scheme places no restrictions on the bounds of movement between two adjacent configurations. This could raise problems in a dynamic environment. On the otherhand, the path tracking method which we shall introduce restricts the vehicle's motion to the interior boundaries set by the intersecting paths. Thereby increasing control over the vehicle's motion, while reducing the information needed to program a vehicle. In the path tracking method the configurations lose their importance to the more general entity of the paths, which are represented as traffic lanes.

B. THESIS ORGANIZATION

The objective of this paper is to develop a mathematical model to support a path to path motion control system. The scope of our study will be limited to include simple planar curves, such as lines and circles. Upon deriving a mathematical foundation, we shall translate the work into a working simulator. This simulator should give the user the capability to fully test the algorithm, to include merging a vehicle onto simple planar paths, and to transition from path to path. After successful development and testing, the algorithm shall be incorporated into the motion control system of an operating vehicle, specifically the Yamabico-11 robot.

The layout of the thesis shall reflect the outline of the research. Chapter II presents a detailed problem statement to include our assumptions and requirements for a path tracking algorithm. Chapter III will discuss the mathematical groundwork necessary to develop our system. In this section we shall derive a control function which will be suitable to merge an autonomous vehicles onto a reference path. Chapter IV will then develop the equations necessary to support the control function. In the development phase we will separate our study into two cases, lines and circles. With the development of an appropriate framework, Chapter V will discuss the inherent problems in transitions between paths, and the specifics of our transition scheme. Chapter VI will elaborate on the details of implementation of the path tracking algorithm into the software of the Yamabico-11 robot. Finally, Chapter VII shall be a look at future areas of research and some conclusions. An appendix shall contain the path tracking simulator which was written in C. and a users manual for operating the path tracking algorithm.

II. PROBLEM STATEMENT

The problem can be stated as follows: find a smooth path from an initial configuration p_i to a reference path p_{ref} . The basic concept of our method shall be to vary the vehicles instantaneous curvature in order to manipulate the vehicle's position and heading. To facilitate this, we shall expand the concept of a configuration to be a quadruple (x, y, θ, κ) , which describes the vehicles position by it's cartesian coordinates coupled with it's heading and instantaneous curvature. The reference path will be defined by a similar quadruple. This permits a line or a circle to be represented by the same structure depending on the value of κ . For example, we might want to solve the problem presented earlier, concerning the car entering a traffic lane from a curb position and then making a right turn. The vehicles curb position would be represented $p_i = (0, 0, 0, 0)$, while the traffic lanes would be represented by $p_{ref1} = (0, 10, 0, 0)$, and $p_{ref2} = (50, 0, \pi/2, 0)$ as shown in Figure 1.

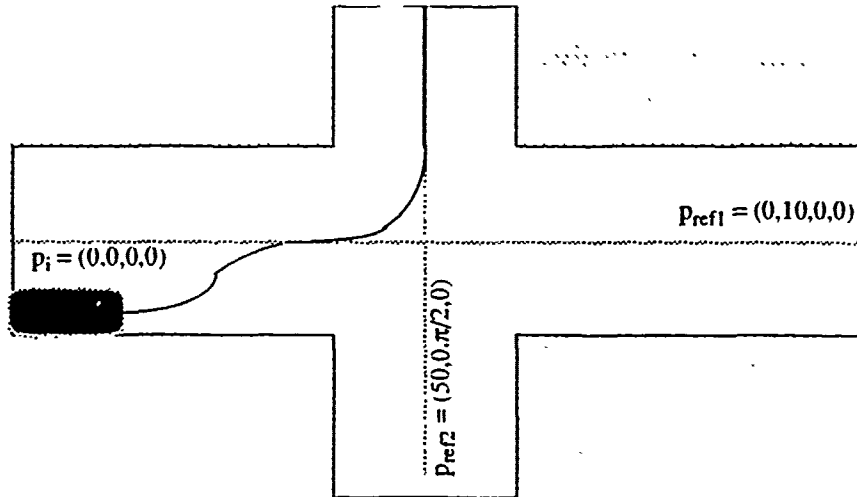


Figure 1. Vehicle Pulling Away From the Curb

In developing our solution we shall ignore the size and shape of the vehicle by assuming our vehicle is a point robot, and where the clearance to obstacles has been

assessed at a higher level within the system. This will simplify our work by removing the finite curvature limitation and clearance requirements necessary for a rigid body robot. In addition we shall assume the velocity of the vehicle is a constant positive value, and that the vehicle's motion will be controlled solely through altering the vehicle's instantaneous curvature. These assumptions will restrict the research to a spatial problem and alleviate the need to consider the dynamic relationships inherent to the problem, e.g., time, speed, acceleration, and rotation. By making these assumptions we will simplify the mathematical model required in finding a suitable path tracking algorithm.

In our initial problem statement we required that the proposed algorithm produce a smooth path. The derivative of curvature is the only control variable within the mathematical model. Since the derivative of the curvature dk/ds is finite, the resultant vehicle's trajectory is "smooth" in the sense that the tangent orientation, curvature, and derivative of curvature exist at every point on the trajectory [Ref. 2]. Although, this concept of smoothness is essential, our requirements must be more stringent. The generated path must be suitable to be followed by a robot's mechanical power train system. However, smoothness is a subjective quality, which must be clearly defined before attempting to calculate a viable solution. The problem can be characterized by how rapidly we wish to converge onto a desired reference path. If we attempt to converge in too short of a distance, the vehicle's motion will be unstable and the vehicle will lose tracking precision. On the otherhand, if the distance to converge is too large, then the vehicle's motion will be inhibited by a need for excessive maneuvering space. Therefore, our method will incorporate a distance constant, S_0 , to regulate the smoothness of the generated paths. The distance constant will balance the need for rapid response with the smoothness requirement. It's value will be dependant on the maneuvering characteristics of the vehicle, and will not adversely effect the mathematical correctness of our work.

III. METHOD

In developing any system the first step is to create a model of the system in order to analyze the system. The objective of this is to translate the real world mechanics of the system into a mathematical theory which accurately describes the system. This allows one to fully explore the nature of the problem and mathematically verify the solution. As the previous chapter outlined our system, this chapter's purpose is to develop the necessary mathematical equations to support our previously stated goals and requirements.

A. PATH CONTROL BY CURVATURE

When deriving a mathematical equation to simulate a real world problem, it is often obvious what type of equation is necessary to solve the problem. Physical models often directly translate into applicable mathematical models. However, in our situation we undertook this research without a clear idea of the final form of our mathematical model. Although we feel it may be possible to solve the problem using traditional control theory methods, we desire an easier method which may prove to yield equally powerful results. Therefore, we predicated our solution on the belief that the problem could be solved using analysis through differential equations.

In our problem statement we introduced the concept of the configuration as a quadruple, which defines a vehicle's positional status by stating the vehicle's present coordinate position, its' present direction of motion, and a measure of the change in direction of motion. We also defined our desired position by way of a reference path defined by the same variables as that of the vehicle's position. This similarity allows us to make quick accurate comparisons between the two quadruples. Therefore, we can define a function which relates our desired motion with respect to the initial and goal positions. Using this idea consider controlling the vehicle by changing its instantaneous curvature as determined by a function of the initial and goal quadruples.

$$\frac{d\kappa}{ds} = f(\kappa, \theta, x, y) \quad (3.1)$$

Equation (3.1) can be simplified by using the x , and y coordinates to determine, d , the distance between the present position and the desired reference path position. However, to accomplish this we must narrow the infinite number of available points on the reference path down to a specific point of interest. The most logical point would be the point on the path closest to the vehicle's position. We shall define this point to be the image of the vehicle's position, and reference it as p_{image} . Therefore, at all times there will be an image point which will provide the positional information of a theoretical vehicle located and tracking along the reference path. The establishment of this point allows us to quickly calculate the closest distance, d , between the vehicle's position and the reference path. Thereby equation (3.1) becomes

$$\frac{d\kappa}{ds} = f(\kappa, \theta, d) \quad (3.2)$$

Equation (3.2) restates our desire to control the vehicle through changing the instantaneous curvature, which is dependent on the vehicular position, p_i , and the reference path, p_{ref} . The equation does not show the underlying relationship between p_i and p_{ref} , or how their values effect the change in curvature. However, since we are attempting to find a simple but powerful solution, a logical choice would be to begin with a linear relationship. Thus, we propose the following general class as a steering function.

$$\frac{d\kappa}{ds} = -[A(p_i\kappa - p_{image}\kappa) + B(p_i\theta - p_{image}\theta) + Cd] \quad (3.3)$$

$$\frac{d\kappa}{ds} + A(p_i\kappa - p_{image}\kappa) + B(p_i\theta - p_{image}\theta) + Cd = 0 \quad (3.4)$$

Where A , B , and C are constants. This steering function is a simple linear equation, in which the constants can be solved for by using differential equations.

B. COEFFICIENTS BY APPROXIMATION

To find the optimal values for A , B , and C , let us analyze the simplest possible case. That is the case in which the reference path, p_{ref} , is the x -axis as shown in Figure 2.

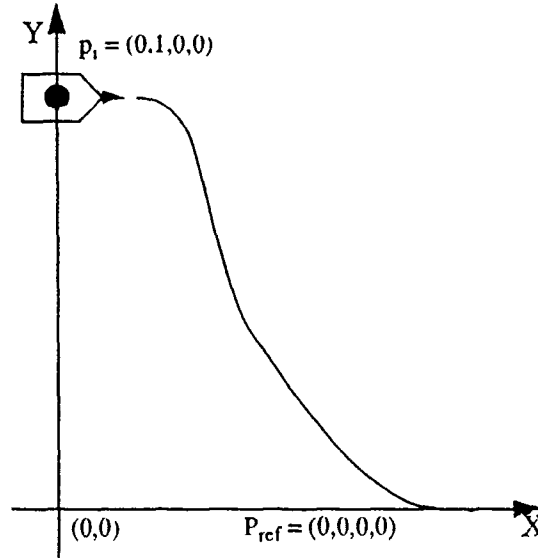


Figure 2. Vehicle Merging Onto X-axis

In this case, we can simplify our steering equation due to the following observations.

$$\begin{aligned} p_{image} \kappa &= 0 \\ p_{image} \theta &= 0 \\ d &= y \end{aligned} \tag{3.5}$$

That is since the x -axis is the desired reference path, our linear equation can be viewed as dependent only on the vehicle's positional quadruple. Therefore, equation (3.4) becomes

$$\frac{d\kappa}{ds} + A\kappa + B\theta + Cy = 0 \tag{3.6}$$

Since we know

$$y = y(x) \tag{3.7}$$

$$\theta = \text{atan}(y') = y' - \frac{(y')^3}{3} + \frac{(y')^5}{5} - \dots \tag{3.8}$$

$$\kappa = \frac{y''}{(1 + (y')^2)^{3/2}} \quad (3.9)$$

$$\frac{d\kappa}{ds} = \frac{\frac{d\kappa}{dx}}{\frac{ds}{dx}} = \frac{d}{dx} \left(\frac{\frac{y''}{(1 + (y')^2)^{3/2}}}{\sqrt{1 + (y')^2}} \right) \quad (3.10)$$

$$\frac{d\kappa}{ds} = y''' (1 + (y')^2)^{-2} - 3y' (y'')^2 (1 + (y')^2)^{-3} \quad (3.11)$$

By equation (3.6) to (3.11) the steering function becomes,

$$\frac{y'''}{(1 + (y')^2)^2} - \frac{3y' (y'')^2}{(1 + (y')^2)^3} + \frac{ay''}{(1 + (y')^2)^{3/2}} + b \left(1 - \frac{(y')^2}{3} + \dots \right) y' + cy = 0 \quad (3.12)$$

However, this equation is too confusing, and as a result very difficult to solve. If we are to maintain a simple solution, we must make some assumptions, concerning the terms of equation(3.12). To accomplish this we shall make use of two reasonable assumptions.

$$\begin{aligned} (y')^2 &\ll 1 \\ y' (y'')^2 &\ll y'' \end{aligned} \quad (3.13)$$

Both assumptions deal with the relative magnitude of the first three derivatives of y. If these assumptions hold, which by all indications they do when the value of x is relatively high, they would significantly simplify our control equation. Using the assumptions of (3.13), equation (3.12) becomes,

$$y''' + Ay'' + By' + Cy = 0 \quad (3.14)$$

Written in differential notation, we have

$$(D^3 + AD^2 + BD + C)y = 0 \quad (3.15)$$

Our steering function thus becomes an ordinary third order differential equation. Since the steering function is of the third order, the equation must have at least one real root. This

real root can be either a positive or a negative value. However, if the real root is a positive value the generated solution would diverge from the reference path. Therefore, to generate a converging solution, the real root must be a negative value. Let's assume that the value of the root is $-k$. Then equation (3.15) becomes,

$$(-k)^3 + A(-k)^2 + B(-k) + C = 0 \quad (3.16)$$

By solving this equation for the constants, and substituting their value back into the equation, we get

$$(D^3 + AD^2 + BD + k^3 - Ak^2 + Bk)y = 0 \quad (3.17)$$

$$(D + k) [D^2 + (A - k)D + k^3 - Ak^2 + Bk]y = 0 \quad (3.18)$$

The second order polynomial of equation (3.18) has two roots. If these roots are imaginary, then the solution would be oscillatory and inappropriate for our goal. Therefore, we restrict the roots of this polynomial to be negative real roots, which we shall assume to be $-k_1$ and $-k_2$. Equation (3.18) now becomes

$$(D + k)(D + k_1)(D + k_2)y = 0 \quad (3.19)$$

Since, there are no advantages of having three distinct values for k , k_1 , and k_2 , we shall assume the three roots are the same.

$$k = k_1 = k_2 \quad (3.20)$$

Therefore, equation (3.19) becomes

$$(D + k)^3 y = 0 \quad (3.21)$$

Now, solving for y we get

$$y = \left(\frac{A}{2}x^2 + Bx + C\right)e^{-kx} \quad (3.22)$$

Using equations (3.15 and 3.21) we can solve the equation for the values of the constants A , B , and C . This is simply accomplished by expanding the third order polynomial and individually solving the coefficient for each order.

$$D^3 + AD^2 + BD + C \equiv (D + k)^3 \quad (3.23)$$

$$D^3 + AD^2 + BD + C \equiv D^3 + 3kD^2 + 3k^2D + k^3 \quad (3.24)$$

Therefore, the value of the constants are as follows.

$$\begin{aligned} A &= 3k \\ B &= 3k^2 \\ C &= k^3 \end{aligned} \quad (3.25)$$

Now that we have calculated the values of the individual constants we can complete our steering function by substituting these values into equation (3.3).

$$\frac{d\kappa}{ds} = -[3k(p_i\kappa - p_{image}\kappa) + 3k^2(p_i\theta - p_{image}\theta) + k^3d] \quad (3.26)$$

Equation (3.26) is the final form of our derived curvature control equation. However, we still must determine an appropriate value for the constant k . We know that the coefficients of the derived equation will affect the responsiveness and smoothness of the path generated by the steering function. Furthermore, we earlier established the distance constant, S_0 , for the same purpose. Thus a logical solution would be to base the value of k on the value of S_0 . Therefore, for our algorithm we shall assign the value of k to be the inverse of S_0 .

$$k = \frac{1}{S_0} \quad (3.27)$$

This completes the derivation of our steering function. The output of equation (3.26) should generate a smooth path which meets our stated requirements. Preliminary results of the steering function are illustrated in Figures 3 and 4. The output displayed in Figure 3 depicts the generated paths for various initial configurations and the x-axis being the reference path. In each of the cases the initial configuration is the point $p_i = (0,1)$ while the vehicle's orientation is the intervals of every 45 degrees. On the otherhand, Figure 4 illustrates the effects of changing the value S_0 has on the path generated by the algorithm. The output illustrated in Figures 3 and 4 are similar to what we generally expected, and fully meet our requirements. However, to verify the output and the equation's suitability, we must check it through experimental results.

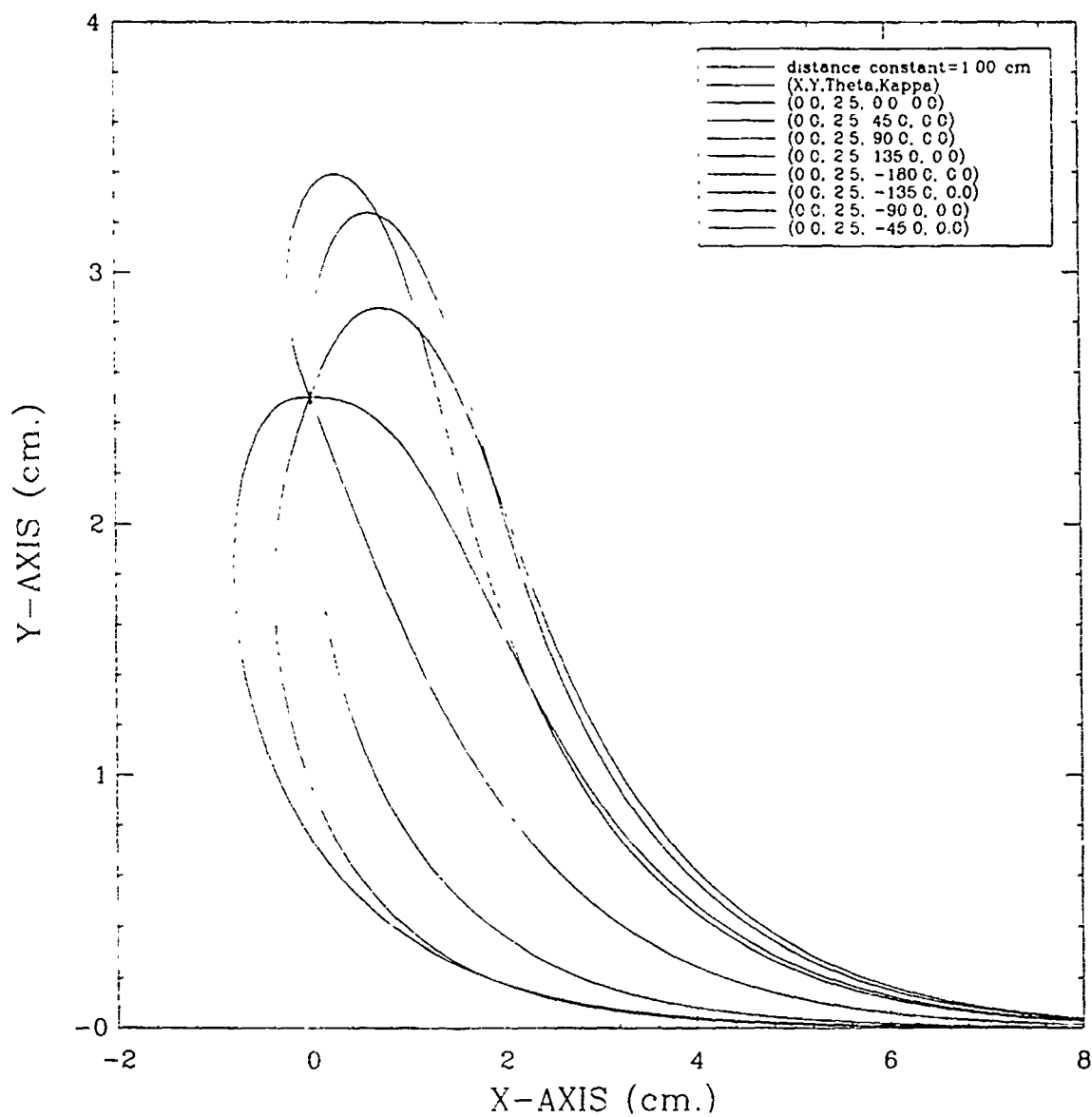


Figure 3. Simulator Output for Various P_i

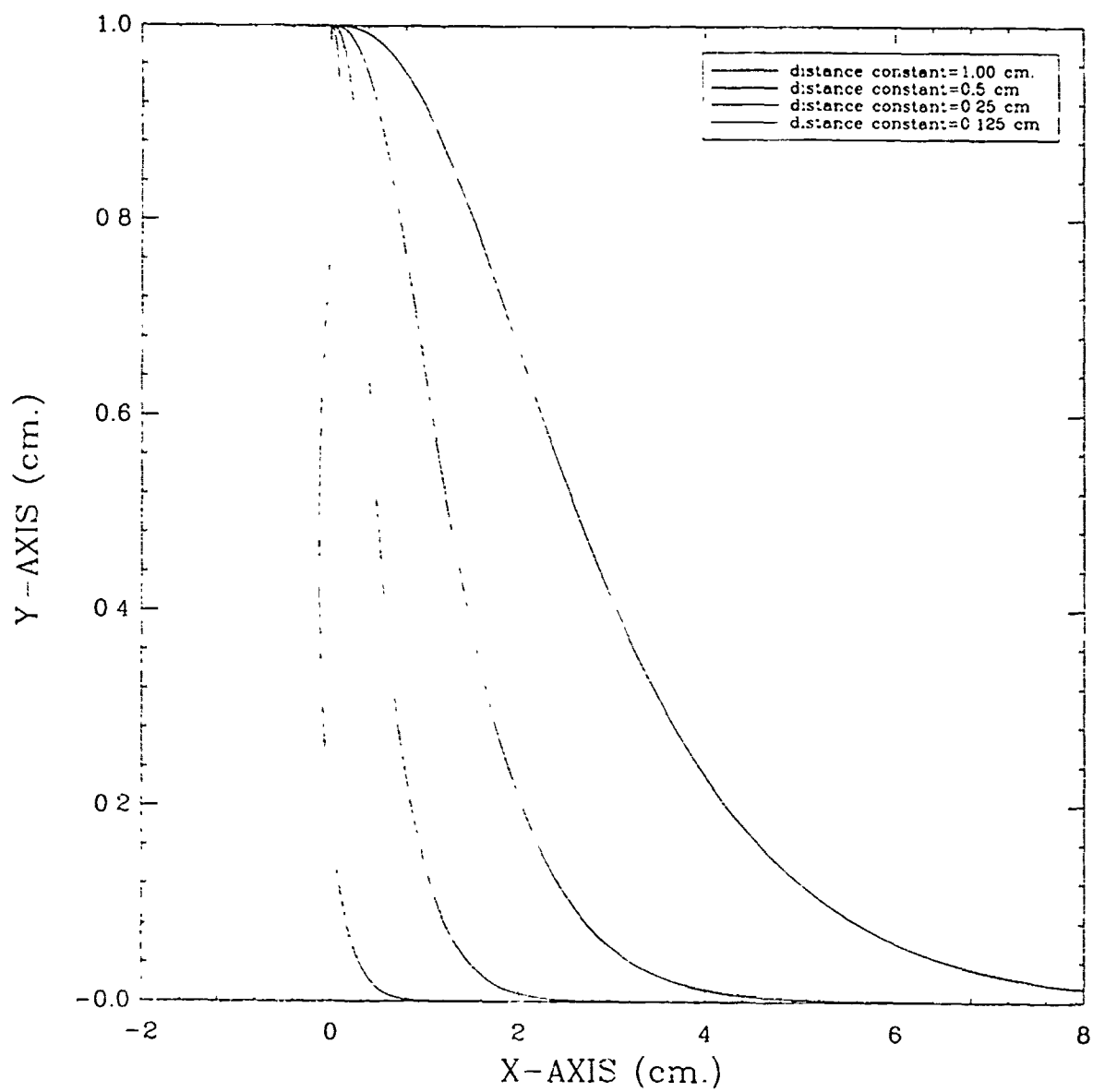


Figure 4. Simulator Output for Various Values of S_0

C. EXPERIMENTAL RESULTS

In deriving our curvature control equation we made some assumption vital to simplifying our equation. These assumptions dealt with the relative magnitude of the derivatives of the y component of the vehicle's position, and the values of the roots of a third order polynomial. Whenever assumptions are made within a mathematical problem error is introduced into the final solution. In order to validate the solution and the assumptions, we must offer some substantiative proof that the work was not drastically flawed by making the assumptions. The best proof is sound mathematical support for the assumptions. We can not offer this in our case. However, we can support our assertions via experimental results.

If we can show that the results of our curvature control equation closely match the output of a similar system which does not take our assumptions in consideration, then we add a degree of validity to our work. Thus, we intend to verify our work by comparing the results of our algorithm to the results of an equation derived without our assumptions. Thus, we shall compare the output results of our curvature control function of equation (3.26) to the assumption free equation(3.22). To accomplish this we shall make the comparison for a specific problem. This problem will be the case of merging a vehicle with a initial position $p_i = (0,1,0,0)$ onto the directed line, $p_{ref} = (0,0,0,0)$, which is simply the x -axis. By using this case we can easily calculate the coefficients for equation (3.22). By solving for the variables and using variable substitution we calculated the coefficients' values;

$$\begin{aligned} A &= k^2 y_0 \\ B &= k y_0 \\ C &= y_0 \end{aligned} \tag{3.28}$$

Details of the actual mathematical process involved can be found in appendix A. Using these values in the assumption free control function we can compare results.

