IDA PAPER P-2940

THE APPLICATION OF FUZZY LOGIC TO SAFOR IN SIMNET

Irvin W. Kay
Bohdan Balko

February 1994

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE February 1994	3. REPORT TYPE AND DATES COVERED Final--June 1993–December 1993

4. TITLE AND SUBTITLE The Application of Fuzzy Logic to SAFOR in SIMNET	5. FUNDING NUMBERS IDA Central Research Project
6. AUTHOR(S) Irvin W. Kay Bohdan Balko	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Defense Analyses 1801 N. Beauregard St. Alexandria, VA 22311-1772	8. PERFORMING ORGANIZATION REPORT NUMBER IDA Paper P-2940
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 180 words)*

This paper describes the history of an experiment to demonstrate the potential utility of fuzzy logic applied to SAFOR algorithms by using fuzzy logic to modify an existing vehicle collision avoidance algorithm. Appended to the paper are instructions on how to create SAFOR exercises on a Silicon Graphics terminal in the IDA Simulation Center network. This report also contains listings of source codes used in various phases of the work.

14. SUBJECT TERMS fuzzy logic, fuzzy set theory, SAFOR algorithms, semiautomated forces, collision avoidance	15. NUMBER OF PAGES 130
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

IDA PAPER P-2940

THE APPLICATION OF FUZZY LOGIC TO SAFOR IN SIMNET

Irvin W. Kay
Bohdan Balko

February 1994

INSTITUTE FOR DEFENSE ANALYSES
IDA Central Research Program

PREFACE

This document has been prepared under IDA's Central Research Program.

We would like to thank the following IDA people for their cooperation and help: Bob Clover, Keith Green, Barron Gibson, Chris Turrell, and Grant Shackelford of the Simulation Center; Peter Brooks of SFRD; Kevin Brown of STD; and Ken Ratkiewicz of PSO.

The implementation of the algorithm in the SAFOR system could not have been done without the cooperation of Bob Clover and Keith Green and the help of Kevin Brown in the early stages of the work. Keith Green taught us how to use the SAFOR system and create exercises and Bob Clover helped us identify certain interesting features in existing logged military exercises. For the stealth room operation during the creation of the video we owe thanks to Barron Gibson, Chris Turrell, and Grant Shackelford.

Peter Brooks deserves our thanks for introducing us to SAFOR and providing helpful information about its use and operation.

Ken Ratkiewicz encouraged us to make a movie of the stealth action showing the improvements we achieved and then provided professional assistance in the creation of the movie. For this we are very grateful.

We would also like to thank Dr. Hillman Dickinson for helpful suggestions to improve battlefield realism.

FOREWORD

This document is essentially a notebook, recording the work done and information acquired during a two-phase experiment to test the potential application of fuzzy logic to the Semi-Automated Forces (SAFOR) system. SAFOR is used in Simulation Network (SIMNET) battle exercises to reduce the number of real human participants by replacing some of them with virtual participants generated by computer software. A major objective of SAFOR is to replicate human behavior with enough fidelity that a human taking part in an exercise will be unable to distinguish between simulated objects controlled by computer and those controlled by other humans. The idea underlying the use of fuzzy logic is that algorithms based on it are more likely to resemble the kind of rough-estimate, semi-qualitative, mental process that humans apply to quick decision making than the mathematically precise algorithms that are more typical of computer programs.

The question addressed in this experiment is whether fuzzy logic can be a general tool for improving an algorithm affecting the action of one or more SAFOR entities by making the action resemble human behavior more than it would have without the improvement. Therefore, in the context of the experiment it is important to avoid changing such an algorithm in any way other than by substituting fuzzy rules for precise calculations that would normally govern a particular action. *An important objective has been to* demonstrate that this can be achieved in a straightforward way by means of what is now a standard procedure in the design of fuzzy rule-based control systems.

ABSTRACT

This paper describes the history of an experiment to demonstrate the potential utility of fuzzy logic applied to SAFOR algorithms by using fuzzy logic to modify an existing vehicle collision avoidance algorithm. Appended to the paper are instructions on how to create SAFOR exercises on a Silicon Graphics terminal in the IDA Simulation Center network. This report also contains listings of source codes used in various phases of the work.

CONTENTS

FIGURES

TABLES

I. INTRODUCTION

A. BACKGROUND

1. SAFOR

Reference 1 describes SIMNET as a joint ARPA and U.S. Army research project with the goal of developing the technology to build an extended network of interactive combat simulators. SAFOR is the term used for the SIMNET Semi-Automated Forces system, the purpose of which, as stated in Ref. 1, is to allow "a few individuals to direct a large number of ground and air vehicles that operate as a unit in the simulated world."

Most of the individual SAFOR vehicle behavior, e.g., avoiding obstacles or maneuvering to stay in a prescribed formation, is under the control of software algorithms rather than humans. On the other hand, it is highly desirable that the human participants in a simulated combat exercise not be able to distinguish between simulated vehicles controlled by other humans and those that are computer driven.

In various recorded SIMNET exercises deviations from realistic vehicle behavior range from the subtle, recognizable only by experienced observers, to the gross or even bizarre. Examples of the first type are: unnecessarily leaving a road while traveling along it, not moving realistically over certain terrain profiles, not changing speed when climbing a hill, not slowing down when encountering another vehicle. Examples of the second are: colliding with other vehicles, driving into a lake, going through a wall. Bizarre behavior occurring in early exercises included several tanks coalescing to form an unrecognizable mass and tanks milling about randomly before lining up in a formation.[1] Recent examples are two or more vehicles performing a kind of dance and single vehicles aimlessly turning.

---

[1] These "monster vehicles" are really not due to a SAFOR deficiency, but the results of initial attempts to de-aggregate a platoon from BBS (an algorithmic war game that operates at the unit level—platoon being the lowest level) into its individual component entities (4 tanks) in SIMNET. Sometimes the de-aggregation algorithm fails to work properly.

2. Fuzzy Logic

In 1965 L. Zadeh, a professor of electrical engineering at the University of California whose special interest was in the design of control systems, introduced the concept of fuzzy sets (Ref. 2) as a generalization of classical set theory to deal with linguistically imprecise notions in a technologically sound way. His point was that to perform satisfactorily, control systems need not be optimal or even based on complete knowledge of the pertinent physical laws of motion. In fact, he expected controls designed using fuzzy sets and associated rules of fuzzy logic to mimic human behavior when applied to ordinary human activities such as parallel parking a car.

With the advent of the microcomputer chip, industrial applications of Zadeh's approach became feasible and, spearheaded by Japanese manufacturers, have multiplied at an ever increasing rate over the last decade. It is now common knowledge that the fuzzy logic methodology in conjunction with microcomputer chips has resulted in many consumer-oriented devices that exercise human-like judgment, such as a self-focusing camcorder that also selects the focal region of interest. Equally well known are more elaborate applications, such as a completely automated cement kiln or subway train, in which a fuzzy controller replaces a human operator.

The success of fuzzy logic as a technique for imitating human behavior in the design of control systems suggests that the same approach may accomplish a similar end if applied to dynamic computer generated simulations like those created by the SAFOR system. The objective of the work reported in this document has been to demonstrate the potential utility of applying fuzzy logic to SAFOR to make its computer-controlled entities behave more as if they were controlled by humans. To keep the demonstration as simple as possible the work has concentrated on a single activity—avoiding collisions between vehicles.

B. PROJECT HISTORY

1. Perspective

The project has had two distinct phases. The early phase concentrated on finding the relevant SAFOR source code. Although we had no documentation describing the actual SAFOR algorithms, and therefore no way of knowing what they were, during this period we also considered possible collision avoidance algorithms. The later phase involved experimenting with SAFOR by creating simple exercises in which vehicle collisions occurred and then replacing the existing avoidance algorithms by fuzzy versions that avoided collisions in identical scenarios.

2. Phase One

While still ignorant of the SAFOR collision avoidance algorithms we attempted to develop some of our own. We started with a three-dimensional concept to leave open the possibility of including aircraft, or at least helicopter, avoidance maneuvering. The vehicle and all obstacles were represented by spheres.

At first we treated ground vehicles as spheres resting on the ground, which was assumed to be a plane surface. Later we transformed the algorithms into two-dimensional versions that use cylindrical instead of spherical entities for representing the vehicle and obstacles.

Chapter II provides a description of the nonfuzzy (also called crisp) algorithms along with a detailed analysis deriving the necessary supporting mathematical formulae. Deleting the z coordinate, which represents height above the ground, accomplishes the transformation of the formulae from the spherical to the cylindrical geometry. Chapter III provides a brief summary of the fuzzy logic approach to algorithm design in general and its application to obstacle avoidance by a vehicle in particular.

The approaches to collision avoidance in Chapters II and III also differ in another respect: the crisp algorithms take into account all obstacles that the vehicle can encounter on its way to its destination, while the fuzzy ones consider at most two neighboring obstacles at a time. The objective in the first case was to devise a crisp algorithm to define a path from the initial position of the vehicle to its destination such that it would not collide with any obstacle along the way. The objective in the second case was to design a fuzzy algorithm that would behave more naturally, avoiding obstacles as they come into view without necessarily knowing their locations in advance. In both cases the algorithms chosen were the simplest possible that appeared to meet the stated conditions.

Both the crisp and the fuzzy algorithms were implemented in PC GW Basic programs that display the spherical or cylindrical vehicle avoiding similarly shaped obstacles on its way as it moves from the left end of a PC terminal screen to the right end. Appendix A contains figures illustrating the path of the vehicle as it avoids obstacles in five different configurations, using both the crisp and fuzzy algorithms, for both the spherical and the cylindrical geometries. Appendix D contains listings of all of the GW Basic programs, including one called REPLOT.BAS, which replots the vehicle path as shown in the figures. Also included in the appendix are files containing data that the programs can use to reproduce the exercises associated with the figures.

3. Phase 2

The second phase began when Kevin Brown, who was working on a different SIMNET programming problem but knew of our interest in SAFOR collision avoidance, located a file called avoid.c containing the source code for the C language function avoid_a_vehicle that implements avoidance of a collision between two vehicles. That file is located in a Silicon Graphics subdirectory along with a large number of other files containing source code and data required for SAFOR. Thus, a search of the other subdirectory files for relevant function names found in avoid.c made it possible to uncover enough source code to completely implement the SAFOR vehicle collision avoidance algorithms.

Chapter IV identifies the location and discusses the nature of the source code for avoiding vehicle collisions in general that the search uncovered. It also considers in detail the calculations made to implement the function avoid_a_vehicle, which pertains specifically to the avoidance of collisions between two vehicles.

At this point it was clear that our preliminary work on avoidance algorithms described in Chapters II and III had concentrated on an essentially different problem than the SAFOR algorithms address. We had been concerned with avoiding multiple obstacles, whereas the comparable SAFOR algorithms could only deal with a vehicle against a single other vehicle in the role of an obstacle. In fact, experimenting with a scenario involving two stationary tanks near each other acting as obstacles and a third attempting to avoid them, demonstrated that the moving tank may avoid one of the stationary ones but collide with the other.

The calculations of Chapter IV indicate that the approach of the SAFOR function avoid_a_vehicle is actually closer to that of our fuzzy than our crisp algorithm because it deals with just one obstacle at a time.[2] This made the insertion of a fuzzy version into SAFOR much simpler than it might otherwise have been. Chapter V describes in detail the new algorithm and how it fits into avoid_a_vehicle. Figures 21 and 22 in Appendix A illustrate the difference between the paths that a cylindrical vehicle takes, controlled first by

[2] In fact, it does not seem possible to use our crisp algorithm when the obstacles, themselves, are moving vehicles.

the regular SAFOR algorithm and then by the fuzzy algorithm, when avoiding two cylindrical objects.

As remarked in Chapter V, when a vehicle is turning into the path of another the regular SAFOR algorithm can fail to prevent a collision, while the fuzzy version averts it when applied to exactly the same scenario. Also discussed in Chapter V is a discovery made after experimenting with various exercises using either the crisp or the fuzzy version: causing vehicles to slow down when starting an avoidance maneuver greatly enhances the avoidance process.

Chapter VI describes a video tape recording of SAFOR anomalies observed in several exercises displayed on the SIMNET Stealth screen. The tape also shows vehicle collision scenes with the crisp SAFOR avoidance algorithm in control, followed by their counterparts obtained by recreating the original scenes with the fuzzy algorithm in control.

To create, save, and replay SAFOR exercises it was necessary to use the SIMNET Silicon Graphics terminals. Although user's manuals for some versions of SAFOR (cf. Ref. 3) have been written, at the time we were unaware that any existed. Fortunately, the SIMNET staff were very helpful, so that learning the procedures by word of mouth was not as difficult as it might seem. Nevertheless, we hope that Appendix B, which outlines the basic operational steps required to carry out the experiments discussed in this report, can still have some value as a primitive guide for others who may be involved in similar programs until a more polished and comprehensive manual becomes available.

II. CRISP OBSTACLE AVOIDANCE

A. PRELIMINARY ANALYSIS

Assume the vehicle, which is a sphere of radius R_V, is located initially at the position r_V and that its final destination is at r_D. Assume that *N* obstacles, which are also spheres of radius R_n, are located at the positions r_n. Assume, also, that the vehicle intends to move along the straight path connecting r_V and r_D. In dealing with obstacle avoidance it is convenient to regard all vectors as relative to the destination point r_D, working with $\Delta r = r - r_D$ instead of r. Figure 1 illustrates an example of the configuration with a single obstacle and also shows the radius vector r_{Cn} to the point on the vehicle path closest to the obstacle.

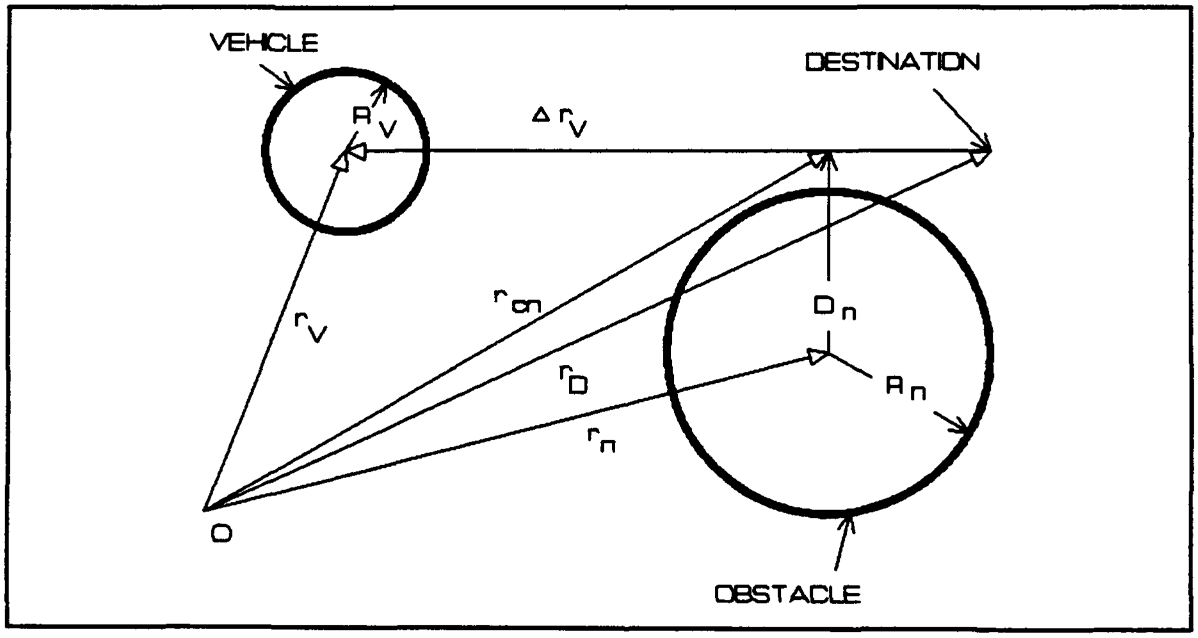

Figure 1. Vehicle Obstacle Collision Diagram

Points r on the path must satisfy the equation

$$\Delta r_V \times \Delta r = 0 \quad . \tag{1}$$

From (1) it follows that

$$\Delta \mathbf{r}_V \times (\Delta \mathbf{r}_V \times \Delta \mathbf{r}) = (\Delta \mathbf{r}_V \cdot \Delta \mathbf{r})\Delta \mathbf{r}_V - |\Delta \mathbf{r}_V|^2 \Delta \mathbf{r} = 0 \quad ,$$

so that

$$\Delta \mathbf{r} = \frac{(\Delta \mathbf{r}_V \cdot \Delta \mathbf{r})}{|\Delta \mathbf{r}_V|^2} \Delta \mathbf{r}_V \quad . \tag{2}$$

If $\mathbf{r}_{Cn}$ is the point on the path closest to the obstacle located at $\mathbf{r}_n$ then $\mathbf{r}_n$ must satisfy

$$\Delta \mathbf{r}_V \cdot (\Delta \mathbf{r}_{Cn} - \Delta \mathbf{r}_n) = \Delta \mathbf{r}_V \cdot (\mathbf{r}_{Cn} - \mathbf{r}_n) = 0 \quad ,$$

so that

$$\Delta \mathbf{r}_V \cdot \Delta \mathbf{r}_{Cn} = \Delta \mathbf{r}_V \cdot \Delta \mathbf{r}_n \quad . \tag{3}$$

But because of (2)

$$\Delta \mathbf{r}_{Cn} = \frac{\Delta \mathbf{r}_V \cdot \Delta \mathbf{r}_{Cn}}{|\Delta \mathbf{r}_V|^2} \Delta \mathbf{r}_V \quad . \tag{4}$$

It then follows from (3) and (4) that

$$\Delta \mathbf{r}_{Cn} = \frac{\Delta \mathbf{r}_V \cdot \Delta \mathbf{r}_n}{|\Delta \mathbf{r}_V|^2} \Delta \mathbf{r}_V \quad . \tag{5}$$

It follows from (5) that

$$|\Delta \mathbf{r}_{Cn}|^2 = \frac{(\Delta \mathbf{r}_V \cdot \Delta \mathbf{r}_n)^2}{|\Delta \mathbf{r}_V|^2} \quad . \tag{6}$$

The distance $D_n$between the point on the path $\mathbf{r}_{Cn}$ closest to the object center $\mathbf{r}_n$ and the object center is given by

$$D_n^2 = |\mathbf{r}_n - \mathbf{r}_{Cn}|^2 = |\Delta \mathbf{r}_n - \Delta \mathbf{r}_{Cn}|^2 = |\Delta \mathbf{r}_n|^2 - 2\Delta \mathbf{r}_n \cdot \Delta \mathbf{r}_{Cn} + |\Delta \mathbf{r}_{Cn}|^2 \quad .$$

Because of (5) and (6) this becomes

$$D_n^2 = |\Delta \mathbf{r}_n|^2 - 2\frac{(\Delta \mathbf{r}_n \cdot \Delta \mathbf{r}_V)^2}{|\Delta \mathbf{r}_V|^2} + \frac{(\Delta \mathbf{r}_n \cdot \Delta \mathbf{r}_V)^2}{|\Delta \mathbf{r}_V|^2} = |\Delta \mathbf{r}_n|^2 - \frac{(\Delta \mathbf{r}_n \cdot \Delta \mathbf{r}_V)^2}{|\Delta \mathbf{r}_V|^2} \quad . \tag{7}$$

The condition that the vehicle moving along the path will collide with obstacle n is

$$D_n^2 \le (R_V + R_n)^2 \quad . \tag{8}$$

The distance D_{Cn} between $\mathbf{r}_{Cn}$ and the position of the vehicle center $\mathbf{r}_{Vn}$ on the path when the collision occurs is given by

$$D_{C_n} = \sqrt{(R_V + R_n)^2 - D_n^2} \quad . \tag{9}$$

The collision occurs when the vehicle position on the path is given by

$$\Delta r_{V_n} = \Delta r_{C_n} + D_{C_n} \frac{\Delta r_V}{|\Delta r_V|} \quad , \tag{10}$$

where Δr_{C_n} is given by (5) and D_{C_n} is given by (9).

In general a minimum approach sphere, with a radius $R = R_V + R_n$, is centered at r_n. When the vehicle comes in contact with the obstacle at r_n its center lies on the minimum approach sphere at the point r_{V_n}, obtained by adding r_D to Δr_{V_n} given by (10). To avoid being impeded by the obstacle the vehicle can move around it, in either of two directions, until it reaches the point where it touches the cone that has its vertex at r_D and is tangent to the minimum approach sphere. The vehicle center will then lie on a conical generator tangent to that sphere and can move along the generator toward the conical apex, which coincides with the destination at r_D. Figure 2 illustrates an example of such a configuration, in which an obstacle directly blocks the vehicle's path to the destination.

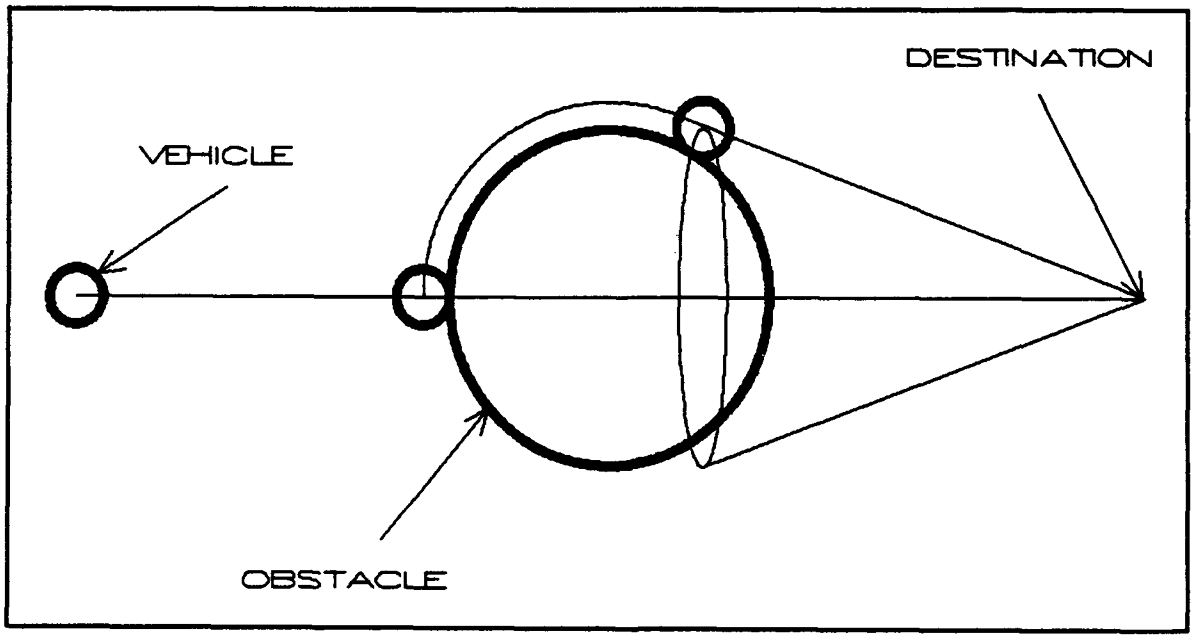

Figure 2. Vehicle Going Around Obstacle

The cone is tangent to the sphere at a circle lying in a plane P that intersects the conical axis at the point r_P, given by

$$\Delta r_P = \Delta r_n - \frac{R^2}{|\Delta r_n|^2}\Delta r_n = \frac{|\Delta r_n|^2 - R^2}{|\Delta r_n|^2}\Delta r_n \quad . \tag{11}$$

The equation

$$\Delta r \cdot \Delta r_P = |\Delta r_P|^2 \quad ,$$

which, because of (11) takes the form

$$\Delta r \cdot \Delta r_P = \frac{\left(|\Delta r_n|^2 - R^2\right)^2}{|\Delta r_n|^2} \tag{12}$$

determines the plane P, whose intersection with the sphere, determined by

$$|\Delta r - \Delta r_n|^2 = R^2 \quad , \tag{13}$$

determines the tangent circle. Substitution of (11) into (12) leads to

$$\Delta r_n \cdot \Delta r = |\Delta r_n|^2 - R^2 \quad . \tag{14}$$

The combination of (13) and (14) leads to

$$|\Delta r|^2 = |\Delta r_n|^2 - R^2 \quad , \tag{15}$$

which points on the circle must also satisfy.

Equations (14) and (15), which together determine the circle where the cone and sphere are tangent, are equivalent to

$$\Delta r_n \cdot \Delta r = L^2 \quad ,$$
$$|\Delta r|^2 = L^2 \quad , \tag{16}$$

where L is the distance from a point on the tangent circle to the destination at the conical apex. For the case in which the vehicle and obstacle are both touching the plane $z = 0$, i.e., are on the ground, the pair of equations (16) determine two points at the intersection of the tangent circle with the plane $z = z_D = z_V$. This is because the position of the vehicle (or, for that matter, of the obstacle) refers to the center of the sphere representing it. Then $\Delta z = 0$.

If for the vector Δr_n the component $\Delta x_n = 0$, the Δx and Δy components of the solution Δr of (16) are given by

$$\Delta y = \frac{L^2}{\Delta y_n} \quad ,$$

$$\Delta x = \pm\sqrt{L^2 - \Delta y^2} \quad , \qquad (17a)$$

and if for the vector Δr_n the component $\Delta y_n = 0$ the Δx and Δy components of the solution Δr can be obtained by interchanging Δx and Δy as well as Δx_n and Δy_n in (17a). Otherwise, those components are given by

$$\Delta x = \frac{L^2 \Delta x_n \pm \Delta y_n \sqrt{L^2\left(\Delta x_n^2 + \Delta y_n^2\right) - L^4}}{\Delta x_n^2 + \Delta y_n^2} \quad ,$$

$$\Delta y = \frac{L^2 - \Delta x_n \Delta x}{\Delta y_n} \qquad (17b)$$

Since two points satisfy (16) the vehicle can choose between two possible new paths. A simple, but not unreasonable, choice is the one requiring the vehicle to travel the least distance to its destination, without taking into account future collisions with other obstacles. For this purpose it is only necessary to select the point determined by the vector Δr for which the distance δ_{VD} given by

$$\delta_{VD} = |\Delta r_n - \Delta r| + |\Delta r| \qquad (18)$$

has the smaller of the two possible values.