We conducted four separate cases for comparison using the problem as stated above. In each case we solved the problem of merging a vehicle onto the x -axis, while we varied

the value of S_0 for each test case. The graphical results of both our curvature control algorithm and the actual output as found by the assumption free function is depicted in Figures 5 to 8. The results of the comparison are very good. In our results we see that the output of our path tracking algorithm are very close to the actual values. The test results show, that as we increase the distance constant, S_0 , the difference in the output becomes even less significant, which meets our expectations. Although, the difference increases as the distance constant is significantly decreased, as in the case of $S_0 = 0.125$, the output is not inappropriate. The results show, that at no instance does an extreme difference in the output between the two functions exist. Therefore, the results fully support the assumption made within our work. Although, this does not fully validate our results, it does provide a significant degree of confidence in our method. With this degree of confidence we have faith that we are on the correct track, and that our algorithm is appropriate.

D. SUMMARY

In this chapter we laid the mathematical groundwork for our path tracking algorithm. Initially, we had few concrete ideas on the form of our controlling equation. However, our guiding factor was to keep the equation simple and powerful. This lead us to guess the format of a suitable equation to be a linear differential equation. From this we generated a simple steering function based on the difference between the vehicle's position and the vehicle's image position on a reference path. The resulting differential equation was cumbersome and difficult to solve. Therefore, we made some assumption about the relative values of the various derivatives of y . This enabled us to reduce the problem to a third order differential equation, which could be solved rather easily. Upon solving the differential equation, we proposed using S_0 as the basis for the coefficients of the resulting equation. Thereby, allowing us to balance the need for responsiveness with the requirement of smoothness. The paths generated by the proposed steering function fully meet our expectations. In addition, we tested our results by making a comparison between the output of our derived curvature control function and an assumption free version of the steering

function. The graphical results showed no significant deviations or aberrant trends between the two functions output. Thereby, these experiments fully support the appropriateness of our equation, lending a degree of validity to our path tracking method.

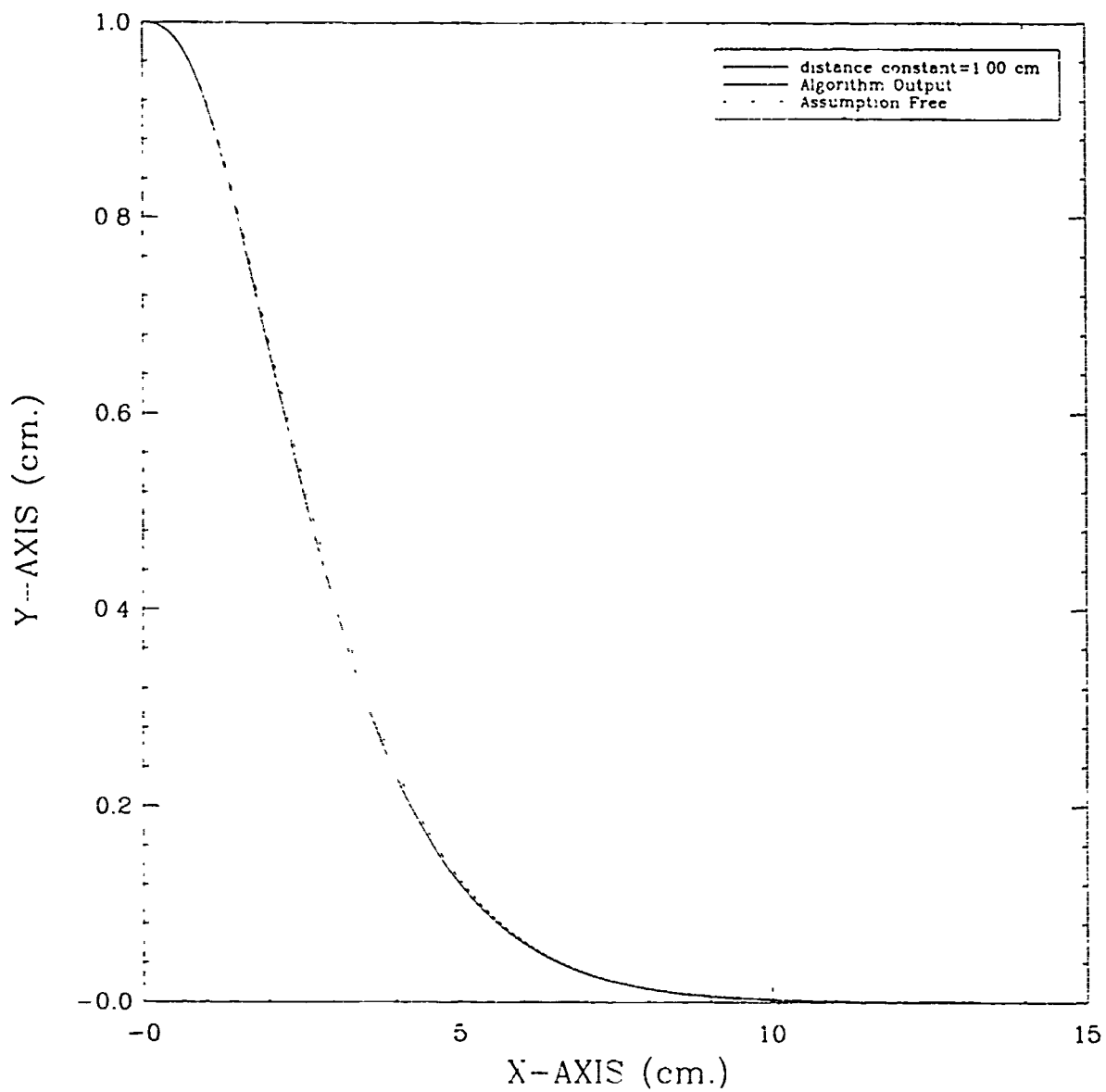


Figure 5. Output Comparison for $S_0=1.0$

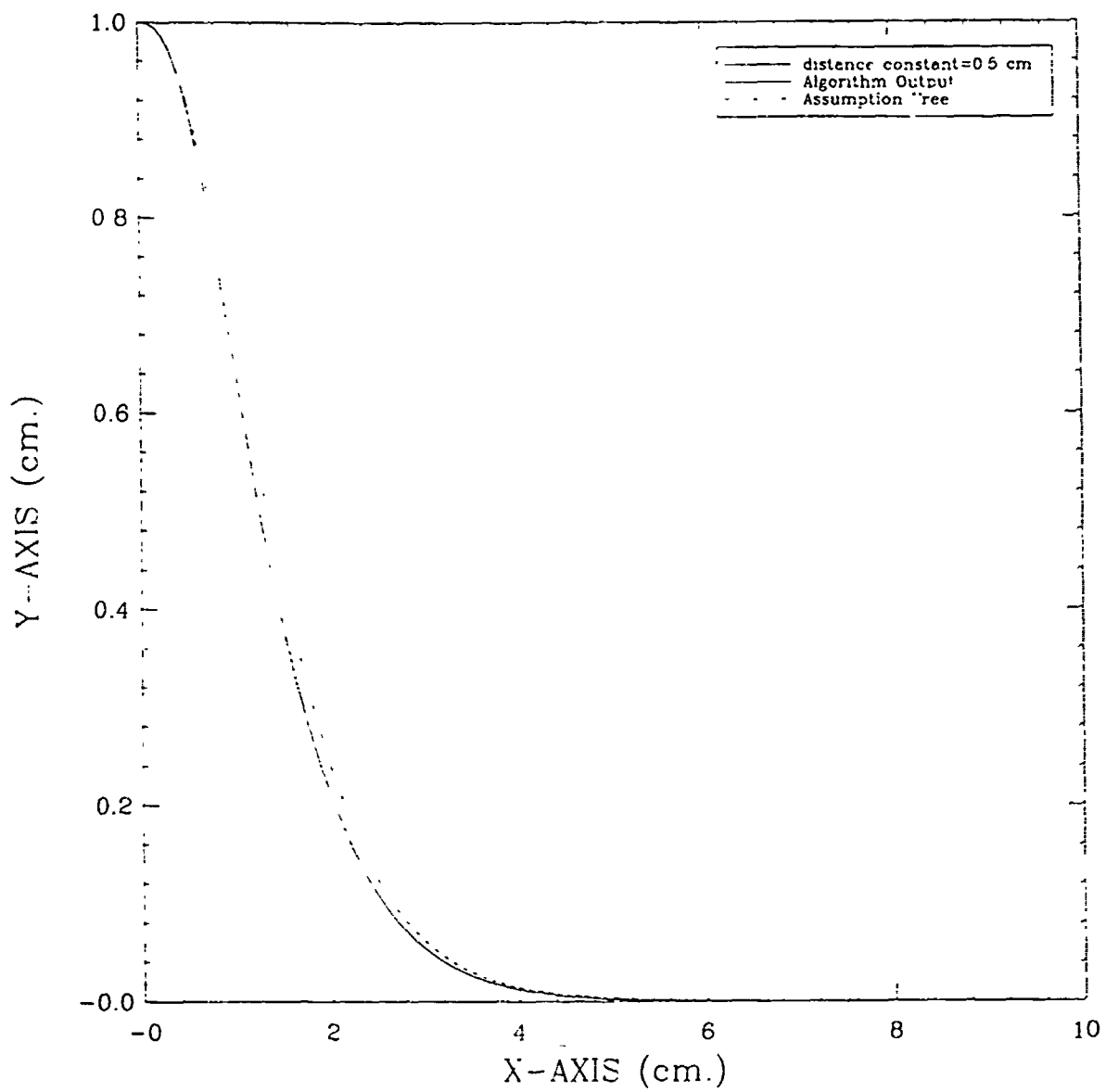


Figure 6. Output Comparison for $S_0 = 0.5$

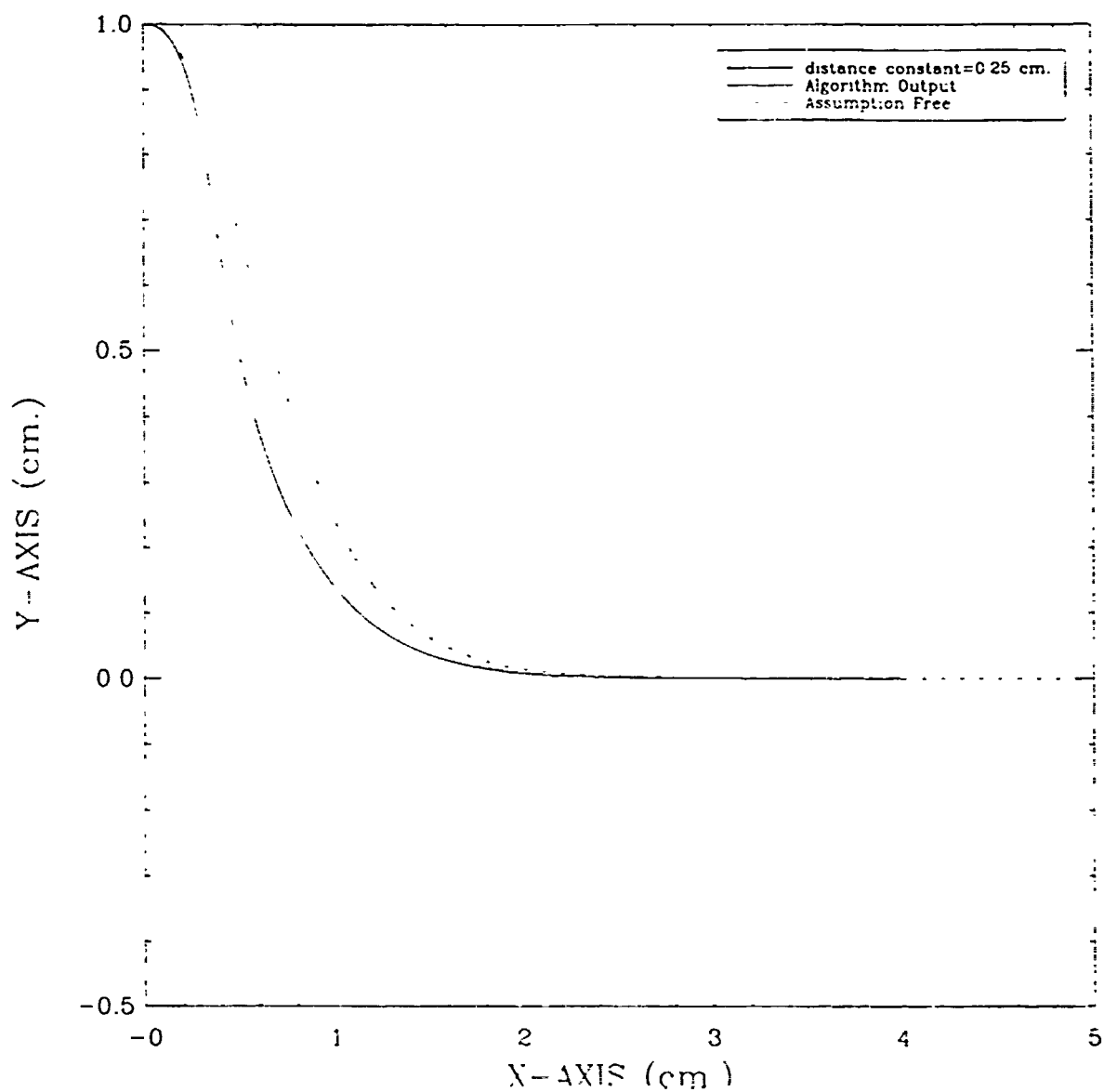


Figure 7. Output Comparison for $S_0 = 0.25$

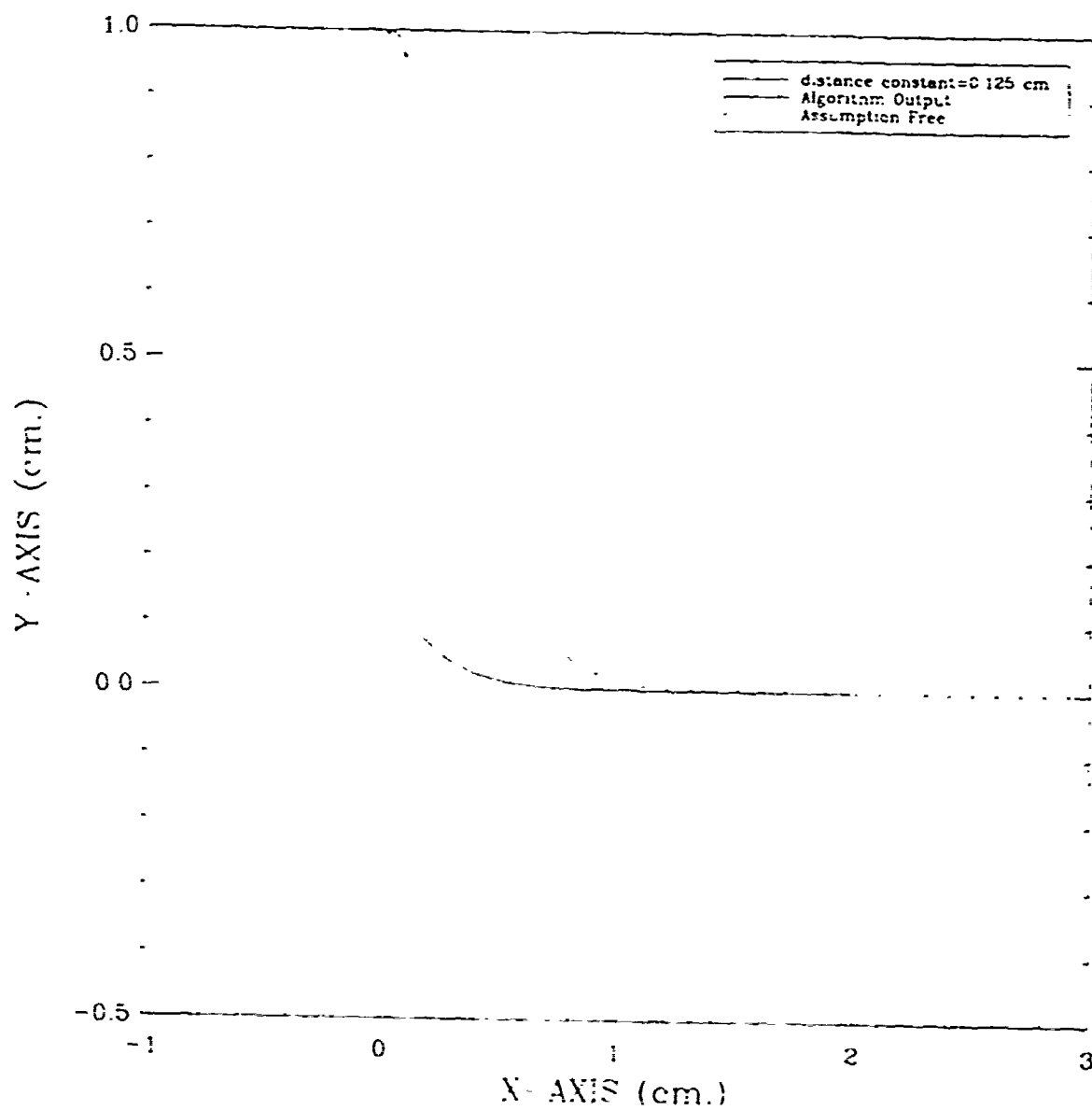


Figure 8. Output Comparison for $S_0 = 0.125$

IV. DETAILED ALGORITHMS

Chapter III successfully derived a curvature control equation for our path tracking algorithm. In this chapter we shall develop the support equations necessary to incorporate the control equation into a broader algorithm for path tracking. The overall objective for the algorithm is to alter a vehicle's location defined by a configuration onto a planar curve. To accomplish this we shall undertake three steps: to calculate the closest distance between p_i and p_{ref} to locate p_{image} ; and to calculate the change in curvature. With these steps accomplished we can construct a program which would successfully maneuver a vehicle onto any given reference path.

A. CLOSEST DISTANCE

The first step in our general scheme is to calculate the shortest distance between p_i and p_{ref} . We shall refer to this distance as d_{close} . There are numerous methods to calculate distance. However, in our case we desire a method which will be versatile enough to calculate this distance given configurations, which may either represent a point, a line, or a circle. We do not know of an equation which will allow us to perform such a calculation. Therefore, we shall derive a distance equation flexible enough to meet our requirements. There are two possible cases in our model; point to line, and point to circle. We shall look at these cases separately, and derive suitable equations for each case.

1. Point to Line

Our objective is to calculate the shortest distance between any directed line $p_{ref} = (a, b, \alpha)$ and a point $p_i = (x, y)$. Thus $dist(p_{ref}, p_i)$ refers to the perpendicular distance between the point and the line. In preparation for this calculation, let's calculate the distance between the point of interest and a special directed line, $p_0 = (0, 0, \alpha)$. This directed

line is simply a line parallel to p_{ref} , which originates at the origin. As Figure 9 illustrates, using simple trigonometry the distance is

$$dist(p_0, p_i) = y \cos(\alpha) - x \sin(\alpha) \quad (4.1)$$

Using the results of equation (4.1) we can generate a general distance equation. This general equation will be the difference between the distance from the point to the reference path and the distance from the reference path to our special directed line. This general equation is illustrated in Figure 10. To calculate the difference we simply apply equation (4.1) twice.

$$dist(p_{ref}, p_i) = dist(p_o, p_i) - dist(p_o, p_{ref}) \quad (4.2)$$

$$= y \cos(\alpha) - x \sin(\alpha) - b \cos(\alpha) + a \sin(\alpha) \quad (4.3)$$

$$= (y - b) \cos(\alpha) - (x - a) \sin(\alpha) \quad (4.4)$$

Note, that $dist < 0$ if the orientation between the directed line and the point is clockwise. Thus, this provides us with a suitable equation for the shortest distance between a point and any directed line.

2. Point to Circle

In deriving the shortest distance from a point $p_i = (x, y)$ to a circle $p_{ref} = (x, y, \theta, \kappa)$, we must realize that two cases exist. The value of κ can be either positive or negative. We shall examine these two cases separately. Both cases are illustrated in Figure 11 and 12, respectively.

a. Positive Curvature

Let's first calculate the origin of the reference circle. This can be done by adding the x , and y components of the radius to the x , and y coordinates of the reference circle.

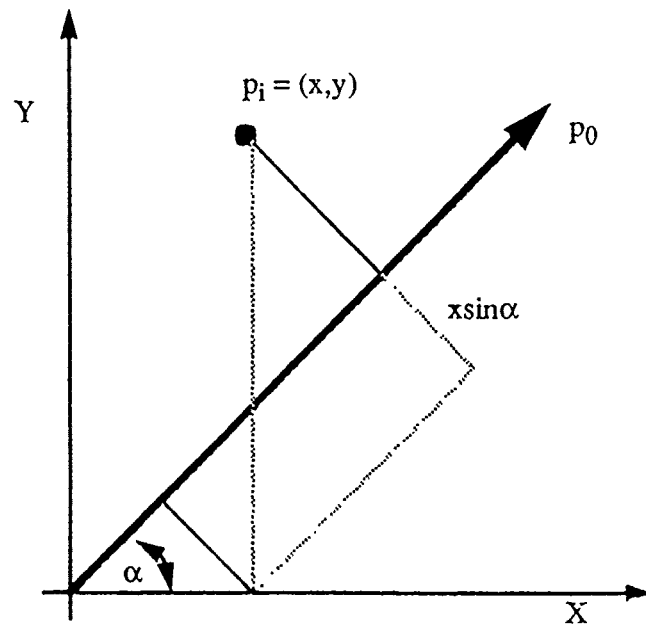


Figure 9. Distance Between P_i and P_0

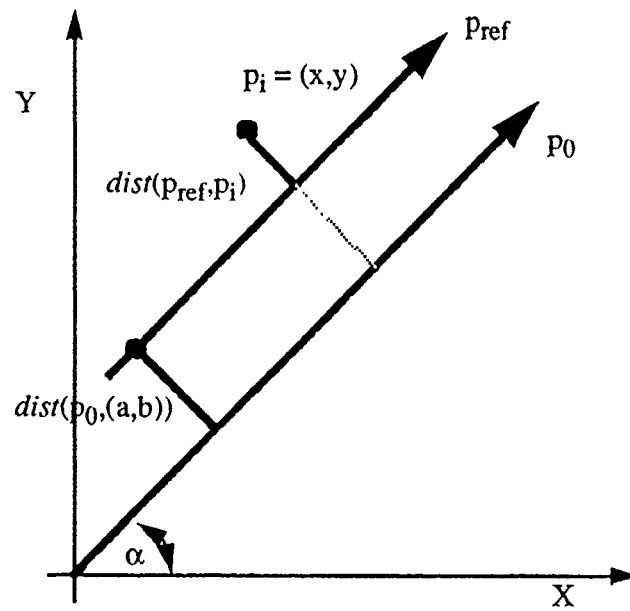


Figure 10. Distance Between P_i and P_{ref}

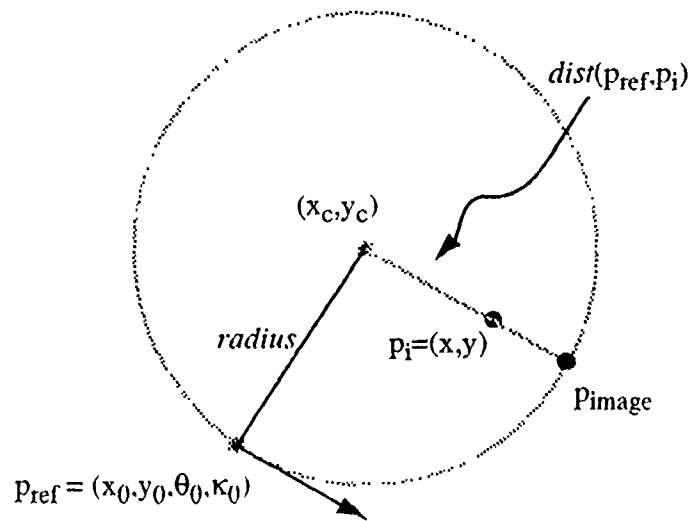


Figure 11. Positive Curvature Case

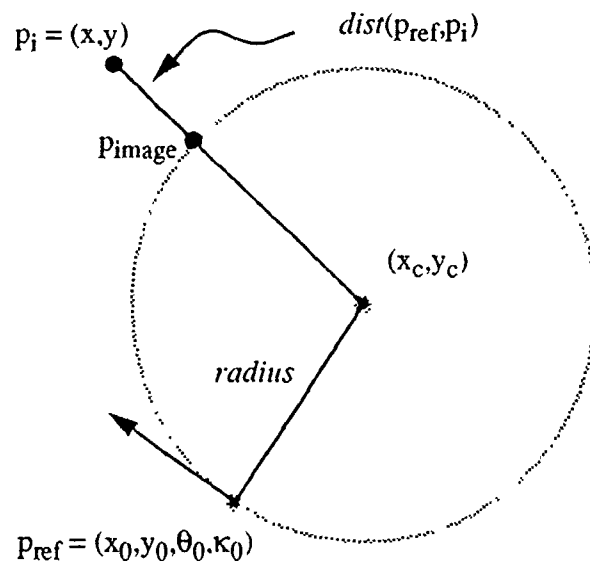


Figure 12. Negative Curvature Case

$$\begin{aligned}x_c &= p_0 + \left(\frac{1}{\kappa_0}\right) \cos(\theta_0 + \pi/2) \\y_c &= p_0 + \left(\frac{1}{\kappa_0}\right) \sin(\theta_0 + \pi/2)\end{aligned}\tag{4.5}$$

$$(x_c, y_c) = (x_0 - \left(\frac{1}{\kappa_0}\right) \sin(\theta_0), y_0 + \left(\frac{1}{\kappa_0}\right) \cos(\theta_0))\tag{4.6}$$