If a pair of (spherical) obstacles are too close together they can be combined into one large sphere enclosing them both. For deciding when they are too close, a safe (but perhaps too stringent)[3] criterion would be: when the shortest distance between the two obstacle surfaces is less than the vehicle diameter, i.e., if the condition

$$|r_n - r_m| < R_n + R_m + 2R_v \qquad (19)$$

is satisfied. If (19) is satisfied, an obstacle with the radius R_{mn}, given by

$$R_{mn} = \frac{1}{2}\left(|r_n - r_m| + R_m + R_n\right) \quad , \qquad (20a)$$

will take the place of the obstacles centered at r_m and r_n with radii R_m and R_n. The center of the replacement will be located by the vector

---

3 In fact, if the vehicle and obstacles are on the ground a straightforward geometrical calculation shows that, instead of (19), the least stringent condition is

$$|r_n - r_m|^2 < (R_m - R_n)^2 + 4R_v\left(\sqrt{R_m} + \sqrt{R_n}\right)^2 \quad .$$

$$r = r_m + (R_{mn} - R_m)\frac{(r_n - r_m)}{|r_n - r_m|} \quad ,$$

which is equivalent to

$$r = \frac{R_m - R_n + |r_n - r_m|}{2|r_n - r_m|} r_m + \frac{R_n - R_m + |r_n - r_m|}{2|r_n - r_m|} r_n \quad . \tag{20b}$$

B. A CRISP AVOIDANCE ALGORITHM FOR A VEHICLE AND OBSTACLES ON THE GROUND

The basic idea of this algorithm is always to move the vehicle in a straight line from its present position toward its final destination whenever possible. Using (8), the algorithm determines the nearest obstacle with which the vehicle will collide if it continues on that path. Then the next step is to find the collision point, using (10), and determine whether the vehicle can go around the obstacle on a circular path to a position, calculated using (17a) or (17b) along with (18), from which it can proceed in a straight line toward the destination. This will be possible if the distance, determined from (19), between the nearest obstacle and all others, or all others but one, is large enough; then if it is obstructed on one side the vehicle can go around the nearest obstacle on the other side.

If more than one other obstacle is close enough to the nearest one to block the vehicle, the algorithm assumes that the blockage occurs for either path around the nearest obstacle, which it replaces, along with one of those that are too close, by a sphere whose center is given by (20b) and whose radius is given by (20a). The sphere encloses, and is tangent to, both obstacles being replaced.

After two obstacles have been replaced by a single sphere in this way the algorithm starts over from the beginning with the new configuration of spheres replacing the old. This procedure continues until it is no longer necessary to replace two spheres, regarded as single obstacles, with one; i.e., the vehicle can avoid all spheres in the final configuration, and therefore all of the original obstacles, without colliding with any.

III. FUZZY OBSTACLE AVOIDANCE

A. PRELIMINARY DEFINITIONS

In fuzzy logic terminology it is customary to refer to a set as defined in classical logic as "crisp." Associated with any set in classical logic is its "characteristic function," defined over all elements in the universe of discourse (i.e., all elements that could possibly be members of the set) as having the value 1 for every element that is a member of the set and the value 0 for every one that is not. Associated with every "fuzzy" set is its "membership function," which for every element in the universe of discourse has a value in the closed interval from 0 to 1. The crisp set characteristic function is obviously a special case of the fuzzy set membership function.

Here fuzzy obstacle avoidance means using an algorithm consisting of some rules expressed in the *modens ponens* form "If A then B," where A and B define fuzzy sets. For example, A might be the phrase "the distance to the obstacle is large" and B the statement "the change in the direction of the vehicle motion is small." The phrases "large distance to the obstacle" and "small change in the direction of the vehicle motion" refer to fuzzy sets defined by specific membership functions.

The membership function of the set "large distance to the obstacle" might be a non-decreasing function that is 0 as the obstacle distance increases from 0 to 50 meters, increases linearly from 0 to 1 as the obstacle distance increases from 50 to 100 meters, and remains equal to 1 for all obstacle distances larger than 100 meters. Figure 3 illustrates a simple piecewise linear membership function for the fuzzy set "large distance to the obstacle."

Membership functions for "medium distance to the obstacle" and "small distance to the obstacle" can be defined in a similar way. Figure 4 illustrates one for "medium distance to the obstacle," and Figure 5 illustrates one for "small distance to the obstacle".

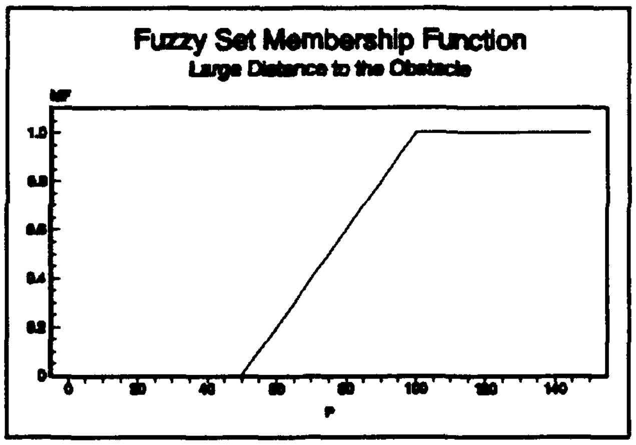

Figure 3. Fuzzy Set Membership Function--Large Distance to the Obstacle

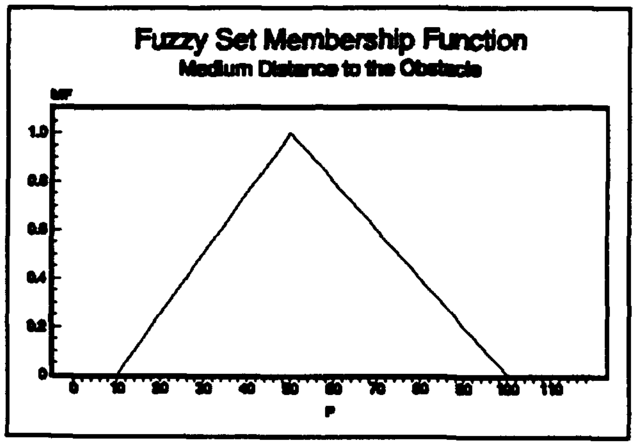

Figure 4. Fuzzy Set Membership Function--Medium Distance to the Obstacle

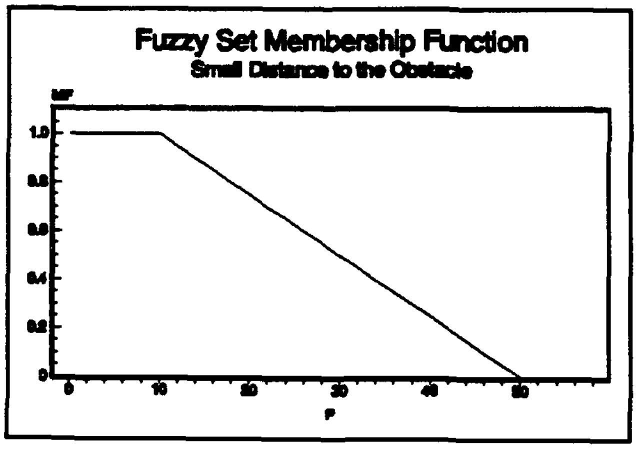

Figure 5. Fuzzy Set Membership Function--Small Distance to the Obstacle

Three parameters P_1, P_2, P_3 are sufficient to characterize any function of the types illustrated in Figures 3-5. The following rules define a generic membership function representing *small*, *medium*, or *large*.

If

$$P_3 < P_2 \quad ,$$

then the fuzzy set is *large* and its membership function is defined by:

$$\begin{cases} P \geq P_2, & MF(P) = 1; \\ P_2 \geq P > P_1, & MF(P) = \dfrac{P - P_1}{P_2 - P_1}; \\ P_1 \geq P, & MF(P) = 0 . \end{cases}$$

If

$$P_1 < P_2 < P_3 \quad ,$$

then the fuzzy set is *medium* and its membership function is defined by:

$$\begin{cases} P < P_1, & MF(P) = 0; \\ P_1 < P < P_2, & MF(P) = \dfrac{P - P_1}{P_2 - P_1}; \\ P_2 \leq P < P_3, & MF(P) = \dfrac{P_3 - P}{P_3 - P_2}; \\ P_3 < P, & MF(P) = 0 . \end{cases}$$

If

$$P_2 < P_1 \quad ,$$

then the fuzzy set is *small* and its membership function is defined by:

$$\begin{cases} P \geq P_2, & MF(P) = 1; \\ P_2 \leq P < P_3, & MF(P) = \dfrac{P_3 - P}{P_3 - P_2}; \\ P_3 \leq P, & MF(P) = 0 . \end{cases}$$

The membership function $\mu_{A \cap B}(P)$ of the intersection of two fuzzy sets A and B with membership functions $\mu_A(P)$ and $\mu_B(P)$ is given by the function, which for each value of P is equal to the smaller of the two membership functions at that value, i.e.,

$$\mu_{A \cap B}(P) = \min[\mu_A(P), \mu_B(P)] \quad . \tag{21}$$

For example, the intersection of "large distance to the obstacle" with "medium distance to the obstacle," the curves of whose membership functions appear separately in Figures 3 and 4 and together in Figure 6, has a membership function depicted by the curve in Figure 7.

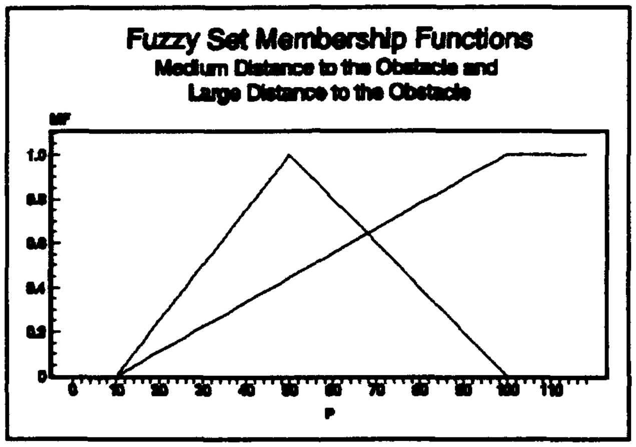

Figure 6. Fuzzy Set Membership Functions--Medium Distance to the Obstacle and Large Distance to the Obstacle

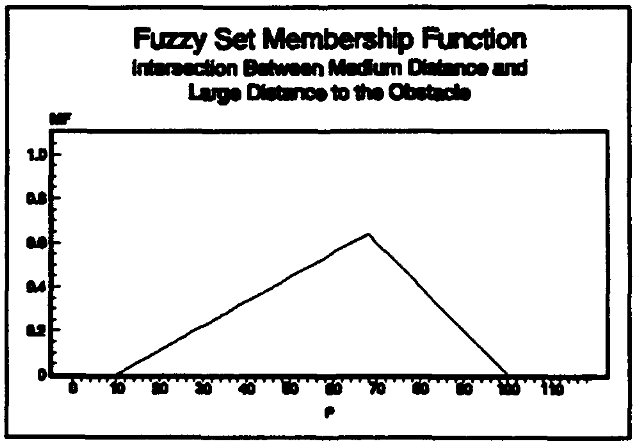

Figure 7. Fuzzy Set Membership Function--Intersection Between Medium Distance and Large Distance to the Obstacle

The membership function $\mu_{A \cup B}(P)$ for the union of two fuzzy sets with membership functions $\mu_A(P)$ and $\mu_B(P)$ is given by the function, which for each value of P is equal to the larger of the two membership functions at that value, i.e.,

$$\mu_{A \cup B}(P) = \max[\mu_A(P), \mu_B(P)] \quad . \tag{22}$$

The curve in Figure 8 depicts the membership function of the union of the two fuzzy sets whose membership function curves appear in Figure 6.

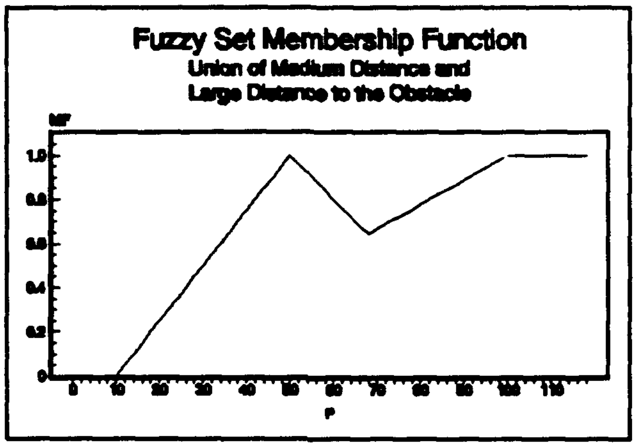

Figure 8. Fuzzy Set Membership Function--Union of Medium Distance and Large Distance to the Obstacle

A rule of the form "If A then B" is a fuzzy relation. The elements of the sets to which the statements A and B refer are, in general, from different universes of discourse. For example, suppose that A represents the phrase "the distance to the obstacle is large" and B the phrase "add a small velocity increment." Clearly, no element in the set of "large distances to the obstacle," to which the premise A refers, can be a candidate for inclusion in the set of "small velocity increments," or vice versa. However, the elements in the set of "large distances to the obstacle" and the elements of the set of "small velocity increments," although from different universes of discourse, exist simultaneously in the cartesian product of the two universes of discourse, which is a two-dimensional space, representing a combined universe of discourse wherein the elements of the sets referenced by A and B belong to mutually orthogonal cylindrical sets.[4] The complex set referenced by the relation "if A then B" is the intersection of the two cylindrical sets in the product space.

In the classical case the intersection can be characterized completely by a two-dimensional characteristic function: defined as 1 for every pair of elements (a,b) such that a is in the set referenced by A and b is in the set referenced by B, while for all other pairs the characteristic function is defined as 0. In the fuzzy case, of interest here, the two-dimensional membership function $\mu_{A\cap B}(P,Q)$ is defined by

$$\mu_{A\cap B}(P,Q) = \min[\mu_A(P), \mu_B(Q)] \tag{23}$$

exactly as in (21), except that the result is a function of P and Q instead of just P.

A standard fuzzy set computational procedure called composition provides a method for implementing a rule such as "If A then B" in an algorithm. Suppose that when A is "the distance to the obstacle is large" and B is "add a small velocity increment" an observed value of the distance to the obstacle is better described by "the distance to the obstacle is small." Then, given the membership function $\mu_O(P)$ of the observed quantity, composition determines the membership function $\mu_{action}(Q)$ of the quantity associated with the action implied by the rule, i.e., the increment that should be added to the velocity in place of the *small* increment that the rule prescribes when the distance to the obstacle is *large*. The calculation for this purpose is

$$\mu_{action}(Q) = \max_P\{\min[\mu_O(P), \mu_{A\cap B}(P,Q)]\} \quad , \tag{24}$$

where the minimum implied by "min" is the smaller of the two membership functions inside the square brackets, as in (21) and (23), but the maximum implied by "max" is the largest value of that result for all possible values of P.

If an algorithm consists of more than one "If A then B" rule, the action fuzzy set implied by the whole algorithm is the union of all of those defined by the separate rules. Therefore, the membership function $\mu_{alg}(P)$ associated with the algorithm can be calculated for each value of P by taking the largest corresponding value of all action membership functions associated with the separate rules and calculated using (24); i.e.,

$$\mu_{alg}(P) = \max_i[\mu_{action,i}(P)] \quad , \tag{25}$$

4 By definition, sets consisting of element pairs that include all elements from one of the original universes of discourse are cylindrical sets.

where $\mu_{action,i}(P)$ is the membership function associated with the action fuzzy set defined by rule i.

B. FUZZY OBSTACLE AVOIDANCE RULES

It is assumed here that in avoiding an obstacle a vehicle adds an increment to its own velocity vector, which is assumed to be constant except during the avoidance maneuver, and that the direction of the increment is orthogonal to the line of sight between the vehicle and the obstacle.[5] The magnitude of the increment is regarded as a fuzzy set depending on two observable fuzzy sets: the distance between the vehicle and the obstacle (measured from center to center) and the obstacle's radius.

Table 1 defines a set of nine rules for determining the velocity increment magnitude, given the observable sets. The phrase A in the "If A then B" form of a rule in this case refers to a complex set: the intersection of an "obstacle distance from the vehicle" with an "obstacle radius." Three possibilities exist for each type of set: *small*, *medium*, and *large*.

Table 1. Fuzzy Vehicle Velocity Avoidance Increment Dependence On Obstacle Radius and Distance from Vehicle

		Obstacle Radius		
		Small	Medium	Large
Distance	Small	Medium	Large	Large
From	Medium	Small	Medium	Large
Vehicle	Large	Small	Small	Medium

The phrase B in each rule applies to the fuzzy velocity increment size that appears at the intersection of the row and column in the table labeled in accordance with the fuzzy sizes of the two quantities whose intersection is specified by the phrase A in the rule. For example, the first rule given by Table 1, at the intersection of row 1 with column 1, is: "If the distance between the vehicle and the obstacle is small and the obstacle radius is small, then the velocity increment should be medium."

---

[5] An alternative, used in the current SAFOR avoidance algorithm when the obstacle is another vehicle, is to make the increment direction orthogonal to the vehicle's own velocity. At least for spherical obstacles, this does not work as well as choosing the direction to be orthogonal to the line of sight, as is done here.

Since the actual calculations will generally be numerical, although the membership functions are theoretically continuous functions of the independent variables, the actual values used for both the independent and dependent variables will be limited to a discrete, finite set. Also, in the end even a fuzzy algorithm must lead to a single number to be useful (i.e., the algorithm's action fuzzy set must be defuzzified). The most popular method for accomplishing this is to calculate the centroid given by

$$P_{alg} = \frac{\sum_i P_i \mu_{alg}(P_i)}{\sum_i \mu_{alg}(P_i)} \tag{26}$$

where P_{alg} is the value of the action quantity recommended by the algorithm, e.g., the magnitude of an increment to be applied to a vehicle's velocity in the case of a fuzzy obstacle avoidance algorithm.

C. A FUZZY OBSTACLE AVOIDANCE ALGORITHM FOR A VEHICLE AND OBSTACLES ON THE GROUND

This algorithm uses Table 1 to calculate the magnitude of a vector increment added to the vehicle path direction to avoid a collision with an obstacle. The direction of the increment is orthogonal to the line through the centers of the vehicle and the obstacle.

The fuzzy membership functions used for this purpose are those illustrated in Figures 3–5 and defined earlier, as well as similar membership functions for fuzzy obstacle radius sizes. Since, as in the case of the "distance from obstacle" sizes, they are all of the generic type, three parameters completely specify each of the "obstacle radius" size membership functions. The parameter sets are as follows:

For "small radius"	$P_1 = 35, P_2 = 5, P_3 = 20;$
For "medium radius"	$P_1 = 5, P_2 = 20, P_3 = 35;$
For "large radius"	$P_1 = 20, P_2 = 35, P_3 = 5.$

As mentioned earlier, in each rule of the form "if A then B" A is the intersection of the "distance from obstacle" and the "obstacle radius," which, although possibly measured in the same units, are different types of quantities. Therefore, the independent variables must be different in their respective membership functions, from which it follows that the membership function for A must be a function of two variables. In fact, according to (23) in rule ij it must be given by

$$\mu_{Aij}(P,Q) = \min\left[\mu_{di}(P), \mu_{rj}(Q)\right] \quad , \tag{27}$$

where $\mu_{di}(P)$ is the membership function for a "distance from obstacle" size i and $\mu_{rj}(Q)$ is the membership function for an "obstacle radius" size j. Similarly, (23) also determines the membership function for the intersection of the observed "distance from obstacle" and "obstacle radius" sizes, so that the result $\mu_O(P,Q)$ has the same form as that given by (27):

$$\mu_O(P,Q) = \min\left[\mu_{Od}(P), \mu_{Or}(Q)\right] \quad , \tag{28}$$

where $\mu_{Od}(P)$ is the membership function for the observed "distance from obstacle" size and $\mu_{Or}(Q)$ is the membership function for the observed "obstacle radius" size.

If the "vector increment" magnitude, which is B in the rule, has a membership function $\mu_{inc,ij}(R)$, (23) determines the rule relation membership function $\mu_r(P,Q,R)$, which is then given by

$$\mu_{rij}(P,Q,R) = \min\left[\mu_{Aij}(P,Q)\mu_{inc,ij}(R)\right] \quad , \tag{29}$$

where $\mu_{Aij}(P,Q)$ is given by (27). Then for the "vector increment" magnitude specified by the rule ij, (24) provides the membership function $\mu_{action,ij}(R)$, given by

$$\mu_{action,ij}(R) = \max_{P,Q}\left\{\min\left[\mu_O(P,Q), \mu_{rij}(P,Q,R)\right]\right\} \quad , \tag{30}$$

where the quantities on the right side of (30) are given by (28) and (29).

After calculating all of the individual rule membership functions $\mu_{action,ij}(R)$ given by (30), the next step is calculate the single velocity increment membership function $\mu_{inc}(R)$, which is the membership function for the union of the sets defined by all of the rules. According to (25), it is given by

$$\mu_{inc}(R) = \max_{ij}\left[\mu_{action,ij}(R)\right] \quad , \tag{31}$$

where $\mu_{ij,action}(R)$ is given by (30).

The final calculation based on (26) uses $\mu_{inc}(R)$ to estimate the vector increment magnitude. The result is given by

$$R_{inc} = \frac{\sum_n R_n \mu_{inc}(R_n)}{\sum_n \mu_{inc}(R_n)} \quad , \tag{32}$$

where $\mu_{inc}(R)$ is given by (31). In (32) each continuous variable, i.e., the membership function and its argument, is replaced by a discrete set of sample values. As a practical matter this must be done, in any case, before numerical results can be obtained.

When two or more obstacles are too close together for the vehicle to pass between them the fuzzy algorithm does not replace them with a single sphere as the crisp algorithm does. Instead, the fuzzy algorithm proceeds as follows.

If, when it moves in the normal direction, avoiding one obstacle, the vehicle will collide with another, the algorithm reverses its direction for one incremental path distance and then has it attempt to resume normal motion toward the destination. If the vehicle will move closer to the last obstacle encountered and the distance between the last obstacle and the nearest one ahead is too small to permit passage, the algorithm reverses the vehicle's direction, causes it to move the minimum incremental distance, and then continues to have it proceed toward the destination.

IV. SAFOR OBSTACLE AVOIDANCE

A. SAFOR AVOIDANCE IN GENERAL

Source code in the file called driver.c governs the avoidance of obstacles by vehicles in the SAFOR simulation. The functions used by code in driver.c for this purpose depend on the type of obstacle to be avoided. In particular, the function for avoiding vehicles, other than fixed wing aircraft but including dismounted infantry, is called avoid_vehicles, and its source code can be found in avoid.c.

The SAFOR function, defined by source code in driver.c, for avoiding extended obstacles, i.e., lakes, rivers, canopies, buildings, and tree lines, is avoid_objects. That function treats any of those obstacles as a collection of line segments. It avoids them by means of the repetitive use of the function called avoid_line, the source code for which is also in driver.c.

The work described in this document considers only the SAFOR avoidance of ground vehicles. The function that the code in driver.c uses for this purpose, avoid_vehicles, refers to another function, constant_velocity_avoid, which in turn refers to a third function, avoid_a_vehicle.[6] The source code for each of these functions is in the file avoid.c.[7]

B. SAFOR VEHICLE AVOIDANCE

The function avoid_a_vehicle implements the basic algorithm for collision avoidance. The steps involved are as follows.

1. Calculate the 2-dimensional vector r_{pos} representing the (instantaneous) relative position of the avoidee vehicle relative to the avoider by subtracting the

[6] The same functions also apply to the avoidance of helicopters. However, the function for avoiding fixed wing aircraft is more elaborate and quite different. It is called fixed_wing_avoid and its source code is in the file avoid.c.

[7] The file avoid.c also contains source code for another function called avoid_an_object, which closely resembles avoid_a_vehicle. However, a search of the appropriate SAFOR source code files (those in ./usr/safdevel/developer/src/host) uncovered no reference to that function by any other function in the present version (SAF 4.3.3) of SAFOR, indicating that the system no longer uses it.

2-dimensional position vector of the avoider from that of the avoidee and correcting for the difference between the tick times associated with the two vehicles.

2. Calculate the distance distance_at_cpa of the avoidee from the closest point of approach (cpa) of the avoider. This is done by first calculating the 2-dimensional relative velocity vector v_{rel}, which is the difference between the velocity vector of the avoider and that of the avoidee, and then calculating the cross product of that relative velocity and the relative position divided by the relative speed:

$$\mathrm{distance\_at\_cpa} = \frac{v_{rel} \times r_{pos}}{|v_{rel}|} \quad .$$

   Since the vectors are 2-dimensional their cross product has just one non-zero component, which is orthogonal to the plane of v_{rel} and r_{pos}, and can therefore be treated as a scalar. The sign of its value distance_at_cpa depends on which side of the avoidee the relative velocity is directed.

3. If the actual distance that is equal to the absolute value of distance_at_cpa is greater than a threshold called Passing_Dist, no avoidance maneuver is required. At the beginning of avoid.c the quantity Passing_Dist is defined as 5 if both vehicles are dismounted infantry, 10 if the vehicles are on the ground and at least one of them is not dismounted infantry, and 50 if the vehicles are helicopters.[8]

4. If avoidance is required, calculate the distance to the cpa, which is the absolute value of the quantity given by

$$\mathrm{dist\_to\_cpa} = \frac{v_{rel} \cdot r_{pos}}{|v_{rel}|} \quad .$$

   If dist_to_cpa is negative the avoider has passed the avoidee, so that avoidance is no longer necessary. If not, then calculate the time to the cpa, which is given by

$$\mathrm{time\_to\_cpa} - \frac{\mathrm{dist\_to\_cpa}}{|v_{rel}|} \quad .$$

   If time_to_cpa exceeds a threshold called GRND_LOOK_AHEAD, which is defined at the beginning of avoid.c as 8,[9] then quit because the obstacle is too far away to require an avoidance maneuver.

---

[8] Since the numbers refer to distances the units are presumed to be meters.

[9] In this case the threshold must be in units of time, which are assumed to be seconds.

5. If there is a destination point, determine whether the quantity called speed, which is the magnitude of the avoider velocity, is less than a threshold called SMALL, which is defined as 0.0005 at the beginning of a header file called util.h. If it is, then to avoid dividing by 0 skip the avoidance process; otherwise, continue by calculating the vector position r_{point} of the destination relative to the avoider by subtracting the avoider position vector from that of the destination. Then the time to the destination point is given by

$$\text{time\_to\_point} = \frac{|r_{point}|}{\text{speed}} \quad .$$

6. If

$$\text{time\_to\_point} < \text{time\_to\_cpa} \quad ,$$

then the avoider will arrive at the destination before the collision, the avoidance of which is therefore unnecessary. Otherwise, calculate a quantity called scale_factor, given by

$$\text{scale\_factor} = \text{sgn}(\text{dist\_at\_cpa}) \cdot \text{AVOID\_FACTOR} \cdot \frac{\text{Passing\_Dist} \cdot |\text{dist\_at\_cpa}|}{\max(\text{time\_to\_cpa}, \text{MINIMUM\_TIME})} \quad .$$

The quantities MINIMUM_TIME, which limits the possible size of scale_factor, and AVOID_FACTOR are both defined as 0.5 at the beginning of avoid.c.

7. Define a vector increment Δv, in terms of the unit vector v_o orthogonal to the relative velocity v_{rel} of the avoider, by

$$\Delta v = (\text{scale\_factor}) v_o$$

where, in terms of the components (v_1, v_2) of v_{rel}, the components (v_{o1}, v_{o2}) of v_o are given by

$$v_{o1} = \frac{v_2}{|v_{rel}|}, v_{o2} = -\frac{v_1}{|v_{rel}|} \quad .$$

Then change the avoider velocity by adding the increment Δv to it.

V. SAFOR VEHICLE AVOIDANCE USING A FUZZY ALGORITHM

A. THE CONCEPT

As observed in Section IV.B, the present, crisp, SAFOR algorithm for avoiding ground vehicle collisions accomplishes this by adding an orthogonal vector increment to the velocity of an avoider vehicle relative to that of the avoidee. The magnitude of the corrective increment is a scale factor based on the time to the cpa and a fixed passing distance (pd) threshold of 10 meters.