Now, that we have the coordinates of the origin, we can calculate the distance between the origin and the point by using Euler's distance equation. Furthermore, since we know the distance between the origin and the reference path, the radius, we can calculate the distance between p_i and p_{ref} . This is accomplished by taking the difference between the two calculated distances.

$$dist(p_{ref}p_i) = radius - dist(p_{origin}p_i)\tag{4.7}$$

$$= \frac{1}{\kappa_0} - \sqrt{\left[x - \left(x_0 - \frac{\sin\theta_0}{\kappa_0}\right)\right]^2 + \left[y - \left(y_0 + \frac{\cos\theta_0}{\kappa_0}\right)\right]^2}\tag{4.8}$$

We can simplify equation (4.8) by multiplying through with an appropriate factor.

$$\begin{aligned}&= \frac{\left(\frac{1}{\kappa_0}\right)^2 - \left[\left(x - x_0\right) + \frac{\sin\theta_0}{\kappa_0}\right]^2 + \left[\left(y - y_0\right) - \frac{\cos\theta_0}{\kappa_0}\right]^2}{\frac{1}{\kappa_0} + \sqrt{\left[\left(x - x_0\right) + \frac{\sin\theta_0}{\kappa_0}\right]^2 + \left[\left(y - y_0\right) + \frac{\cos\theta_0}{\kappa_0}\right]^2}}\end{aligned}\tag{4.9}$$

By factoring we get,

$$\begin{aligned}&= \frac{-(x - x_0) \left(x - x_0 - \frac{2\sin\theta_0}{\kappa_0}\right) - (y - y_0) \left(y - y_0 - \frac{2\cos\theta_0}{\kappa_0}\right)}{\frac{1}{\kappa_0} + \sqrt{\left[\left(x - x_0\right) + \frac{\sin\theta_0}{\kappa_0}\right]^2 + \left[\left(y - y_0\right) + \frac{\cos\theta_0}{\kappa_0}\right]^2}}\end{aligned}\tag{4.10}$$

Finally, we can multiply by κ_0/κ_0 to simplify equation (4.10).

$$= \frac{-(x-x_0) [\kappa_0 (x-x_0) + 2 \sin \theta_0] - (y-y_0) [\kappa_0 (y-y_0) - 2 \cos \theta_0]}{1 + \sqrt{[\kappa_0 (x-x_0) + \sin \theta_0]^2 + [\kappa_0 (y-y_0) - \cos \theta_0]^2}} \quad (4.11)$$

This provides us with the final form of our equation. It will calculate the distance between any point and any positive curvature circle. Also note that with this equation $dist(p_{ref}, p_i) < 0$ if the point p_i is not circumscribed by the circle.

b. Negative Curvature

Figure 10 illustrates the situation when p_{ref} curvature is negative. As this situation is very similar to the case with a positive curvature, we use the same method to calculate the $dist(p_{ref}, p_i)$. However, since the curvature of the circle is negative, this will give us a negative value for the radius. Thus, in this case we subtract the length of the radius from the distance between the origin and p_i .

$$dist(p_{ref}, p_i) = \sqrt{\left[x - \left(x_0 - \frac{\sin \theta_0}{\kappa_0}\right)\right]^2 + \left[y - \left(y_0 - \frac{\cos \theta_0}{\kappa_0}\right)\right]^2} - \frac{1}{\kappa_0} \quad (4.12)$$

We can simplify this equation by the method used earlier for the positive curvature case. The results of this method will produce the same equation as the positive curvature case. However, in this case $dist(p_{ref}, p_i) < 0$ when p_i is circumscribed by the circle.

3. General Distance Equation

Our objective was to derive a single general equation to calculate the shortest distance between a point and a reference path. However, it seems we have derived two separate equations to fulfill our needs. Yet, if we take a closer look at the two equations we shall reveal an interesting fact. We can consider a directed line, a special case of a circle with an infinite radius. The line's curvature is defined to be equal to zero. Therefore, if we implement equation (4.11) in the case of a directed line, we get,

$$dist = \frac{-(x-x_0) [\kappa_0 (x-x_0) + 2 \sin \theta_0] - (y-y_0) [\kappa_0 (y-y_0) - 2 \cos \theta_0]}{1 + \sqrt{[\kappa_0 (x-x_0) + \sin \theta_0]^2 + [\kappa_0 (y-y_0) - \cos \theta_0]^2}} \quad (4.13)$$

$$= \frac{-(x-x_0)[2\sin\theta_0] - (y-y_0)[-2\cos\theta_0]}{1 + \sqrt{[\sin\theta_0]^2 + [-\cos\theta_0]^2}} \quad (4.14)$$

$$= -(x-x_0)\sin\theta_0 + (y-y_0)\cos\theta_0 \quad (4.15)$$

This suggests that the distance equation derived in the case that the reference path is a circle subsumes both possible reference path cases. Thus, we have established a single general equation to determine the shortest distance between any point and any simple planar reference curve.

B. THE IMAGE

The control equation derived in chapter III was predicated on the establishment of a point referred to as p_{image} . That is the configuration of a theoretical vehicle, which is continuously maintaining position along the reference path. This configuration includes the theoretical vehicle's coordinate position, orientation, and curvature. The difficulty in calculating p_{image} is that the reference path could be either a line or a circle. Unfortunately, there is not a single method, to our knowledge, that would suffice for both of these cases. Therefore, we must derive two unique methods to calculate p_{image} . To accomplish this we shall address the two cases separately.

1. Lines

The case of finding p_{image} for a vehicle on a directed line is quite simply. Since the curvature of a directed line is defined to be equal to zero, and the orientation of all points on a directed line are defined to be equal to the orientation of the directed line itself, the calculation of p_{image} 's curvature and orientation is trivial. That is,

$$\begin{aligned} p_{image}\theta &= p_{ref}\theta \\ p_{image}\kappa &= 0 \end{aligned} \quad (4.16)$$

Thus, in the case of the reference path being a line, to calculating p_{image} is reduced to calculating the x, and y coordinates of p_{image} . Basically, to accomplish this we need to find

the shortest distance between the vehicle's position and the directed line. This situation is illustrated in Figure 13.

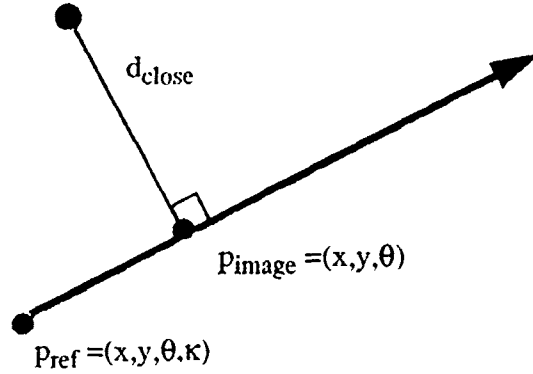


Figure 13. Calculating P_{image} for a Line

To calculate the shortest distance between p_i and p_{ref} we use equation (4.11), which was developed as a general equation for calculating the closest distance. However, we can simplify this equation since the curvature of a line is defined as zero. Therefore, the equation for the shortest distance between a point and a directed line as defined with configurations is,

$$d_{close} = (p_i y - p_{ref} y) \cos(\alpha) - (p_i x - p_{ref} x) \sin(\alpha) \quad (4.17)$$

Given the closest distance we can now calculate the coordinate position of p_{image} using simple trigonometry. Since the distance between the vehicle's position and the reference path is the shortest possible distance, the orientation of the line segment which connect the two points must be perpendicular to the orientation of the reference path, as shown in Figure 13. The coordinates of p_{image} can then be calculated by adding the x , and y components of d_{close} to the vehicle's coordinate position. Therefore, the coordinates of P_{image} are.

$$\begin{aligned} p_{image}^x &= p_i^x + d_{close}(\cos(p_i\theta - \pi/2)) \\ p_{image}^y &= p_i^y + d_{close}(\sin(p_i\theta - \pi/2)) \end{aligned} \quad (4.18)$$

2. Circles

Finding p_{image} for the case of p_{ref} being a circle is somewhat trickier than the line case, but mathematically the calculations are straightforward. We shall be able to distinguish between the two cases by examining $p_{ref} \kappa$ value. If the value is non-zero then the desired path is a circle. Otherwise, the path is a line. A circle will be defined by an x , and y point which lies on the circle, with this point's specific orientation and curvature. The curvature of a circle is defined to be a constant real number. Therefore, each point on the circle will have the same curvature value as p_{ref} curvature.

$$p_{image} \kappa = p_{ref} \kappa \quad (4.19)$$

The first step in calculating the other components of the image position's quadruple is to calculate the origin of the reference circle. To do so we calculate the radius of the circle, which is defined as the inverse of the curvature.

$$radius = \frac{1}{p_{ref} \kappa} \quad (4.20)$$

The orientation of a point on a circle is perpendicular to the orientation from the origin to the point on the circle, as shown in Figures 11 and 12. Knowing the orientation of the point defined as p_{ref} , thus allows us to calculate the orientation from the reference path to the origin. Now, by dividing the radius into its' x , and y components along this orientation we will find the coordinates of the origin. The origin shall be referred to as p_{origin} .

$$\begin{aligned} p_{origin}^x &= p_{ref}^x + radius(\cos(p_{ref}\theta - \pi/2)) \\ p_{origin}^y &= p_{ref}^y + radius(\sin(p_{ref}\theta - \pi/2)) \end{aligned} \quad (4.21)$$

Which can be written as,

$$p_{origin} = (p_{ref}^x + radius(\sin(p_{ref}\theta)) \cdot p_{ref}^y + radius(\cos(p_{ref}\theta))) \quad (4.22)$$

The next step in calculating the coordinate position of p_{image} is to calculate the orientation between the origin and the initial position, which we shall refer to as γ . This orientation can be found using the inverse-tangent function. Thus, γ is calculated

$$\gamma = \text{atan} \left(\frac{p_i^y - p_{origin}^y}{p_i^x - p_{origin}^x} \right) \quad (4.23)$$

Unfortunately, this method has two shortcomings. We want to distinguish between pairs of points which may have the same inverse-tangent value but different orientations, one such pair is $-\pi/4$ and $\pi/4$. A second problem is that equation (4.24) is undefined for all pairs of points in which $p_i^x = p_{origin}^x$. However, we can solve these problems by introducing a variation of the normal inverse-tangent function [Ref. 2].

$$\gamma = \text{atan2}(p_i^y - p_{origin}^y, p_i^x - p_{origin}^x) \quad (4.24)$$

In equation (4.25) we employ an inverse-tangent function of two arguments. This function has a range between $[\pi, -\pi]$. Therefore, it distinguishes between equivalent tangent values of different quadrants, and is defined for all values of x .

With the orientation between p_i and p_{origin} we can now calculate the coordinates of p_{image} for a circle. The image position is simply the distance of the radius from the origin in the direction of γ . Therefore, p_{image} becomes

$$\begin{aligned} p_{image}^x &= p_{origin}^x + |radius| \cos(\gamma) \\ p_{image}^y &= p_{origin}^y + |radius| \sin(\gamma) \end{aligned} \quad (4.25)$$

The final value we need to calculate for p_{image} is the orientation at the image point. Once again we shall use the fact that any point on a circle is perpendicular to the orientation between the origin and the point. Thus, we can calculate the image's orientation by adding or subtracting $\pi/2$ to the orientation between the origin and the initial position. If the reference path's curvature is negative we subtract, while if it is positive we add. We can take advantage of the absolute value function to incorporate both cases into one equation. Thus, the orientation of the image point is

$$p_{image} \theta = \gamma + (\pi/2) \left(\frac{p_{ref} \kappa}{|p_{ref} \kappa|} \right) \quad (4.26)$$

We have successfully collected all the data needed to find p_{image} . With this point established we can then go onto creating an algorithm which takes advantage of the curvature control equation developed in chapter III.

C. EVALUATE NEW CONFIGURATION

We have now calculated all the data we need to assemble a working path tracking algorithm. After a user inputs the vehicle's configuration and the desired reference path's configuration, our system would calculate p_{image} , then calculate the closest distance between the vehicle and the reference path by our derived d_{close} equation, and finally calculate the necessary change in curvature needed to move the vehicle towards and onto the reference path. This process would be completed at predetermined intervals, allowing for a means to update the vehicle's positional configuration. However, before we could update the vehicle's configuration we would have to calculate two values, the distance traveled by the vehicle each interval, $\delta dist$, and the change in the vehicle's orientation each interval due to the vehicle's instantaneous curvature, $\delta \theta$. These values are calculated as follows.

$$\begin{aligned} \delta \theta &= \delta dist (p_i \kappa) \\ \text{if } \delta \theta &= 0 \text{ then } \delta dist &= \delta \tau \times v \\ \text{if } \delta \theta &\neq 0 \text{ then } \delta dist &= (\delta \tau \times v) \frac{2 \cos \frac{\delta \theta}{2}}{\delta \theta} \end{aligned} \quad (4.27)$$

Where $\delta \tau$ is the duration of the interval, and v is the vehicle's constant velocity. The difference in the calculations for $\delta dist$ is a correction factor which is intended to correct the vehicle's coordinate position when it has a non-zero curvature. By calculating these values we can update the vehicle's positional configuration

$$\begin{aligned}
p_i \kappa &= p_i \kappa + \frac{d\kappa}{ds} \\
p_i \theta &= \text{norm}(p_i \theta + \delta \text{dist}(p_i \kappa)) \\
p_i x &= p_i x + \delta \text{dist}(\cos(p_i \theta + \delta \theta)) \\
p_i y &= p_i y + \delta \text{dist}(\sin(p_i \theta + \delta \theta))
\end{aligned} \tag{4.28}$$

The algorithm would continuously update the vehicle's position, and calculate the needed change in curvature. This process would effectively smoothly merge the vehicle onto the path and maintain it on the path after merging.

D. SUMMARY

In this chapter we developed the necessary equation to support an algorithm based on the curvature control equation developed in chapter III. This basically consisted of the development of the image point on the reference path, and a consistent method to calculate distance. With this information we can establish a system which will continuously calculate the necessary change in the instantaneous curvature, and update the vehicle's position. This will effectively merge the vehicle onto the desired reference path. With this accomplished we are ready to implement the algorithm into a vehicle simulator to test the results, and verify performance. With satisfactory testing accomplished we could then begin to work on a scheme to transition between multiple path.

V. TRANSITIONING

Our path tracking algorithm successfully merged a vehicle onto a reference path, therefore the next logical step in developing our path tracking algorithm is to expand the algorithm to be able to handle several successive paths. In transitioning between paths we must insist that the vehicle's motion is restricted within the boundary formed by the intersecting paths. This requirement is to ensure vehicle safety within an unknown environment. Since our algorithm was designed to work for all possible planar lines and circles, the problem of executing multiple paths in succession is reduced to the transitioning method between successive paths. The primary question to answer is, when do we begin our transition from one path onto the next?

A. POSSIBLE METHODS

Since we have limited our research to a spatial problem, we have limited the factors which effect our transition time. Our problem is not concerned with time, speed, and other factors which usually have a bearing on the moment which actions are to occur. Rather, our problem is solely dependent on distance. Thus, we have reduced the question of when to transition, to at what distance from the next reference path do we begin to transition. There are many different options available, however we want to limit the complexity while maximizing the effectiveness and fulfilling our safety requirements. Before actually detailing our transitioning scheme, let's look at a few possible options.

1. Minimum Distance

The simplest scheme can be devised to transition when the vehicle is within a given distance from the next reference path. This distance can be either the shortest distance between the vehicle and reference path, or the distance between the vehicle and the intersection of the two reference paths. The option chosen is very significant. When the

vehicle reaches the determined minimum distance, we simply begin using the next reference path in the path tracking process. This scheme is simple, and will satisfactorily work in many cases. However, the minimum distance scheme has a serious flaw.

The minimum distance scheme begins its' transition the same distance from the next reference path for every situation. This scenario works well when the desired turn angle approaches 90 degrees. However, problems arise when the interior angle between the present reference path and the next reference path deviate greatly from 90 degrees. When the turn angle is large the vehicle often does not have sufficient space available to transition without crossing through the new reference path. An example of this can be seen in Figure 14, where a vehicle attempts a 170 degree turn with a minimum transitioning distance of three. Although, this distance was sufficient for a 90 degree turn, figure 14 shows that the vehicle significantly overshoots the desired reference path. This wastes energy and time, and could represent a very serious hazard in an unknown environment.

Alternatively, when the incident turn angle is small, the moment of transition is often earlier than desired. This is due to when we execute the transition, the image is often located a significant distance before the intersection point of the two successive paths. In many cases this will cause the vehicle to cross the previous reference path during its' convergence to the new reference path. In extreme cases the vehicle may even merge onto the new reference path prior to the intersection point of the paths. This scenario is depicted in figure 15, where the vehicle attempts a ten degree turn with a minimum transitioning distance of three. Once again this deviation from the desired motion wastes energy and time, and represents a danger to the vehicle. Therefore, the minimum distance scheme must be considered unsuitable for our system.

2. Dynamic Transitioning

A second transitioning option would be to vary the transitioning distance according to the path tracking problem. The system would dynamically calculate the optimum distance to make a smooth efficient turn. To accomplish this method we would

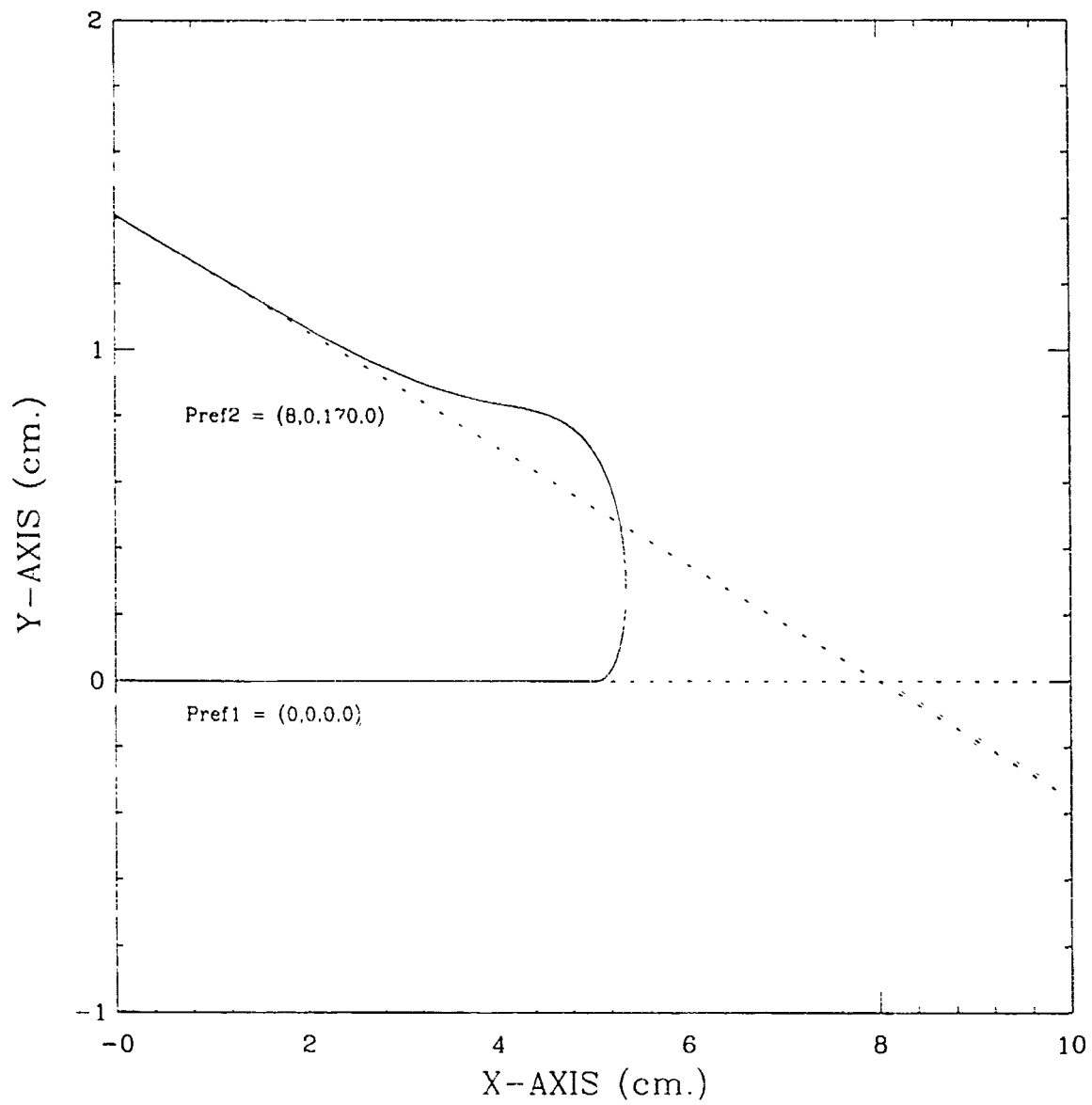


Figure 14. Vehicle Overshoot of a 170 Degree Turn

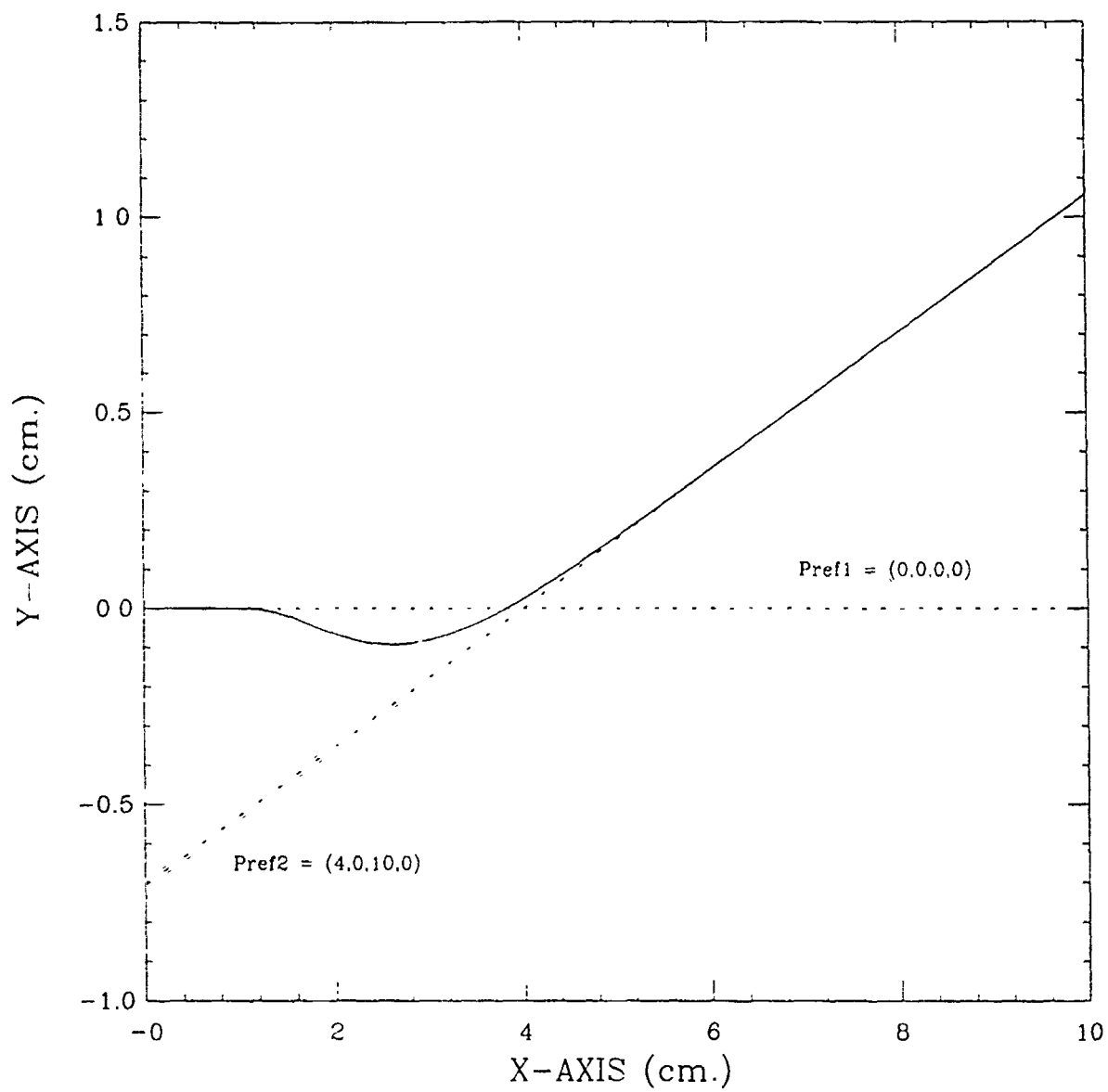


Figure 15. Premature Convergence of a 10 Degree Turn

establish a turn angle as a reference point on which to vary the transitioning distance. This benchmark would be a 90 degree turn, with a transitioning distance of two times the distance constant, S_0 . The dynamic scheme would increase this distance for tighter turns, and decrease the distance for wider turns. We could further improve this scheme by incorporating other factors which affect the vehicle's maneuverability, e.g. curvature limitations, velocity, distance constant, and the vehicle's maneuverability characteristics.

The dynamic transitioning described would be extremely effective in optimizing the transitions between paths. It would maintain the vehicle's movement to within the perimeter outlined by the reference path. This would increase vehicle efficiency, maneuverability, and control, which would thereby enhance safety. However, this type of system has some disadvantages. The system would require a complex function or a large data table to properly determine the correct transitioning distance. Either method would be costly in time and hardware support. Furthermore, both options would require exhaustive experimentation to derive the necessary data. This would be time consuming and dreary for a computer science student. Therefore, we feel although this method yields excellent results, it is not appropriate for our system at this time.

B. A SUITABLE SCHEME

The transitioning schemes we introduced were unsatisfactory for our system. The minimum distance scheme did not fulfill our requirements, while the dynamic scheme was too costly. Therefore, the scheme which we settled on was a simplified variation of the dynamic transitioning scheme. Our variation will have a transitioning distance function based on the turn angle and the distance constant. The transitioning distance will refer to the distance from the vehicle to the intersection point of the paths. We will conduct transitioning experiments to collect the necessary data concerning the relationship between these factors. Given the appropriate data we shall extrapolate a general function which meets our requirements. The idea is to derive a function which may not provide the optimum transitioning distance, but will provide an appropriate distance in all cases.

1. Intersection Point

Our transitioning distance shall be measured from the vehicle to the intersection point of the successive paths. Therefore, we must have a procedure which locates the intersection point for all possible transition. There are four transition possibilities; line to line, line to circle, circle to line, and circle to circle. The use of circles in our path tracking algorithm is for obstacle avoidance. Therefore, it would be inappropriate to transition between two circles. This being the case we shall exclude the circle to circle case in deriving the intersection point for our research.

a. Line to Line

The normal line to line intersection problem is very simple when we have the lines in the slope intersect format. However, our lines are in the configuration format. The method we shall use to calculate the intersection point is based on the Law of Sines. To calculate the point of intersection we are going to construct a triangle from the two reference paths, calculate the distance of one of the sides of the triangle, and calculate the interior angles of the triangle. With this information we can calculate any of the side's distances, and the coordinates of the intersection point. A graphic description of a line intersection problem can be seen in Figure 16. We shall now calculate S , the distance between the two reference configurations. This can be accomplished using the Euler distance equation,

$$S = \sqrt{(p_{ref2}^x - p_{ref1}^x)^2 + (p_{ref2}^y - p_{ref1}^y)^2} \quad (5.1)$$

We then calculate the orientation between these two configurations, Γ , using the inverse-tangent function described in equation (4.25).

$$\Gamma = \text{atan2}(p_{ref2}^y - p_{ref1}^y, p_{ref2}^x - p_{ref1}^x) \quad (5.2)$$

With this orientation we can construct a triangle from the two directed lines, by projecting a line segment in the orientation of Γ between the two reference paths. We now have the necessary information to calculate the interior angles, as shown in Figure 17.

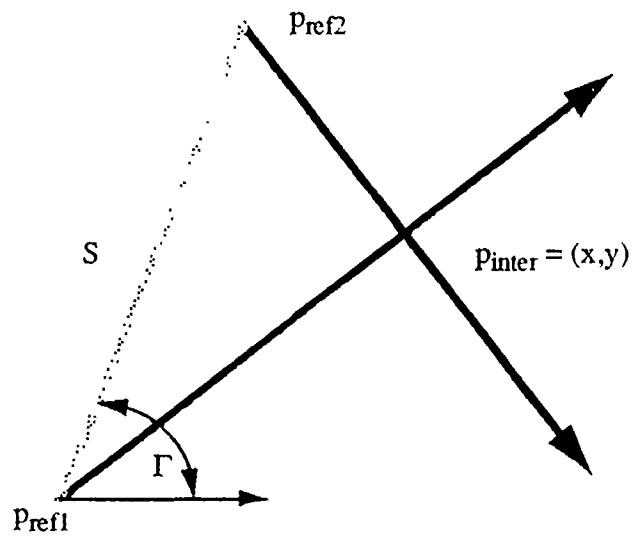


Figure 16. Intersection of Two Lines

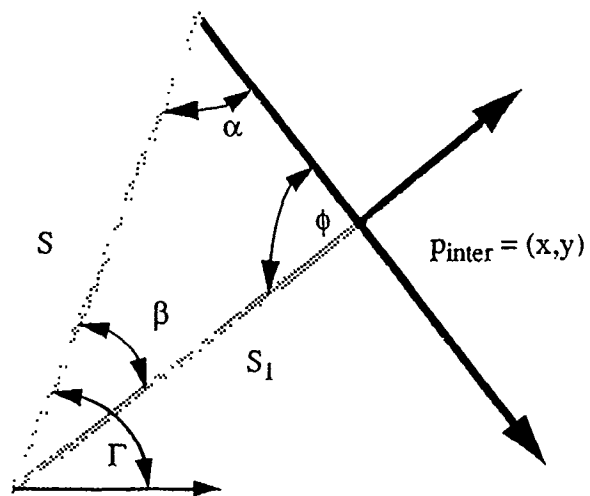


Figure 17. Calculating P_{inter}

$$\beta = |p_{ref1}\theta - \Gamma| \quad (5.3)$$

$$\alpha = |norm(\pi - \Gamma) - p_{ref2}\theta| \quad (5.4)$$

$$\Phi = (\pi - \beta - \alpha) \quad (5.5)$$

With the interior angles calculated we can use the Law of Sines to calculate the distance of one of the other two sides of the triangle. We will calculate S_1 , the distance from the intersection point and the first reference path.

$$S_1 = S\left(\frac{\sin(\alpha)}{\sin(\Phi)}\right) \quad (5.6)$$

This provides us with the needed information to calculate the intersection point. The coordinates of the intersection are calculated by dividing S_1 into its x , and y components, and adding to the coordinates of the reference path configuration

$$\begin{aligned} p_{inter}^x &= p_{ref1}^x + S_1 \cos(p_{ref1}\theta) \\ p_{inter}^y &= p_{ref1}^y + S_1 \sin(p_{ref1}\theta) \\ p_{inter}\theta &= p_{ref2}\theta - p_{ref1}\theta \end{aligned} \quad (5.7)$$

The orientation of p_{inter} is simply equal to the interior angle formed by the intersecting reference paths. This value is being calculated for future use in determining the transition distance required. It will be referred to in the future as the turn angle.

b. Line to Circle/ Circle to Line

The transition between line to circle and circle to line are very similar. We assume the user inputs two path which intersect. Given this assumption, a line will intersect with a circle in either one or two points. In either case both scenarios reflect both transition possibilities depending on which point of intersection we choose. This example is illustrated in Figure 18. The first step in determining the intersection point is to calculate the origin of the circular reference path. We calculate the radius, and then use equation (4.6) to find the origin's coordinates.

$$radius = \frac{1}{p_{ref}^K}$$

$$p_{origin}^x = p_{ref}^x - radius (\sin (p_{ref}^{\theta}))$$

$$p_{origin}^y = p_{ref}^y + radius (\cos (p_{ref}^{\theta}))$$
(5.8)

The trick now is to create a triangle with p_{origin} and the directed line in order to find the intersection point. After creating this triangle we can use the properties of a triangle to calculate the measurements of the sides, and to then find the coordinates of the intersection point, as shown in Figure 18. To begin let's calculate the distance from the origin to the directed line, A , using equation (4.18).

$$A = (p_{origin}^y - p_{ref}^y) (\cos (p_{ref}^{\theta})) - (p_{origin}^x - p_{ref}^x) (\sin (p_{ref}^{\theta}))$$
(5.9)

Using the Pythagorean theorem we can calculate the distance B . If $|A| > |radius|$, then there exists no intersection point between the two paths.

$$B = \sqrt{\left(\frac{1}{p_{ref}^K}\right)^2 - A^2}$$
(5.10)

We can calculate the image point of the origin onto the reference path by using the value of A , and the fact that this distance is perpendicular from the origin with respect to the directed line. The image is

$$p_{image}^x = p_{origin}^x + A \sin (p_{ref}^{\theta})$$

$$p_{image}^y = p_{origin}^y - A \cos (p_{ref}^{\theta})$$
(5.11)

With the image calculated we can now find the intersection point.

$$p_{inter}^x = p_{image}^x \pm B \cos (p_{ref}^{\theta})$$

$$p_{inter}^y = p_{image}^y \pm B \sin (p_{ref}^{\theta})$$
(5.12)

This equation is suitable for both line to circle and circle to line transitions. If we are transferring from a line to a circle we would subtract the components of B from the image, and if we go from a circle to line we add the components. To calculate the

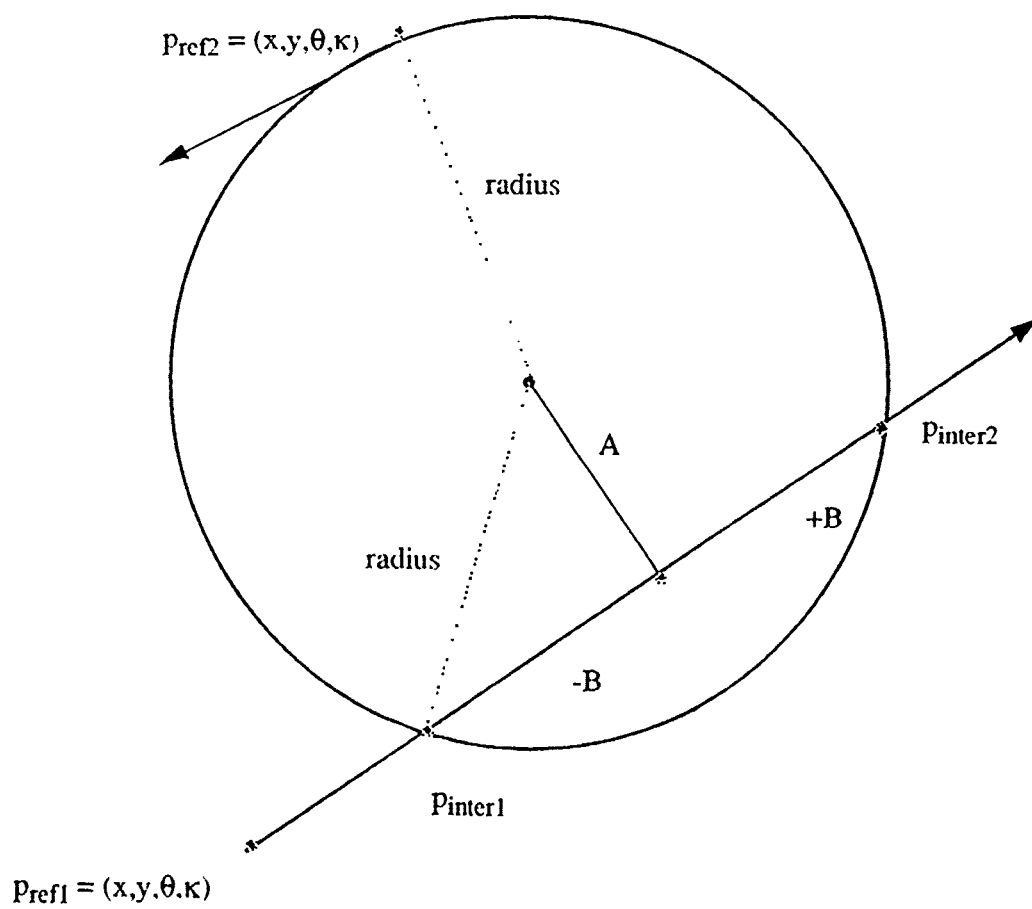


Figure 18. Calculate P_{inter} Between a Line and Circle

orientation of the intersection point we once again use our variation of the inverse-tangent function.

$$\Phi = \text{atan2}(p_{inter}^y - p_{origin}^y, p_{inter}^x - p_{origin}^x) \quad (5.13)$$

This orientation is perpendicular to the orientation of the intersection point, which can be calculated by adding or subtracting $\pi/2$ radians, depending on the sign of the curvature of the circular reference path.

$$p_{inter}^{\theta} = \Phi + \left(\frac{p_{ref}^K}{|p_{ref}^K|} \right) \quad (5.14)$$

Thus, we have successfully calculate the intersection point for all three transition possibilities. Using this point we shall be able to calculate the distance from the intersection to control the time of transition. Now, we are ready to proceed to determining the transitioning distance function.

2. Transitioning Distance

Our transitioning scheme is based on transferring between successive paths at a time determined by the distance from the vehicle to the next path. This distance shall be referred to as the transitioning distance. To optimize the efficiency of transitioning the transitioning distance can not be a constant value. Therefore, it is our idea to find a function that will determine the appropriate distance between the vehicle and path to produce an efficient transfer. However, it is highly unlikely that a general function will produce the optimum transitioning distance for all possible situations. Thus, our objective is to derive a function which will ensure all transfers between paths meet our requirements for safety in an efficient manner. The safety requirements are based on maintaining positive control over the vehicle to the extent that the vehicle avoid oscillation, and does not cross the bounds established by the reference paths. Our method of deriving the function will be based on gathering experimental data, which will translate into an appropriate function.

To derive a transitioning function we intend to conduct experimental tests to determine the critical distance necessary for a safe turn. A safe turn is defined to be a turn

in which the vehicle negotiates the intended turn without crossing the present reference path or overshooting the next successive path. The experimental tests will maintain the turn angle and the distance constant, while varying the available distance to maneuver the vehicle. This process will be repeated until we find the minimum distance needed to fulfill the requirements. The range of the experiment will consist of turn angles between $[0, \pi]$ at intervals of $\pi/12$. We shall also complete the experiments for various values of S_0 . The results of our experiments will be put into a table format to enable quick comparison of results. We shall then develop a distance function which will approximate the results of our experimental data.

In conducting these experiments the major concern is determining if the generated path oscillates or crosses the reference path. To accomplish this we shall manipulate the vehicle's initial position to simulate the desired turn and transitioning distance. To illustrate this, we refer to Figure 19.

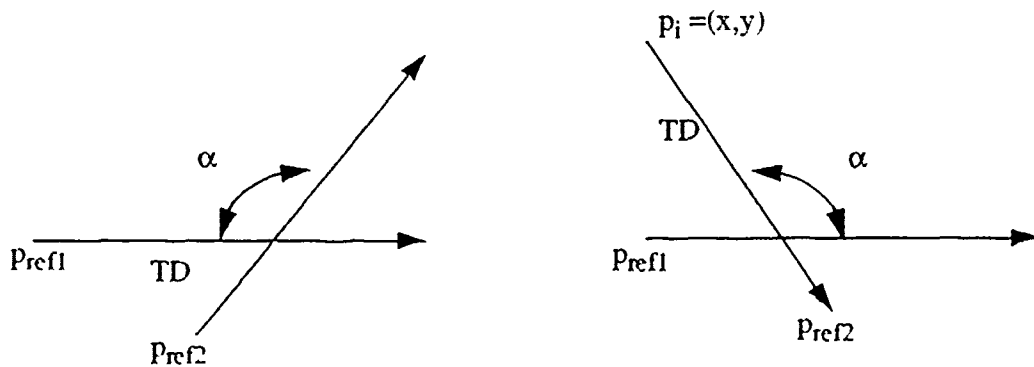


Figure 19. Rotation of a Transition Problem

The problems displayed in both figures are identical and will be handled by the simulator in the same manner. We are simply changing the vehicle's initial configuration to simulate our desired transition. Thus, we convert every transitioning situation into a transition to the x-axis. In this case we can quickly determine if the path oscillates or crosses the reference

path by simply checking the sign of the vehicle's position. If the value of the vehicle's y component ever goes negative, then the vehicle has crossed the reference path, and the transitioning distance is insufficient for that scenario. Thus, it is our desire to translate all experimental transitioning into transitions onto the x-axis.

The graphical results of many of the experiments can be viewed in appendix B. Table 1 below shows the experimental data for the simulations. Thus, the table is a concise means to display the relationship between the turn angle, distance constant, and transitioning distance. The rows represent different distance constants while the columns are different turn angle. The entry under any specific row /column is the required minimum transitioning distance for that particular problem. For example, the minimum transitioning distance for a 90 degree turn with a distance constant of 0.5 is 1.3units, depending on the units of S_0 .

3. Deriving A Transitioning Function

Given this experimental data we can now determine a general function for the transitioning distance. Examining the entries of the Table 1, we quickly come to the conclusion, that there exists no simple function to determine the transitioning distance for all different situations. Therefore, let's attack the problem by dividing the problem into two parts, the relationship between the turn angle and the transitioning distance, and the relationship between the distance constant and the transitioning distance. By solving these two simpler problems and then combining the results, we can derive a composite function which is suitable to fulfill our requirements.

a. Relationship Between TD and Turn Angle

First let us examine the relationship between the turn angle and the transitioning distance. It is an obvious observation that it takes a greater transitioning distance to complete a sharper turn than a wider turn. To find the underlying relationship between the two values we can plot the entries of the Table 1 for a given value of the

distance constant. We can then use this plot to extrapolate a distance function from the graph. The example we plotted was the case of $S_0 = 0.25$, and is illustrated in Figure 20.

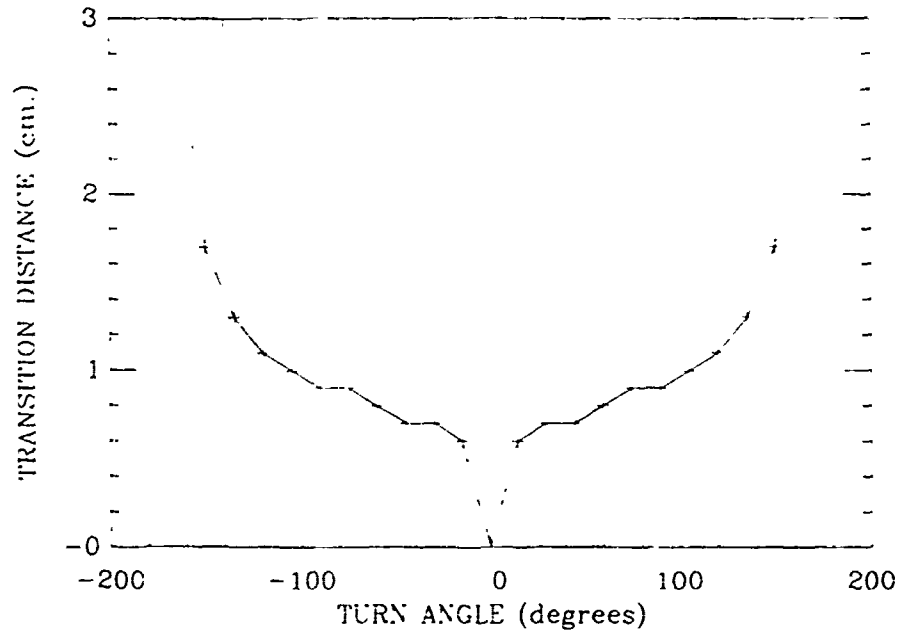


Figure 20. Experimental Minimum Transition Distance

We plotted the turn angle along the x-axis from -165 to 165 degrees. Since the negative turn angles are turns of the same magnitude but in the opposite direction as the positive turns, they require the same transitioning distance as their positive counterparts. On the y-axis is plotted the minimum required distance to complete a given turn angle. The results of our efforts clearly reveal a relationship between these two variables. The shape of the graph is similar to that of a parabola, however the base of the curve is much flatter. What we desire is to find a function which will map to this plot as close as possible, but never produce a transitioning distance smaller than that produced by our experimental data. Thus, let's assume the relationship is in the form of a fourth order polynomial, which share the basic shape of the parabola.

$$TD(\phi) = \frac{1}{1 - \left(\frac{\phi}{\pi}\right)^4} \quad (5.15)$$

Table 1: EXPERIMENTAL TRANSITIONING DISTANCE

Turn Angle (Degrees)	Distance Constant			
	$S_0=1.0$	$S_0=0.5$	$S_0=0.25$	$S_0=0.125$
0	0.0	0.0	0.0	0.0
15	2.0	1.1	0.6	0.3
30	2.1	1.1	0.7	0.3
45	2.1	1.2	0.7	0.4
60	2.1	1.2	0.8	0.4
75	2.2	1.2	0.9	0.5
90	2.3	1.3	0.9	0.5
105	2.5	1.4	1.0	0.5
120	2.9	1.6	1.1	0.6
135	3.5	1.9	1.3	0.7
150	4.8	2.6	1.7	1.0
165	8.8	4.8	3.0	1.9
180	----	----	----	----

Where, ϕ is the turn angle in radians and TD is the transitioning distance. By converting the turn angle into a ratio between the turn angle and the maximum possible turn, we simulate the curve of Figure 20. Equation (5.15) will produce a flat plot when the turn ratio is close to zero, while exponentially increasing the transitioning distance as the turn ratio approaches positive or negative unity.

If we calculate the transitioning distance by this function for those points in which we collected experimental data, we can determine the appropriateness of the function. We set up a plot similar to that of Figure 20 to display the results of equation (5.15). However, we included all turn angles between -170 and 170 degrees. Figure 21 displays the results. The plot illustrates a very similar curve to that of the experimental data. A quick check of the individual values also reveals that the function produces results which at all times is greater than those of the experimental data for the case of S_0 equal to 0.25. Therefore, this function is an appropriate approximation for determining transitioning distance. However, further comparisons between the output of this function and entries in Table 1 clearly illustrates that this function does not fulfill our needs. The problem is that the function does not take into account the effect altering the value of the distance constant has on the minimum transitioning distance.

b. Relationship Between TD and S_0

Equation (5.15) is unsuitable for non constant S_0 . However, this equation does provide us with a base case transitioning function. That is we can use this function to determine the transitioning constant, but modify its' output as appropriate dependent on the actual value of the distance constant. Thus, we are basically creating a composite function of a function dependent on the turn angle and a function dependent on the distance constant. To derive the function dependent on the distant constant we shall follow the same method we used with the turn angle. Therefore, once again let's plot the experimental results.