Experimenting with simple exercises involving two tanks or a single armored personnel carrier and a platoon of tanks has shown that collisions can occur in certain situations, in particular, when a vehicle is turning into the path of another vehicle. The reason in that case is probably a consequence of the algorithm's basic assumption that all velocities are constant. Thus, since fuzzy rules in algorithms are inherently nonlinear it is not surprising that in exactly the same scenarios replacing the scale factor by one derived from fuzzy rules averted the collisions.

The idea was to use rules like those defined in Table 1, with quantities PD and DISTANCE TO CPA replacing OBSTACLE RADIUS and DISTANCE FROM VEHICLE. After starting with some common sense guesses, to establish the rules a procedure, called "tuning" in the fuzzy logic literature, which involves testing them in some typical examples and making appropriate changes until they work satisfactorily, can be used. In the collision experiment starting with rules corresponding to those identified with Table 1 and then tuning them resulted in Table 2, which, in fact, differs only slightly from Table 1.

Table 2. Fuzzy Vehicle Velocity Avoidance Increment Dependence On Passing Distance and Distance to Closest Point of Approach

		PD		
		Small	Medium	Large
Distance	Small	Medium	Large	Large
To	Medium	Small	Large	Large
Cpa	Large	Small	Medium	Medium

B. THE IMPLEMENTATION

It was also necessary to apply the tuning process to the definition of the *small*, *medium*, and *large* membership functions associated with the pd, distance to cpa (dcpa) and scale factor (sclf), each of which was assumed to be of the generic type defined in Section III.A and therefore specified by three parameters. Actually, it is only necessary to assign three parameters to each of the quantity types: pd, dcpa, sclf; the order of the parameters determines whether the modifier is *small*, *medium*, or *large*.

The quantity pd in the existing crisp version of the SAFOR avoidance algorithm is assigned the possible values 5, 10, or 50, depending on whether the avoider and avoidee vehicles are both dismounted infantry, both are on the ground, and at least one is not dismounted infantry, or they are both helicopters. Therefore, a natural choice of parameters PD1, PD2, PD3 associated with the fuzzy pd membership function seemed to be: PD1 = 0, PD2 = 5, PD3 = 10, in the first case; PD1 = 5, PD2 = 10, PD3 = 50 in the second case; PD1 = 10, PD2 = 50, PD3 = 100 in the third case. For the parameters associated with the dcpa membership function, the choice for ground vehicles, which were the only type actually considered in the experiment, was: DCPA1 = 10, DCPA2 = 50, DCPA3 = 100. The first choice for sclf was: SCLF1 = 1, SCLF2 = 3, SCLF3 = 5. However, it was quickly observed that those values were too small, but that doubling each of them led to good results; i.e., the final choice was: SCLF1 = 2, SCLF2 = 6, SCLF3 = 10.

For each type of variable the associated parameter values increase in the same order as the subscripts on the symbols representing them. Thus, by virtue of the properties of the generic membership function determined by a set of three parameters, as defined in Section III.A, the membership functions determined by the triplets (PD3, PD1, PD2), (DCPA3, DCPA1, DCPA2), (SCLF3, SCLF1, SCLF2) are all associated with the size *small*. Those determined by (PD1, PD2, PD3), (DCPA1, DCPA2, DCPA3), (SCLF1, SCLF2, SCLF3) are associated with the size *medium*, and those determined by (PD1, PD3, PD2), (DCPA1, DCPA3, DCPA2), (SCLF1, SCLF3, SCLF2) with the size *large*.

C. CODE CHANGES

The SAFOR source code is written in the C programming language. A program written in C consists of various functions, the source codes of which may be collected together in arbitrarily defined groups and each group stored in a separate file. The corresponding file names all have the extension c, like the files driver.c and avoid.c mentioned in Section IV.A. [10]

As observed earlier, the source code of the function avoid_a_vehicle, which is the primary function that causes all vehicles but fixed wing aircraft to avoid each other, appears in the file avoid.c. The source code for the version of avoid_a_vehicle currently used in SAFOR is in Appendix C, from p. C-3 to p. C-6. The source code for the fuzzy version of avoid_a_vehicle is in Appendix C, from p. C-10 to p. C-13.

A comparison of the two versions of avoid_a_vehicle shows that the only effective difference between them is the way each calculates the quantity scale_factor. The fuzzy calculation replaces the non-fuzzy calculation

```
"scale_factor =
    SGN( dist_at_cpa ) * AVOID_FACTOR * ( Passing_Dist - ( abs( dist_at_cpa ) ) )
    / ( MAX(time_to_cpa, MINIMUM_TIME) );"
```

in the present SAFOR version by

```
"fsf = fuzzy_scale_factor(Passing_Dist, dist_to_cpa, speed);
    scale_factor =
    SGN( dist_at_cpa ) * AVOID_FACTOR * fsf;".
```

The code for the function fuzzy_scale_factor that calculates the quantity fsf is in Appendix C, from p. C-17 to p. C-24, followed by code for functions needed to support the calculation.

After viewing a video tape showing preliminary results of the collision avoidance experiment, Dr. Robert Roberts suggested that the vehicle behavior would be more realistic if the vehicles slowed down before engaging in their avoidance maneuvers. Adding this feature turned out to be quite effective, causing more realistic behavior in both the fuzzy and non-fuzzy versions of SAFOR. In fact, when the standard crisp SAFOR algorithm was modified to include slowing down it became impossible to make the vehicles collide at all.

[10] Another type of file, called a header, contains definitions and has the extension h. Both file types are compiled into object files with the extension o, which are then linked to create an executable file.

Fortunately, incorporating this feature in either avoidance algorithm is a simple matter. If the scale_factor is less than 1, it is assumed that the slow down is unnecessary. Otherwise, the algorithm divides both components of the avoider's current velocity vector by the scale factor before adding the increment required for the avoidance maneuver. The easiest way to see how this was done is to compare the original version of the function avoid_a_vehicle in Appendix C, pp. C-3 to C-6, with a version in Appendix C, pp. C-6 to C-8 that includes the slow down feature. The fuzzy version of avoid_a_vehicle in Appendix C, p. C-10 to p. C-13 can also be compared with the fuzzy version that includes the slow down feature, and which can be found in Appendix C, p. C-13 to p. C-17.

VI. VIDEO ACTION

In this chapter we describe a video that shows a dramatic result obtained from introducing a fuzzy algorithm into SAFOR. The video is divided into two parts. The first part deals with SAFOR and fuzzy logic and illustrates some unrealistic vehicle behavior in SAFOR exercises. The second part concentrates on vehicle collision avoidance problems.

The video starts with a scene from a Hunter-Liggett exercise with tanks moving in formation across a flat terrain and over hills. Enemy tanks are seen moving along a road. A battle ensues and the enemy tanks are destroyed one at a time. As they go up in flames the last tank in the platoon goes around the burning pile and continues on the course eventually to be destroyed also. Some of the tanks take questionable actions in the scenes shown, but nothing unusual or obviously unrealistic happens, with the exception of a tank running through a tree,[11] until the tanks approach a hilly area where one bumps into the one ahead of it. Soon they coalesce into a single tank. A short time later a third tank is seen approaching the two in what appears to be a "mating dance." The three combine into a single "monster vehicle" with three turrets independently moving, presenting a bizarre scene to the viewer. As explained in footnote 1 on page 1, these effects are not due to any SAFOR deficiency.

The second and third scenarios show tanks unsuccessfully avoiding a lake and a building. In the second scene one tank gets into the water, stops and rotates aimlessly in place. The following tank stops a few meters from the water and also rotates in place. In the third scene a tank is seen approaching a building, which it tries to avoid, but not, as becomes apparent very quickly, very successfully. About 25 percent of the tank gets into the building perhaps because the avoidance was not initiated early enough or the turn was not sharp enough. Either action could have avoided the problem. The tank gets out of the building and makes a reasonable attempt to get around its corner. It then travels parallel to the side of the building in a perfectly reasonable manner until suddenly, about half of the way to the other corner, it turns and, for no apparent reason, moves right through the wall.

[11] There is no provision made for avoiding isolated trees in SAFOR, clearly a giveaway in an exercise but not a difficult problem to fix.

These three scenes were included in the video to point out some obvious SAFOR problems not involving vehicle collision avoidance.

Next the video shows a vehicle collision course depicted in Figure 9. The arrows indicate the direction of the vehicle's movement. A platoon of tanks goes along the road from the top to the bottom, making two turns, first to the right and then to the left. An Armored Personnel Carrier (APC) will enter from the left and move along a road toward the intersection where the tanks make the left turn.

Although the scenario and troop movements were taken from an actual military exercise, the timing was rearranged to test the SAFOR collision avoidance algorithm. This was done by initializing the APC movement so that a collision would occur unless the algorithm prevented it. When the action in the video starts, we see the platoon following the road. Figure 10(a) shows the lead tank approaching the intersection from the right. The APC at this time is out of the picture to the left of the intersection. The leader makes a right turn, then a left, and then gets clobbered by the APC at the spot shown on the map in Figure 10(a). The SAFOR algorithm did not work. Figure 10(b) shows a close up of the collision.

The next action scene starts with the same initial conditions, but with the SAFOR code modified by replacing the original collision algorithm with a fuzzy version. This time the collision is avoided.

A closer analysis indicates that even though the collision was avoided, the action did not seem realistic. For example, the tanks did not slow down during the avoidance maneuver and the miss distance was too small. Figure 10(c) shows the APC just missing the tank. What the picture does not reveal is that both vehicles were moving rather fast at this time.[12]

We decided to supplement collision avoidance with slowdown. We inserted a slowdown operation (dividing the vehicle velocity by the avoidance scale factor whenever it is greater than 1) into the code and reran the action. This time the fuzzy algorithm did much better. Both the APC and the tank slowed down perceptively and the tank, after making the

[12] Some viewers commented that, realistically, the APC would stop and let the column of tanks pass. The collision avoidance algorithm, however, does not discriminate between different types of ground vehicles, although perhaps this could be changed.

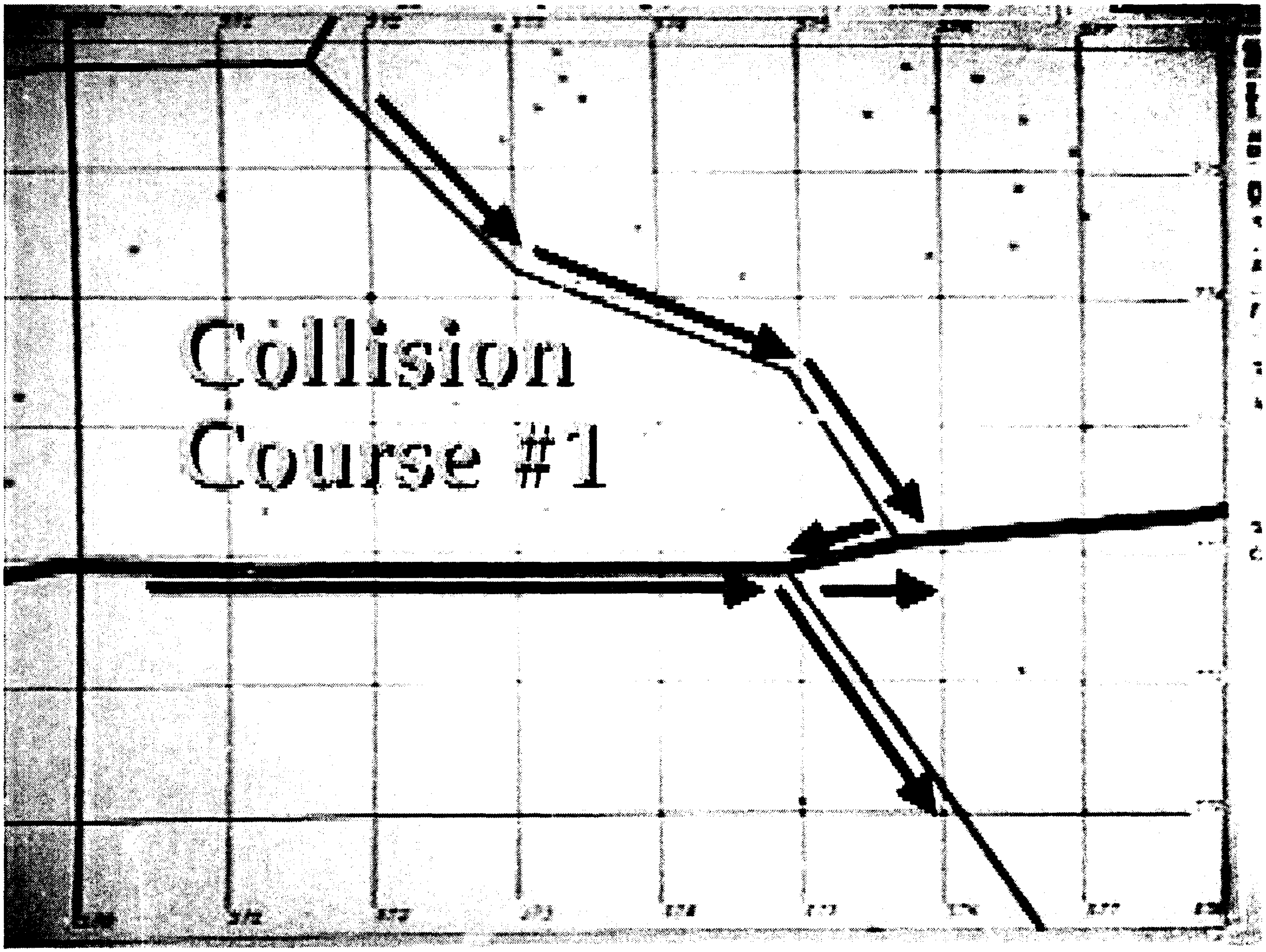

Figure 9. An Overhead of the Area of Interest With Movement of the Platoon and APC Indicated

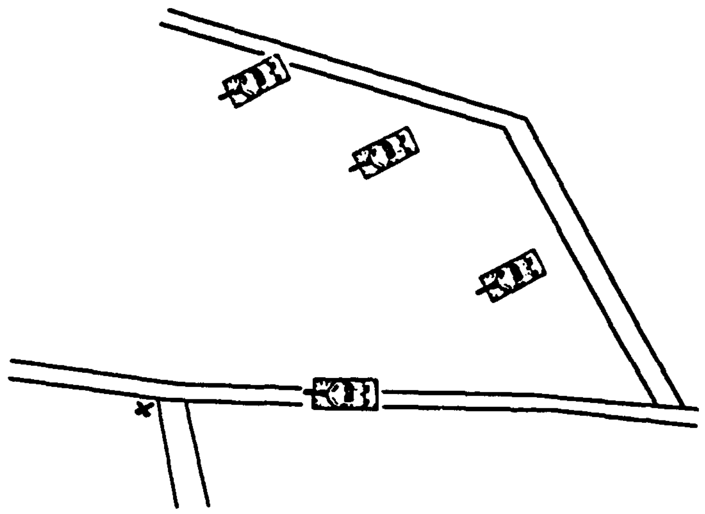

Figure 10(a). Overhead View of the Lead Tank Approaching the Intersection. This scenario is the initial condition for all the runs shown here.

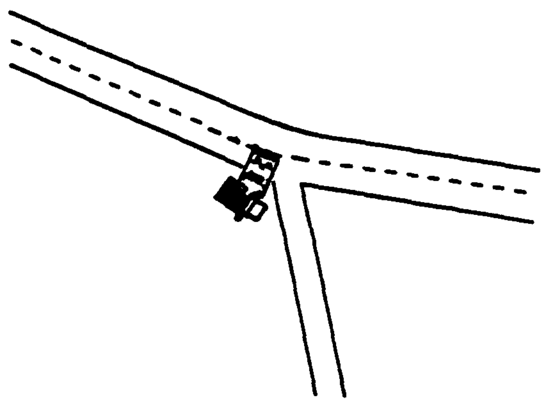

Figure 10(b). Collision Between Tank and APC When SAFOR Algorithm Is Used

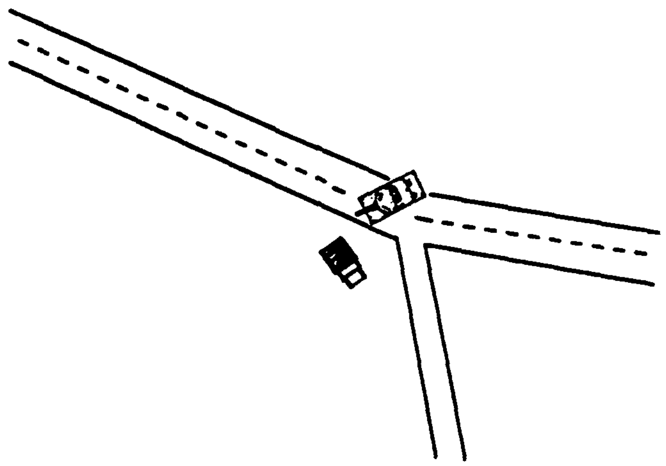

Figure 10(c). Avoidance of Collision When Fuzzy Algorithm Is Used

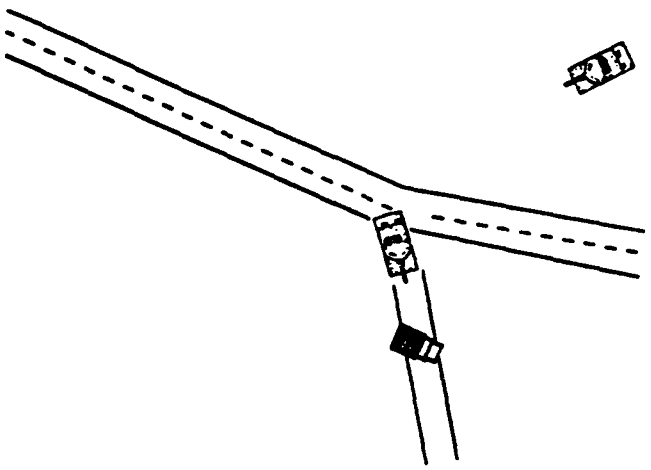

Figure 10(d). Avoidance of Collision When Fuzzy and Slowdown Algorithms Are Used left turn, actually stopped and let the APC pass.[13] Figure 10(d) shows an overhead view with the tank actually stopped and the APC avoiding it by about a tank's length.

We also tried the slowdown with the original crisp SAFOR algorithm. The collision was avoided but again the avoidance was just barely successful and the vehicles went past each other faster than reasonable.

[13] Whether a real tank would actually stop to let an APC pass is questionable. Further refinements may be necessary.

REFERENCES

1. A.R. Pope and R.L. Schaffer, "The SIMNET Network and Protocols," Rep. No. 7627, BBN Systems and Technologies, Cambridge, Mass., June 1991.

2. L.A. Zadeh, "Fuzzy Sets," *Informat. Control*, V. 8, 1965, pp. 338-353.

3. M.R. Saffi, "The OBG/SAF Interface, Version 4.2.0, User Guide," July 29, 1992, ©1992 BBN.

APPENDIX A

VEHICLE PATH WHILE AVOIDING OBSTACLES

APPENDIX A

VEHICLE PATH WHILE AVOIDING OBSTACLES

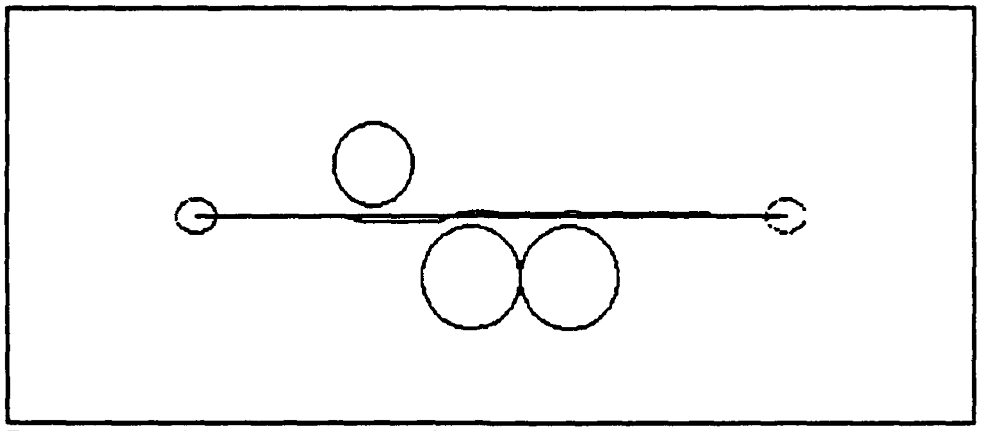
Figure 1 Crisp Algorithm Avoidance of Spherical Obstacles Set 1

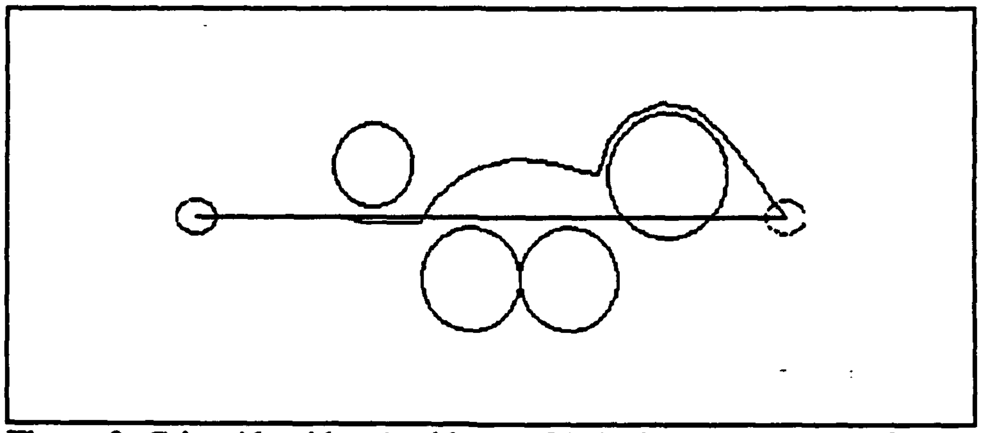
Figure 2 Crisp Algorithm Avoidance of Spherical Obstacles Set 2

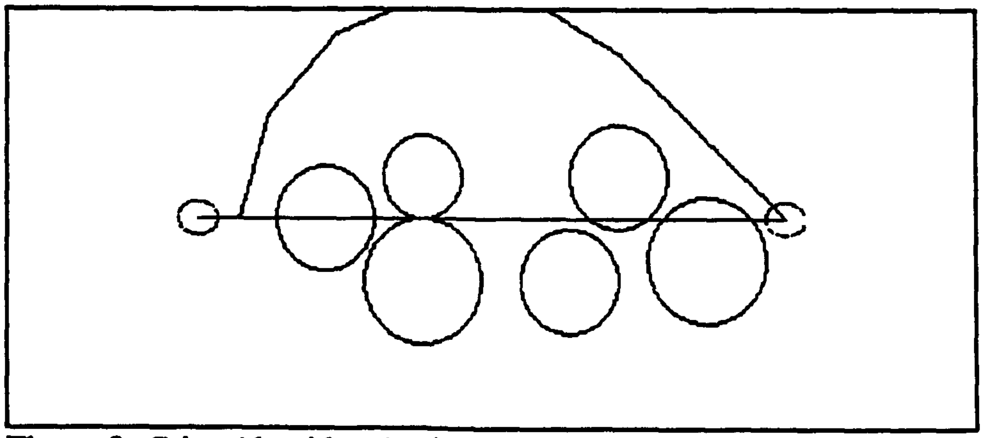
Figure 3 Crisp Algorithm Avoidance of Spherical Obstacles Set 3

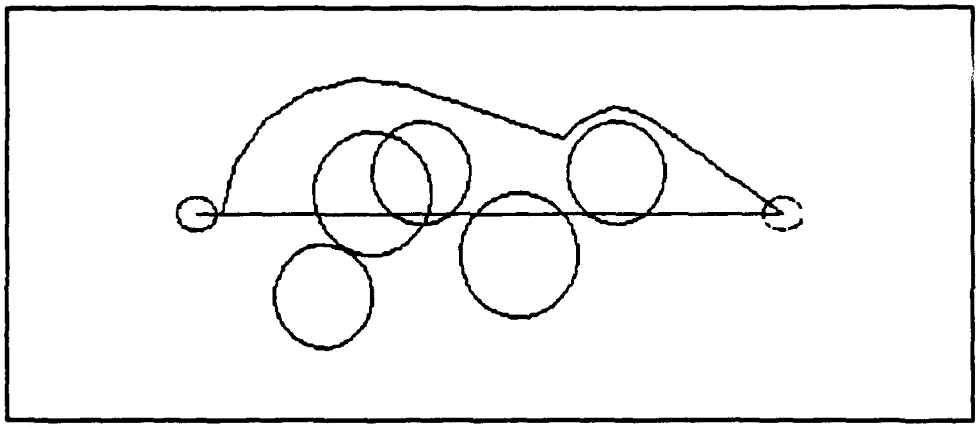
Figure 4 Crisp Algorithm Avoidance of Spherical Obstacles Set 4

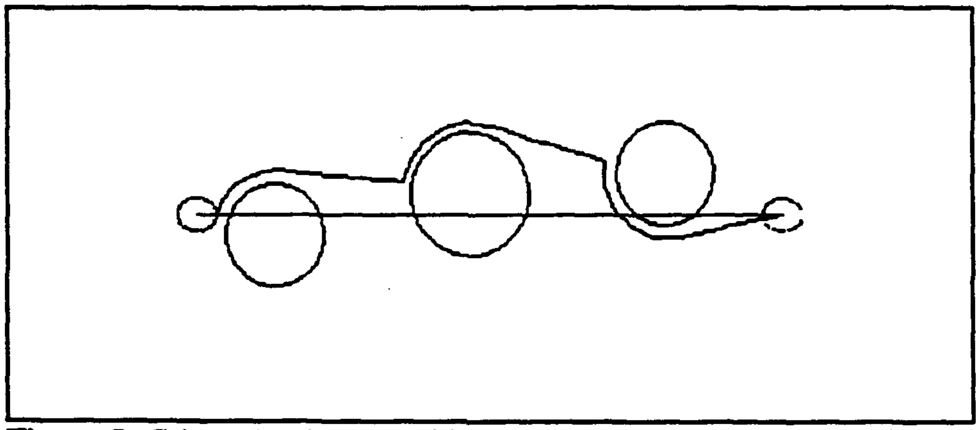
Figure 5 Crisp Algorithm Avoidance of Spherical Obstacles Set 5