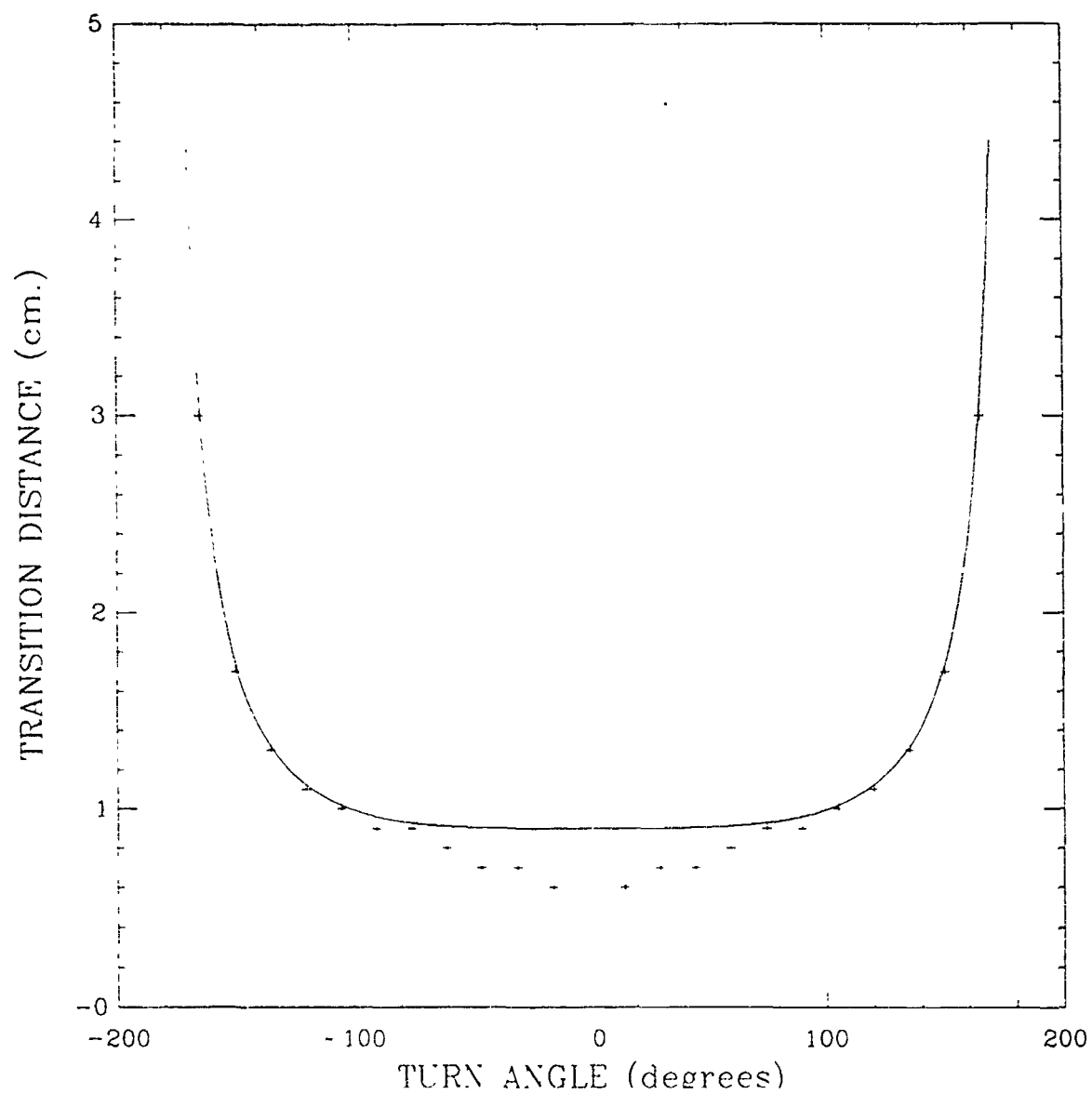


Figure 21. TD as Calculated by Table 1 and Equation (5.15)

This would produce 12 independent curves, one for each turn angle, which illustrate the relationship between the value of S_0 and TD . The case for turn angles of 150 and 15 degrees are displayed in Figure 22.

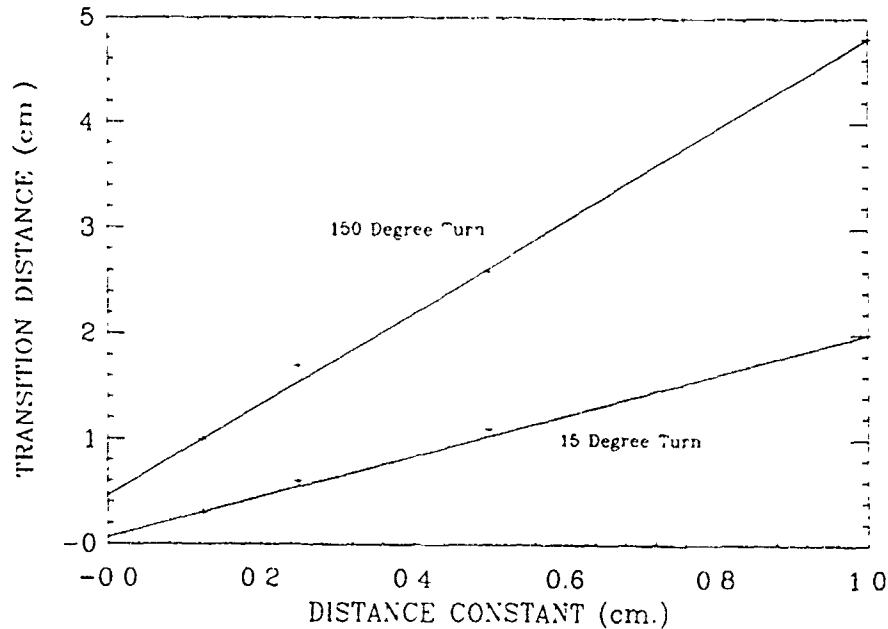


Figure 22. Relationship Between TD and S_0

This figures show that the relationship between the value of S_0 and TD is linear. In fact, each of the twelve plots indicate that this relationship is linear. Therefore, it may be possible to extrapolate a linear equation from these plots which will provide a correction factor for different distance constants.

If we begin looking for a relationship between the 12 individual plots, we quickly realize that the slope of the twelve plots vary significantly. This would make it very difficult to find a linear equation which would satisfy all possible cases. However, since our transitioning distance function (5.15) is based on the case of $S_0 = 0.25$, we can standardize the entries of Table 1. By plotting the standardized results we can more effectively isolate the effect the value of the distance constant has on the transitioning distance. Thus, Table 2 is an abridged version of Table 1 with the entries converted into ratios. The row/ column entries of Table 2 are the ratios between the transitioning distance for a specific turn angle

and S_0 , and the standard transitioning distance for that turn with $S_0 = 0.25$. The standard transitioning distance is the value outputted by equation (5.15) for a specific turn angle.

Table 2: TRANSITION DISTANCE RATIO

Turn Angle (Degrees)	Standard TD	Distance Constant Ratio			
		4.0	2.0	1.0	0.5
15	1.000	2.0	1.1	0.6	0.3
90	1.067	2.16	1.22	0.84	0.47
165	3.402	2.56	1.41	0.88	0.56

Now, if we plot the results of Table 2, we can find an appropriate correction factor. Figure 23 depicts the relationship between the ratios of transitioning distance and turn angles.

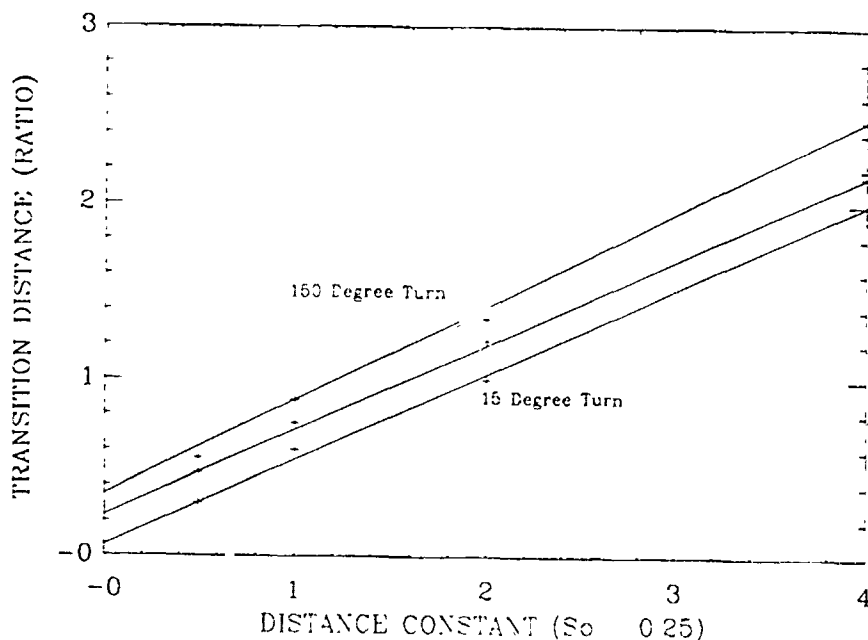


Figure 23. Plot of the Ratios of TD and S_0

In Figure 23 we plotted all three turn angles included in Table 2. As we can see in each case the resultant curve is very similar, with nearly the same slope. Thus, this indicates that we

can derive a linear correction factor for various distance constants. Using the slope intercept line equation we get,

$$CF(S_0) = \left(\frac{2.56 - 0.88}{4.0 - 1.0} \right) x + b \quad (5.16)$$

$$CF(S_0) = 0.56x + b \quad (5.17)$$

Where CF is the correction factor, x is the distance constant standardized with respect to the value of 0.25, and b is the y-intercept. Using equation (5.17) and the point (1, 0.88) from our plot we can solve for b . The value of b is 0.32, and equation (5.17) becomes,

$$CF(S_0) = (0.56) \frac{S_0}{0.25} + 0.32 \quad (5.18)$$

This equation was a suitable correction factor for various distance constant values, however we desire to match the experimental data as close as possible. Therefore, we adjusted this linear equation to produce results which would match the results of the transitioning function with those of the experimental data. We accomplished this simply by analyzing the twelve cases and interpolating appropriate values for the slope and intercept. The final results were,

$$CF(S_0) = 0.6 \left(\frac{S_0}{0.25} \right) + 0.3 \quad (5.19)$$

By combining the results of both relationships we can create our composite function. Thus, our function for determining the transitioning distance for all turn angle and distance constants is

$$TD(\phi, S_0) = [2.4S_0 + 0.3] \left[\frac{1}{1 - \left(\frac{\phi}{\pi} \right)^4} \right] \quad (5.20)$$

This equation is simply, but robust enough to apply in all possible transition problems. To verify the effectiveness of this equation, we shall compare the results of the equation to the experimental results. Table 3 gives us the transitioning distance results by using the derived

Table 3: CALCULATED TRANSITIONING DISTANCE

Turn Angle (Degrees)	Distance Constant			
	$S_0=1.0$	$S_0=0.5$	$S_0=0.25$	$S_0=0.125$
0	0.0	0.0	0.0	0.0
15	2.70	1.50	0.90	0.60
30	2.702	1.501	0.90	0.60
45	2.711	1.505	0.904	0.602
60	2.734	1.519	0.911	0.608
75	2.784	1.547	0.928	0.619
90	2.880	1.60	0.960	0.640
105	3.054	1.696	1.079	0.679
120	3.365	1.869	1.121	0.748
135	3.950	2.194	1.317	0.878
150	5.215	2.897	1.738	1.159
165	9.186	5.103	3.062	2.041
180	----	----	----	----

function. As we can see by simple comparison of the entries of Table 1 and Table 3, in all cases our derived function outputs a value equal to or greater than that derived by experimentation. Therefore, we can assume that our derived equation is appropriate, and will ensure sufficient maneuvering space while minimizing waste.

C. SUMMARY

In this chapter we introduced some possible transitioning schemes for our algorithm. The scheme on which we settled for is a variation of a dynamic scheme. Our scheme is based on transitioning a variable distance from the intersection point of two successive paths. This distance is determined by a general function which takes into account the degree of turn involved and the desired distance constant. The function was derived by gathering experimental data through use of our simulator. With the results of these experiments we interpolated a general function, which would fulfill our safety requirements while maximizing transitioning efficiency. Some graphical results are included in figures 24 to 26, which illustrate the effectiveness of our transitioning system. In Figure 24 we are completing a simple 90 degree turn with a S_0 value of 1.0. Figure 25 combine the line circle transitioning combinations, and illustrates the obstacle avoidance problem. In this graph we can assume the vehicle isolates an obstacle ahead on its' present path. The vehicle thus, transitions to an appropriate dimension circle to avoid the obstacle. When the vehicle is clear of the obstacle it transitions back onto its' original path. The remaining figure displays the transitioning between two circles.

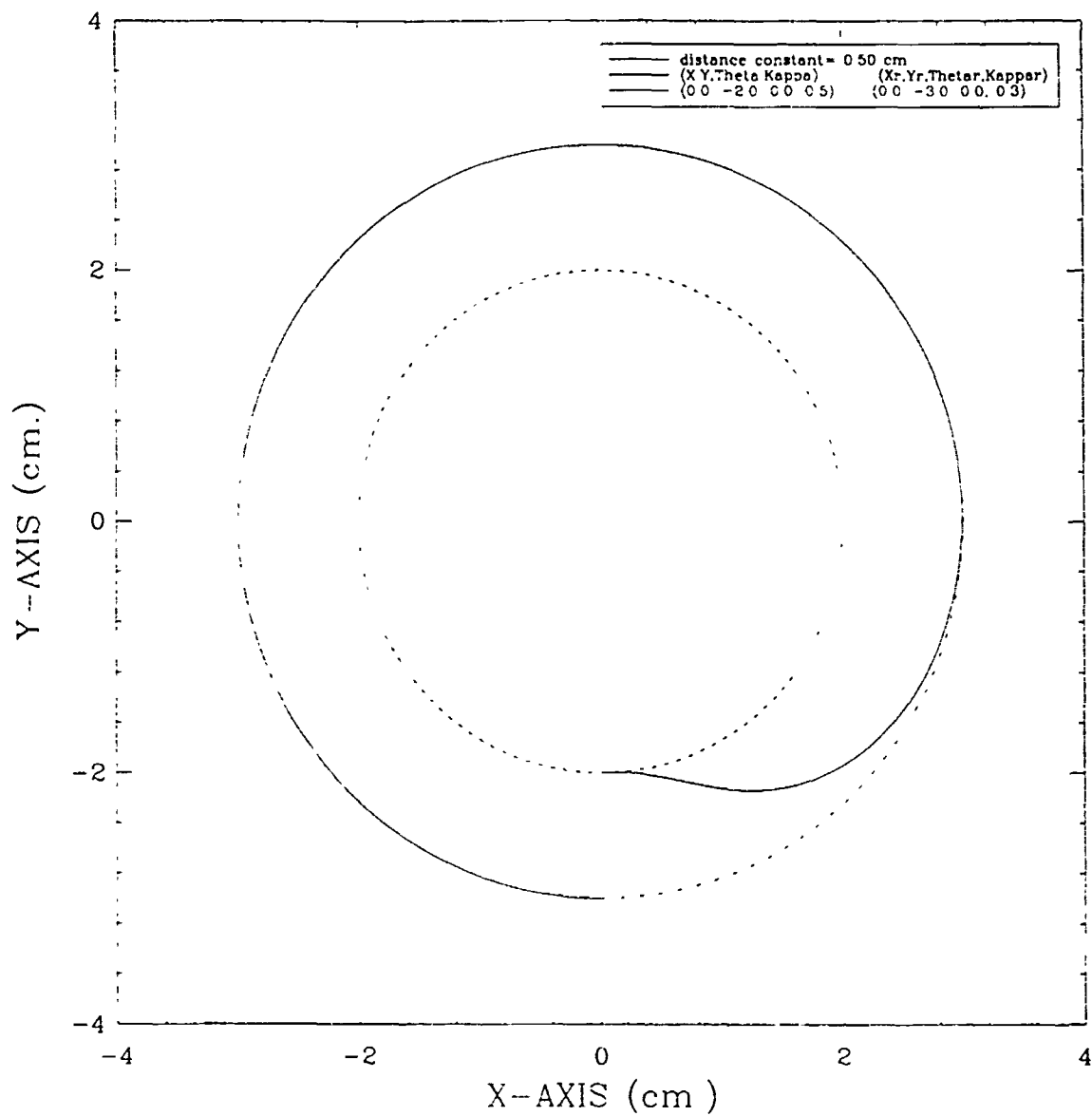


Figure 24. Transition From Inner to Outer Circle

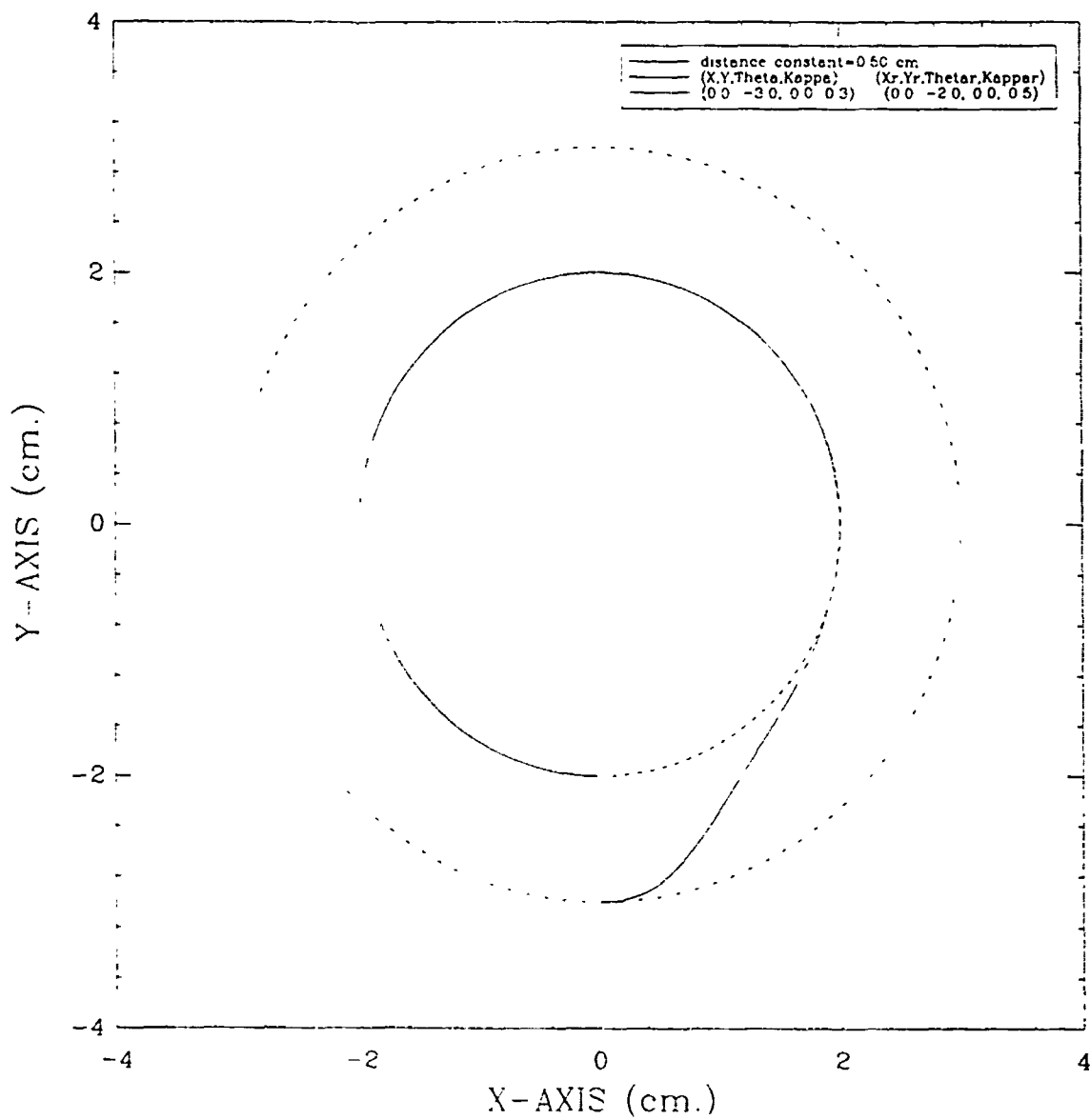


Figure 25. Transition From Outer to Inner Circle

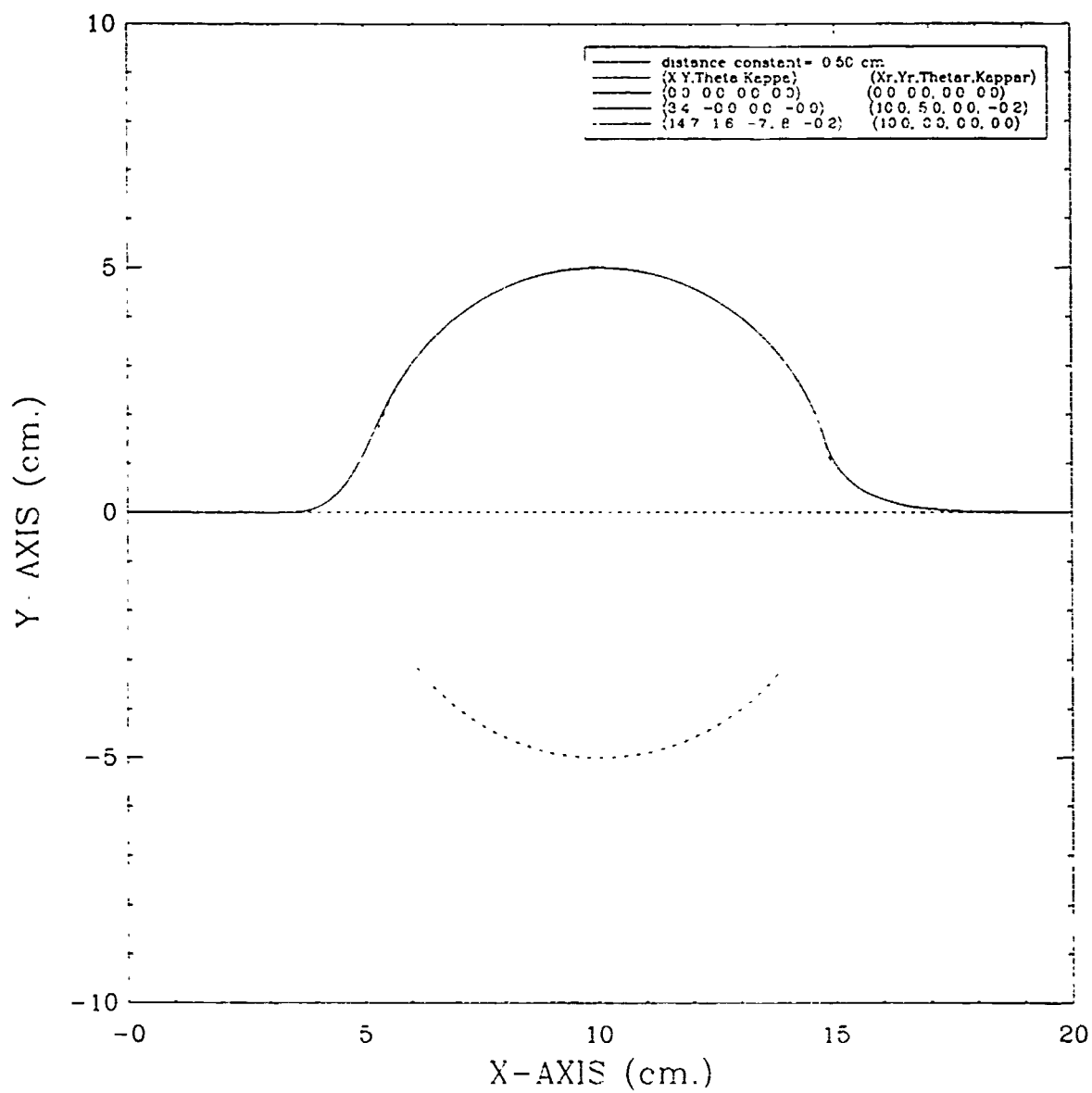


Figure 26. Transitions Involved in Obstacle Avoidance

VI. IMPLEMENTATION

One of the primary objectives of our research was to provide greater flexibility and maneuverability for an autonomous vehicle. The path tracking method which we propose would accomplish this by providing a simple, but powerful means to control a vehicle's motion. However, to maximize the benefits of our pathtracking algorithm, we should incorporate it with a traditional path planning algorithm, such as that described for Kanayama's Yamabico-11 [Ref. 3]. The resultant algorithm will enjoy flexibility, maneuverability, ease of operation, and increased control. Therefore, our final objective is to implement our path tracking algorithm into the software system of an autonomous vehicle. Our system has been designed with regards for all categories of vehicles. Therefore, our algorithm will be compatible for the Yamabico-11 robot.

A. OVERVIEW OF SYSTEM

1. MML System

The Yamabico-11 robot presently operates within the MML software environment. MML is an abbreviation for Model-based Mobile robot Language, and is a on-line library of mobile robot oriented functions. MML is a real time motion control system for off-line programming of a mobile robot. It consists of three primary components; a path planner, a motion generator, and a tracking controller module. The system accepts the user's desires in the form of a user program. The path planning module takes this input and converts it into a description of the path. This path description is used in conjunction with the velocity and acceleration specifications to produce reference configurations which track along the desired path. This data is compared by the tracking controller module to minimize the difference between the vehicle's position and the reference configuration. The controller module also consists of all the necessary mechanisms for locomotion capabilities including the vehicle's motor and wheels [Ref. 3].

B. PATH TRACKING SYSTEM

The system we envision shall be comparable to the existing system. There shall be two modules which parallel the path planner, and the generator. These modules shall be referred to as the path descriptor, and the executor. The output of these modules shall provide input to the existing tracking controller module. The process shall be initiated by the creation and linking of a user module to the path tracking system. Our system shall be written in the C language within the Unix environment.

1. User Program

The user program shall be designed for off-line programming of the robot. Its purpose is to provide a means for the user to program the robot's motion. The user module shall consist of an indeterminate number of path tracking locomotion functions, which describe the robot's desired motion. The available commands are described in appendix D. The sequence of locomotion commands must be completed by listing the execute command after all other commands. Execute acts as a signal to the system to transfer control to the executor module. The user program shall be executed under the supervision of the MML operating system. The path tracking functions within the user module shall be linked only at compile time. This module thereby acts the same as it did for the user program of the MML system.

2. Path Descriptor

The path descriptor module shall perform a similar function to the path planner module of the MML system. It shall take a user program and convert it into a description of the desired motion in terms acceptable to the executor module. This shall be accomplished by taking the sequence of commands of the user program and placing their corresponding path information onto an instruction stack. The stack shall comprise of three components: the class, the reference path, and the transition point. An example of a typical instruction stack is given in Table 4.

Table 4: INSTRUCTION STACK

Class	Reference Path	Intersection
spath	$(x_{r5}, y_{r5}, \theta_{r5}, \kappa_{r5})$	---
ppath	$(x_{r4}, y_{r4}, \theta_{r4}, \kappa_{r4})$	$(x_{i3}, y_{i3}, \theta_{i3}, \kappa_{i3})$
path	$(x_{r3}, y_{r3}, \theta_{r3}, \kappa_{r3})$	$(x_{i2}, y_{i2}, \theta_{i2}, \kappa_{i2})$
ppath	$(x_{r2}, y_{r2}, \theta_{r2}, \kappa_{r2})$	$(x_{i1}, y_{i1}, \theta_{i1}, \kappa_{i1})$
path	$(x_{r1}, y_{r1}, \theta_{r1}, \kappa_{r1})$	---

The class is the type of path locomotion function. There are only three possible options: path, ppath, and spath. The path function is an indefinite move command onto the path described by the given configuration. The function ppath is a partial path function, which is simply a path with a distinct endpoint for transitioning purposes. The path and the endpoint are described by the same configuration. The last available class is spath, which is a stop path function. The accompanying reference configuration refers to the location on the path for which the robot is to stop.

The reference path and the intersection are configurations which describe the desired motion. The user program shall have a path configuration for each command entered to delineate a specific path. This configuration shall be placed on the stack as the desired reference path. The system shall then use the class and successive reference path configurations to determine the intersection point. This intersection point shall be used to determine when to transition between successive paths.

3. Executor

The executor module will be initiated by the user program. After all locomotion functions have been placed onto the instruction stack the command execute will pass control to the executor. The objective of the executor is to derive the dk/ds necessary to

merge onto the reference path. This is simply a call to our path tracking algorithm with the specific instruction stack. After the robot's image has reached the determined transitioning point the next entry of the instruction stack shall be executed. This shall be continued until all instructions have been executed. At this point the robot shall switch to the stop state.

C. SUMMARY

One of the primary objectives of our research was to expand the motion control capabilities of all categories of autonomous vehicles. In accordance with this we developed a simple but powerful path tracking algorithm. To show the benefits of such an algorithm we intend to implement it within the operating system of a mobile robot, Yamabico-11. The implementation that we decided to undertake was based on a parallel system to the present Yamabico-11 operating system, MML. This would allow us to employ applicable existing modules of the MML system, while developing other modules to tailor the system to our path tracking algorithm. Our ultimate desire is to create a operating system which could be quickly operational, and expanded to subsume the MML system in the future.

VII. SUMMARY AND CONCLUSIONS

A. CONTRIBUTIONS OF RESEARCH

The benefits of our path tracking algorithm are primarily within two areas. The first area is concerned with the effective control of an autonomous vehicle. Most present motion control algorithms for autonomous vehicles are based on a exact positioning. That is, the vehicle is controlled to pass through certain desired points. This method is extremely useful and versatile. However, many instances exist where the exact positioning of the vehicle is not as important as the orientation. In our path tracking algorithm we emphasize translating our desired motion into simple terms. We make it considerably easier for the user to program a vehicle's motion by alleviating the need to predetermine points along the motion path. In addition, the general concept of motion behind our path tracking algorithm provides a more general approach, which is as encompassing as the more prevalent point to point approach. The path tracking approach allows the user to maintain the vehicle on curves which would be impossible to replicate in the point to point approach. Both methods offer the user some advantages, however it is best when the methods are combined within a vehicle's software. This greatly enhances the vehicle's motion control flexibility and user operability.

The second area which this research directly benefits is that of obstacle avoidance. In our path tracking scheme we included tracking along circular paths. The primary reason we implemented a path tracking algorithm for circles is to provide a means for obstacle avoidance. Our intentions are, that when a vehicle's sensors detect an obstacle, we would have an automatic transition from the path the vehicle is presently on to a circular path which would circumvent the obstacle. After avoiding the obstacle the vehicle would then transition from the circle back onto the original path. A more complex scheme can be

developed on this idea to handle multiple obstacles and dynamic moving obstacles. However, the path tracking scheme presents the foundation for these ideas.

B. FUTURE RESEARCH

The path tracking scheme developed in this research is a simple algorithm. This simple nature provides ample opportunities to improve or expand the system. The possible improvements can be conducted mainly in two areas, the mathematical model, and the transitioning scheme.

We developed the algorithm primarily through experimentation. That is we attempted to find a valid control method through hunches, and gut instincts. This is not the most appropriate means, and it usually does not yield the best results. In deriving our mathematical model we included assumptions to ensure a simple control function. In addition, the output of the system is not precisely our desired output. A very slight oscillation exist in simulations conducted with the value of S_0 less than one. Although the magnitude of the oscillation is on the order of 10^{-6} , it may indicate a problem with our method. Therefore, for these reasons it may be beneficial to invest further research into a similar mathematical model. This may provide a solid mathematical approach, which produces superior results to those presently attained.

The second area which could be improved upon is the transitioning scheme. We developed our transitioning scheme primarily as a result of the limits of our resources, both time and money. With greater resources available we could investigate whether an expanded transitioning scheme would be feasible and economical. If the system merited improvement, we could expand the transitioning scheme by taking more factors into account in determining the appropriate transitioning distance. The goal would be a fully dynamic scheme, which provides the safest, most efficient transitions possible.

Other research can be accomplished by expanding the present algorithm. The algorithm could be expanded by expanding the systems' applicability, and functionality. The applicability of the system could be expanded by increasing the scope of possible

reference paths. Thus, we could implement a system in which a robot could path track along any given planar curve, including ovals, parabolas, and cubic spirals. This would be extremely useful in developing a path tracking scheme based on voronoi boundaries within an environment. Another possibility is to expand the algorithm to a three dimensional system. This could be useful in motion control of an unteathered submarine, or a mobile drone. These options can be accomplished by manipulating the present motion control aspects of our system and implementing them in other areas.

Another research area which could be started in conjunction with our work is obstacle avoidance. The path tracking algorithm lays the groundwork for a possible obstacle avoidance algorithm for autonomous vehicles. This is by far the most significant area of possible future research. The path tracking algorithm is simple, and provides a simple reliable means to control motion of a vehicle. This is ideal for an obstacle avoidance scheme. The groundwork is established, with further work along these lines an effective means to transit unknown environments could be close at hand.

APPENDIX A

To test the correctness of our control equation, we shall compare the results of the output of our simulator with that of an assumption free equation based on our system. To do this we shall use equation (3.22) and solve for the coefficients A, B, and C. Let's look at equation (3.22) and its' first and second derivative.

$$y = \left(\frac{A}{2}x^2 + Bx + C\right) \epsilon^{-kx} \quad (\text{A.1})$$

$$y' = \left(Ax + B - \frac{kA}{2}x^2 - kBx - kC\right) \epsilon^{-kx} \quad (\text{A.2})$$

$$y'' = \left(A - kAx - kB - k\left(Ax + B - \frac{kA}{2}x^2 - kBx - kC\right)\right) \epsilon^{-kx} \quad (\text{A.3})$$

With these equations we can solve for the coefficients.

$$y_{x=0} = y_0 \quad C = y_0 \quad (\text{A.4})$$

$$y'_{x=0} = \tan \theta_0 \quad B = ky_0 + \tan \theta_0 \quad (\text{A.5})$$

$$\left. \frac{y'}{(1 + (y')^2)^{3/2}} \right|_{x=0} = \frac{A - 2kB + k^2C}{(1 + \tan^2 \theta_0)^{3/2}} = \kappa_0 \quad (\text{A.6})$$

$$A = 2k(\tan \theta_0 + ky_0) - k^2y_0 + \kappa_0 |\cos \theta_0|^3 \quad (\text{A.7})$$

To find the specific solution for the coefficients we use the initial position of the robot for a given problem. To simplify the problem we shall choose a problem in which the robots initial x coordinate is equal to zero. Thus, we shall solve the coefficients for the problem of the robot merging from an initial position $p_i = (0, 1, 0, 0)$ onto the x-axis. Now, we can calculate the actual coefficients for this problem. Using equation (A.4) we solve for C.

$$C = y_0 = 1 \quad (\text{A.8})$$

Now, we solve for B using equations (A.5) and (A.8).

$$B = k(1) + \tan(0) = k \quad (\text{A.9})$$

And the value of A is calculated,

$$A = 2k(k) - k^2(1) + (0)|1|^3 = k^2 \quad (\text{A.10})$$

Since, we have assigned the value of k to be equal to the inverse of S_0 , we have all the information to plot the desired path free from our earlier assumptions. Thus, we established an algorithm which would plot the following function.

$$y = \left(\frac{k^2}{2}x + k\lambda + 1\right) e^{-kx} \quad (\text{A.11})$$

Thus the output of this equation was plotted in conjunction with the output of our simulator for various values of S_0 . The results of both outputs clearly show the degree of difference between our simulator and the assumption free output.

APPENDIX B

In deriving our transition distance function, TD , we conducted numerous simulations of transitioning between paths. The objective was to find the minimum distance required to complete a specific turn angle. We also conducted these experiments to determine the effect varying the value of S_0 had on the minimum distance. The results were reported in Table 1, and the graphical results can be found in figure 22 to 32.

In conducting the minimum transitioning distance simulation we found a problem with the output of our algorithm. The objective of the testing was to find the minimum distance to complete the turn without any oscillations or crossing of the reference path. However, as we conducted the experiment we found that this requirement was impossible to meet in some circumstances. As we decreased the value of S_0 we found that a small degree of oscillation was present, regardless of how much we increased the transitioning distance. This oscillation was present when the value of S_0 was 0.5, 0.25, and 0.125. The oscillation was a single crossing of the x-axis on the order of 10^{-6} . Therefore, we regarded this oscillation as insignificant, and proceeded with our testing. However, in the cases where the oscillation existed we determined the minimum distance as the point where this slight oscillation first appears.

The output of our testing was four individual paths for each specific turn angle. All of the paths were plotted on the same graph for easy comparison. The paths can be distinguished from one another since those with the widest and slowest turns have the higher values for S_0 . Also included on the graph was a dotted directed line, which provides the vehicle's initial reference orientation. The ledger found in the lower right-hand corner of each graph tells the necessary initial position of the vehicle to complete the turn for a given S_0 value.