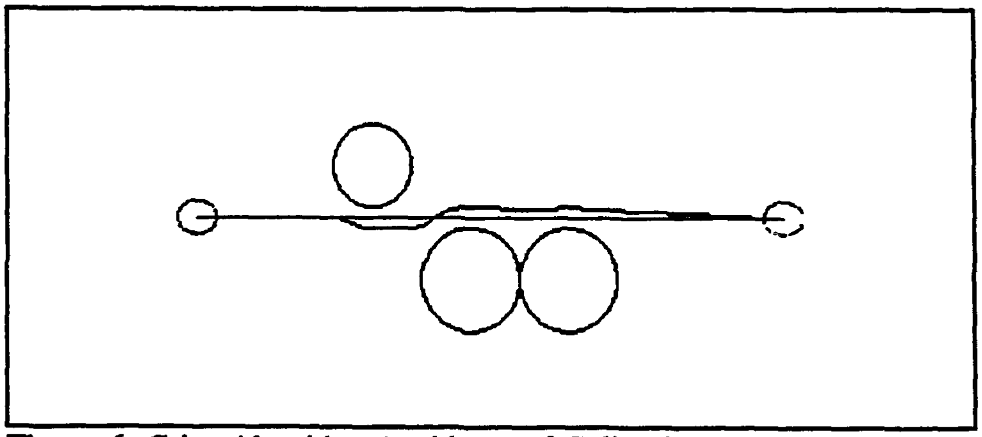
Figure 6 Crisp Algorithm Avoidance of Cylindrical Obstacles Set 1

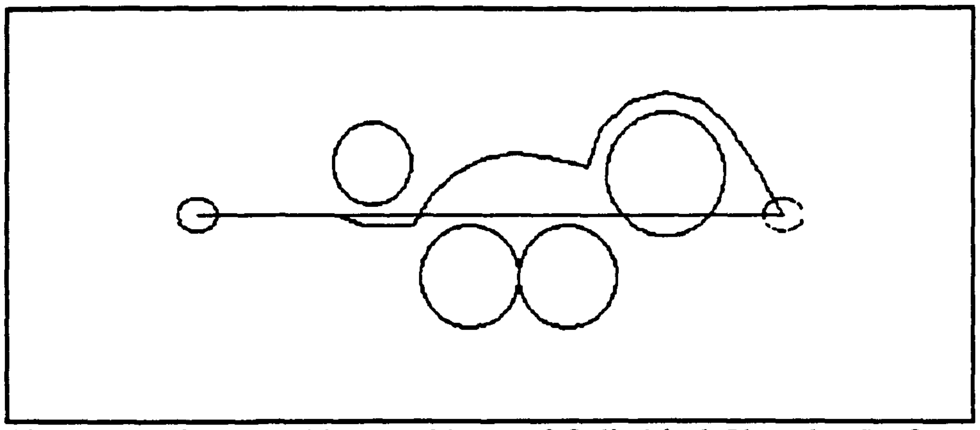
Figure 7 Crisp Algorithm Avoidance of Cylindrical Obstacles Set 2

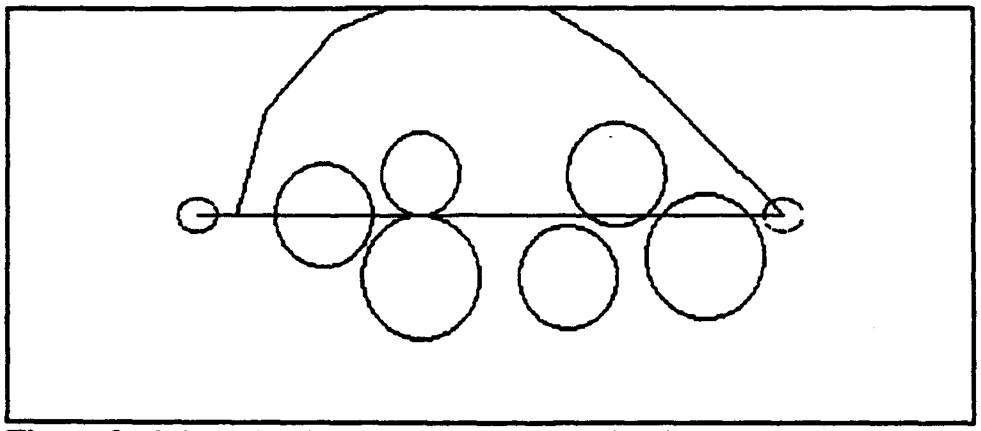
Figure 8 Crisp Algorithm Avoidance of Cylindrical Obstacles Set 3

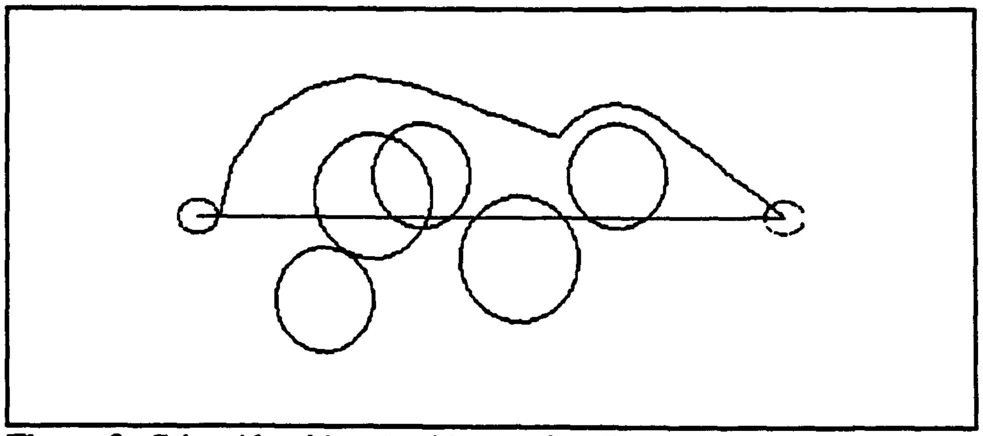
Figure 9 Crisp Algorithm Avoidance of Cylindrical Obstacles Set 4

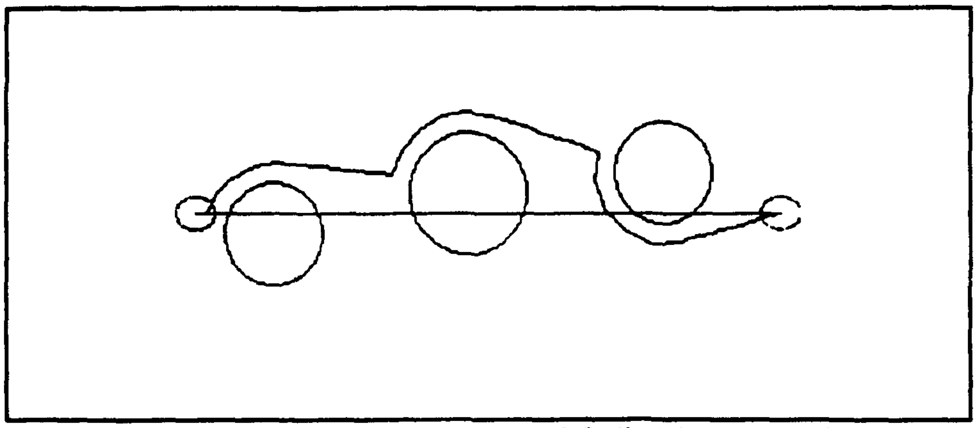
Figure 10 Crisp Algorithm Avoidance of Cylindrical Obstacles Set 5

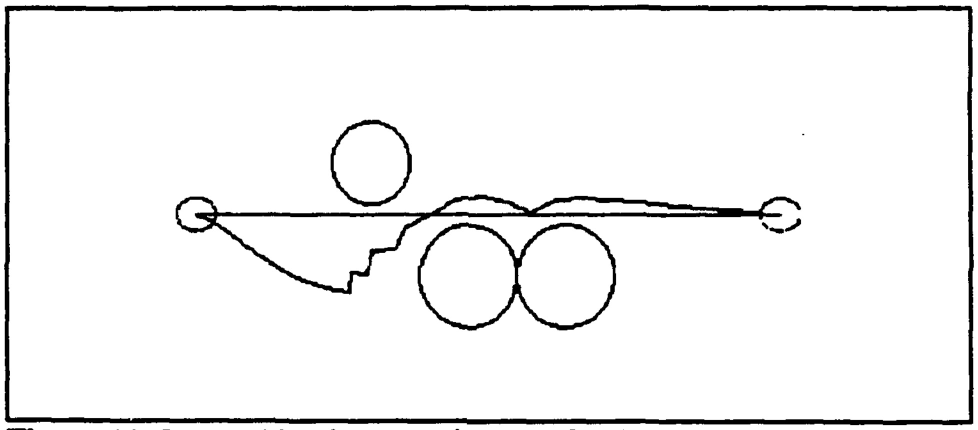
Figure 11 Fuzzy Algorithm Avoidance of Spherical Obstacles Set 1

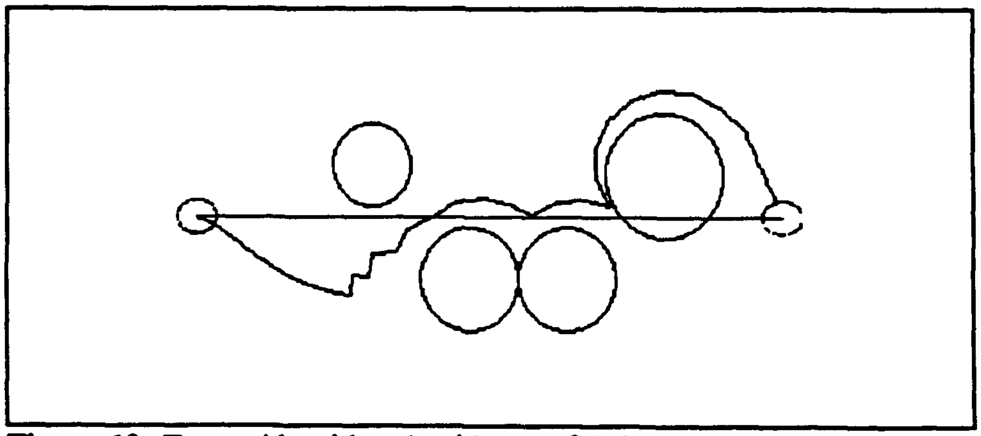
Figure 12 Fuzzy Algorithm Avoidance of Spherical Obstacles Set 2

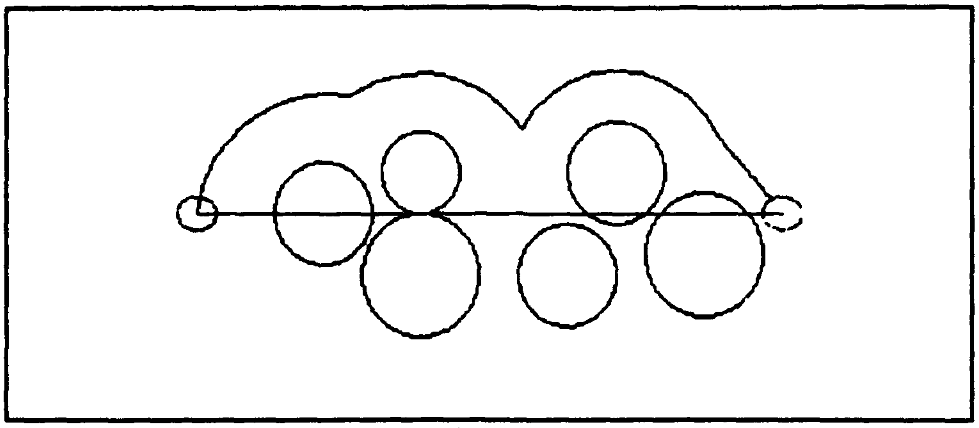
Figure 13 Fuzzy Algorithm Avoidance of Spherical Obstacles Set 3

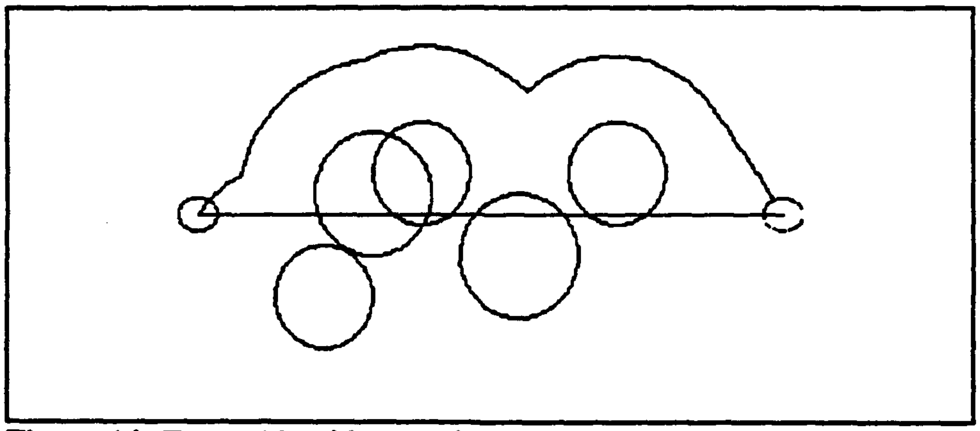
Figure 14 Fuzzy Algorithm Avoidance of Spherical Obstacles Set 4

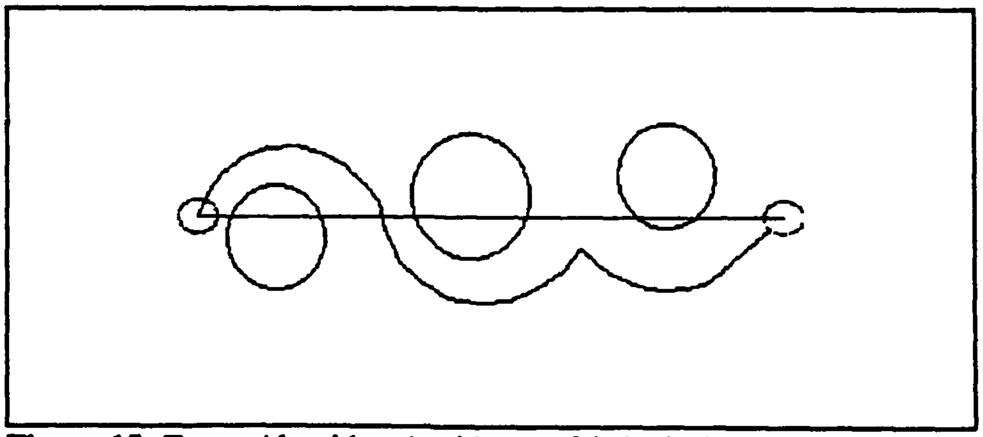
Figure 15 Fuzzy Algorithm Avoidance of Spherical Obstacles Set 5

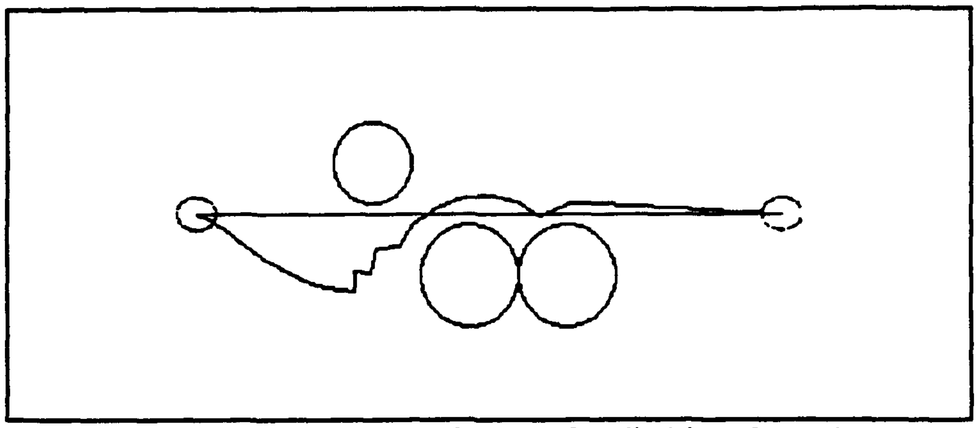

Figure 16 Fuzzy Algorithm Avoidance of Cylindrical Obstacles Set 1

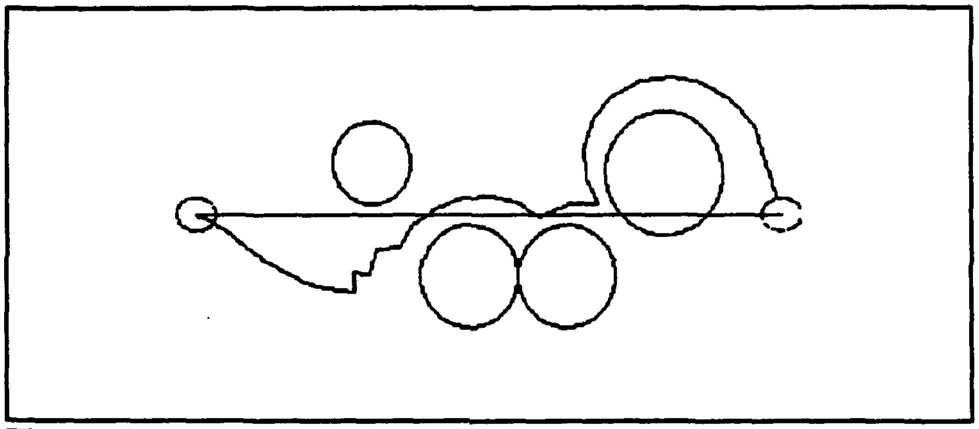

Figure 17 Fuzzy Algorithm Avoidance of Cylindrical Obstacles Set 2

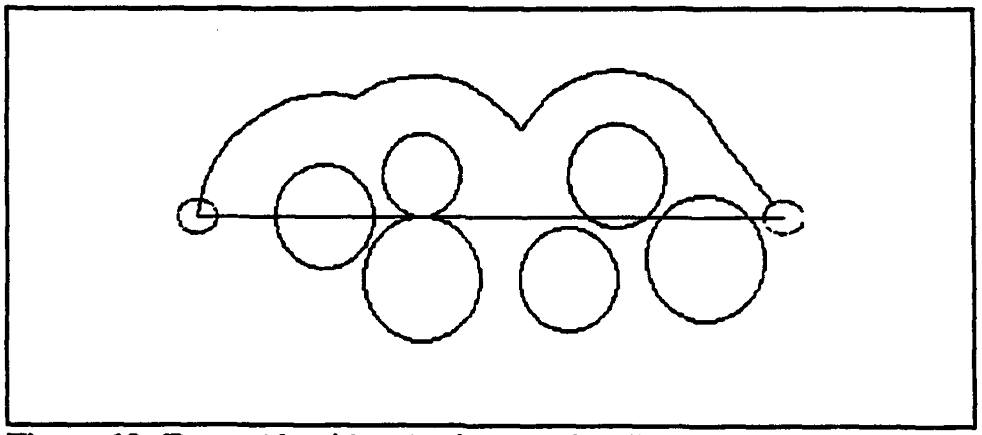

Figure 18 Fuzzy Algorithm Avoidance of Cylindrical Obstacles Set 3

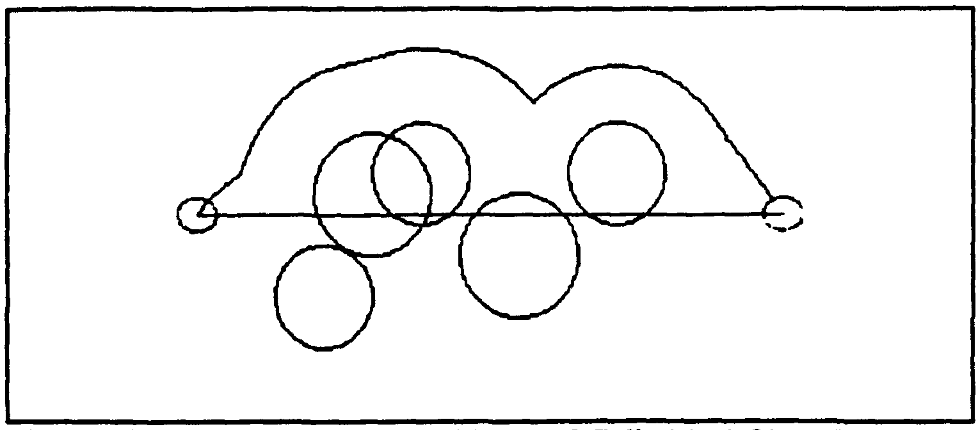
Figure 19 Fuzzy Algorithm Avoidance of Cylindrical Obstacles Set 4

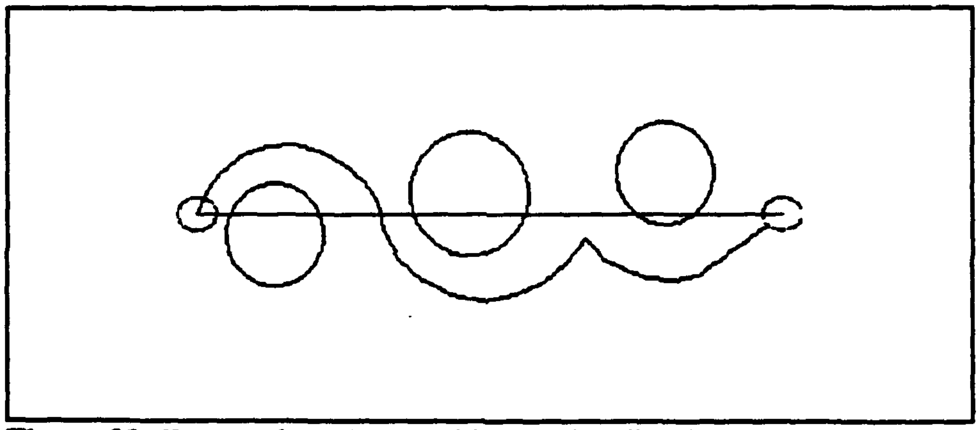
Figure 20 Fuzzy Algorithm Avoidance of Cylindrical Obstacles Set 5

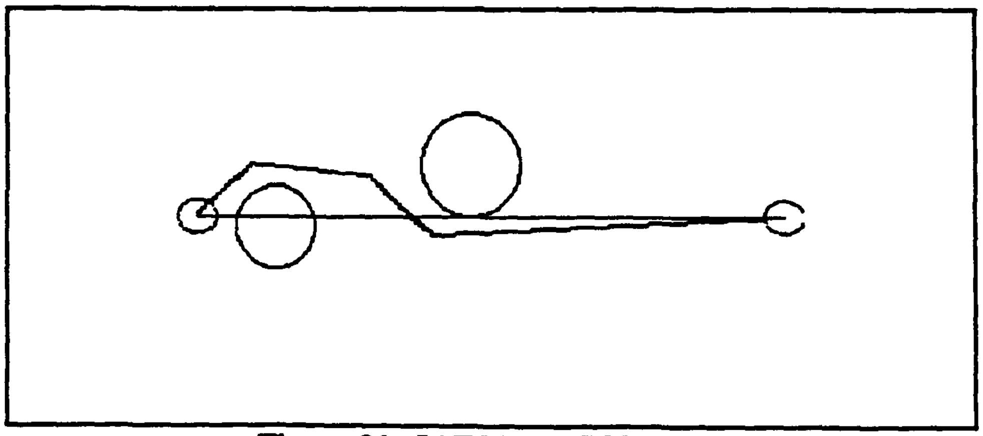
Figure 21 SAFOR ALGORITHM

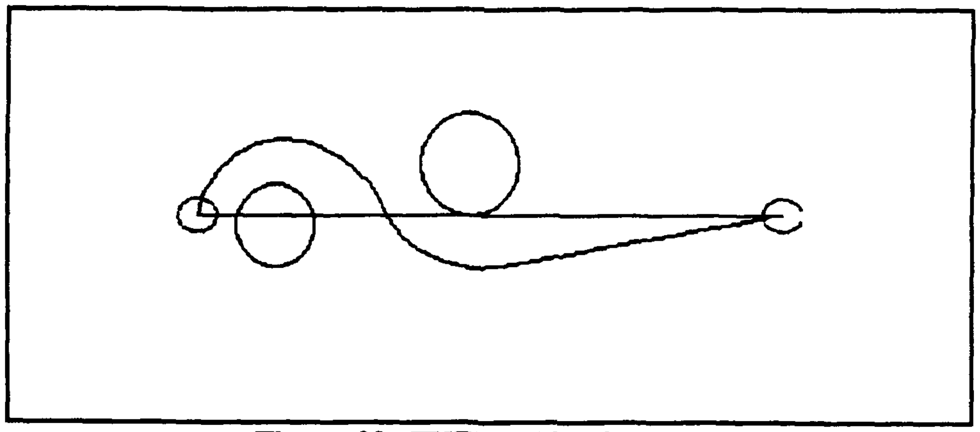
Figure 22 FUZZY ALGORITHM

APPENDIX B

CREATING, STORING, AND REPLAYING SAFOR EXERCISES ON SIMNET

APPENDIX B
CREATING, STORING, AND REPLAYING SAFOR EXERCISES ON SIMNET

1. SILICON GRAPHICS WORK STATIONS

a. Introduction

Creating a SAFOR scenario or playing back one that has been saved in a file requires two Silicon Graphics work stations, one called the front end and the other the back end. Logging a scenario, i.e., saving it for future playback, requires a third. However, without interfering with any other user that it may have, the third work station can be accessed via the network and displayed on the back end terminal.

A logged scenario saved in a file can not only be played back using two Silicon Graphics work stations, it can also be run on the Stealth system. VCR equipment in the Stealth room can record the scenario, or selected portions of it viewed in any of the available Stealth modes, on videotape.

Since the Silicon Graphics operating system is Unix an elementary knowledge of Unix is occasionally helpful in carrying out the steps, e.g., for deleting or changing the name of a file. A paperback text on the subject is sometimes available in the SIMNET area.

The next two sections detail the basic steps needed to create, record, and play back SAFOR scenarios. The steps, which are in the necessary order of their application, are numbered for easy reference. Italicized words or phrases in the text are commands or labels as they actually appear on a Silicon Graphics terminal screen.

b. Steps for Creating a Scenario

(1) Set up two Silicon Graphics terminals: alex and springfield, making alex the back end and springfield the front end. Start by logging in on each terminal and then going to the appropriate SAFOR subdirectory, which is the same for both, by entering the command:

cd /usr/saf/4.3.3. On alex set up the back end connection with the terrain data base[1] and assign an exercise number by entering the command: *phantom -terrain hunter-0110 -e 18*. To set up the front end on springfield use the command:

ws -terrain hunter-0110 -sim_ex 18 -po_ex 18. As indicated, the exercise is labeled 18, which is arbitrary but chosen here because it is unlikely to be the same as the number associated with a Stealth exercise that may be going on at the same time, since the default number is 1. After a short wait a map will appear on the springfield screen.

(2) Use the following procedure to connect the front end with the back end. At the top of the screen click on *File*--then, in the menu that drops down, on *Connect*--then, in the new menu appearing on the right, on *alex*. A message at the top of the screen will announce the connection when it occurs, or a failure if, for some reason it does not.

(3) Select the region of interest on the map and enlarge its image on the screen. To do this move the cursor to the center of the region and press the middle mouse button to zoom in. By dragging the number in the panel labelled *Vehicle Scale* at the bottom of the screen on the right, at any time the mouse can be used to set a scale that determines the size of the icons representing vehicles on the map. The number increases when dragged to the right and decreases when dragged to the left.

(4) Create an Overlay by first clicking on *Overlay* at the top of the screen and then on *Create* in the resulting menu that drops down. A panel will appear at the bottom of the screen requesting an Overlay name. Enter a name in the designated area; then click on *OK*.