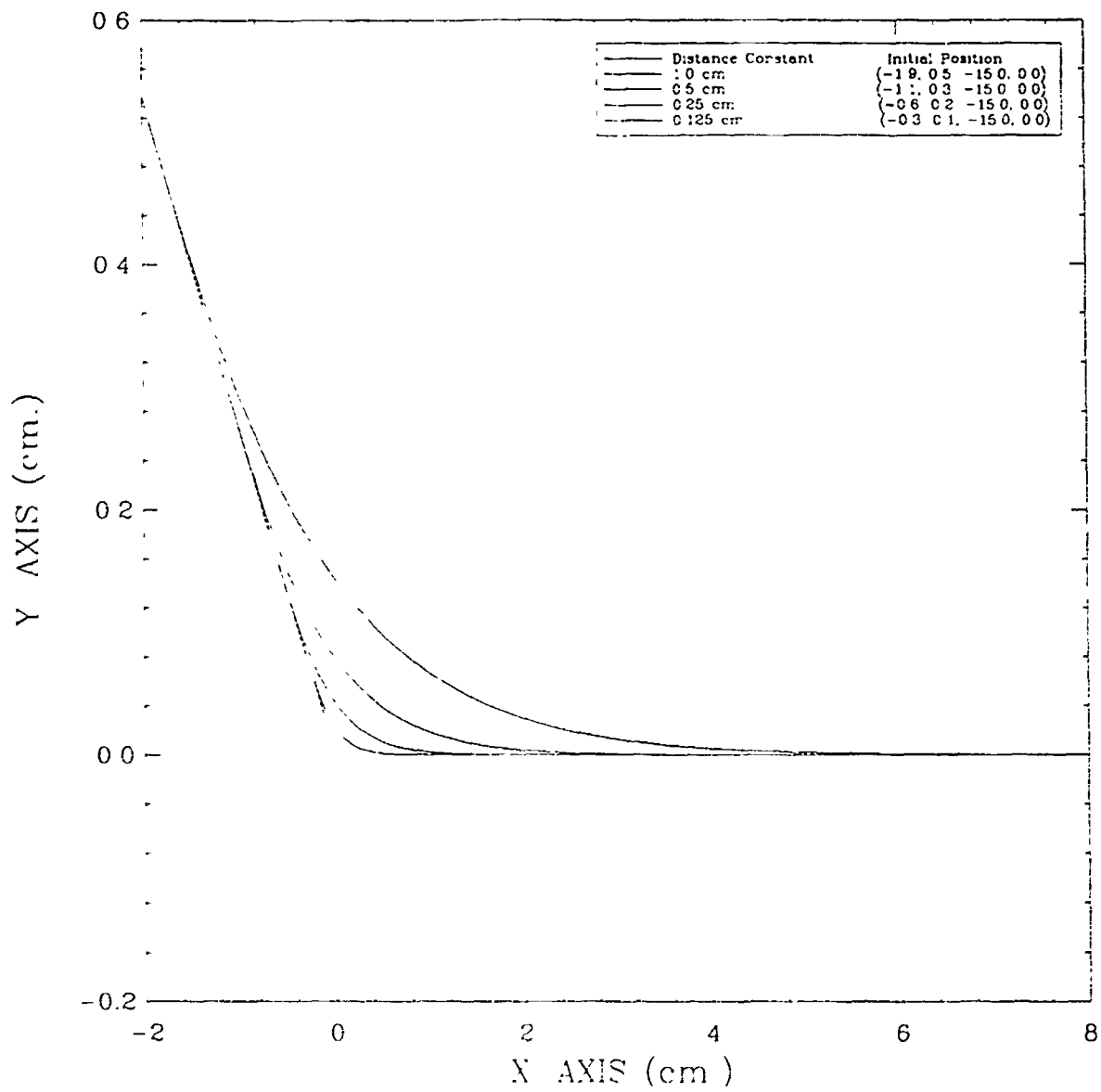


Figure 27. A 15 Degree Turn

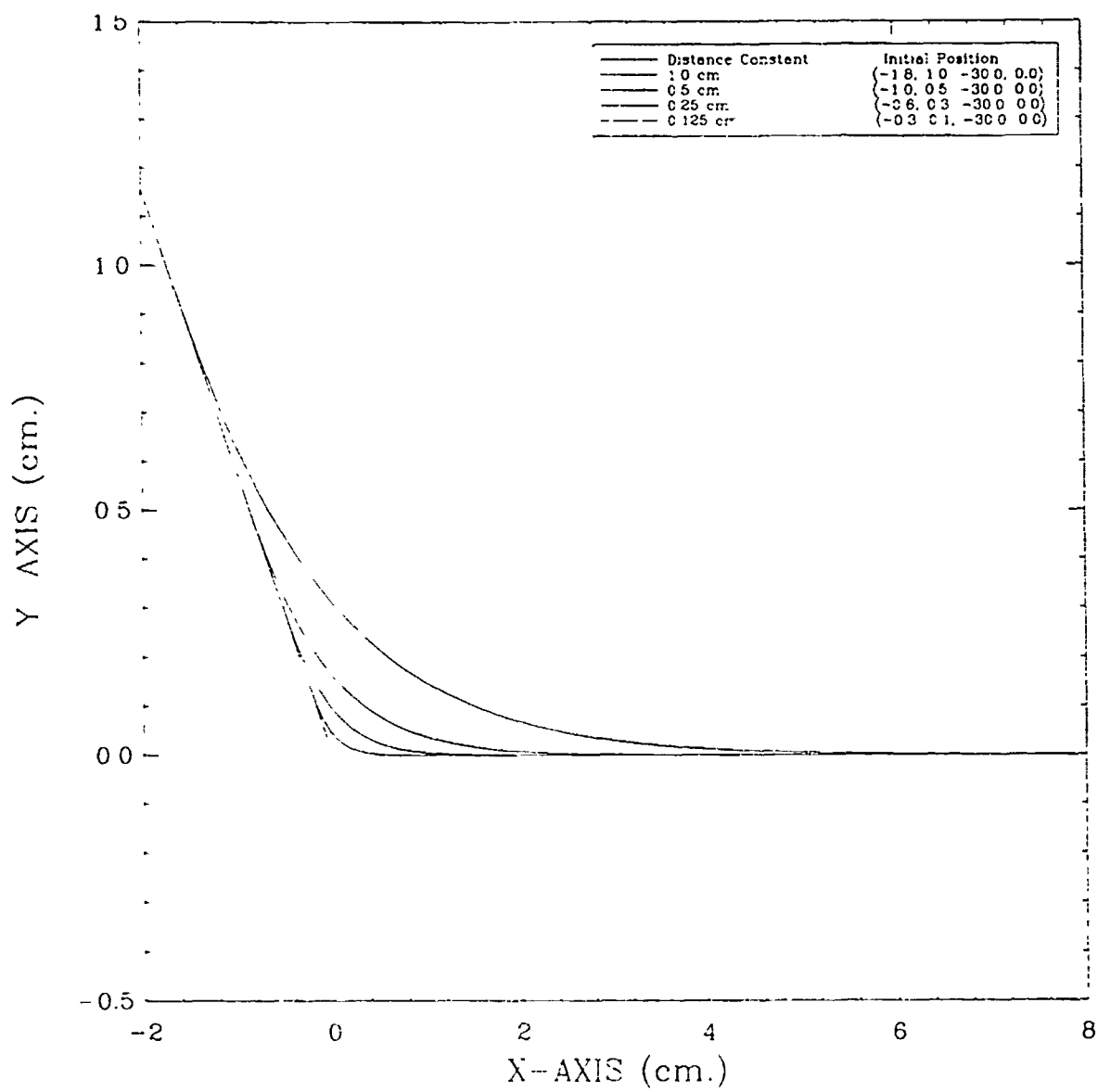


Figure 28. A 30 Degree Turn

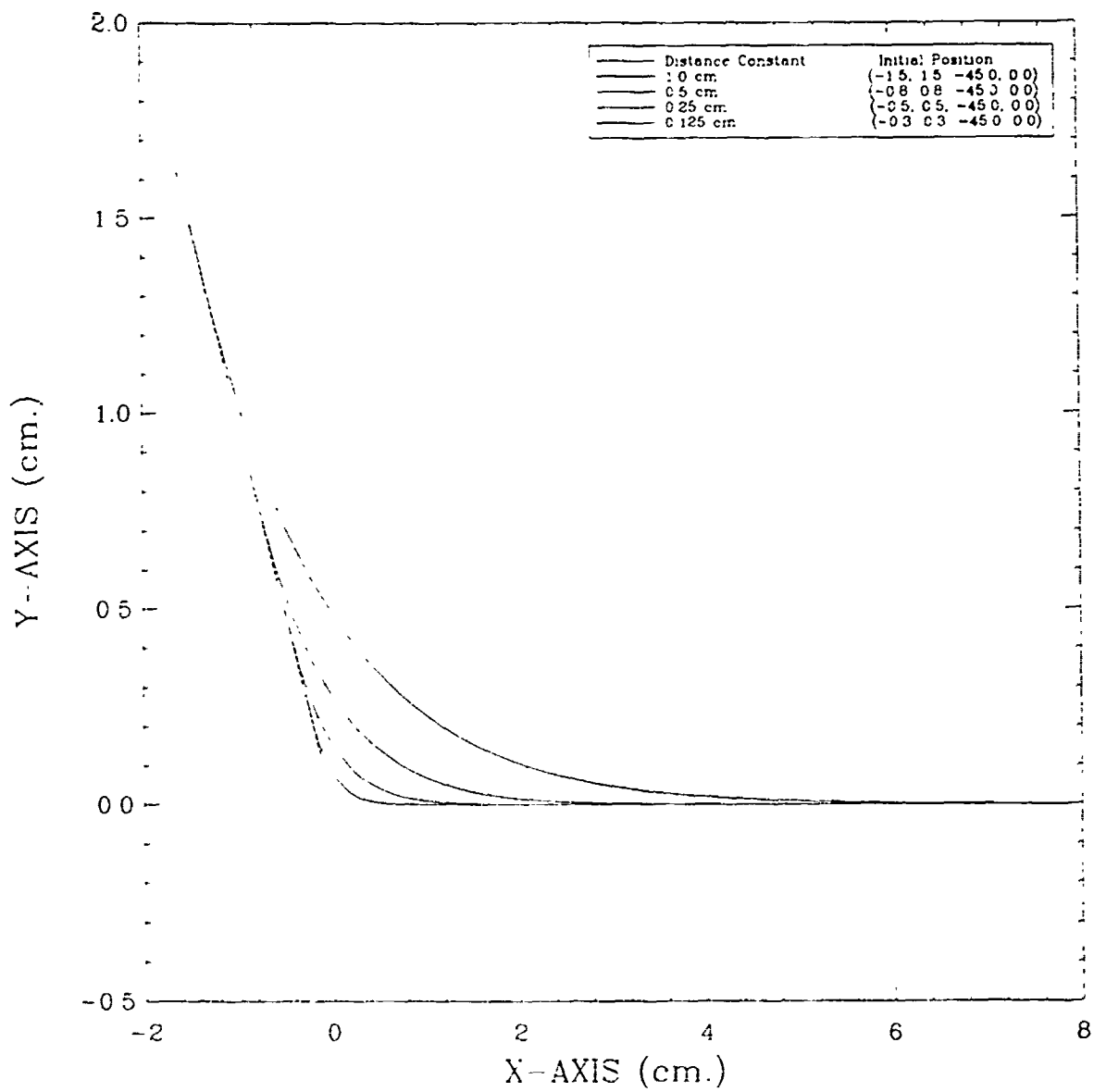


Figure 29. A 45 Degree Turn

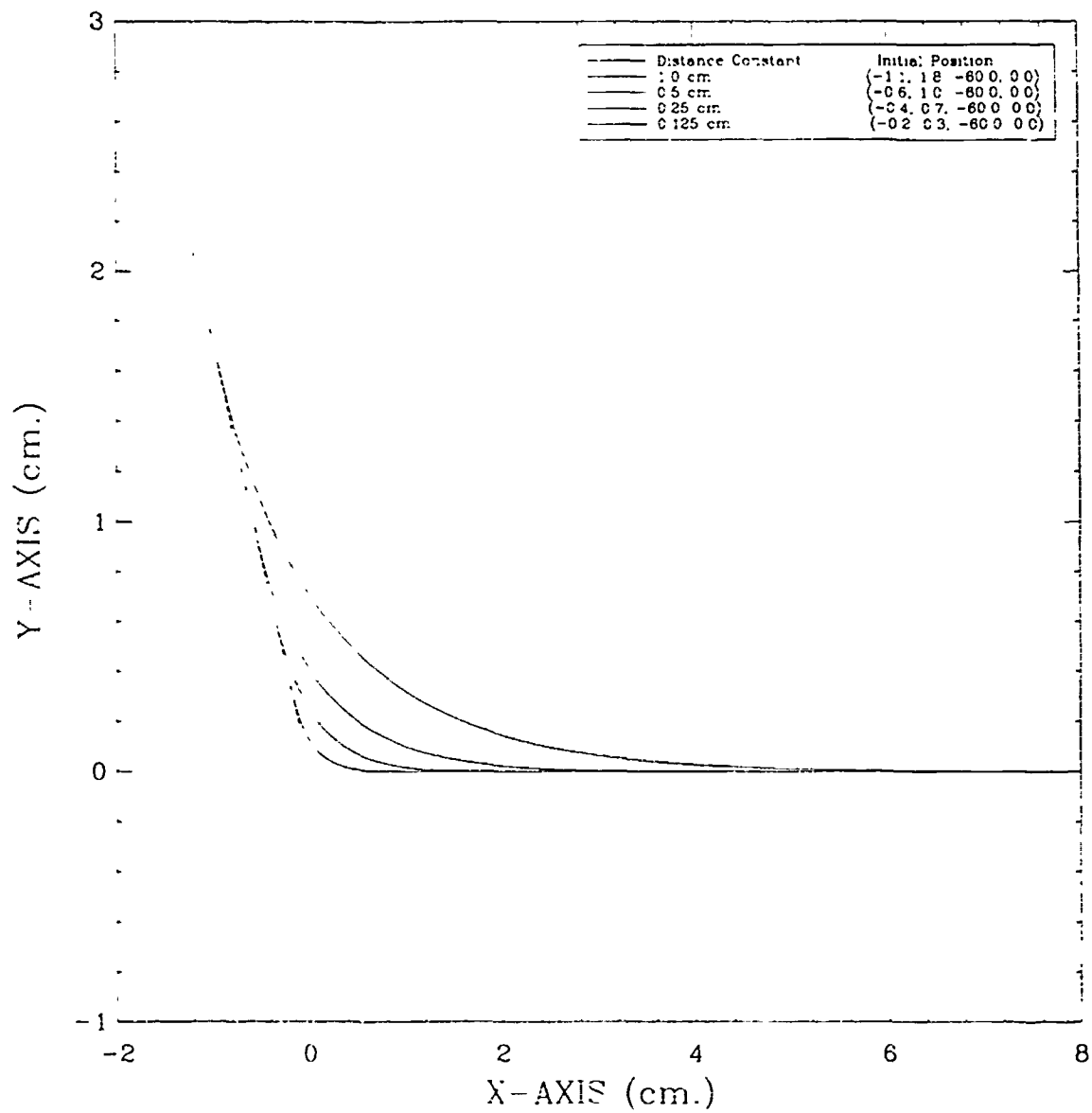


Figure 30. A 60 Degree Turn

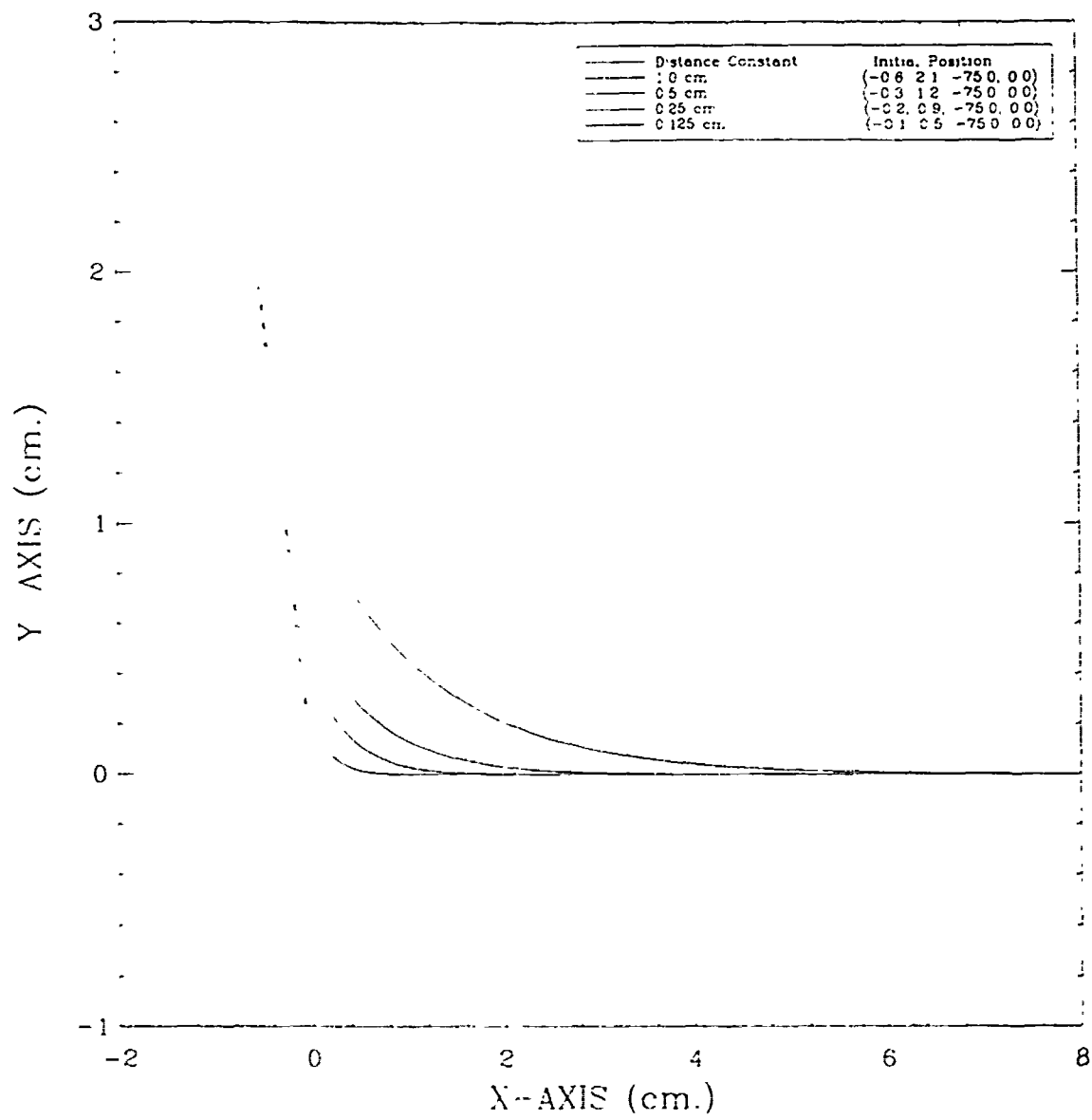


Figure 31. A 75 Degree Turn

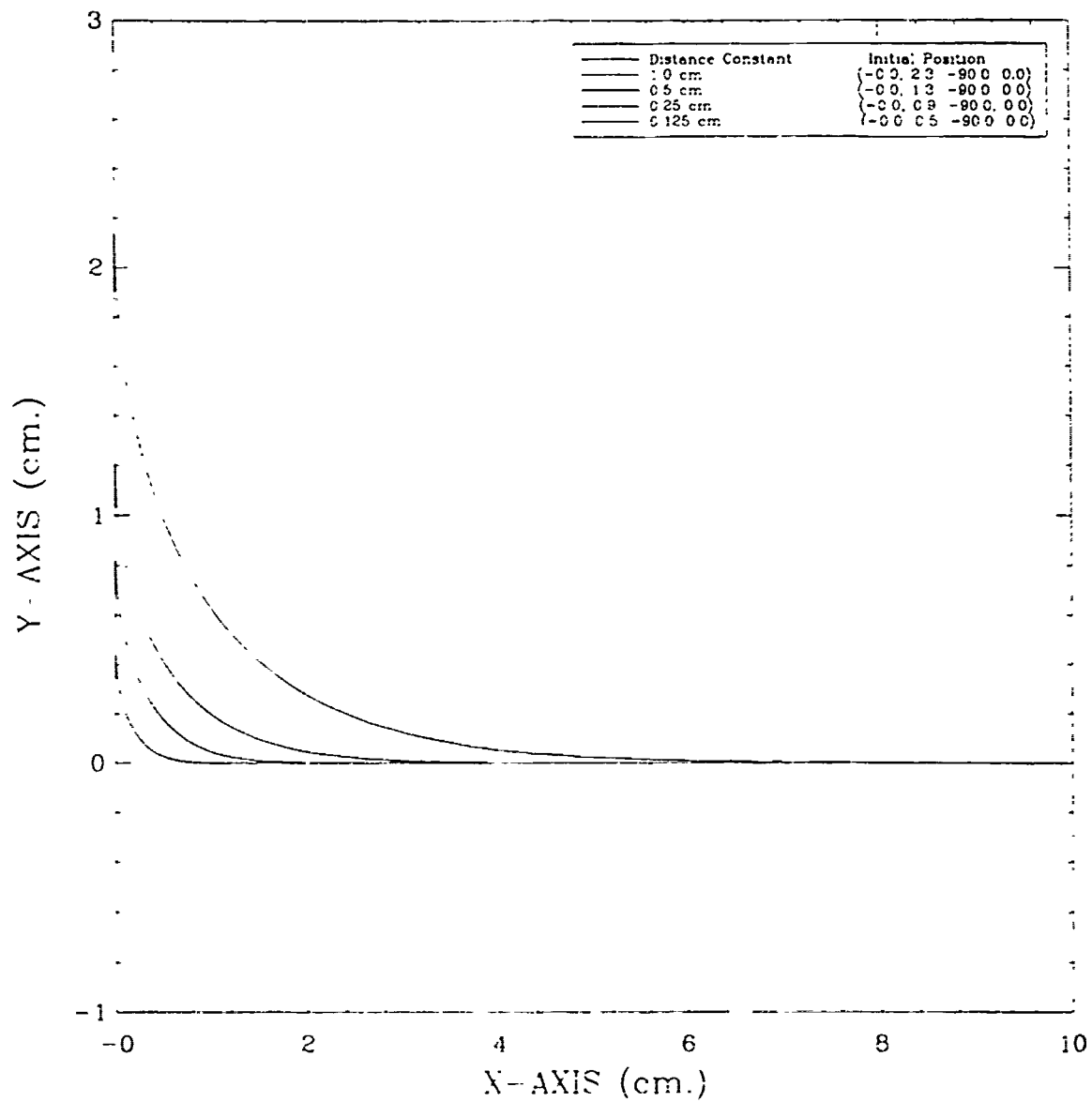


Figure 32. A 90 Degree Turn

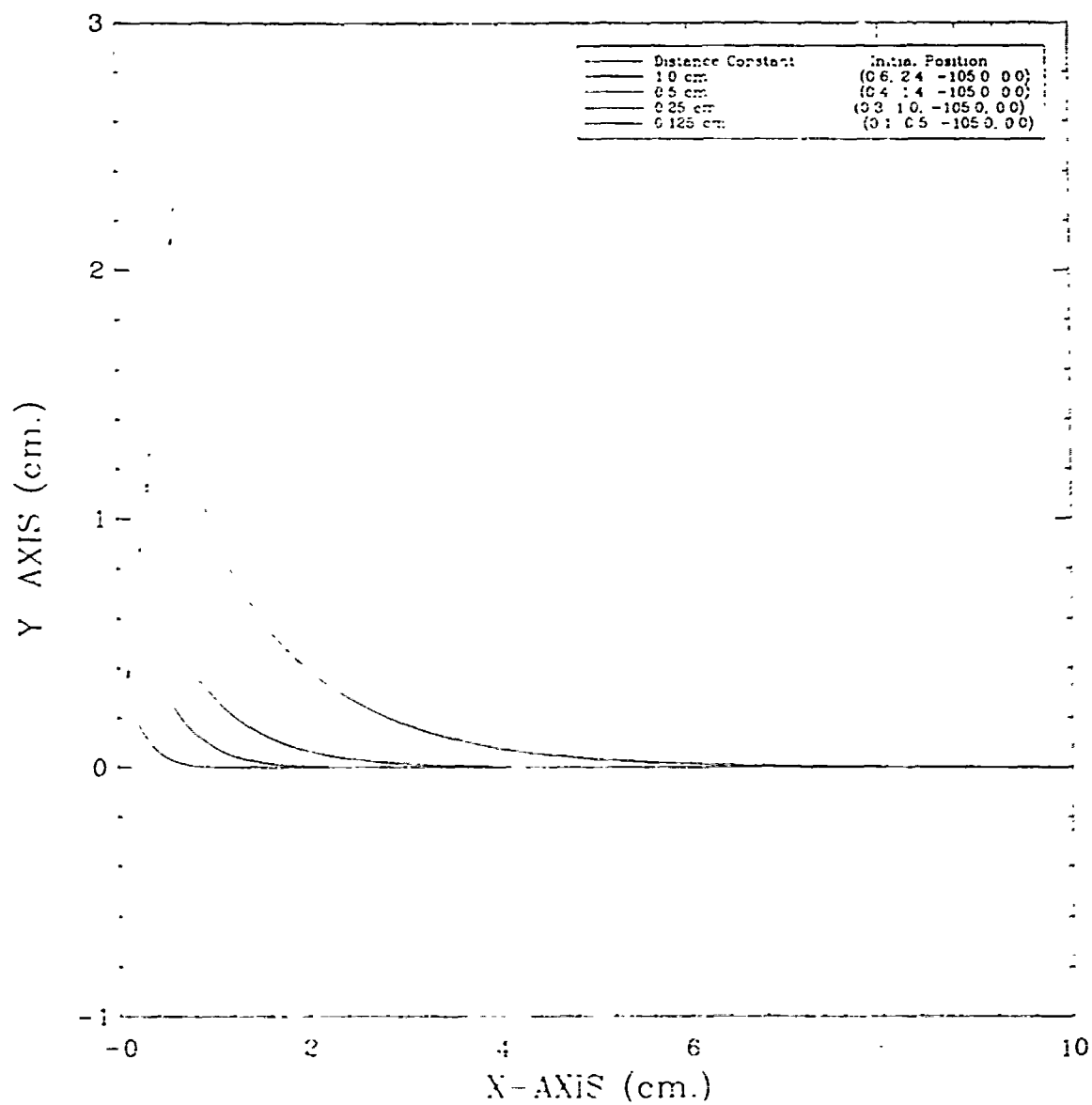


Figure 33. A 105 Degree Turn

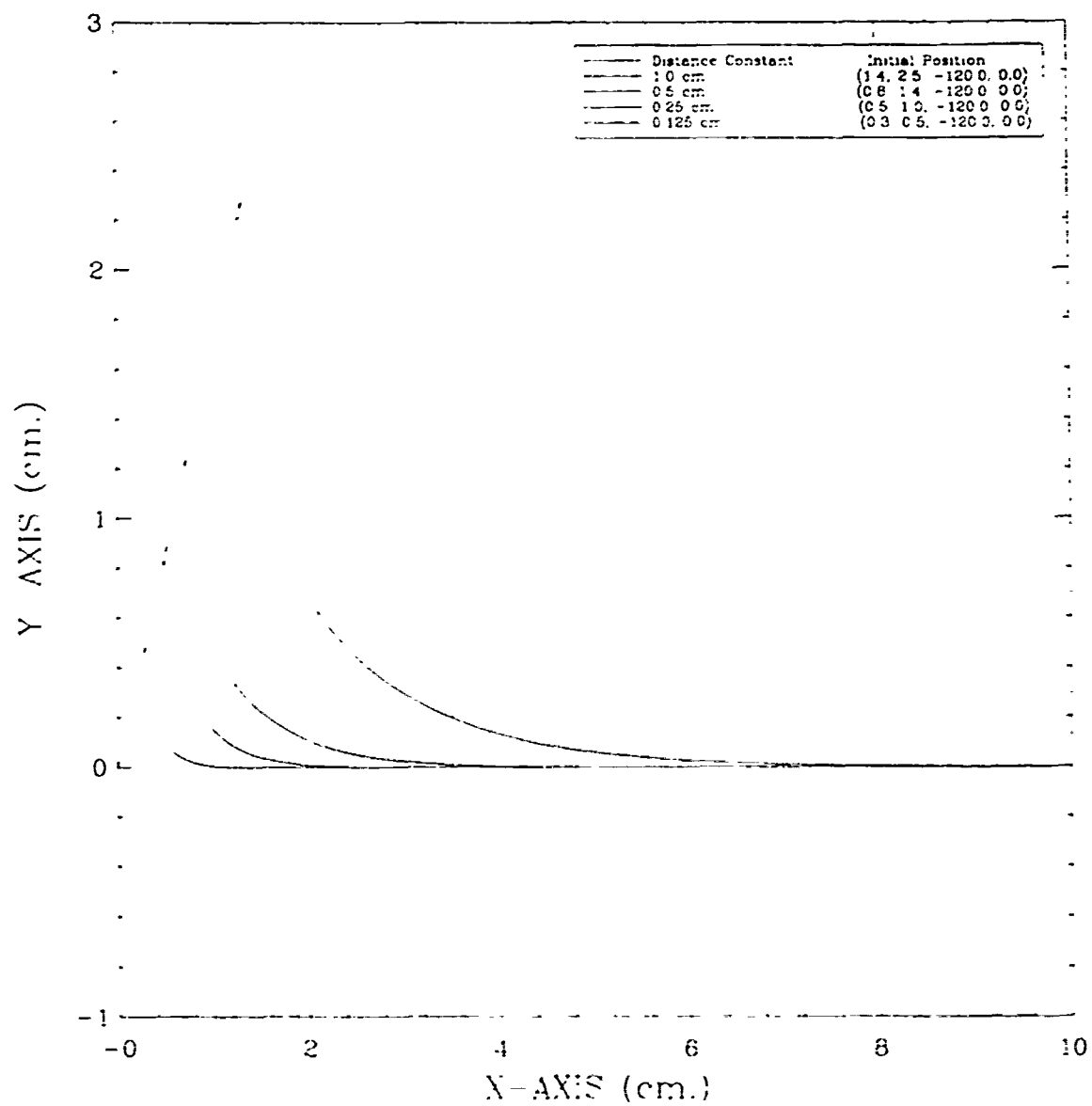


Figure 34. A 120 Degree Turn

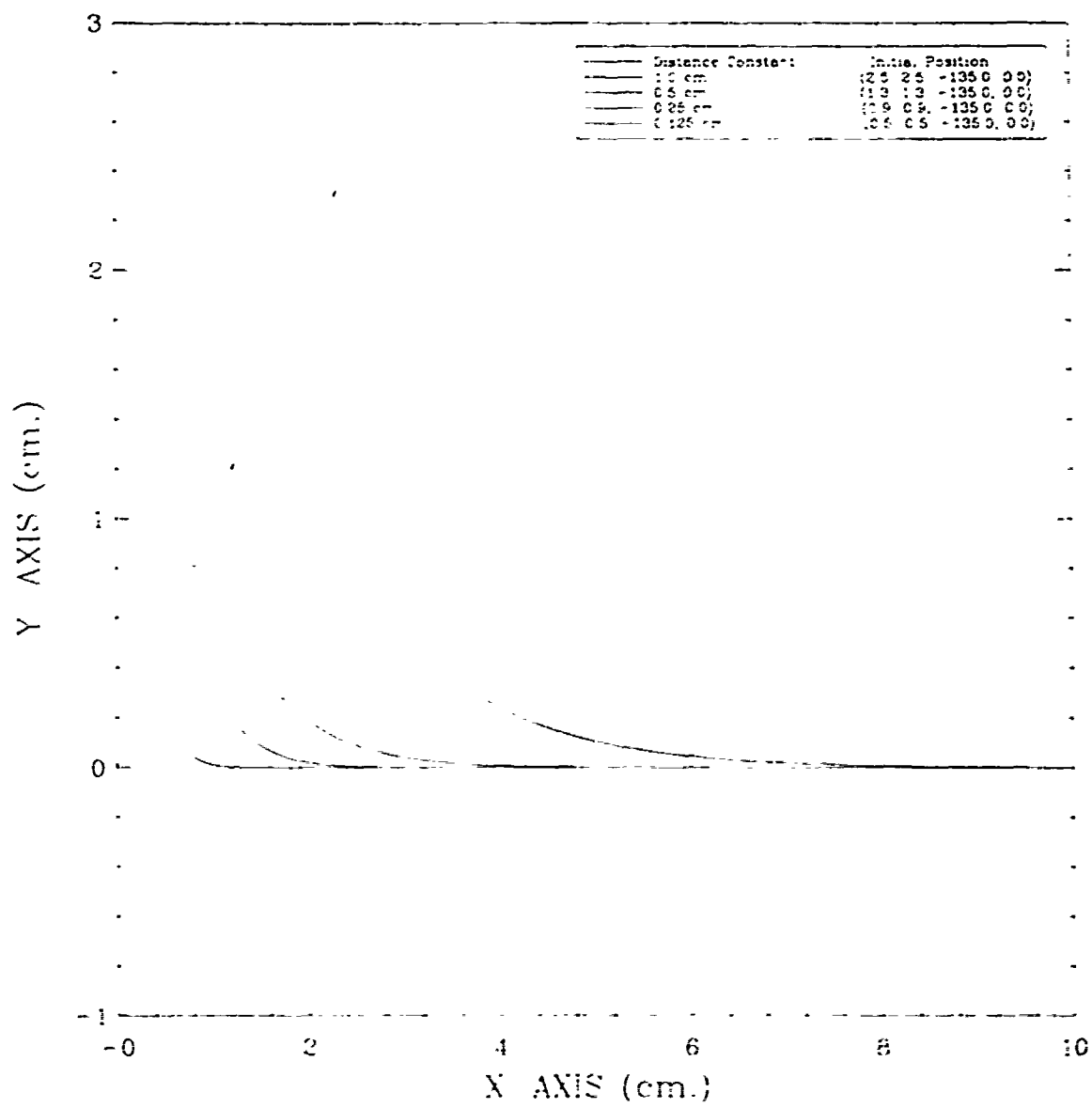


Figure 35. A 135 Degree Turn

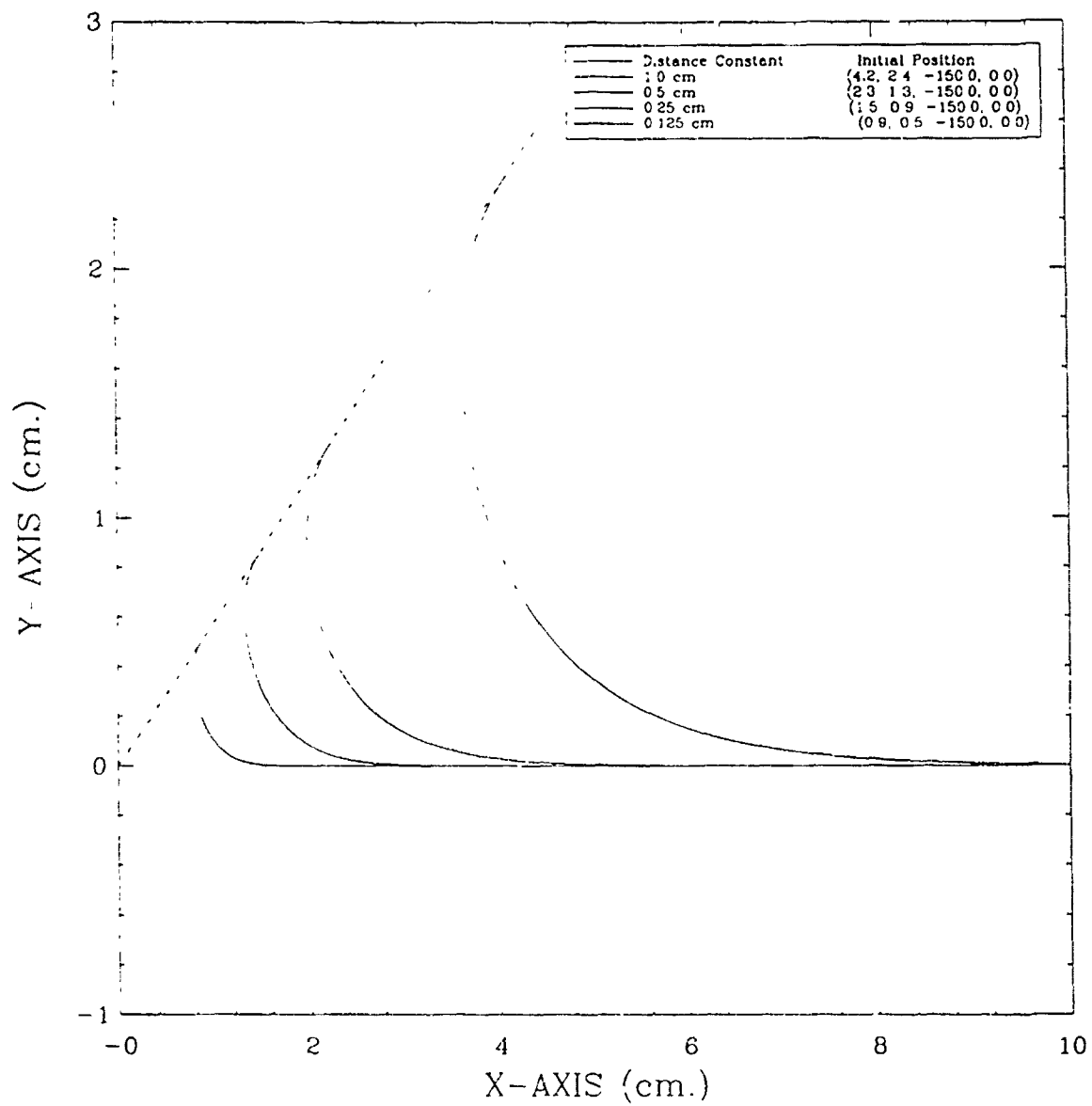


Figure 36. A 150 Degree Turn

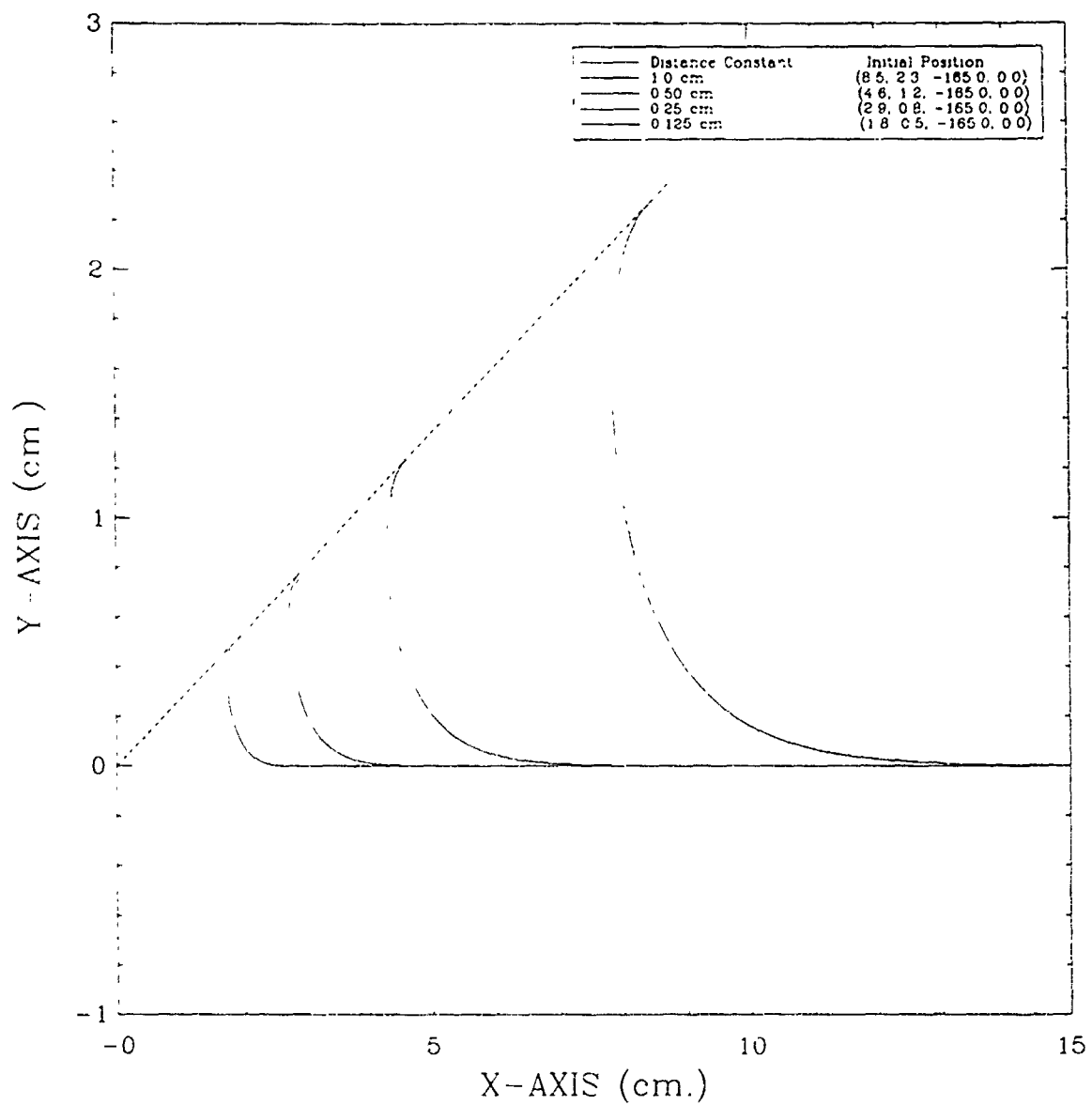


Figure 37. A 165 Degree Turn

APPENDIX C

```
/* The header file used in conjunction with the path tracking
algorithm of pathld.c */
```

```
/* Constants needed within the program. */
```

```
#define PI 3.14159265
#define DPI 6.28318530
#define RAD 57.2957795 /* 180 / PI */
#define DELTA_TIME 0.001
#define MAXSTRING 80
#define VELOCITY 10.0
#define MAX_DELTA_KAPPA 1.0
#define N 10
```

```
/* Structure for reference paths, vehicle's position, image,
and the intersection point between paths. */
```

```
typedef struct {
    double x;
    double y;
    double theta;
    double kappa;
} CONFIG;
```

```
/* Global variables. */
```

```
typedef struct {
    double x0;
    double y0;
} POINT;
```

```
/* Procedures defined within the program. */
```

```
CONFIG current_config, intersect;
FILE *out_path, *info_file;
CONFIG path_array[N];
POINT intersect;
int ITERATIONS;
```

```
double i_x0, i_y0, dkappa, delta_d, DIST_CONSTANT;
```

```
CONFIG      initial_configuration();  
CONFIG      initial_reference();  
void        initialize_parameters();  
void        initialize_files();  
CONFIG      intersection_point();  
void        print_to_file();  
void        update_velocity();  
CONFIG      update_configuration();  
double      calc_kappa();  
CONFIG      calc_image();  
double      calc_kappa_dot();  
void        time_to_transition();  
double      norm();
```

```

#include "stdio.h"
#include "math.h"
#include "path.h"

main()
{
    int    path_num = 0;
    int    done = 0;
    double dkappa_ds = 0.0, speed, delta_dist, aa, bb, cc;
    CONFIG path, image;

    vehicle = initial_configuration();
    path = initial_reference();
    initialize_parameters(&aa, &bb, &cc);
    initialize_files(path);
    intersect = intersection_point(path_num);
    do {
        print_to_file(image path, dkappa_ds);
        update_speed(&speed, &delta_dist);
        update_image(&image, vehicle, path);
        delta_d = delta_distance(vehicle, image);
        dkappa_ds = kappa_dot(vehicle, image, delta_d, delta_dist,
                               aa, bb, cc);
        update_configuration(&vehicle, dkappa_ds, delta_dist);
        time_to_transition(&path, &path_num, &done, image);
    } while (!(done));
}

CONFIG initial_configuration()
{
    CONFIG vehicle;
    double dgre;

    printf("Enter the robots current configuration (X Y DEGREES\n
           KAPPA): ");
    scanf("%lf%lf%lf%lf", &vehicle.x, &vehicle.y, &dgre,
           &vehicle.kappa);
}

```

```

    vehicle.theta = norm(dgre / RAD);
    return(vehicle);
}

```

```

CONFIG initial_reference()
{
    double    degree, kpa;
    int       i, num_of_paths;

    printf("How many reference paths do you desire in the robots
           motion: ");
    scanf("%d",&num_of_paths);
    for (i=0; i< num_of_paths; ++i) {
        printf("\n%s%d%s", "Enter the equation for number ",
              (i+1)," reference path (X Y THETA KAPPA)");
        scanf("%lf%lf%lf%lf", &path_array[i].x, &path_array[i].y,
              &degree, &kpa);
        path_array[i].theta = norm(degree / RAD);
        path_array[i].kappa = kpa;
    }
    path_array[num_of_paths].x = 0.0;
    path_array[num_of_paths].y = 0.0;
    path_array[num_of_paths].theta = 6.28;
    path_array[num_of_paths].kappa = 0.0;
    return(path_array[0]);
}

```

```

void initialize_parameters(aa, bb, cc)
double *aa, *bb, *cc;
{
    double kk;

    printf("What is the desired value of the distance
           constant: ");
    scanf("%lf", &DIST_CONSTANT);
    kk = 1.0 / DIST_CONSTANT;
    *aa = 3.0 * kk;
}

```

```

*bb = *aa * kk;
*cc = *bb * kk / 3.0;
}

```

```

void initialize_files(path)
CONFIG path;
{
    char    name[MAXSTRING], name1[MAXSTRING];

    printf("What is the name of the graph output file? ");
    scanf("%s", name);
    printf("What is the name of the info output file? ");
    scanf("%s", name1);
    info_file = fopen(name1, "w");
    out_path = fopen(name, "w");
    fprintf(out_path, "%s%.1f\n%s\n", "#cs ", 0.5, "#lg 2");
    fprintf(out_path, "%s%.2f%s\n",
        "#leg \"distance constant=", DIST_CONSTANT, "\"");
    fprintf(out_path, "%s%27s\n", "#leg \"(X,Y,Theta,Kappa)",
        "(Xr,Yr,Thetar,Kappar)\");    fprintf(out_path,
        "%s%.1f, %.1f, %.1f, %.1f%s5s%.1f, %.1f, %.1f,
        %.1f%s\n", "#leg \"(", vehicle.x, vehicle.y,
        vehicle.theta*RAD, vehicle.kappa,")", "(",
        path.x,path.y,path.theta * RAD, path.kappa,")\");
}

```

```

void print_to_file(image, path, dkappa_ds)
CONFIG image, path;
double dkappa_ds;
{

    printf( "%1f ", vehicle.x);
    printf( " %1f\n ", vehicle.y);
    fprintf(out_path, "%1f ", vehicle.x);
    fprintf(out_path, " %1f\n ", vehicle.y);
    fprintf(info_file, "%s%1f ", "x=", vehicle.x);
    fprintf(info_file, "%s%1f ", "y=", vehicle.y);
    fprintf(info_file, "%s%1f ", "theta=", vehicle.theta);
}

```

```

fprintf(info_file, "%s%lf\n ", "kappa=", vehicle.kappa);
fprintf(info_file, "%s%lf ", "itheta=", image.theta);
fprintf(info_file, "%s%.5f ", "image_x=", i_x0);
fprintf(info_file, "%s%.5f ", "image_y=", i_y0);
fprintf(info_file, "%s%lf\n ", "close_dist=", delta_d);
fprintf(info_file, "%s%lf ", "dkappa_ds=", dkappa_ds);
fprintf(info_file, "%s%lf ", "inter_x=", intersect.x);
fprintf(info_file, "%s%lf\n ", "inter_y=", intersect.y);
fprintf(info_file, "%s%lf ", "path.x=", path.x);
fprintf(info_file, "%s%lf ", "path.y=", path.y);
fprintf(info_file, "%s%lf ", "path.t=", path.theta);
fprintf(info_file, "%s%lf\n\n ", "path.k=", path.kappa);
}

```

```

void update_speed(speed, delta_dist)
double *speed, *delta_dist;
{
    *speed = VELOCITY;
    *delta_dist = DELTA_TIME * *speed;
}

```

```

double delta_distance(vehicle, path)
CONFIG vehicle, path;
{
    double distance;

    distance = (-(vehicle.x - path.x) * (path.kappa *
        (vehicle.x - path.x) + 2 * sin(path.theta)) -
        (vehicle.y - path.y) * (path.kappa *
        (vehicle.y - path.y) - 2 * cos(path.theta))) /
        (1 + sqrt(pow(path.kappa * (vehicle.x - path.x)
        + sin(path.theta), 2.0) + pow(path.kappa *
        (vehicle.y - path.y) - cos(path.theta), 2.0)));
    return(distance);
}

```

```

void update_image(image, vehicle, path)
CONFIG *image, vehicle, path;
{
    double radius, gamma, close_dist;
    POINT origin;

    if (path.kappa == 0.0) {
        close_dist = (((vehicle.y - path.y) * cos(path.theta)) -
            ((vehicle.x - path.x) * sin(path.theta)));
        (*image).x = vehicle.x + close_dist * sin(path.theta);
        (*image).y = vehicle.y - close_dist * cos(path.theta);
        (*image).theta = path.theta;
        (*image).kappa = path.kappa;
    }
    else {
        radius = (1.0 / path.kappa);
        origin.x0 = path.x - radius * (sin(path.theta));
        origin.y0 = path.y + radius * (cos(path.theta));
        gamma = atan2(vehicle.y - origin.y0,
            vehicle.x - origin.x0);
        (*image).x = origin.x0 + fabs(radius) * (cos(gamma));
        (*image).y = origin.y0 + fabs(radius) * (sin(gamma));
        (*image).theta = norm(gamma + (PI/2)*(path.kappa/
            fabs(path.kappa)));
        (*image).kappa = path.kappa;
    }
    i_x0 = (*image).x;
    i_y0 = (*image).y;
}

```

```

double kappa_dot(vehicle, image, delta_d, delta_dist,
    aa, bb, cc)
CONFIG vehicle, image;
double delta_dist, delta_d, aa, bb, cc;
{
    double delta_kappa, dkappal;

```

```

    dkappal = -aa * (vehicle.kappa - image.kappa)
              -bb * (norm(vehicle.theta - image.theta))
              -cc * delta_d;
    delta_kappa = dkappal * delta_dist;
    return(delta_kappa);
}

```

```

void update_configuration(vehicle, dkappa_ds, delta_dist)
CONFIG *vehicle;
double dkappa_ds, delta_dist;
{
    double delta_theta, delta_dist1, kappa;
    double epsilon = 0.00001;

    kappa = (*vehicle).kappa + dkappa_ds;
    delta_theta = delta_dist * kappa;
    delta_dist1 = delta_dist;
    if (fabs(delta_theta) <= epsilon) {
        delta_dist1 = delta_dist * (sin(delta_theta/2) /
                                     (delta_theta/2));
    }
    (*vehicle).x += (cos((*vehicle).theta + delta_theta / 2.0)
                    * delta_dist1);
    (*vehicle).y += (sin((*vehicle).theta + delta_theta / 2.0)
                    * delta_dist1);
    (*vehicle).theta = norm((*vehicle).theta + delta_theta);
    (*vehicle).kappa = kappa;
}

```

```

void time_to_transition(path, path_num, done, image)
CONFIG *path, image;
int *path_num, *done;
{
    double distance, turn_angle, TDist;

    if (path_array[*path_num+1].theta == 6.28){
        if (((fabs(delta_d) < 0.00001) && (fabs((*path).kappa -
            vehicle.kappa) <= 0.00001) && (fabs(norm(vehicle.theta -

```

```

    image.theta)) <= 0.00001)) || (ITERATIONS > 2000))
        *done = 1;
    else {
        *done = 0;
        ITERATIONS += 1;
    }
}
else {
    turn_angle = intersect.theta;
    distance = sqrt(pow(image.x - intersect.x, 2.0) +
        pow(image.y - intersect.y, 2.0));
    TDist = (2.4 * DIST_CONSTANT + 0.3) * (1/(1- pow(turn_angle
        / PI, 4.0)));
    if (distance < TDist ) {
        *path = path_array[*path_num + 1];
        *path_num += 1;
        intersect = intersection_point(*path_num);
        *done = 0;
        fprintf(out_path,"%s%.1f, %.1f, %.1f, %.1f%s%.1f, %.1f,
            %.1f, %.1f%s\n", "#leg \"(", vehicle.x, vehicle.y,
            vehicle.theta*RAD, vehicle.kappa, ")", "(",
            (*path).x, (*path).y, (*path).theta * RAD,
            (*path).kappa, ")\n");
    }
    else
        *done = 0;
}
}

```

```

CONFIG intersection_point(path_num)
int path_num;
{
    double dist_refs, ref_orient, beta, alpha, sigma, inter_d,
        distance, distancel, phi, intersect_orient;
    POINT imagel, origin;
    CONFIG inter;

    if (path_array[path_num+1].theta == 6.28) {