(5) To add vehicles to the Overlay click on the tank shaped icon on the right side of the screen. A menu will appear requesting the echelon level. Click on the one desired, and drag the mouse to the left. Then a menu with a list of vehicle types will drop down. After clicking on one of those listed, a military symbol of the choice will appear on the map. To change the symbol to one more pictorially representative of the actual vehicle, click on the designation *Non-Military* among the items listed on the bottom left of the screen. Other items listed in the same area, such as *Contour Lines*, which displays constant terrain altitude curves on the map, may also be useful.

[1] All scenarios discussed in this document take place on terrain defined by data stored in the Hunter-Liggett data base.

(6) To edit or get information about an Overlay vehicle click on the icon in either the military or nonmilitary representation. This will produce a small menu including the option *Describe*, which, when selected, provides the vehicle type and I.D. number.

Another option is *Edit*, which can be used to make certain changes, including the vehicle's location. Clicking on *Edit* produces a panel containing options such as *Alignment*, which allows the user to select a *Friendly* (US) or *Enemy* (Soviet) type of vehicle, or *Echelon*, which allows the user to change to any echelon in the range from a single vehicle to a battalion. Among the options included is one labelled *Direction (mils)* with an associated number in a narrow horizontal track. Clicking on the number, which is at the left end of the track and is 0 initially, and dragging it to the right has two effects. It causes the vehicle to turn in the clockwise direction and the number to increase. Dragging to the left reverses both effects. Also, while in the edit mode, clicking anywhere on the map will cause the vehicle to move to the cursor position. To leave the edit mode click on *OK*.

(7) To use the overlay vehicles in an exercise it is necessary to simulate them by clicking on *Simulate* in the Overlay menu. A military symbol will replace each vehicle icon on the map. To return to the nonmilitary form click on *Show As* located at the top of the screen on the left--then on *Vehicle* in the resulting menu that appears.

Although both are initially at the same location, the simulated vehicle has an identity separate from the one originally created in the Overlay. Clicking on the simulated vehicle produces a menu that includes the options: *Describe*, *Edit*, *TAC/E*, and *Unit Tasking*.

The first two items perform the same functions for the simulated vehicle as those in the menu associated with the original Overlay vehicle. However, the changes that can be made in the panel created by clicking on *Edit* are now limited to changing the orientation of the simulated vehicle and moving it to another location.

(8) To get the vehicles moving in the exercise use either the *TAC/E* or the *Unit Tasking* option. Clicking on *TAC/E* produces a menu containing several vehicle override and command options.

Important override settings are *Speed*, which allows values ranging from 0 to 85 km/hr and *Formation*. Clicking on *formation* and then on *new formation* in the panel that appears leads to a menu on the left containing from 5 to 10 options, depending on the echelon level.

Several useful commands under *TAC/E* are *Go To Location*, *Go To Location Via Road*, and *Follow Route*. Clicking on either *Go To Location* option causes the appearance of a panel that is best ignored. Instead, just click on the desired location and on *OK* in the panel; the vehicle will then move, directly in the first case and in a roundabout way in the second, to the indicated spot. To accomplish the same end by having the vehicle follow a prescribed route, use *Unit Tasking*.

(9) To create a route, which is necessary before a vehicle can follow it, click on the route icon on the right side of the screen. Before doing so, however, it is advisable to click on *Overlay* at the top of the screen and then on *Show*. In the resulting panel make sure that, along with the button labeled *Simulation* the one labeled *Overlay* is depressed; otherwise, the route path will not be visible. Then click on the starting point of the route on the map. Click on the next route point connected by a straight line with the first. Continue to add straight line segments to the route by clicking on the locations of their endpoints until the route is completed. Note that in the accompanying panel a choice between solid or dashed line segments is available. After completing the route definition click on *OK* in the panel.

(10) There are two ways to command a vehicle, or a higher echelon unit of vehicles, to follow the route. One is to use the *Follow Route* option under *TAC/E* in the menu obtained by clicking on the vehicle or unit icon. The other is to invoke *Unit Tasking* rather than *TAC/E*. The second method produces a panel that requires confirmation of the Overlay name, the route name, and the CIS[2], which for route following expects the *Roadmarch* option. Click on each item, making sure that each corresponding name appears under the appropriate heading in the sub-panel below. When this happens a CIS editor panel appears. Select the vehicle speed and click on *OK*. The Unit Tasking panel will then reappear. To start the vehicle on its way click on *Execute* at the bottom line; then click on *OK* to remove the panel.

(11) To save an overlay, which is useful for modifying a scenario after running one version without having to go through all of the preliminary steps again, just click on *Overlay* and then on *Save* in the ensuing menu. A panel requesting a file name will appear. Enter a file name and click on *OK*.

(12) To start over using the same Overlay, including vehicles and routes, click on *File*, then on *Delete All* in the menu that drops down. After confirming the command by

2 Combat Instruction Set

clicking on *OK* in the ensuing panel, click on *Overlay*, followed by *Load* in the resulting menu that appears. A menu containing the list of saved Overlay files will then appear. Clicking on the filename of a previously saved Overlay will recreate it. **N.B.**: to continue it is necessary to simulate the appropriate Overlay entities by clicking on *Simulate* in the Overlay menu, as well as the correct Overlay name appearing on the right.

c. Some Useful Aids in Designing and Viewing a Scenario

Clicking on the middle mouse button and dragging creates a rectangular frame, the upper left hand corner of which starts at the initial cursor position. The lower right hand corner follows the cursor. When the mouse button is released the region inside the frame becomes the center of the screen view, occupying a larger fraction of the screen than it did before. The smaller the frame the more the scene within it will be magnified in this process.

To change the time between updates of the moving vehicle positions click on *Options* at the top of the screen--then click on *Pvd Display Options* in the menu that drops down. A panel will appear at the bottom of the screen containing a track labeled *Seconds Between Update*. The number 5 appears a short distance from the track's left end. Clicking on the number and dragging to the left reduces it and, therefore, the time between updates. This makes the vehicle motion less jerky. Dragging as far as possible to the left reduces the number to 0 and provides the smoothest possible motion.

Setting a marker at a point on a route can be a useful device in synchronizing the motion of two different vehicles, e.g., delaying the start of one until the instant the other passes the marker. One way to mark a point without interfering with the scenario is to create another Overlay and then from among the entities available in the vehicle menu, choose the dismounted infantry. After clicking on the corresponding icon, which should be set to the nonmilitary version as described in Step 5, use the *Edit* option in the resulting menu that appears, as described in Step 6, to move it to the point where the marker is needed. Do not simulate the Overlay in which the dismounted infantry used for a marker was created. To be able to repeat the initial scene, save that Overlay and the Overlay containing the vehicles and routes in separate files.

When redoing a scenario by restoring its initial state, restore each Overlay separately by clicking on *Load* in the Overlay menu and selecting its filename in the resulting list that appears. Simulate the Overlay containing the vehicle entities and the routes, but do not simulate the dismounted infantry Overlay.

d. Steps in Saving a Scenario

(1) The process begins on roanoke, which can access the required logging program. It is not necessary to log in on the roanoke terminal. Instead, while logged in on alex create a new window and enter the command *telnet roanoke*. At the resulting prompt log in as usual. Then enter the command *cd /usr/saf/4.3.3* followed by *Otblgr -display alex:0.0*, which allows the display of the logger control panel to appear on the alex terminal screen.

(2) After the logger control panel appears type in the name of the file in which the scenario is to be stored. The filename should include a path starting with */usr/people/* followed by a subdirectory in which the user has write privileges on roanoke. Press the Return key; type in the exercise number, i.e., 18, in the designated place, and press the Return key.

(3) Set up alex and springfield as in step b(1) and connect them as in step b(2), if the setup and connection do not already exist. On alex this can be done by bringing up another window for the purpose. Once the connection is made switch back, making the roanoke window containing the logger control panel active.

(4) On springfield create and run the scenario to be recorded as in steps b(3)–b(11). Click on *Record* at the top of the logger control panel on the alex terminal to start the recording session. This can be done either before or after the scenario is started on springfield. To end the recording session, click on *Stop* at the top of the logger control panel on alex.

(5) To play back a recorded scenario springfield should not be connected to alex, but the logger control panel on roanoke must be created and displayed either on roanoke or on alex. To create and display it on alex follow the step c(1) procedure. Otherwise, log in on roanoke, type the command *cd /usr/saf/4.3.3*, press the Return key, and then type the command *Otblgr*. The logger control panel will then appear on roanoke.

In either case, in the designated place enter the file name, including the correct path, which starts out with */usr/people/*. Press the Return key. The exercise number 18 used for the recording session should appear in the place reserved for it on the panel, and a number should appear in *Time Remaining* indicating the amount of time the scenario will run. The playback can be speeded up or slowed down by clicking on the speed factor track labelled *1.0xSpeed* and dragging it to the right or left. The number replacing 1.0 gets smaller while dragging to the left and larger while dragging to the right.

To start the scenario click on *Play* or *Loop Play* at the top of the panel. To stop the play back and begin again, click on *Stop*--then click on and drag the button in the *Time* track as far as it will go to the left--then click on *Play* or *Loop Play* once more.

2. PLAYING BACK AN EXERCISE ON STEALTH

The first step requires that an authorized person log in as root on the right hand terminal. Once this is done, enter the same commands as those used on the Silicon Graphics terminal springfield in step b(1) except that 4.3.3 must be replaced by 4.3.6 and ./ws must be used in place of ws. The same map that would appear on springfield will appear on the right hand Stealth terminal.

Next, on the left hand terminal follow the same procedure as in d(1), first entering *telnet roanoke*, then logging in (not necessarily as root) and entering *cd /usr/saf/4.3.3*. Then enter *Otblgr -display idaobgl:0.1 &*, which is similar to, but not the same as, the command in d(1). The logger control panel will appear on the left hand terminal just as it does on alex in the procedure of d(1).

Follow the procedure described in d(2) to play back a stored exercise, but before playing it go to the right hand terminal. Under *Options* at the top of the screen click on the last item, which has the form $FC(n_1n_2)$ with numbers n_1 and n_2 that may vary, and enter the correct exercise number, i.e., 18, in the *Flying Carpet* panel that appears.

Clicking on the terminal screen causes a white arrow that locates the viewing position on the displayed map and points in the viewing direction to jump to the cursor position. At the same time the observer's stealth viewpoint moves to the new location indicated by the arrow on the map.

Other options on the *Flying Carpet* panel are dynamic viewpoints, which include attaching or tethering to a vehicle. To select one click on the panel and then on the desired vehicle.

The observer's viewpoint can also be controlled by the spaceball to the right of the right hand terminal. The ball can change the viewpoint location in three dimensions and its numerical altitude relative to the ground is displayed in a panel on the terminal screen. A line of buttons numbered from 1 to 8 at the front end of the spaceball base control the Flying Carpet view as indicated in Table B-1.

Table B-1. Spaceball Quick Reference Flying Carpet

Constant Altitude 1	Rotation On/Off 2	Dominant Mode 3	Level View 4
Free Fly 5	Increase Sensitivity 6	Decrease Sensitivity 7	Re-Zero 8

APPENDIX C

SAFOR VEHICLE COLLISION AVOIDANCE CODE

APPENDIX C

SAFOR VEHICLE COLLISION AVOIDANCE CODE

1. Present Crisp Version

```
/*****************************************************************************
* Modify the desired velocity of the avoiding vehicle by
* a quantity appropriate to making a greater gap between the two
* vehicles at their projected cpa (closest point of approach)
*****************************************************************************/
void avoid_a_vehicle( avoider, avoidee, desired_vel, goal_pt )

  SAF_OBJECT *avoider, *avoidee;
  VECTOR    desired_vel, goal_pt;
{
 VECTOR rel_pos, rel_vel, delta_vel, delta_pos, rel_pos_of_point;
 double dist_at_cpa, dist_to_cpa, time_to_cpa, rel_speed, scale_factor;
 double time_to_point, dist_to_point, speed;
 double time_stagger = ( avoider->tickable->now - avoidee->tickable->now )
               * .001;
 void  increment_desired_vel();

 /* calculate relative position taking into consideration the difference
```

```
  in tick times */
VEC2_SUB( avoidee->entity->position, avoider->entity->position, rel_pos );
VEC2_MULT( time_stagger, avoidee->entity->velocity, delta_pos );
VEC2_ADD( delta_pos, rel_pos, rel_pos );

/* calculate distance at closest point of approach */
VEC2_SUB( avoider->entity->velocity, avoidee->entity->velocity, rel_vel );
rel_speed = vec2_mag( rel_vel );
if( rel_speed < SMALL )
  return;
dist_at_cpa = VEC2_CROSS( rel_vel, rel_pos ) / rel_speed;

/* if this distance is greater than a threshold, no need to avoid */
if( abs( dist_at_cpa ) > Passing_Dist )
  return;

/* calculate time to closest point of approach */
dist_to_cpa = VEC2_DOT( rel_vel, rel_pos ) / rel_speed;
if( dist_to_cpa <= 0 )
  return;
time_to_cpa = dist_to_cpa / rel_speed;
if( time_to_cpa > GRND_LOOK_AHEAD )
```

```
   return;



  /* if there is a goal point, calculate time to it */
  if( goal_pt != Null_Vector ) {
   speed = vec2_mag( avoider->entity->velocity );
   if( speed < SMALL )
    return;
   VEC2_SUB( goal_pt, avoider->entity->position, rel_pos_of_point );
   dist_to_point = vec2_mag( rel_pos_of_point );
   time_to_point = dist_to_point / speed;

   /* if will reach goal point, before collision, don't do avoidance */
   if( time_to_point < time_to_cpa )
    return;
  }

  /* calculate magnitude in change of velocity in direction normal to
    relative velocity away from collision */
  scale_factor =
   SGN( dist_at_cpa ) * AVOID_FACTOR * ( Passing_Dist - ( abs( dist_at_cpa ) ) )
    / ( MAX(time_to_cpa, MINIMUM_TIME) );
```

```
 delta_vel[ 0 ] =  rel_vel[ 1 ] * scale_factor/rel_speed;

 delta_vel[ 1 ] = - rel_vel[ 0 ] * scale_factor/rel_speed;


 VEC2_ADD( desired_vel, delta_vel, desired_vel );

}

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

2. Crisp Version with Slow Down

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

*  Modify the desired velocity of the avoiding vehicle by

*  a quantity appropriate to making a greater gap between the two

*  vehicles at their projected cpa (closest point of approach)

/*************************************************************************/

void avoid_a_vehicle( avoider, avoidee, desired_vel, goal_pt )


   SAF_OBJECT *avoider, *avoidee;

   VECTOR     desired_vel, goal_pt;

{

 VECTOR rel_pos, rel_vel, delta_vel, delta_pos, rel_pos_of_point,old_vel;

 double dist_at_cpa, dist_to_cpa, time_to_cpa, rel_speed, scale_factor;

 double time_to_point, dist_to_point, speed, scf;

 double time_stagger = ( avoider->tickable->now - avoidee->tickable->now )
```

```
                * .001;
    void  increment_desired_vel();


     VEC2_COPY(avoider->entity->velocity,old_vel);


    /* calculate relative position taking into consideration the difference
      in tick times */
    VEC2_SUB( avoidee->entity->position, avoider->entity->position, rel_pos );
    VEC2_MULT( time_stagger, avoidee->entity->velocity, delta_pos );
    VEC2_ADD( delta_pos, rel_pos, rel_pos );


    /* calculate distance at closest point of approach */
    VEC2_SUB( avoider->entity->velocity, avoidee->entity->velocity, rel_vel );
    rel_speed = vec2_mag( rel_vel );
    if( rel_speed < SMALL )
      return;
    dist_at_cpa = VEC2_CROSS( rel_vel, rel_pos ) / rel_speed;


    /* if this distance is greater than a threshold, no need to avoid */
    if( abs( dist_at_cpa ) > Passing_Dist )
      return;
```

```
/* calculate time to closest point of approach */
dist_to_cpa = VEC2_DOT( rel_vel, rel_pos ) / rel_speed;

/* if avoider is already past avoidee then return to desired velocity and
quit */
if( dist_to_cpa <= 0 )
        {
        if (!(IS_AIRCRAFT(OBJ_OBJECT_TYPE(avoider))))
                {
                avoider->entity->velocity[0] = old_vel[0];
                avoider->entity->velocity[1] = old_vel[1];
                }
        return;
        }
time_to_cpa = dist_to_cpa / rel_speed;
if( time_to_cpa > GRND_LOOK_AHEAD )
  return;

/* if there is a goal point, calculate time to it */
if( goal_pt != Null_Vector ) {
  speed = vec2_mag( avoider->entity->velocity );
  if( speed < SMALL )
```

```
    return;
  VEC2_SUB( goal_pt, avoider->entity->position, rel_pos_of_point );
  dist_to_point = vec2_mag( rel_pos_of_point );
  time_to_point = dist_to_point / speed;


  /* if will reach goal point, before collision, don't do avoidance */
  if( time_to_point < time_to_cpa )
    return;
 }


/* calculate magnitude in change of velocity in direction normal to
   relative velocity away from collision */
  scf =
    AVOID_FACTOR * ( Passing_Dist - ( abs( dist_at_cpa ) ) )
   / ( MAX(time_to_cpa, MINIMUM_TIME) );
/* if the scale factor is not too small slow down before incrementing the velocity */
if(!(IS_AIRCRAFT(OBJ_OBJECT_TYPE(avoider))))
        {
        if(scf > 1.0)vec2_div(scf,avoider->entity->velocity, desired_vel);
        }
scale_factor = SGN(dist_at_cpa)*scf;
delta_vel[ 0 ] =  rel_vel[ 1 ] * scale_factor/rel_speed;
```

```
 delta_vel[ 1 ] = - rel_vel[ 0 ] * scale_factor/rel_speed;
 VEC2_ADD( desired_vel, delta_vel, desired_vel );
}
/****************************************************************************/
```

3. Fuzzy Version

```
/****************************************************************************/
* Modify the desired velocity of the avoiding vehicle by
* a quantity appropriate to making a greater gap between the two
* vehicles at their projected cpa (closest point of approach)
*****************************************************************************/
void avoid_a_vehicle( avoider, avoidee, desired_vel, goal_pt )

   SAF_OBJECT *avoider, *avoidee;
   VECTOR    desired_vel, goal_pt;
{
 float fuzzy_scale_factor();
 VECTOR rel_pos, rel_vel, delta_vel, delta_pos, rel_pos_of_point;
 double dist_at_cpa, dist_to_cpa, time_to_cpa, rel_speed, scale_factor;
 double time_to_point, dist_to_point, speed;
 double time_stagger = ( avoider->tickable->now - avoidee->tickable->now )
                 * .001;
```

```
float fsf;
void  increment_desired_vel();


/* calculate relative position taking into consideration the difference
  in tick times */
VEC2_SUB( avoidee->entity->position, avoider->entity->position, rel_pos );
VEC2_MULT( time_stagger, avoidee->entity->velocity, delta_pos );
VEC2_ADD( delta_pos, rel_pos, rel_pos );


/* calculate distance at closest point of approach */
VEC2_SUB( avoider->entity->velocity, avoidee->entity->velocity, rel_vel );
rel_speed = vec2_mag( rel_vel );
if( rel_speed < SMALL )
 return;
dist_at_cpa = VEC2_CROSS( rel_vel, rel_pos ) / rel_speed;


/* if this distance is greater than a threshold, no need to avoid */
if( abs( dist_at_cpa ) > Passing_Dist )
 return;


/* calculate time to closest point of approach */
dist_to_cpa = VEC2_DOT( rel_vel, rel_pos ) ' el_speed;
```

```
if( dist_to_cpa <= 0 )
  return;
time_to_cpa = dist_to_cpa / rel_speed;


/* if there is a goal point, calculate time to it */
if( goal_pt != Null_Vector ) {
  speed = vec2_mag( avoider->entity->velocity );
  if( speed < SMALL )
    return;
  VEC2_SUB( goal_pt, avoider->entity->position, rel_pos_of_point );
  dist_to_point = vec2_mag( rel_pos_of_point );
  time_to_point = dist_to_point / speed;


  /* if will reach goal point, before collision, don't do avoidance */
  if( time_to_point < time_to_cpa )
    return;
}


/* calculate magnitude in change of velocity in direction normal to
  relative velocity away from collision */
fsf = fuzzy_scale_factor(Passing_Dist, dist_to_cpa, speed);
scale_factor =
   SGN( dist_at_cpa ) * AVOID_FACTOR * fsf;
  delta_vel[ 0 ] =  rel_vel[ 1 ] * scale_factor/rel_speed;
  delta_vel[ 1 ] = - rel_vel[ 0 ] * scale_factor/rel_speed;
  VEC2_ADD( desired_vel, delta_vel, desired_vel );
}
/******************************************************************************/
```

4. Fuzzy Version with Slow Down

```
/******************************************************************************/
*  Modify the desired velocity of the avoiding vehicle by
*  a quantity appropriate to making a greater gap between the two
*  vehicles at their projected cpa (closest point of approach)
******************************************************************************/
void avoid_a_vehicle( avoider, avoidee, desired_vel, goal_pt )

    SAF_OBJECT *avoider, *avoidee;
    VECTOR     desired_vel, goal_pt;
{
  float fuzzy_scale_factor();
  VECTOR rel_pos, rel_vel, delta_vel, delta_pos, rel_pos_of_point,old_vel;
  double dist_at_cpa, dist_to_cpa, time_to_cpa, rel_speed, scale_factor;
  double time_to_point, dist_to_point, speed;
double time_stagger = ( avoider->tickable->now - avoidee->tickable->now )

                * .001;

float fsf;

void  increment_desired_vel();


VEC2_COPY(avoider->entity->velocity,old_vel);


/* calculate relative position taking into consideration the difference

  in tick times */

VEC2_SUB( avoidee->entity->position, avoider->entity->position, rel_pos );

VEC2_MULT( time_stagger, avoidee->entity->velocity, delta_pos );

VEC2_ADD( delta_pos, rel_pos, rel_pos );


/* calculate distance at closest point of approach */

VEC2_SUB( avoider->entity->velocity, avoidee->entity->velocity, rel_vel );

rel_speed = vec2_mag( rel_vel );

if( rel_speed < SMALL )

  return;

dist_at_cpa = VEC2_CROSS( rel_vel, rel_pos ) / rel_speed;


/* if this distance is greater than a threshold, no need to avoid */

if( abs( dist_at_cpa ) > Passing_Dist )
```

```
  return;

 /* calculate time to closest point of approach */
 dist_to_cpa = VEC2_DOT( rel_vel, rel_pos ) / rel_speed;
 if( dist_to_cpa <= 0 )
        {
        if (!(IS_AIRCRAFT(OBJ_OBJECT_TYPE(avoider))))
                {
                avoider->entity->velocity[0] = old_vel[0];
                avoider->entity->velocity[1] = old_vel[1];
                }
        return;
        }
time_to_cpa = dist_to_cpa / rel_speed;


 /* if there is a goal point, calculate time to it */
 if( goal_pt != Null_Vector ) {
  speed = vec2_mag( avoider->entity->velocity );
  if( speed < SMALL )
   return;
  VEC2_SUB( goal_pt, avoider->entity->position, rel_pos_of_point );
  dist_to_point = vec2_mag( rel_pos_of_point );

  time_to_point = dist_to_point / speed;


  /* if will reach goal point, before collision, don't do avoidance */

  if( time_to_point < time_to_cpa )

    return;

 }


 /* calculate magnitude in change of velocity in direction normal to

   relative velocity away from collision */

 fsf = fuzzy_scale_factor(Passing_Dist, dist_to_cpa, speed);


 /* if avoider is not an aircraft and the scale factor is not too small slow

 down */

 if(!(IS_AIRCRAFT(OBJ_OBJECT_TYPE(avoider))))

        {

        if(fsf>1.0) vec2_div(fsf,avoider->entity->velocity,desired_vel);

        }

scale_factor =

  SGN( dist_at_cpa ) * AVOID_FACTOR * fsf;

 delta_vel[ 0 ] =  rel_vel[ 1 ] * scale_factor/rel_speed;

 delta_vel[ 1 ] = - rel_vel[ 0 ] * scale_factor/rel_speed;
 VEC2_ADD( desired_vel, delta_vel, desired_vel );
}
/*******************************************************************************/
```

5. Fuzzy Functions

```
/*******************************************************************************
* This function calculates a fuzzy scale_factor replacing the one used in the
*SAFOR function avoid.c.
*******************************************************************************/
float fuzzy_scale_factor(pd, dcpa, speed)


#define DCPA1 10.00
#define DCPA2 50.00
#define DCPA3 100.00
#define SCLF1 2.00
#define SCLF2 6.00
#define SCLF3 10.00
#define PDNUM 10
#define DCPANUM 10
#define SCFNUM 3


float pd, dcpa, speed;
```

```
{
        float mf_scale_factor();
        int i, j, k;
        float opd[4], odcpa[4], rpd[4][4], rdcpa[4][4], rsclf[4][4][4],
scf_pt = 0, scf_inc, scf_mf, term, num_sum = 0, den_sum = 0,  scale_factor,
max_dcpa, max_pd, max_scf = SCLF3, PD1, PD2, PD3;
max_dcpa = 8.0*speed;

if (Passing_Dist == DI_PASSING_DIST)
        {
        PD1 = 0;
        PD2 = 5.00;
        PD3 = 10.00;
        }
if (Passing_Dist == GRND_PASSING_DIST)
        {
        PD1 = 5.00;
        PD2 = 10.00;
        PD3 = 50.00;
        }
if (Passing_Dist == HELO_PASSING_DIST)
        {
```

```
        PD1 = 10.00;
        PD2 = 50.00;
        PD3 = 100.00;
        }
    max_pd = PD3;


    /* Enter the array elements defining the membership functions of the fuzzy sets
    in the rules that determine the scale factor. */
    rpd[1][1] = PD3;
    rpd[1][2] = PD1;
    rpd[1][3] = PD2;
    rpd[2][1] = PD1;
    rpd[2][2] = PD2;
    rpd[2][3] = PD3;
    rpd[3][1] = PD2;
    rpd[3][2] = PD3;
    rpd[3][3] =  PD1;
    rdcpa[1][1] = DCPA3;
    rdcpa[1][2] = DCPA1;
    rdcpa[1][3] = DCPA2;
    rdcpa[2][1] = DCPA1;
    rdcpa[2][2] = DCPA2;
```

```
rdcpa[2][3] = DCPA3;

rdcpa[3][1] = DCPA2;

rdcpa[3][2] = DCPA3;

rdcpa[3][3] = DCPA1;

rsclf[1][1][1] = SCLF1;

rsclf[1][1][2] = SCLF2;

rsclf[1][1][3] = SCLF3;

rsclf[1][2][1] = SCLF3;

rsclf[1][2][2] = SCLF1;

rsclf[1][2][3] = SCLF2;

rsclf[1][3][1] = SCLF3;

rsclf[1][3][2] = SCLF1;

rsclf[1][3][3] = SCLF2;

rsclf[2][1][1] = SCLF2;

rsclf[2][1][2] = SCLF3;

rsclf[2][1][3] = SCLF1;

rsclf[2][2][1] = SCLF2;

rsclf[2][2][2] = SCLF3;

rsclf[2][2][3] = SCLF1;

rsclf[2][3][1] = SCLF1;

rsclf[2][3][2] = SCLF2;

rsclf[2][3][3] = SCLF3;
```

```
rsclf[3][1][1] = SCLF2;

rsclf[3][1][2] = SCLF3;

rsclf[3][1][3] = SCLF1;

rsclf[3][2][1] = SCLF2;

rsclf[3][2][2] = SCLF3;

rsclf[3][2][3] = SCLF1;

rsclf[3][3][1] = SCLF1;

rsclf[3][3][2] = SCLF2;

rsclf[3][3][3] = SCLF3;


/* Get the array elements defining the membership functions of the observed

(i.e., prescribed for the object to be avoided) passing distance. */

if (pd == PD1)

        {

        opd[1] = PD3;

        opd[2] = PD1;

        opd[3] = PD2;

        }

else

        {

        opd[1] = PD1;

        opd[2] = PD2;
        opd[3] = PD3;
        }


/* Get the array elements defining the membership functions of the observed
distance_to_cpa. */
if (dcpa <= DCPA1)
        {
        odcpa[1] = DCPA3;
        odcpa[2] = DCPA1;
        odcpa[3] = DCPA2;
        goto FINISH;
        }
if (dcpa >= DCPA3)
        {
        odcpa[1] = DCPA2;
        odcpa[2] = DCPA3;
        odcpa[3] = DCPA1;
        goto FINISH;
        }
odcpa[1] = DCPA1;
odcpa[2] = DCPA2;
odcpa[3] = DCPA3;
```

```
/* Then calculate the centroid of the scale_factor membership function.  The
centroid is the "best estimate" of the actual scale_factor to be used.
  First get the increment scf_inc of the scale_factor membership function's
argument scf_pt.  The increment will determine the points at which the
membership values are to be calculated. */
FINISH: scf_inc = max_scf/SCFNUM;
while (scf_pt < max_scf)
        {
        scf_pt += scf_inc;

/* Use the function scf_mf to calculate the scale_factor membership function
value at the point scf_pt. */
        scf_mf = mf_scale_factor(opd, odcpa, rpd, rdcpa, rsclf, max_pd,
        max_dcpa, scf_pt);

/* Use the result to calculate the centroid of the scale_factor membership
function. */
        term = scf_mf*scf_pt;
        num_sum += term;
        den_sum += scf_mf;
        }
if (den_sum > 0)
```

```
        return (num_sum/den_sum);
else
        return (0);
}


/*****************************************************************************
* This function calculates the membership function for the scale_factor, given
* the membership functions of the observed passing_distance obs_pd, the observed
* distance_to_cpa obs_dcpa, and 3 modens ponens rules that prescribe the
* scale_factor for each combination of the passing_distance pd and the
* distance_to_cpa dcpa.
*****************************************************************************/
float mf_scale_factor(obs_pd, obs_dcpa, pd, dcpa, sclfac, max_pd, max_dcpa,
scf_pt)

float obs_pd[4], obs_dcpa[4], pd[4][4], dcpa[4][4], sclfac[4][4][4], max_pd,
max_dcpa, scf_pt;
{
        float composition();
        int i,j,k;
        float opd[4], odcpa[4], rpd[4], rdcpa[4], rsclf[4], comp_val, max_val =
        0;
```

```
/* Get all obs_pd and obs_dcpa  array components. */

for (i = 1; i < 4; i++)

          {

          opd[i] = obs_pd[i];

          odcpa[i] = obs_dcpa[i];

          }


/* Get the membership function value of the scale_factor, which is the union of

the sets determined by each of the rules associated with all possible

combinations of a pd and a dcpa.  Start by getting the required array

components. */

for (i = 1; i < 4; i++)

          {

          for (j = 1; j< 4;j++)

                    {

                    for (k = 1; k < 4; k++)

                              {

                              rpd[k] = pd[i][k];

                              rdcpa[k] = dcpa[j][k];

                              rsclf[k] = sclfac[i][j][k];

                              }
/* The result of each rule is determined by the composition function. */
                comp_val = composition(opd, odcpa, rpd, rdcpa, rsclf,
                                max_pd, max_dcpa, scf_pt);


/* Then calculate the union membership function value. */
                if (comp_val > max_val)
                            max_val = comp_val;
                }
        }
return (max_val);
}


/*******************************************************************************
* This function calculates the membership function of the intersection of the
* observed passing_distance obs_pd and the observed distance_to_dcpa obs_dcpa,
* both of which have generic membership functions.  It does the same for the
* corresponding quantities whose intersection is the modens in a specific modens
* ponens rule: the passing_distance rule_pd and the distance_to_cpa rule_dcpa,
* both of which also have generic membership functions.  It then calculates the
* relation defined by the rule and then the composition of the intersection of
* the observed quantities with the relation.
*******************************************************************************/
```

```
float composition(obs_pd, obs_dcpa, rule_pd, rule_dcpa, sclfac, max_pd,
max_dcpa, scf_pt)

float obs_pd[4], obs_dcpa[4], rule_pd[4], rule_dcpa[4], sclfac[4], max_pd,
max_dcpa, scf_pt;
{
        float intersection(), membfunc(), fuzzy_and();
        int i;
        float opd[4], odcpa[4], pd_inc, dcpa_inc, pd_pt = 0, dcpa_pt = 0,
rpd[4], rdcpa[4], mf_obs_dist, mf_rule_dist, mf_relation, scf[4], mf_scf,
rule_mf, max_val = 0;

/* First get the increment of the 2 independent variables of the intersection
membership functions by dividing the continuous range of each, max_pd and
max_dcpa, by the number, PDNUM and DCPANUM, of discrete values to be
considered in either case. Then get the components of each of the arrays
involved. */
pd_inc = max_pd/PDNUM;
dcpa_inc = max_dcpa/DCPANUM;
for (i = 1; i < 4; i++)
        {
        opd[i] = obs_pd[i];
        odcpa[i] = obs_dcpa[i];
        rpd[i] = rule_pd[i];
        rdcpa[i] = rule_dcpa[i];
        scf[i] = sclfac[i];
        }

/* Get the value of the scale_factor membership function mf_scf, which is
generic. */
mf_scf = membfunc(scf[1], scf[2], scf[3], scf_pt);

/* Now do the promised composition inference calculation, which involves a
max-min product. */
while (pd_pt < max_pd)
        {
        pd_pt += pd_inc;
        while (dcpa_pt < max_dcpa)
                {
                dcpa_pt += dcpa_inc;

/* Get the membership function mf_obs_dist of the intersection of obs_pd and
obs_dcpa. */
                mf_obs_dist = fuzzy_and(opd, odcpa, pd_pt, dcpa_pt);
/* Get the membership function mf_rule_dist of the intersection of rule_pd and
rule_dcpa. */
                    mf_rule_dist = fuzzy_and(rpd, rdcpa, pd_pt, dcpa_pt);

/* Get the membership function mf_relation of the rule relation, which is
defined by the intersection of mf_rule_dist and mf_scf. */
                    mf_relation = intersection(mf_rule_dist, mf_scf);

/* Get the membership function rule_mf of the intersection of obs_dist and
mf_relation. */

                    rule_mf = intersection(mf_obs_dist, mf_relation);

/* Finally, get the composition value, which is defined as the maximum value of
rule_mf for all points pd_pt and dcpa_pt. */
                    if (rule_mf > max_val)
                         max_val = rule_mf;
                    }
          dcpa_pt = 0;
          }
return (max_val);
}
```

```
/******************************************************************************
* This function determines the value of the membership function associated
* with the intersection of two fuzzy sets having the membership function values
* membfunc1 and membfunc2.
******************************************************************************/
float intersection(membfunc1, membfunc2)


float membfunc1, membfunc2;
{
if (membfunc1 < membfunc2)
        return (membfunc1);
else
        return (membfunc2);
}


/******************************************************************************
* This function determines the value at a point p of the generic membership
* function determined by the parameters p1, p2, p3.
******************************************************************************/
float membfunc(p1, p2, p3, p)


float p1, p2, p3, p;
```

```
{
        float MF;
        if (p1<p2) goto NOT_SMALL;
        if (p<=p2)
                {
                MF=1;
                goto FINISH;
                }
        else
                goto DOWN;
NOT_SMALL: if (p<=p1)
                {
                MF=0;
                 goto FINISH;
                }
        if (p<p2)
                {
                MF=(p-p1)/(p2-p1);
                goto FINISH;
                }
DOWN:           if (p<p3)
                {
```

```
                MF=(p3-p)/(p3-p2);
                goto FINISH;
                }
        if (p3<p2)
                {
                MF=1;
                goto FINISH;
                }
        if (p>=p3)
                MF=0;
FINISH:         return (MF);
}


/*****************************************************************************/
* This function determines the value of the membership function of the
* intersection of two fuzzy sets having generic membership functions.
*****************************************************************************/
float fuzzy_and(membf1, membf2, pt1, pt2)
float membf1[4], membf2[4], pt1, pt2;


{
        int i;
```

```
          float par[4], memf1, memf2, MF;

for (i = 1; i < 4; i++)
          {
          par[i] = membf1[i];
          }
memf1 = membfunc(par[1], par[2], par[3], pt1);
for (i = 1; i < 4; i++)
          {
          par[i] = membf2[i];
          }
memf2 = membfunc(par[1], par[2], par[3], pt2);
MF = intersection(memf1, memf2);
return (MF);
}
/*****************************************************************************/
```