        inter.x=path_array[path_num].x +
            1000*cos(path_array[path_num].theta);
    }
}

```

```

inter.y=path_array[path_num].y +
    1000*sin(path_array[path_num].theta);
)
else if ((path_array[path_num].kappa == 0.0) &&
    (path_array[path_num+1].kappa == 0.0)){
    dist_refs=sqrt(pow(path_array[path_num+1].x -
        path_array[path_num].x, 2.0) +
        pow(path_array[path_num+1].y -
        path_array[path_num].y, 2.0));
    ref_orient = atan2(path_array[path_num+1].y -
        path_array[path_num].y,
        path_array[path_num+1].x -
        path_array[path_num].x);
    beta = path_array[path_num].theta - ref_orient;
    alpha = norm(PI - ref_orient) - path_array[path_num +
        1].theta;
    sigma = norm(PI - beta - alpha);
    inter_d = dist_refs * (sin(alpha)/sin(sigma));
    inter.x=path_array[path_num].x +
        inter_d*cos(path_array[path_num].theta);
    inter.y=path_array[path_num].y +
        inter_d*sin(path_array[path_num].theta);
    inter.theta = norm(path_array[path_num + 1].theta -
        path_array[path_num].theta);
}
else if(path_array[path_num].kappa == 0.0) {
    origin.x0 = path_array[path_num+1].x - (1/
        path_array[path_num+1].kappa) *
        (sin(path_array[path_num+1].theta));
    origin.y0 = path_array[path_num+1].y + (1/
        path_array[path_num+1].kappa) *
        (cos(path_array[path_num+1].theta));
    distance = (origin.y0 - path_array[path_num].y) *
        cos(path_array[path_num].theta) - (origin.x0 -
        path_array[path_num].x) *
        sin(path_array[path_num].theta);
    distancel =sqrt(pow(1/path_array[path_num+1].kappa,2.0)-
        pow(distance,2.0));
    imagel.x0 = origin.x0 + distance *
        sin(path_array[path_num].theta);
    imagel.y0 = origin.y0 - distance *
        cos(path_array[path_num].theta);
    inter.x= imagel.x0 - distancel

```

```

        *cos(path_array[path_num].theta);
inter.y= imag1.y0 - distancel
        *sin(path_array[path_num].theta);
phi = atan2(inter.y - origin.y0, inter.x - origin.x0);
intersect_orient = norm(phi + (PI/2) *
        (path_array[path_num + 1].kappa /
        fabs(path_array[path_num + 1].kappa)));
inter.theta = norm(intersect_orient -
        path_array[path_num].theta);
}
else {
    origin.x0 = path_array[path_num].x - (1/
        path_array[path_num].kappa) *
        (sin(path_array[path_num].theta));
    origin.y0 = path_array[path_num].y + (1/
        path_array[path_num].kappa) *
        (cos(path_array[path_num].theta));
    distance = (origin.y0 - path_array[path_num+1].y) *
        cos(path_array[path_num+1].theta) - (origin.x0 -
        path_array[path_num+1].x) *
        sin(path_array[path_num+1].theta);
    distancel = sqrt(pow(1/path_array[path_num].kappa,2.0) -
        pow(distance,2.0));
    imag1.x0 = origin.x0 + distance *
        sin(path_array[path_num+1].theta);
    imag1.y0 = origin.y0 - distance *
        cos(path_array[path_num+1].theta);
    inter.x= imag1.x0 + distancel
        *cos(path_array[path_num+1].theta);
    inter.y= imag1.y0 + distancel
        *sin(path_array[path_num+1].theta);
    phi = atan2(inter.y - origin.y0, inter.x - origin.x0);
    intersect_orient = norm(phi + (PI/2) *
        (path_array[path_num].kappa /
        fabs(path_array[path_num].kappa)));
    inter.theta = norm(- intersect_orient +
        path_array[path_num + 1].theta);
}
return(inter);
}

```

```

double norm(angle)
double angle;
{

    while ((angle > PI) || (angle <= -PI)) {
        if (angle > PI)
            angle -= DPI;
        else
            angle += DPI;
    }
    return(angle);
}

```

APPENDIX D

We propose a path tracking algorithm which is applicable to any autonomous vehicle. This algorithm would be operable in conjunction with the present motion control software system MML for Yamabico. An overview of the present locomotion functions is presented by Kanayama [Ref. 4], which second as Yamabico's user manual. This reference gives the reader a clear understanding of the MML software environment, and the available motion control commands. The modification which would be necessary to combine the two independent systems was explained in chapter VII. In this appendix we would like to explain the additions necessary to the library functions in the MML language by expanding the user's manual.

The changes which are necessary are concerned exclusively with the sequential locomotion functions, such as move and stop. To append the users manual we must simply add three locomotion functions to the system. These three commands shall allow the user to intersperse the functional features of the path tracking algorithm with the established point to point motion control scheme. The added commands will be labeled as follows; *path*, *ppath*, and *spath*. It is our hope, that by incorporating both systems together with as few as possible additional commands, we can maintain the entire MML environment. In addition, we feel this will greatly expand the robot's functionality and versatility without complicating the operating procedure of the robot. Let's take a in-depth look at each new sequential locomotion function.

PATH TRACKING LOCOMOTION FUNCTIONS:

MOVE PATH

SYNOPSIS: *path(c);*
 CONFIGURATION *c;

DESCRIPTION: This function moves the robot from the robot's current configuration $p_i = (x_i, y_i, \theta_i, \kappa_i)$ onto the directed reference path $= (x, y, \theta, \kappa)$. The present speed and acceleration, v_c and a_c , are used for the motion. The robot's motion should smoothly merge onto the reference path and continue to track along the path indefinitely or until another motion function is encountered. The only acceptable command following a `path()` is another path tracking locomotion function, or an immediate function. The locomotion function which follows this command must consist of a path which intersects the present desired path. If the two consecutive paths do not intersect, the robot stays on the current reference path.

ERROR: If the command following `path()` is not an acceptable option, the robot stops and an error code is returned.

SEE ALSO: `ppath()`, `spath()`, `speed()`, `acc()`, `movei()`.

MOVE PARTIAL PATH

SYNOPSIS: `ppath(c);`
`CONFIGURATION *c;`

DESCRIPTION: This function is a variation of the `path` function. It moves the robot from the robot's current configuration $p_i = (x_i, y_i, \theta_i, \kappa_i)$ onto the desired reference path. The present speed and acceleration, v_c and a_c , are used for the motion. The robot's should smoothly merge onto the desired reference path and continue to track along the path until its image reaches $c = (x, y, \theta, \kappa)$. At this point control goes to the next path function.

ERROR: If the destination point `c` is the same as the current nominal point, the robot stops and an error code is returned.

SEE ALSO: `path()`, `spath()`, `speed()`, `acc()`, `movei()`, `move()`, `stop()`.

STOP PATH

SYNOPSIS:

spath(c);
CONFIGURATION *c;

DESCRIPTION:

This function is a variation of the *path()* function. It moves the robot from the robot's current configuration $p_i = (x_i, y_i, \theta_i, \kappa_i)$ onto the desired reference path. The present speed and acceleration, v_c and a_c , are used for the motion. The robot should smoothly merge onto the desired reference path and continue to track along the path until its image stops at $c = (x, y, \theta, \kappa)$. At this point the robot switches to the STOP state. In this state the robot can complete any command of either system.

SEE ALSO:

path(), *ppath()*, *speed()*, *acc()*, *movei()*, *move()*, *stop()*.

SPECIFYING S_0

SYNOPSIS:

dist_const(x);
double x;

DESCRIPTION:

This function allows the user to adjust the value of S_0 for a particular maneuver. The command can only be used while the robot is in the STOP state.

ERROR:

If the user inputs a negative value for the distance constant, the robot stops and an error code is returned. If this command is given while the robot is not in the STOP state, the robot shall stop and an error code returned.

LIST OF REFERENCES

1. Kanayama, Y., and Hartman B.I., "Smooth Local Path Planning for Autonomous Vehicles, Part I: Symmetetricity," *Proceedings IEEE Journal of Robotics and Automation*, pp. 1265-1270, 1989.
2. Kanayama, Y., "Introduction to Two Dimensional Spatial Reasoning," paper presented at the Naval Postgraduate School, Monterey, CA., 24 March 1991.
3. Technical Report of Naval Postgraduate School, "Locomotion Functions in the Mobile Robot Language, MML," Kanayama, Y., and Onishi, M., pp.1 - 23, 1990.

BIBLIOGRAPHY

1. Akman, V., *Unobstructed Shortest Paths in Polyhedral Environments*, Springer-Verlag, 1987.
2. Brady, M., *Robot Motion: Planning and Control*, MIT Press, 1982.
3. Canny, J., *The Complexity of Robot Motion Planning*, MIT Press, 1988.
4. Gordpasture, R. P., *A Computer Simulation Study of an Expert System for Walking Machine Motion Planning*, National Technical Information Service, 1987.
5. Kanayama, Y., and Hartman B.I., "Smooth Local Path Planning for Autonomous Vehicles, Part II: Cubic Spirals," *Proceedings IEEE Journal of Robotics and Automation*, 1989.
6. Kanayama, Y., and Noguchi, T., "Locomotion Functions for a Mobile Robot Language," *Proceedings IEEE/RSJ International Workshop on Robot Programming Languages*, 1989.
7. Kanayama, Y., and Onishi, M., "Locomotion Functions in the Mobile Robot Language, MML," *Proceedings IEEE Journal of Robotics and Automation*, pp. 1110 -1115, 1991.
8. Kanayama, Y., and Yuta, S., "Vehicle Path Specification by a Sequence of Straight Lines," *IEEE Journal of Robotics and Automation*, vol. 4, no. 3, 1988.
9. Nelson, W., and Cox, I., "Local Path Control for an Autonomous Vehicle," *Proceedings IEEE Conference on Robotics and Automation*, 1988.

10. Richbourg, R. F., *Solving a Class of Spatial Reasoning Problems: Minimal-Cost Path Planning in the Cartesian Plane*, National Technical Information Service, 1987.
11. Sanders, D. W., *A feasibility Study in Path Planning Applications Using Optimization Techniques*, National Technical Information Service, 1987.
12. Schwartz, J. T., Sharir, M., and Hopcroft, J., *Planning Geometry, and Complexity of Robot Motion*, Ablex Pub, 1987.
13. Smith, W., *Local Path Planning using Optimal Control Techniques*, National Technical Information Service, 1988.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Dr. Yutaka Kanayama, Code CS/Ka Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
LT Richard J. Abresch 1640 Amberlea Dr. Dunedin, FL 33528	2