APPENDIX D

GW BASIC OBSTACLE AVOIDANCE PROGRAMS

APPENDIX D

GW BASIC OBSTACLE AVOIDANCE PROGRAMS

```
0 'PROGRAM TO AVOID SPHERICAL OBJECTS USING A CRISP ALGORITHM
10 'INITIALIZE THE COLLISION ANALYSIS STATUS LAST. DISPLAY ONLY WHEN LAST=1.
20 LAST=0
30 INPUT "USE PARAMETERS IN FILE Y(N)";ANS$
40 IF ANS$="Y" OR ANS$="y" THEN OPEN "I",1,"OBSTACLE.DAT" ELSE GOTO 80
50 'YES, SO GET THE PARAMETERS FROM THE FILE CALLED 'OBSTACLE.DAT'.
60 GOSUB 1000:GOTO 140
70 'NO, SO GET THE PARAMETERS DIRECTLY FROM THE KEYBOARD.
80 OPEN "O",1,"OBSTACLE.DAT"
90 'GET THE VEHICLE RADIUS AND ITS INITIAL AND DESTINATION (X,Y,Z) LOCATIONS.
100 GOSUB 1500
110 'GET THE NUMBER OF OBSTACLES, THEIR RADII, AND THEIR (X,Y,Z) LOCATIONS.
120 GOSUB 2000
130 'OPEN A FILE TO RECORD THE VEHICLE'S OBSTACLE AVOIDANCE PATH.
140 OPEN "O",2,"VCOORDS.DAT"
150 XV=XV0:YV=YV0:ZV=ZV0
160 'FIND THE OBSTACLE COLLISION POINTS ON THE VEHICLE PATH FROM ITS INITIAL
170 'LOCATION TO ITS DESTINATION AND DETERMINE THE EARLIEST ONE.
180 GOSUB 3500
190 'IF NO COLLISIONS OCCUR PRINT THAT MESSAGE AND QUIT.
200 IF MINI=0 THEN PRINT "NO COLLISIONS":PRINT:GOTO 690
210 'IF MINI>0 A COLLISION WITH OBSTACLE NUMBER MINI WILL OCCUR.
220 'FIND THE VEHICLE POSITION AT THE COLLISION POINT. IF THE COLLISION POINT
230 'IS FURTHER TO THE LEFT THAN THE INITIAL VEHICLE POSITION AN OBSTACLE IS
240 'TOO CLOSE TO THE VEHICLE POSITION. ANNOUNCE THE FACT AND QUIT. OTHERWISE
250 'CONTINUE.
260 GOSUB 4000:IF XA<XV0 THEN GOTO 9040 ELSE DXA=XA-XD:DYA=YA-YD
270 'IF NO MORE OBSTACLES ARE AHEAD MOVE AND DISPLAY THE VEHICLE.
280 IF LAST=1 THEN GOSUB 6500
290 'OTHERWISE EXAMINE THE OBSTACLES AHEAD. IF MORE THAN TWO OBSTACLES IN THE
300 'VEHICLE'S PATH ARE TOO CLOSE TOGETHER FOR THE VEHICLE TO PASS BETWEEN THEM
310 'THEN THE REMAINING CONFIGURATION OF OBSTACLES IS DIFFERENT FROM THE
320 'ORIGINAL ONE, SO REINITIALIZE THE VEHICLE POSITION AND START OVER.
330 GOSUB 4500:IF NC>1 THEN GOSUB 2500:GOTO 150
340 'IF NO TWO OBSTACLES IN THE VEHICLE'S PATH ARE TOO CLOSE TOGETHER FOR THE
350 'VEHICLE TO PASS BETWEEN THEM THEN CONTINUE WITH THE SAME OBSTACLE
360 'CONFIGURATION.
370 IF NC=0 THEN 460
380 'OTHERWISE CHOOSE THE BEST SIDE OF THE OBSTACLE FOR THE VEHICLE TO GO
390 'AROUND AND MOVE THE VEHICLE TO THE OBSTACLE AND AROUND IT ON THAT SIDE.
```

```
400 IF (DY(CI)>DYD1) OR (DY(CI)>DYD2) THEN 430
410 IF DYD2<DYD1 THEN XV=XD+DXD1:YV=YD+DYD1:GOTO 520
420 GOTO 500
430 IF (DY(CI)<DYD1) OR (DY(CI)<DYD2) THEN 460
440 IF DYD2>DYD1 THEN XV=XD+DXD1:YV=YD+DYD1:GOTO 520
450 GOTO 500
460 R21=SQR(DXD1^2+DYD1^2)+SQR((DXA-DXD1)^2+(DYA-DYD1)^2)
470 R22=SQR(DXD2^2+DYD2^2)+SQR((DXA-DXD2)^2+(DYA-DYD2)^2)
480 'GIVEN A CHOICE, CHOOSE THE SHORTEST DISTANCE AROUND THE OBSTACLE.
490 IF R21<R22 THEN XV=XD+DXD1:YV=YD+DYD1:GOTO 520
500 XV=XD+DXD2:YV=YD+DYD2
510 'MAKE SURE THE VEHICLE LOCATION IS TO THE RIGHT OF ITS INITIAL POSITION.
520 IF XV<XV0 THEN XV=XV0:YV=YV0
530 'MOVE THE VEHICLE AROUND THE OBSTACLE AND DISPLAY THE MOTION ON SCREEN.
540 GOSUB 7500
550 'IF THERE IS A COLLISION AT ALL CHECK FOR NEXT COLLISION.
560 IF MINI>0 THEN GOSUB 3500
570 'IF NO REMAINING COLLISION CAN OCCUR DISPLAY THE VEHICLE MOVE TO THE
580 'DESTINATION AND QUIT.
590 IF (LAST=1 AND MINI=0 AND XA<XD) THEN XA=XD:YA=YD:GOSUB 6500:GOTO 690
600 'IF NO COLLISION IS PREDICTED THE OBSTACLE CONFIGURATION HAS BEEN
610 'MODIFIED BY BEING COMBINED INTO A SINGLE OBSTACLE, SO START OVER.
620 IF (LAST=0 AND MINI=0) THEN LAST=1:GOTO 150
630 'IS THE CONDITION FOR COLLISION WITH OBSTACLE NUMBER MINI SATISFIED?
640 'IF SO REPEAT THE AVOIDANCE PROCESS WITH THE PRESENT OBSTACLE
650 'CONFIGURATION.
660 IF D(MINI)<= RSUM2(MINI) THEN 260
670 'IF NOT DISPLAY THE VEHICLE MOVE TO THE DESTINATION AND QUIT.
680 XA=XD:YA=YD:GOSUB 6500
690 IF INKEY$="" THEN 690
700 SCREEN 2:SCREEN 0:CLOSE #1
710 END
720 '
1000 'GET THE PARAMETERS FROM THE FILE.
1010 I=0:INPUT #1,XV0,YV0,ZV0,RV
1020 INPUT #1,XD,YD,ZD
1030 WHILE NOT(EOF(1))
1040 I=I+1
1050 INPUT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
1060 WEND
1070 'COUNT THE NUMBER OF OBSTACLES NO.
1080 NO=I:NOB=NO:NOB0=NOB
1090 'NOTE THAT THE DATA IS INPUT FROM A FILE.
```

```
1100 FIL$="I"
1110 'GET THE INITIAL DISPLAY OF THE VEHICLE AND OBSTACLES ON SCREEN.
1120 GOSUB 2100
1130 RETURN
1140 '
1500 'INPUT THE VEHICLE PARAMETERS FROM THE KEYBOARD.
1510 INPUT "VEHICLE INITIAL X,Y,Z COORDINATES";XV0,YV0,ZV0
1520 INPUT "VEHICLE RADIUS";RV
1530 INPUT "DESTINATION X,Y,Z COORDINATES";XD,YD,ZD
1540 'IF THE DATA COMES FROM A FILE THEN DON'T PRINT IT TO A FILE.
1550 IF FIL$="I" THEN 1570
1560 PRINT #1,XV0,YV0,ZV0,RV:PRINT #1,XD,YD,ZD
1570 RETURN
1580 '
2000 'GET THE OBSTACLE DATA FROM THE KEYBOARD.
2010 INPUT "NUMBER OF OBSTACLES";NO:NOB=NO:NOB0=NOB
2020 WHILE I<NO
2030 I=I+1
2040 PRINT "OBSTACLE #";I
2050 CH=0:INPUT "X,Y,Z COORDINATES";X(I),Y(I),Z(I)
2060 INPUT "RADIUS";R(I):IF I>1 THEN GOSUB 8500:IF CH=1 THEN 2050
2070 'SAVE THE PARAMETERS IN THE DESIGNATED FILE.
2080 PRINT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
2090 WEND
2100 'PRINT THE FIRST OBSTACLE DIAGRAM ON THE SCREEN.
2110 GOSUB 6000
2120 'ADD THE VEHICLE IN ITS INITIAL POSITION.
2130 XX=XV0:YY=YV0:RR=RV:GOSUB 7000
2140 IF INKEY$="" THEN 2140
2150 RETURN
2160 '
2500 'REASSIGN OBSTACLE NUMBERS AND PARAMETERS AFTER COMBINING 2 OBSTACLES.
2510 'ZERO THE OBSTACLE FLAG.  IF FEWER THAN 2 OBSTACLES REMAIN NO MORE
2520 'OBSTACLE COMBINING IS NEEDED.
2530 OB=0:IF NO<2 THEN 2780
2540 'IF MORE THAN 1 OBSTACLE IS PRESENT CHECK IF TWO OBSTACLES ARE CLOSE
2550 'ENOUGH TO BE COMBINED INTO ONE.  IF SO COMBINE THEM, SET FLAGS, UPDATE
2560 'THE OBJECT COUNTERS.
2570 FOR I=2 TO NO
2580 FOR J=1 TO I-1
2590 RHO2(I,J)=(X(I)-X(J))^2+(Y(I)-Y(J))^2+(Z(I)-Z(J))^2:RHO(I,J)=SQR(RHO2(I,J)):RIJ=R(I)+R(J)
2600 DRO(I,J)=RHO(I,J)-(RIJ+2*RV)
2610 IF DRO(I,J)<0 THEN GOSUB 3000:W(I)=-1:OB=1:J=I:I=NO+1
2620 NEXT
```

```
2630 NEXT
2640 'AFTER COMBINING TWO OBSTACLES REASIGN OBSTACLE NUMBERS, OR RETURN IF NONE
2650 'WERE COMBINED.
2660 IF OB=1 THEN J=0 ELSE GOTO 2860
2670 FOR I=1 TO NO
2680 'SKIP ITS NUMBER IF AN OBSTACLE IS REMOVED BY COMBINING IT WITH ANOTHER
2690 'OBSTACLE.
2700 IF W(I)=-1 THEN NOB=NOB-1:GOTO 2720
2710 J=J+1:X(J)=X(I):Y(J)=Y(I):Z(J)=Z(I):R(J)=R(I)
2720 NEXT
2730 'IF THE PRESENT NUMBER OF RECOGNIZED OBSTACLES AGREES WITH THE NUMBER
2740 'LEFT AFTER COMBINING PAIRS AND MORE THAN 1 IS LEFT, CHECK OUT THE NEW
2750 'CONFIGURATION.
2760 IF NOB=NO THEN 2850
2770 'IF ONE OR MORE OBSTACLES WERE REMOVED RESET THE NUMBER OF OBSTACLES.
2780 NO=NOB
2790 'GET THE DIAGRAM ON THE SCREEN IF THE LAST OBSTACLE HAS BEEN ENCOUNTERED.
2800 IF LAST=1 THEN GOSUB 6000:ELSE 2830
2810 'IF MORE THAN 1 OBSTACLE REMAINS OR NO MORE THAN 1 OBSTACLE EXISTED BEFORE
2820 'THE LATEST COMBINATION OF OBJECTS TOOK PLACE THEN RETURN.
2830 IF NO>1 THEN 2860
2840 'IF MORE THAN 1 OBSTACLE IS LEFT CHECK OUT THE NEW CONFIGURATION.
2850 IF NOB>1 THEN 2530
2860 RETURN
2870 '
3000 'GET THE POSITION COORDINATES AND RADIUS OF THE NEW OBSTACLE FORMED BY
3010 'COMBINING 2 OLD ONES.
3020 CD=2*RHO(I,J):FACI=(RHO(I,J)+R(I)-R(J))/CD:FACJ=(RHO(I,J)+R(J)-R(I))/CD
3030 X(J)=FACI*X(I)+FACJ*X(J):Y(J)=FACI*Y(I)+FACJ*Y(J):Z(J)=FACI*Z(I)+FACJ*Z(J)
3040 R(J)=(RHO(I,J)+R(I)+R(J))/2
3050 RETURN
3060 '
3500 'GET THE OBSTACLE COLLISION POINTS ON THE VEHICLE PATH TO ITS DESTINATION.
3510 'CALCULATE THE COMPONENTS OF THE VECTOR FROM THE VEHICLE'S INITIAL
3520 'LOCATION TO ITS DESTINATION.
3530 DXD=XV-XD:DYD=YV-YD:DZD=ZV-ZD
3540 'CALCULATE THE DIRECT VEHICLE PATH LENGTH SQUARED.
3550 DEN=DXD*DXD+DYD*DYD+DZD*DZD
3560 MIND=DEN:MINI=0
3570 FOR I=1 TO NO
3580 'CHECK IF VEHICLE IS HEADED TOWARD THE OBSTACLE. IF NOT SKIP THE COLLISION
3590 'TEST.
3600 IF .01+(X(I)-XV)*DXD+(Y(I)-YV)*DYD+(Z(I)-ZV)*DZD>=0 THEN 3770
3610 DX(I)=X(I)-XD:DY(I)=Y(I)-YD:DZ(I)=Z(I)-ZD
```

```
3620 DD(I)=DX(I)*DX(I)+DY(I)*DY(I)+DZ(I)*DZ(I)
3630 NUM=DXD*DX(I)+DYD*DY(I)+DZD*DZ(I):RSUM=RV+R(I):RSUM2(I)=RSUM*RSUM
3640 FAC=NUM/DEN:D(I)=DD(I)-FAC*NUM
3650 'D(I) IS THE SQUARE OF THE DISTANCE OF THE CENTER OF OBSTACLE I FROM
3660 'THE VEHICLE PATH.
3670 X0(I)=XD+FAC*DXD:Y0(I)=YD+FAC*DYD:Z0(I)=ZD+FAC*DZD
3680 'X0(I),Y0(I),Z0(I) ARE COORDINATES OF THE POINT ON THE VEHICLE PATH
3690 'CLOSEST TO THE CENTER OF OBSTACLE I.
3700 DIST(I)=(NUM-DEN)^2/DEN
3710 'DIST(I) IS THE SQUARED DISTANCE FROM THE INITIAL VEHICLE LOCATION
3720 'TO THE POINT (X0(I),Y0(I),Z0(I)).
3730 IF D(I)>=RSUM2(I) THEN 3770
3740 IF DIST(I)<MIND THEN MINI=I:MIND=DIST(I)
3750 'IF MIND IS A MINIMUM THEN THE COLLISION WITH OBSTACLE NUMBER MINI IS THE
3760 'EARLIEST.
3770 NEXT
3780 'IF MIND<DEN THEN THE EARLIEST COLLISION POINT IS AT 'X0(MINI),Y0(MINI),
3790 'Z0(MINI). GET DISTANCES BETWEEN OBJECT MINI AND ALL OTHER OBJECTS.
3800 FOR I=1 TO NO
3810 IF I=MINI GOTO 3830
3820
RHO2(MINI,I)=(X(I)-X(MINI))^2+(Y(I)-Y(MINI))^2+(Z(I)-Z(MINI))^2:RHO(MINI,I)=SQR(RHO2(MINI,I))
3830 NEXT
3840 RETURN
3850 '
4000 'GET THE LAST VEHICLE POSITION BEFORE WHICH COLLISION CAN BE AVOIDED.
4010 LB=RSUM2(MINI)-D(MINI)
4020 FAC2=SQR(LB/DEN)
4030 XA=X0(MINI)+FAC2*DXD:YA=Y0(MINI)+FAC2*DYD:ZA=Z0(MINI)+FAC2*DZD
4040 RETURN
4050 '
4500 'GET THE NEW VEHICLE POSITION AFTER GOING AROUND AN OBSTACLE TO AVOID
4510 'A COLLISION.
4520 'CHECK IF A COLLISION WILL OCCUR WHILE THE VEHICLE GOES AROUND AN
4530 'OBSTACLE. IF MORE THAN 1 SUCH COLLISION IS PREDICTED RETURN.
4540 GOSUB 5500:IF NC>1 THEN 4610
4550 'CALCULATE THE VEHICLE POSITION COORDINATE INCREMENTS IN GOING AROUND THE
4560 'OBSTACLE.
4570 L2=DD(MINI)-RSUM2(MINI)
4580 IF ABS(DX(MINI))>0 AND ABS(DY(MINI))>0 THEN GOSUB 5000:GOTO 4610
4590 IF ABS(DX(MINI))>ABS(DY(MINI)) THEN GOSUB 5300:GOTO 4610
4600 DYD1=L2/DY(MINI):DYD2=DYD1:DXD1=SGN(DXD)*SQR(L2-DYD1*DYD1):DXD2=-DXD1
4610 RETURN
4620 '
```

```
5000 DENOM=DD(MINI)-DZ(MINI)*DZ(MINI)
5010 TERM2=SGN(DXD)*DY(MINI)*SQR(L2*DENOM-L2*L2)
5020 TERM1=L2*DX(MINI):DXD1=(TERM1+TERM2)/DENOM:DXD2=(TERM1-TERM2)/DENOM
5030 DYD1=(L2-DX(MINI)*DXD1)/DY(MINI):DYD2=(L2-DX(MINI)*DXD2)/DY(MINI)
5040 RETURN
5050 '
5300 DXD1=L2/DX(MINI):DXD2=DXD1:DYD1=SGN(DYD)*SQR(L2-DXD1*DXD1):DYD2=-DYD1
5310 RETURN
5320 '
5500 'FIND THE OBSTACLE PAIRS IN THE VEHICLE PATH THAT ARE TOO CLOSE FOR THE
5510 'VEHICLE TO PASS BETWEEN THEM.
5520 RMV=R(MINI)+2*RV:NC=0
5530 FOR I=1 TO NO
5540 IF I=MINI THEN 5560
5550 IF RHO(MINI,I)<RMV+R(I) THEN NC=NC+1:CI=I
5560 NEXT
5570 RETURN
5580 '
6000 'GET THE BASIC SCREEN DIAGRAM.
6010 CLS:SCREEN 1
6020 LINE(XV0,200-YV0)-(XD,200-YD):CIRCLE(XD,200-YD),RV,1
6030 FOR C=1 TO NOB0
6040 CIRCLE(X1(C),200-Y1(C)),R1(C),,,,1
6050 NEXT
6060 RETURN
6070 '
6500 'DISPLAY THE VEHICLE MOVING TO THE COLLISION POINT AT AN OBSTACLE AND
6510 'RECORD THE VEHICLE PATH POSITION AND RADIUS AND OBSTACLE RADIUS IN THE
6520 'OUTPUT FILE.
6530 DXV=(XA-XV)/10:DYV=(YA-YV)/10
6540 FOR I=1 TO 10
6550 XX=XV:YY=YV:RR=RV
6560 IF LAST=0 THEN 6580 ELSE GOSUB 6000:GOSUB 7000:PRINT #2,XV,YV,RV,R(MINI)
6570 XV=XV+DXV:YV=YV+DYV:IF XV<XV0 THEN XV=XV0:YV=YV0
6580 'CHECK FOR A COLLISION
6590 GOSUB 8050
6600 GOSUB 6000:GOSUB 7000:PRINT #2,XV,YV,RV,R(MINI)
6610 NEXT
6620 XX=XV:YY=YV:PRINT #2,XV,YV,RV,R(MINI)
6630 GOSUB 6000:GOSUB 7000:PRINT #2,XV,YV,RV,R(MINI)
6640 RETURN
6650 '
7000 'DRAW THE VEHICLE.
7010 CIRCLE(XX,200-YY),RR,2,,,,1
```

```
7020 RETURN
7030 '
7500 ' MOVE THE VEHICLE AROUND AN OBSTACLE.
7510 DXX=XX-X(MINI):DYY=YY-Y(MINI):DXV=XV-X(MINI):DYV=YV-Y(MINI)
7520 ROB=SQR((X(MINI)-XA)^2+(Y(MINI)-YA)^2)
7530 'CALCULATE THE VEHICLES ANGULAR POSITION RELATIVE TO THE OBSTACLE.
7540 AB=DXX:ORD=DYY:GOSUB 8000:TH1=TH0
7550 AB=DXV:ORD=DYV:GOSUB 8000:TH2=TH0
7560 DTH=(TH2-TH1)/5:TH=TH1
7570 FOR Q=1 TO 5
7580 TH=TH+DTH
7590 XX=X(MINI)+ROB*COS(TH):YY=Y(MINI)+ROB*SIN(TH)
7600 'THEN REFRESH SCREEN DISPLAY.
7610 IF LAST=0 THEN 7620 ELSE GOSUB 6000:GOSUB 7000:PRINT #2,XX,YY,RV,R(MINI)
7620 NEXT
7630 IF LAST=0 THEN 7650
7640 'IF INKEY$="" THEN 7130
7650 RETURN
7660 '
8000 'CALCULATE THE ANGLE.
8010 TH0=ATN(ORD/AB)
8020 IF SGN(AB)=-1 THEN TH0=TH0+3.1415926#:GOTO 8040
8030 IF SGN(ORD)=-1 THEN TH0=TH0+6.2831852#
8040 RETURN
8050 'CHECK FOR A COLLISION WITH ANY OBSTACLE.
8060 FOR OBNUM=1 TO NO
8070 DXY=(XV-X1(OBNUM))^2+(YV-Y1(OBNUM))^2+(ZV-Z1(OBNUM))^2:DRR=(RV+R1(OBNUM))^2
8080 IF .01+DXY<DRR THEN 9000
8090 NEXT
8100 RETURN
8110 '
8500 IF SQR((X(I)-X(I-1))^2+(Y(I)-Y(I-1))^2+(Z(I)-Z(I-1))^2)<R(I)+R(I-1) THEN CH=1:PRINT "OBJECT TOO
          CLOSE TO THE LAST ONE."
8510 RETURN
8520 '
9000 SCREEN 2:SCREEN 0:PRINT "COLLISION!!!":PRINT
          XV;YV;ZV,X1(OBNUM);Y1(OBNUM);Z1(OBNUM),DXY,DRR
9010 PRINT #2,-1,-1,DXY,DRR:PRINT #2,I, X1(I),Y1(I):PRINT #2,"COLLISION!!!"
9020 IF INKEY$="" THEN 9020
9030 XV0=XA:YV0=YA:GOTO 9080
9040 SCREEN 2:SCREEN 0:PRINT "OBSTACLES TOO CLOSE TO VEHICLE STARTING POINT.
9050 PRINT "COLLISION OCCURS AT X =";XA;", Y =";YA;", Z =";ZA;"."
9060 IF INKEY$="" THEN 9060
9070 GOSUB 6000:GOTO 9110
```

```
9080 FOR C=1 TO NOBO
9090 CIRCLE(X(C),200-Y(C)),R(C)
9100 NEXT
9110 XX=XV0:YY=YV0:RR=RV:GOSUB 7000
9120 IF INKEY$="" THEN 9120
9130 SCREEN 2:SCREEN
```

```
0 'PROGRAM TO AVOID CYLINDRICAL OBJECTS USING A CRISP ALGORITHM
10 'INITIALIZE THE COLLISION ANALYSIS STATUS LAST. DISPLAY ONLY WHEN LAST=1.
20 LAST=0
30 INPUT "USE PARAMETERS IN FILE Y(N)";ANS$
40 IF ANS$="Y" OR ANS$="y" THEN OPEN "I",1,"OBSTACLE.DAT" ELSE GOTO 80
50 'YES, SO GET THE PARAMETERS FROM THE FILE CALLED 'OBSTACLE.DAT'.
60 GOSUB 1000:GOTO 140
70 'NO, SO GET THE PARAMETERS DIRECTLY FROM THE KEYBOARD.
80 OPEN "O",1,"OBSTACLE.DAT"
90 'GET THE VEHICLE RADIUS AND ITS INITIAL AND DESTINATION (X,Y,Z) LOCATIONS.
100 GOSUB 1500
110 'GET THE NUMBER OF OBSTACLES, THEIR RADII, AND THEIR (X,Y,Z) LOCATIONS.
120 GOSUB 2000
130 'OPEN A FILE TO RECORD THE VEHICLE'S OBSTACLE AVOIDANCE PATH.
140 OPEN "O",2,"VCOORDS.DAT"
150 XV=XV0:YV=YV0:ZV=ZV0
160 'FIND THE OBSTACLE COLLISION POINTS ON THE VEHICLE PATH FROM ITS INITIAL
170 'LOCATION TO ITS DESTINATION AND DETERMINE THE EARLIEST ONE.
180 GOSUB 3500
190 'IF NO COLLISIONS OCCUR PRINT THAT MESSAGE AND QUIT.
200 IF MINI=0 THEN PRINT "NO COLLISIONS":PRINT:GOTO 690
210 'IF MINI>0 A COLLISION WITH OBSTACLE NUMBER MINI WILL OCCUR.
220 'FIND THE VEHICLE POSITION AT THE COLLISION POINT. IF THE COLLISION POINT
230 'IS FURTHER TO THE LEFT THAN THE INITIAL VEHICLE POSITION AN OBSTACLE IS
240 'TOO CLOSE TO THE VEHICLE POSITION. ANNOUNCE THE FACT AND QUIT. OTHERWISE
250 'CONTINUE.
260 GOSUB 4000:IF XA<XV0 THEN GOTO 9040 ELSE DXA=XA-XD:DYA=YA-YD
270 'IF NO MORE OBSTACLES ARE AHEAD MOVE AND DISPLAY THE VEHICLE.
280 IF LAST=1 THEN GOSUB 6500
290 'OTHERWISE EXAMINE THE OBSTACLES AHEAD. IF MORE THAN TWO OBSTACLES IN THE
300 'VEHICLE'S PATH ARE TOO CLOSE TOGETHER FOR THE VEHICLE TO PASS BETWEEN THEM
310 'THEN THE REMAINING CONFIGURATION OF OBSTACLES IS DIFFERENT FROM THE
320 'ORIGINAL ONE, SO REINITIALIZE THE VEHICLE POSITION AND START OVER.
330 GOSUB 4500:IF NC>1 THEN GOSUB 2500:GOTO 150
340 'IF NO TWO OBSTACLES IN THE VEHICLE'S PATH ARE TOO CLOSE TOGETHER FOR THE
350 'VEHICLE TO PASS BETWEEN THEM THEN CONTINUE WITH THE SAME OBSTACLE
```

```
360 'CONFIGURATION.
370 IF NC=0 THEN 460
380 'OTHERWISE CHOOSE THE BEST SIDE OF THE OBSTACLE FOR THE VEHICLE TO GO
390 'AROUND AND MOVE THE VEHICLE TO THE OBSTACLE AND AROUND IT ON THAT SIDE.
400 IF (DY(CI)>DYD1) OR (DY(CI)>DYD2) THEN 430
410 IF DYD2<DYD1 THEN XV=XD+DXD1:YV=YD+DYD1:GOTO 520
420 GOTO 500
430 IF (DY(CI)<DYD1) OR (DY(CI)<DYD2) THEN 460
440 IF DYD2>DYD1 THEN XV=XD+DXD1:YV=YD+DYD1:GOTO 520
450 GOTO 500
460 R21=SQR(DXD1^2+DYD1^2)+SQR((DXA-DXD1)^2+(DYA-DYD1)^2)
470 R22=SQR(DXD2^2+DYD2^2)+SQR((DXA-DXD2)^2+(DYA-DYD2)^2)
480 'GIVEN A CHOICE, CHOOSE THE SHORTEST DISTANCE AROUND THE OBSTACLE.
490 IF R21<R22 THEN XV=XD+DXD1:YV=YD+DYD1:GOTO 520
500 XV=XD+DXD2:YV=YD+DYD2
510 'MAKE SURE THE VEHICLE LOCATION IS TO THE RIGHT OF ITS INITIAL POSITION.
520 IF XV<XV0 THEN XV=XV0:YV=YV0
530 'MOVE THE VEHICLE AROUND THE OBSTACLE AND DISPLAY THE MOTION ON SCREEN.
540 GOSUB 7500
550 'IF THERE IS A COLLISION AT ALL CHECK FOR NEXT COLLISION.
560 IF MINI>0 THEN GOSUB 3500
570 'IF NO REMAINING COLLISION CAN OCCUR DISPLAY THE VEHICLE MOVE TO THE
580 'DESTINATION AND QUIT.
590 IF (LAST=1 AND MINI=0 AND XA<XD) THEN XA=XD:YA=YD:GOSUB 6500:GOTO 690
600 'IF NO COLLISION IS PREDICTED THE OBSTACLE CONFIGURATION HAS BEEN
610 'MODIFIED BY BEING COMBINED INTO A SINGLE OBSTACLE, SO START OVER.
620 IF (LAST=0 AND MINI=0) THEN LAST=1:GOTO 150
630 'IS THE CONDITION FOR COLLISION WITH OBSTACLE NUMBER MINI SATISFIED?
640 'IF SO REPEAT THE AVOIDANCE PROCESS WITH THE PRESENT OBSTACLE
650 'CONFIGURATION.
660 IF D(MINI)<= RSUM2(MINI) THEN 260
670 'IF NOT DISPLAY THE VEHICLE MOVE TO THE DESTINATION AND QUIT.
680 XA=XD:YA=YD:GOSUB 6500
690 IF INKEY$="" THEN 690
700 SCREEN 2:SCREEN 0:CLOSE #1
710 END
720 '
1000 'GET THE PARAMETERS FROM THE FILE.
1010 I=0:INPUT #1,XV0,YV0,ZV0,RV
1020 INPUT #1,XD,YD,ZD
1030 WHILE NOT(EOF(1))
1040 I=I+1
1050 INPUT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):R1(I)=R(I)
1060 WEND
```

```
1070 'COUNT THE NUMBER OF OBSTACLES NO.
1080 NO=I:NOB=NO:NOB0=NOB
1090 'NOTE THAT THE DATA IS INPUT FROM A FILE.
1100 FIL$="I"
1110 'GET THE INITIAL DISPLAY OF THE VEHICLE AND OBSTACLES ON SCREEN.
1120 GOSUB 2100
1130 RETURN
1140 '
1500 'INPUT THE VEHICLE PARAMETERS FROM THE KEYBOARD.
1510 INPUT "VEHICLE INITIAL X,Y,Z COORDINATES";XV0,YV0,ZV0
1520 INPUT "VEHICLE RADIUS";RV
1530 INPUT "DESTINATION X,Y,Z COORDINATES";XD,YD,ZD
1540 'IF THE DATA COMES FROM A FILE THEN DONT PRINT IT TO A FILE.
1550 IF FIL$="I" THEN 1570
1560 PRINT #1,XV0,YV0,ZV0,RV:PRINT #1,XD,YD,ZD
1570 RETURN
1580 '
2000 'GET THE OBSTACLE DATA FROM THE KEYBOARD.
2010 INPUT "NUMBER OF OBSTACLES";NO:NOB=NO:NOB0=NOB
2020 WHILE I<NO
2030 I=I+1
2040 PRINT "OBSTACLE #";I
2050 CH=0:INPUT "X,Y,Z COORDINATES";X(I),Y(I),Z(I)
2060 INPUT "RADIUS";R(I):IF I>1 THEN GOSUB 8500:IF CH=1 THEN 2050
2070 'SAVE THE PARAMETERS IN THE DESIGNATED FILE.
2080 PRINT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
2090 WEND
2100 'PRINT THE FIRST OBSTACLE DIAGRAM ON THE SCREEN.
2110 GOSUB 6000
2120 'ADD THE VEHICLE IN ITS INITIAL POSITION.
2130 XX=XV0:YY=YV0:RR=RV:GOSUB 7000
2140 IF INKEY$="" THEN 2140
2150 RETURN
2160 '
2500 'REASSIGN OBSTACLE NUMBERS AND PARAMETERS AFTER COMBINING 2 OBSTACLES.
2510 'ZERO THE OBSTACLE FLAG.  IF FEWER THAN 2 OBSTACLES REMAIN NO MORE
2520 'OBSTACLE COMBINING IS NEEDED.
2530 OB=0:IF NO<2 THEN 2780
2540 'IF MORE THAN 1 OBSTACLE IS PRESENT CHECK IF TWO OBSTACLES ARE CLOSE
2550 'ENOUGH TO BE COMBINED INTO ONE.  IF SO COMBINE THEM, SET FLAGS, UPDATE
2560 'THE OBJECT COUNTERS.
2570 FOR I=2 TO NO
2580 FOR J=1 TO I-1
2590 RHO2(I,J)=(X(I)-X(J))^2+(Y(I)-Y(J))^2:RHO(I,J)=SQR(RHO2(I,J)):RIJ=R(I)+R(J)
```

```
2600 DRO(I,J)=RHO(I,J)-(RIJ+2*RV)
2610 IF DRO(I,J)<0 THEN GOSUB 3000:W(I)=-1:OB=1:J=I:I=NO+1
2620 NEXT
2630 NEXT
2640 'AFTER COMBINING TWO OBSTACLES REASIGN OBSTACLE NUMBERS, OR RETURN IF NONE
2650 'WERE COMBINED.
2660 IF OB=1 THEN J=0 ELSE GOTO 2860
2670 FOR I=1 TO NO
2680 'SKIP ITS NUMBER IF AN OBSTACLE IS REMOVED BY COMBINING IT WITH ANOTHER
2690 'OBSTACLE.
2700 IF W(I)=-1 THEN NOB=NOB-1:GOTO 2720
2710 J=J+1:X(J)=X(I):Y(J)=Y(I):R(J)=R(I)
2720 NEXT
2730 'IF THE PRESENT NUMBER OF RECOGNIZED OBSTACLES AGREES WITH THE NUMBER
2740 'LEFT AFTER COMBINING PAIRS AND MORE THAN 1 IS LEFT, CHECK OUT THE NEW
2750 'CONFIGURATION.
2760 IF NOB=NO THEN 2850
2770 'IF ONE OR MORE OBSTACLES WERE REMOVED RESET THE NUMBER OF OBSTACLES.
2780 NO=NOB
2790 'GET THE DIAGRAM ON THE SCREEN IF THE LAST OBSTACLE HAS BEEN ENCOUNTERED.
2800 IF LAST=1 THEN GOSUB 6000:ELSE 2830
2810 'IF MORE THAN 1 OBSTACLE REMAINS OR NO MORE THAN 1 OBSTACLE EXISTED BEFORE
2820 'THE LATEST COMBINATION OF OBJECTS TOOK PLACE THEN RETURN.
2830 IF NO>1 THEN 2860
2840 'IF MORE THAN 1 OBSTACLE IS LEFT CHECK OUT THE NEW CONFIGURATION.
2850 IF NOB>1 THEN 2530
2860 RETURN
2870 '
3000 'GET THE POSITION COORDINATES AND RADIUS OF THE NEW OBSTACLE FORMED BY
3010 'COMBINING 2 OLD ONES.
3020 CD=2*RHO(I,J):FACI=(RHO(I,J)+R(I)-R(J))/CD:FACJ=(RHO(I,J)+R(J)-R(I))/CD
3030 X(J)=FACI*X(I)+FACJ*X(J):Y(J)=FACI*Y(I)+FACJ*Y(J)
3040 R(J)=(RHO(I,J)+R(I)+R(J))/2
3050 RETURN
3060 '
3500 'GET THE OBSTACLE COLLISION POINTS ON THE VEHICLE PATH TO ITS DESTINATION.
3510 'CALCULATE THE COMPONENTS OF THE VECTOR FROM THE VEHICLE'S INITIAL
3520 'LOCATION TO ITS DESTINATION.
3530 DXD=XV-XD:DYD=YV-YD
3540 'CALCULATE THE DIRECT VEHICLE PATH LENGTH SQUARED.
3550 DEN=DXD*DXD+DYD*DYD
3560 MIND=DEN:MINI=0
3570 FOR I=1 TO NO
3580 'CHECK IF VEHICLE IS HEADED TOWARD THE OBSTACLE. IF NOT SKIP THE COLLISION
```

```
3590 'TEST.
3600 IF .01+(X(I)-XV)*DXD+(Y(I)-YV)*DYD>=0 THEN 3770
3610 DX(I)=X(I)-XD:DY(I)=Y(I)-YD
3620 DD(I)=DX(I)*DX(I)+DY(I)*DY(I)
3630 NUM=DXD*DX(I)+DYD*DY(I):RSUM=RV+R(I):RSUM2(I)=RSUM*RSUM
3640 FAC=NUM/DEN:D(I)=DD(I)-FAC*NUM
3650 'D(I) IS THE SQUARE OF THE DISTANCE OF THE CENTER OF OBSTACLE I FROM
3660 'THE VEHICLE PATH.
3670 X0(I)=XD+FAC*DXD:Y0(I)=YD+FAC*DYD
3680 'X0(I),Y0(I) ARE COORDINATES OF THE POINT ON THE VEHICLE PATH
3690 'CLOSEST TO THE CENTER OF OBSTACLE I.
3700 DIST(I)=(NUM-DEN)^2/DEN
3710 'DIST(I) IS THE SQUARED DISTANCE FROM THE INITIAL VEHICLE LOCATION
3720 'TO THE POINT (X0(I),Y0(I)).
3730 IF D(I)>=RSUM2(I) THEN 3770
3740 IF DIST(I)<MIND THEN MINI=I:MIND=DIST(I)
3750 'IF MIND IS A MINIMUM THEN THE COLLISION WITH OBSTACLE NUMBER MINI IS THE
3760 'EARLIEST.
3770 NEXT
3780 'IF MIND<DEN THEN THE EARLIEST COLLISION POINT IS AT 'X0(MINI),Y0(MINI).
3790 'GET DISTANCES BETWEEN OBJECT MINI AND ALL OTHER OBJECTS.
3800 FOR I=1 TO NO
3810 IF I=MINI GOTO 3830
3820 RHO2(MINI,I)=(X(I)-X(MINI))^2+(Y(I)-Y(MINI))^2:RHO(MINI,I)=SQR(RHO2(MINI,I))
3830 NEXT
3840 RETURN
3850 '
4000 'GET THE LAST VEHICLE POSITION BEFORE WHICH COLLISION CAN BE AVOIDED.
4010 LB=RSUM2(MINI)-D(MINI)
4020 FAC2=SQR(LB/DEN)
4030 XA=X0(MINI)+FAC2*DXD:YA=Y0(MINI)+FAC2*DYD
4040 RETURN
4050 '
4500 'GET THE NEW VEHICLE POSITION AFTER GOING AROUND AN OBSTACLE TO AVOID
4510 'A COLLISION.
4520 'CHECK IF A COLLISION WILL OCCUR WHILE THE VEHICLE GOES AROUND AN
4530 'OBSTACLE.  IF MORE THAN 1 SUCH COLLISION IS PREDICTED RETURN.
4540 GOSUB 5500:IF NC>1 THEN 4610
4550 'CALCULATE THE VEHICLE POSITION COORDINATE INCREMENTS IN GOING AROUND THE
4560 'OBSTACLE.
4570 L2=DD(MINI)-RSUM2(MINI)
4580 IF ABS(DX(MINI))>0 AND ABS(DY(MINI))>0 THEN GOSUB 5000:GOTO 4610
4590 IF ABS(DX(MINI))>ABS(DY(MINI)) THEN GOSUB 5300:GOTO 4610
4600 DYD1=L2/DY(MINI):DYD2=DYD1:DXD1=SGN(DXD)*SQR(L2-DYD1*DYD1):DXD2=-DXD1
```

```
4610 RETURN
4620 '
5000 DENOM=DD(MINI)
5010 TERM2=SGN(DXD)*DY(MINI)*SQR(L2*DENOM-L2*L2)
5020 TERM1=L2*DX(MINI):DXD1=(TERM1+TERM2)/DENOM:DXD2=(TERM1-TERM2)/DENOM
5030 DYD1=(L2-DX(MINI)*DXD1)/DY(MINI):DYD2=(L2-DX(MINI)*DXD2)/DY(MINI)
5040 RETURN
5050 '
5300 DXD1=L2/DX(MINI):DXD2=DXD1:DYD1=SGN(DYD)*SQR(L2-DXD1*DXD1):DYD2=-DYD1
5310 RETURN
5320 '
5500 'FIND THE OBSTACLE PAIRS IN THE VEHICLE PATH THAT ARE TOO CLOSE FOR THE
5510 'VEHICLE TO PASS BETWEEN THEM.
5520 RMV=R(MINI)+2*RV:NC=0
5530 FOR I=1 TO NO
5540 IF I=MINI THEN 5560
5550 IF RHO(MINI,I)<RMV+R(I) THEN NC=NC+1:CI=I
5560 NEXT
5570 RETURN
5580 '
6000 'GET THE BASIC SCREEN DIAGRAM.
6010 CLS:SCREEN 1
6020 LINE(XV0,200-YV0)-(XD,200-YD):CIRCLE(XD,200-YD),RV,1
6030 FOR C=1 TO NOBO
6040 CIRCLE(X1(C),200-Y1(C)),R1(C),,,,1
6050 NEXT
6060 RETURN
6070 '
6500 'DISPLAY THE VEHICLE MOVING TO THE COLLISION POINT AT AN OBSTACLE AND
6510 'RECORD THE VEHICLE PATH POSITION AND RADIUS AND OBSTACLE RADIUS IN THE
6520 'OUTPUT FILE.
6530 DXV=(XA-XV)/10:DYV=(YA-YV)/10
6540 FOR I=1 TO 10
6550 XX=XV:YY=YV:RR=RV
6560 IF LAST=0 THEN 6580 ELSE GOSUB 6000:GOSUB 7000:PRINT #2,XV,YV,RV,R(MINI)
6570 XV=XV+DXV:YV=YV+DYV:IF XV<XV0 THEN XV=XV0:YV=YV0
6580 'CHECK FOR A COLLISION
6590 GOSUB 8050
6600 GOSUB 6000:GOSUB 7000:PRINT #2,XV,YV,RV,R(MINI)
6610 NEXT
6620 XX=XV:YY=YV:PRINT #2,XV,YV,RV,R(MINI)
6630 GOSUB 6000:GOSUB 7000:PRINT #2,XV,YV,RV,R(MINI)
6640 RETURN
6650 '
```

```
7000 'DRAW THE VEHICLE.
7010 CIRCLE(XX,200-YY),RR,2,,,1
7020 RETURN
7030 '
7500 ' MOVE THE VEHICLE AROUND AN OBSTACLE.
7510 DXX=XX-X(MINI):DYY=YY-Y(MINI):DXV=XV-X(MINI):DYV=YV-Y(MINI)
7520 ROB=SQR((X(MINI)-XA)^2+(Y(MINI)-YA)^2)
7530 'CALCULATE THE VEHICLES ANGULAR POSITION RELATIVE TO THE OBSTACLE.
7540 AB=DXX:ORD=DYY:GOSUB 8000:TH1=TH0
7550 AB=DXV:ORD=DYV:GOSUB 8000:TH2=TH0
7560 DTH=(TH2-TH1)/5:TH=TH1
7570 FOR Q=1 TO 5
7580 TH=TH+DTH
7590 XX=X(MINI)+ROB*COS(TH):YY=Y(MINI)+ROB*SIN(TH)
7600 'THEN REFRESH SCREEN DISPLAY.
7610 IF LAST=0 THEN 7620 ELSE GOSUB 6000:GOSUB 7000:PRINT #2,XX,YY,RV,R(MINI)
7620 NEXT
7630 IF LAST=0 THEN 7650
7640 'IF INKEY$="" THEN 7130
7650 RETURN
7660 '
8000 'CALCULATE THE ANGLE.
8010 TH0=ATN(ORD/AB)
8020 IF SGN(AB)=-1 THEN TH0=TH0+3.1415926#:GOTO 8040
8030 IF SGN(ORD)=-1 THEN TH0=TH0+6.2831852#
8040 RETURN
8050 'CHECK FOR A COLLISION WITH ANY OBSTACLE.
8060 FOR OBNUM=1 TO NO
8070 DXY=(XV-X1(OBNUM))^2+(YV-Y1(OBNUM))^2:DRR=(RV+R1(OBNUM))^2
8080 IF .01+DXY<DRR THEN 9000
8090 NEXT
8100 RETURN
8110 '
8500 IF SQR((X(I)-X(I-1))^2+(Y(I)-Y(I-1))^2<R(I)+R(I-1) THEN CH=1:PRINT"OBJECT TOO CLOSE TO THE
        LAST ONE."
8510 RETURN
8520 '
9000 SCREEN 2:SCREEN 0:PRINT "COLLISION!!!":PRINT XV;YV,X1(OBNUM);Y1(OBNUM),DXY,DRR
9010 PRINT #2,-1,-1,DXY,DRR:PRINT #2,I, X1(I),Y1(I):PRINT #2,"COLLISION!!!"
9020 IF INKEY$="" THEN 9020
9030 XV0=XA:YV0=YA:GOTO 9080
9040 SCREEN 2:SCREEN 0:PRINT "OBSTACLES TOO CLOSE TO VEHICLE STARTING POINT.
9050 PRINT "COLLISION OCCURS AT X =";XA;", Y =";YA;"."
9060 IF INKEY$="" THEN 9060
```

```
9070 GOSUB 6000:GOTO 9110
9080 FOR C=1 TO NOBO
9090 CIRCLE(X(C),200-Y(C)),R(C)
9100 NEXT
9110 XX=XV0:YY=YV0:RR=RV:GOSUB 7000
9120 IF INKEY$="" THEN 9120
9130 SCREEN 2:SCREEN 0:STOP
```

```
0 'PROGRAM TO AVOID SPHERICAL OBSTACLES USING A FUZZY ALGORITHM
10 INPUT "USE OBSTACLE PARAMETERS IN FILE Y(N)";ANS$
20 IF ANS$="Y" OR ANS$="y" THEN OPEN "I",1,"OBSTACLE.DAT" ELSE GOTO 60
30 'YES, SO GET PARAMETERS FROM FILE
40 GOSUB 1000:GOTO 110
50 'NO, SO GET PARAMETERS DIRECTLY FROM KEYBOARD
60 OPEN "O",1,"OBSTACLE.DAT"
70 'GET VEHICLE RADIUS AND ITS INITIAL AND DESTINATION (X,Y,Z) LOCATIONS
80 GOSUB 2000
90 'GET NUMBER OF OBSTACLES, THEIR RADII, THEIR (X,Y,Z) LOCATIONS
100 GOSUB 2500
110 INPUT "USE MEMBERSHIP FUNCTION PARAMETERS IN FILE Y(N)";ANS$
120 IF ANS$="Y" OR ANS$="y" THEN GOSUB 1500:GOTO 150
130 'NO SO ENTER PARAMETERS WITH KEYBOARD
140 OPEN "O",2,"MF.DAT":GOSUB 3000
150 FOR I=1 TO NO
160 PRINT "OBSTACLE #";I;": X =";X1(I),"Y =";Y1(I),"Z =";Z1(I)
170 NEXT
180 IF INKEY$="" THEN 180
190 'START WITH VEHICLE IN INITIAL POSITION, INITIALIZE CONFIGURATION STATE
200 'AND OPEN FILE TO RECORD VEHICLE PATH
210 XV=XV0:YV=YV0:ZV=ZV0:FIRST=1:LAST=0:OPEN "O",3,"VCOORDS.DAT"
220 CLS:INPUT "BASIC INCREMENT FOR VEHICLE MOTION ALONG PATH";DD
230 'GET NUMBER OF DISCREET MEMBERSHIP FUNCTION VALUES FOR FUZZY SET 'RATIO'
240 'THAT DEFINES THE DIRECTION OF THE VEHICLE MOTION AT A POINT
250 INPUT "NUMBER OF RATIO VALUES";NRAT:DRAT=RAT(1,3,2)/NRAT:GOTO 420
260 'GET DISTANCE OF VEHICLE FROM DESTINATION
270 DXD=XD-XV:DYD=YD-YV
280 DXYD2=DXD*DXD+DYD*DYD:MIND=DXYD2:DXYD=SQR(DXYD2)
290 'IF THE VEHICLE IS CLOSER TO THE DESTINATION THAN THE LENGTH OF ITS OWN
300 'RADIUS LET THE VEHICLE GO STRAIGHT TO THE DESTINATION
310 IF DXYD<RV THEN XV=XD:YV=YD:DEST$="DEST":GOTO 420
320 'IF NOT THEN FIND THE NEAREST OBSTACLE, WHETHER IT IS THE LAST ONE, AND IF
330 'IT IS NOT GET THE DIRECTION ORTHOGONAL TO THE VEHICLE'S LINE OF SIGHT TO
340 'THE OBSTACLE
```

```
350 GOSUB 3500:IF LAST=1 THEN 390
360 'IF THE NEAREST OBSTACLE IS NOT THE LAST GET THE INCREMENT FACTOR FOR THE
370 'VEHICLE MOTION ORTHOGONAL TO THAT DIRECTION
380 GOSUB 4000
390 'CALCULATE NEW VEHICLE POSITION AND SET A FLAG INDICATING THAT IT IS NO
400 'LONGER THE INITIAL POSITION
410 GOSUB 4500:IF FIRST=1 THEN FIRST=0
420 'UPDATE SCREEN AND SAVE LAST VEHICLE POSITION AND DIRECTION
430 GOSUB 9000:XX=XV:YY=YV
440 GOSUB 6000:GOSUB 6500:LXV=XV:LYV=YV:LALPH=ALPHA:LBET=BETA
450 IF FIRST=0 THEN 490
460 'FREEZE THE SCREEN CONFIGURATION BEFORE THE VEHICLE STARTS TO MOVE
470 IF INKEY$="" THEN 470
480 'IF NOT AT DESTINATION GET VEHICLE POSITION UPDATE
490 IF DEST$<>"DEST" THEN 270
500 IF INKEY$="" THEN 500
510 SCREEN 2:SCREEN 0
520 END
530 '
1000 'GET OBSTACLE PARAMETERS FROM FILE
1010 I=0:INPUT #1,XV0,YV0,ZV0,RV
1020 INPUT #1,XD,YD,ZD
1030 WHILE NOT(EOF(1))
1040 I=I+1
1050 INPUT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
1060 WEND
1070 NO=I
1080 RETURN
1090 '
1500 'GET MEMBERSHIP FUNCTION PARAMETERS FROM FILE
1510 OPEN "I",2,"MF.DAT"
1520 FOR I=1 TO 3
1530 FOR J=1 TO 3
1540 INPUT #2,RAD(I,J)
1550 NEXT
1560 NEXT
1570 FOR I=1 TO 3
1580 FOR J=1 TO 3
1590 INPUT #2,DIS(I,J)
1600 NEXT
1610 NEXT
1620 FOR I=1 TO 3
1630 FOR J=1 TO 3
1640 FOR K=1 TO 3
```

```
1650 INPUT #2,RAT(I,J,K)
1660 NEXT
1670 NEXT
1680 NEXT
1690 RETURN
1700 '
2000 'GET INITIAL VEHICLE DATA
2010 INPUT "VEHICLE INITIAL X,Y,Z COORDINATES";XV0,YV0,ZV0
2020 INPUT "VEHICLE RADIUS";RV
2030 INPUT "DESTINATION X,Y,Z COORDINATES";XD,YD,ZD
2040 PRINT #1,XV0,YV0,ZV0,RV:PRINT #1,XD,YD,ZD
2050 RETURN
2060 '
2500 'GET INITIAL OBSTACLE DATA
2510 INPUT "NUMBER OF OBSTACLES";NO
2520 WHILE I<NO
2530 I=I+1
2540 PRINT "OBSTACLE #";I
2550 INPUT "X,Y,Z COORDINATES";X(I),Y(I),Z(I)
2560 INPUT "RADIUS";R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
2570 'IF FILE IS IN INPUT MODE DON'T WRITE IN IT
2580 'IF FIL$="I" THEN 2100
2590 'NEW DATA AND WRITE ONLY FILE MODE SO PRINT PARAMETERS IN FILE
2600 PRINT #1,X(I),Y(I),Z(I),R(I)
2610 WEND
2620 RETURN
2630 '
3000 'GET FUZZY SET PARAMETERS FOR DEFINING SMALL, MEDIUM, OR LARGE FOR THE
3010 'RADIUS OF AN OBSTACLE AND FOR THE DISTANCE OF AN OBSTACLE FROM A VEHICLE
3020 'AND THE PARAMETERS ASSOCIATED WITH THE MEMBERSHIP FUNCTIONS OF THE MATRIX
3030 'ELEMENTS IN THE MATRIX DETERMINED BY THE RULES OF THE FORM 'IF THE
3040 'OBSTACLE RADIUS IS '* AND THE OBSTACLE DISTANCE IS * THEN THE RATIO IS '
3050 PRINT "3 PARAMETERS DEFINING THE MEMBERSHIP FUNCTIONS:":PRINT
3060 PRINT "'SMALL OBSTACLE RADIUS'--":INPUT
      "SRAD1,SRAD2,SRAD3";RAD(1,1),RAD(1,2),RAD(1,3)
3070 WRITE #2,RAD(1,1),RAD(1,2),RAD(1,3)
3080 PRINT "'MEDIUM OBSTACLE RADIUS'--":INPUT
      "MRAD1,MRAD2,MRAD3";RAD(2,1),RAD(2,2),RAD(2,3)
3090 WRITE #2,RAD(2,1),RAD(2,2),RAD(2,3)
3100 PRINT "'LARGE OBSTACLE. RADIUS'--":INPUT
      "LRAD1,LRAD2,LRAD3";RAD(3,1),RAD(3,2),RAD(3,3)
3110 WRITE #2,RAD(3,1),RAD(3,2),RAD(3,3)
3120 PRINT "'SMALL OBST. DISTANCE'--":INPUT "SDIS1,SDIS2,SDIS3";DIS(1,1),DIS(1,2),DIS(1,3)
3130 WRITE #2,DIS(1,1),DIS(1,2),DIS(1,3)
```

```
3140 PRINT "MEDIUM OBST. DISTANCE'--":INPUT "MDIS1,MDIS2,MDIS3";DIS(2,1),DIS(2,2),DIS(2,3)
3150 WRITE #2,DIS(2,1),DIS(2,2),DIS(2,3)
3160 PRINT "LARGE OBST. DISTANCE'--":INPUT "LDIS1,LDIS2,LDIS3";DIS(3,1),DIS(3,2),DIS(3,3)
3170 WRITE #2,DIS(3,1),DIS(3,2),DIS(3,3)
3180 PRINT "RATIO IF 'SMALL OBSTACLE DISTANCE' AND:"
3190 PRINT "'SMALL OBSTACLE RADIUS'--":INPUT
     "SDSR1,SDSR2,SDSR3";RAT(1,1,1),RAT(1,1,2),RAT(1,1,3)
3200 WRITE #2,RAT(1,1,1),RAT(1,1,2),RAT(1,1,3)
3210 PRINT "'MEDIUM OBSTACLE RADIUS'--":INPUT
     "SDMR1,SDMR2,SDMR3";RAT(1,2,1),RAT(1,2,2),RAT(1,2,3)
3220 WRITE #2,RAT(1,2,1),RAT(1,2,2),RAT(1,2,3)
3230 PRINT "'LARGE OBSTACLE RADIUS'--":INPUT
     "SDLR1,SDLR2,SDLR3";RAT(1,3,1),RAT(1,3,2),RAT(1,3,3)
3240 WRITE #2,RAT(1,3,1),RAT(1,3,2),RAT(1,3,3)
3250 PRINT "RATIO IF 'MEDIUM OBSTACLE DISTANCE' AND:"
3260 PRINT "'SMALL OBSTACLE RADIUS'--":INPUT
     "MDSR1,MDSR2,MDSR3";RAT(2,1,1),RAT(2,1,2),RAT(2,1,3)
3270 WRITE #2,RAT(2,1,1),RAT(2,1,2),RAT(2,1,3)
3280 PRINT "'MEDIUM OBSTACLE RADIUS'--":INPUT
     "MDMR1,MDMR2,MDMR3";RAT(2,2,1),RAT(2,2,2),RAT(2,2,3)
3290 WRITE #2,RAT(2,2,1),RAT(2,2,2),RAT(2,2,3)
3300 PRINT "'LARGE OBSTACLE RADIUS'--":INPUT
     "MDLR1,MDLR2,MDLR3";RAT(2,3,1),RAT(2,3,2),RAT(2,3,3)
3310 WRITE #2,RAT(2,3,1),RAT(2,3,2),RAT(2,3,3)
3320 PRINT "RATIO IF 'LARGE OBSTACLE DISTANCE' AND:"
3330 PRINT "SMALL OBSTACLE RADIUS'--":INPUT
     "LDSR1,LDSR2,LDSR3";RAT(3,1,1),RAT(3,1,2),RAT(3,1,3)
3340 WRITE #2,RAT(3,1,1),RAT(3,1,2),RAT(3,1,3)
3350 PRINT "MEDIUM OBSTACLE RADIUS'--":INPUT
     "LDMR1,LDMR2,LDMR3";RAT(3,2,1),RAT(3,2,2),RAT(3,2,3)
3360 WRITE #2,RAT(3,2,1),RAT(3,2,2),RAT(3,2,3)
3370 PRINT "LARGE OBSTACLE RADIUS'--":INPUT
     "LDLR1,LDLR2,LDLR3";RAT(3,3,1),RAT(3,3,2),RAT(3,3,3)
3380 WRITE #2,RAT(3,3,1),RAT(3,3,2),RAT(3,3,3)
3390 RETURN
3400 '
3500 'GET DIRECTION ORTHOGONAL TO LINE OF SIGHT FROM VEHICLE TO NEAREST
3510 'OBSTACLE
3520 'FIRST GET NEAREST OBSTACLE AND OBSTACLE WITH MAXIMUM RADIUS
3530 FOR I=1 TO NO
3540 DX(I)=X(I)-XV:DY(I)=Y(I)-YV:DZ(I)=Z(I)-ZV
3550 DXY(I)=DX(I)*DX(I)+DY(I)*DY(I)+DZ(I)*DZ(I):DRR(I)=(R(I)+RV)^2
3560 IF DXY(I)<MIND THEN MINI=I:MIND=DXY(I)
3570 IF R(I)>R(MAXI) THEN MAXI=I
```

```
3580 IF LAST=1 THEN 3620
3590 'IF NOT ALREADY KNOWN TO BE THE LAST OBSTACLE CHECK WHETHER IT IS OR NOT
3600 'AND SET -1 FLAG IF IT IS NOT BY VIRTUE OF ITS DIRECTION
3610 IF DX(I)*DXD+DY(I)*DYD>0 THEN LAST=-1
3620 NEXT
3630 'IF FLAG NOT SET THEN OBSTACLE MUST BE THE LAST
3640 IF LAST=-1 THEN LAST=0 ELSE LAST=1
3650 'CHECK IF THE NEAREST OBSTACLE IS LAST BECAUSE THE VEHICLE IS CLOSER TO
3660 'THE DESTINATION THAN TO THE OBSTACLE AND QUIT IF IT IS
3670 IF DXY(MINI)>DXYD2 THEN LAST=1:GOTO 3770
3680 'GET UNIT VECTOR IN ORTHOGONAL DIRECTION SLANTED TOWARD DESTINATION
3690 'OR, IF ORTHOGONAL TO DESTINATION DIRECTION, AWAY FROM BIGGEST OBSTACLE
3700 IF DY(MINI)<>0 THEN 3730
3710 IF DY(MAXI)<>0 THEN ALPHA=0:BETA=-SGN(DY(MAXI)):GOTO 3770
3720 BETA=1:GOTO 3770
3730 TH=-ATN(DX(MINI)/DY(MINI))
3740 ALPHA=COS(TH):BETA=SIN(TH)
3750 DOTP=ALPHA*DXD+BETA*DYD
3760 IF DOTP<0 THEN ALPHA=-ALPHA:BETA=-BETA
3770 RETURN
3780 '
4000 'GET FUZZY SET 'RATIO' OF MAGNITUDE OF VECTOR IN (ALPHA,BETA) DIRECTION TO
4010 'INCREMENT DD OF VEHICLE MOVEMENT.
4020 'FIRST GET OBSTACLE DISTANCE FROM VEHICLE AND RADIUS
4030 ODIS=SQR(DXY(MINI)):ORAD=R(MINI)
4040 'THEN GET 'RATIO' MEMBERSHIP VALUES AND DEFUZZIFY
4050 RATI=0:NSUM=0:DSUM=0
4060 WHILE RATI<=RAT(1,3,2)
4070 GOSUB 7000
4080 NSUM=NSUM+RATI*MAXMF:DSUM=DSUM+MAXMF
4090 RATI=RATI+DRAT
4100 WEND
4110 IF NSUM=0 THEN RATIO=0 ELSE RATIO=NSUM/DSUM
4120 RETURN
4130 '
4500 'GET NEW VEHICLE POSITION
4510 IF LAST=1 THEN 4580
4520 DD1=RATIO*DD
4530 'IF NEAREST OBSTACLE CHANGES SPECIAL HANDLING REQUIRED
4540 IF MINI<>LMINI AND FIRST=0 THEN GOSUB 5000
4550 XV=XV+DD1*ALPHA:YV=YV+DD1*BETA
4560 IF DIRCH=1 THEN 4620
4570 'GET DIRECTION OF DESTINATION FROM VEHICLE AND INCREMENT VEHICLE POSITION
```

```
4580 DXD=XD-XV:DYD=YD-YV:DXYD=SQR(DXD*DXD+DYD*DYD):DXD0=DXD/DXYD:DYD0=DYD/DX
     YD
4590 'ALONG THAT DIRECTION
4600 XV=XV+DD*DXD0:YV=YV+DD*DYD0
4610 'GET DISTANCE OF VEHICLE IN NEW POSITION FROM NEAREST OBSTACLE
4620 DXY(MINI)=(X(MINI)-XV)^2+(Y(MINI)-YV)^2+(Z(MINI)-ZV)^2
4630 'RECORD THIS DATA IN VEHICLE PATH FILE ALONG WITH THE SUM OF THE VEHICLE
4640 'AND OBSTACLE RADII
4650 PRINT #3,XV,YV,DXY(MINI),DRR(MINI)
4660 'IF THE DISTANCE BETWEEN THE VEHICLE AND THE NEAREST OBSTACLE IS GREATER
4670 'THAN THE SUM OF THE RADII THEN THERE IS NO COLLISION
4680 IF DXY(MINI)>DRR(MINI) THEN GOTO 4720
4690 'OTHERWISE THERE IS A COLLISION THAT MUST BE AVOIDED SO REVERSE THE
4700 'DIRECTION OF THE VEHICLE MOTION AND TRY AGAIN
4710 XV=LXV:YV=LYV:ALPHA=-LALPH:BETA=-LBET:GOTO 4550
4720 RETURN
4730 '
5000 'IF NORMAL DIRECTION BRINGS VEHICLE CLOSER TO FORMER NEAREST OBSTACLE
5010 'CHANGE DIRECTION UNLESS LAST OBSTACLE IS NOT TOO CLOSE TO PRESENT ONE
5020 IF LMINI=0 THEN 5150
5030 SIDE1=R(LMINI)+RV:SIDE2=R(LMINI)-RV:SIDE3=R(MINI)+RV:SIDE4=R(MINI)-RV
5040 ANG1=ATN(SIDE2/SQR(SIDE1^2-SIDE2^2)):ANG2=ATN(SIDE4/SQR(SIDE3^2-SIDE4^2))
5050 ANG=3.1415926#-(ANG1+ANG2):SIDE5=SIDE1^2+SIDE3^2-2*SIDE1*SIDE3*COS(ANG)
5060 DMINI=(X(LMINI)-X(MINI))^2+(Y(LMINI)-Y(MINI))^2+(Z(LMINI)-Z(MINI))^2
5070 'IF THERE IS ROOM TO SPARE THEN CONTINUE
5080 IF SIDE5<.8*DMINI THEN 5150
5090 'IF NOT THEN REVERSE DIRECTION AND TRY AGAIN
5100 TXV=XV:TYV=YV
5110 TXV=TXV+DD1*ALPHA:TYV=TYV+DD1*BETA:TDXD=XD-TXV:TDYD=YD-TYV
5120 TDXYD=SQR(TDXD*TDXD+TDYD*TDYD):TDXD0=TDXD/TDXYD:TDYD0=TDYD/TDXYD
5130 TXV=TXV+DD*TDXD0:TYV=TYV+DD*TDYD0
5140 IF (TXV-X(LMINI))^2+(TYV-Y(LMINI))^2<(XV-X(LMINI))^2+(YV-Y(LMINI))^2 THEN
       ALPHA=-ALPHA:BETA=-BETA:DIRCH=1:GOTO 5160
5150 LMINI=MINI:DIRCH=0
5160 RETURN
5170 '
5500 'COLLISION OCCURRED, SO ANNOUNCE IT ON THE SCREEN AND QUIT
5510 SCREEN 2:SCREEN 0:LOCATE 12,30:PRINT "COLLISION!!!"
5520 PRINT #3,-1,-1,DXY(I),DRR(I):PRINT #3,I, X(I),Y(I):PRINT #3,"COLLISION!!!"
5530 IF INKEY$="" THEN 5530
5540 XX=XV:YY=YV:GOSUB 6000:GOSUB 6500
5550 IF INKEY$=" THEN 5550
5560 SCREEN 2:SCREEN 0:STOP
5570 '
```

```
6000 'DRAW BASIC ORIGINAL OBSTACLES, VEHICLE PATH AND DESTINATION ON SCREEN
6010 CLS:SCREEN 1
6020 LINE(XV0,200-YV0)-(XD,200-YD):CIRCLE(XD,200-YD),RV,1
6030 FOR C=1 TO NO
6040 CIRCLE(X(C),200-Y(C)),R(C),,,,1
6050 NEXT
6060 RETURN
6070 '
6500 'DRAW VEHICLE POSITION ON SCREEN
6510 CIRCLE(XX,200-YY),RV,2,,,1
6520 RETURN
6530 '
7000 'PERFORM COMPOSITION OF RULES FOR CALCULATING RATIO
7010 'GET MEMBERSHIP FUNCTION OF THE INTERSECTION OF 'OBSTACLE RADIUS' AND
7020 ''DISTANCE FROM VEHICLE' FOR ALL COMBINATIONS OF RADIUS AND DISTANCE SIZES
7030 MAXMF=0
7040 FOR DSIZE=1 TO 3
7050 'GET OBSTACLE DISTANCE FROM VEHICLE MEMBERSHIP FUNCTION VALUE MF1
7060 P1=DIS(DSIZE,1):P2=DIS(DSIZE,2):P3=DIS(DSIZE,3):P=ODIS:GOSUB 7500
7070 MF1=MF
7080 'GET OBSTACLE RADIUS MEMBERSHIP FUNCTION VALUE MF2
7090 FOR RSIZE=1 TO 3
7100 P1=RAD(RSIZE,1):P2=RAD(RSIZE,2):P3=RAD(RSIZE,3):P=ORAD:GOSUB 7500
7110 MF2=MF
7120 'GET INTERSECTION MEMBERSHIP FUNCTION VALUE MFDR
7130 IF MF1<MF2 THEN MF=MF1 ELSE MF=MF2
7140 MFDR=MF
7150 'GET MEMBERSHIP FUNCTION VALUE FOR RATIO AS CARTESIAN PRODUCT OF 'OBSTACLE
7160 'DISTANCE' AND 'OBSTACLE RADIUS' ACCORDING TO EACH RULE AND THEN THE UNION
7170 'OF ALL RULES
7180 P1=RAT(DSIZE,RSIZE,1):P2=RAT(DSIZE,RSIZE,2):P3=RAT(DSIZE,RSIZE,3):P=RATI
7190 GOSUB 7500
7200 IF MF<MFDR THEN MFRDR=MF ELSE MFRDR=MFDR
7210 IF MFRDR>MAXMF THEN MAXMF=MFRDR
7220 NEXT
7230 NEXT
7240 RETURN
7250 '
7500 'GENERIC MEMBERSHIP FUNCTION MF(P1,P2,P3)
7510 IF P1<P2 THEN 7530
7520 IF P<=P2 THEN MF=1:GOTO 7580 ELSE GOTO 7550
7530 IF P<=P1 THEN MF=0:GOTO 7580
7540 IF P<P2 THEN MF=(P-P1)/(P2-P1):GOTO 7580
7550 IF P<P3 THEN MF=(P3-P)/(P3-P2):GOTO 7580
```

```
7560 IF P3<P2 THEN MF=1:GOTO 7580
7570 IF P>=P3 THEN MF=0
7580 RETURN
9000 FOR I=1 TO NO
9010 IF DXY(I)<DRR(I) THEN 5500
9020 NEXT
9030 RETURN
```

```
0 'PROGRAM TO AVOID CYLINDRICAL OBSTACLES USING A FUZZY ALGORITHM
10 INPUT "USE OBSTACLE PARAMETERS IN FILE Y(N)";ANS$
20 IF ANS$="Y" OR ANS$="y" THEN OPEN "I",1,"OBSTACLE.DAT" ELSE GOTO 60
30 'IF 'YES', GET PARAMETERS FROM THE FILE 'OBSTACLE.DAT'.
40 GOSUB 1000:GOTO 110
50 'IF 'NO', GET PARAMETERS DIRECTLY FROM THE KEYBOARD AND STORE IN THE FILE:
60 OPEN "O",1,"OBSTACLE.DAT"
70 'GET THE VEHICLE RADIUS AND ITS INITIAL AND DESTINATION (X,Y,Z) LOCATIONS;
80 GOSUB 2000
90 'GET THE NUMBER OF OBSTACLES, THEIR RADII, AND THEIR (X,Y,Z) LOCATIONS.
100 GOSUB 2500
110 INPUT "USE MEMBERSHIP FUNCTION PARAMETERS IN FILE Y(N)";ANS$
120 'IF 'YES', GET PARAMETERS FROM THE FILE 'MF.DAT'.
130 IF ANS$="Y" OR ANS$="y" THEN GOSUB 1500:GOTO 160
140 'IF 'NO', GET PARAMETERS DIRECTLY FROM THE KEYBOARD AND STORE IN THE FILE.
150 OPEN "O",2,"MF.DAT":GOSUB 2620
160 FOR I=1 TO NO
170 PRINT "OBSTACLE #";I;": X =";X1(I),"Y =";Y1(I),"Z =";Z1(I)
180 NEXT
190 IF INKEY$="" THEN 190
200 'START WITH THE VEHICLE IN ITS INITIAL POSITION, INITIALIZE THE
210 'CONFIGURATION STATE, AND OPEN A FILE TO RECORD THE VEHICLE PATH.
220 XV=XV0:YV=YV0:ZV=ZV0:FIRST=1:LAST=0:OPEN "O",3,"VCOORDS.DAT"
230 CLS:INPUT "BASIC INCREMENT FOR VEHICLE MOTION ALONG PATH";DD
240 'GET THE NUMBER OF DISCREET MEMBERSHIP FUNCTION VALUES FOR THE FUZZY SET
250 'RATIO' THAT DEFINES THE CORRECTIVE CHANGE IN THE VEHICLE MOTION DIRECTION.
260 INPUT "NUMBER OF RATIO VALUES";NRAT:DRAT=RAT(1,3,2)/NRAT:GOTO 430
270 'GET THE DISTANCE OF THE VEHICLE FROM THE DESTINATION.
280 DXD=XD-XV:DYD=YD-YV
290 DXYD2=DXD*DXD+DYD*DYD:MIND=DXYD2:DXYD=SQR(DXYD2)
300 'IF THE VEHICLE IS CLOSER TO THE DESTINATION THAN THE LENGTH OF ITS OWN
310 'RADIUS, LET THE VEHICLE GO STRAIGHT TO THE DESTINATION.
320 IF DXYD<RV THEN XV=XD:YV=YD:DEST$="DEST":GOTO 430
330 'IF NOT THEN FIND THE NEAREST OBSTACLE AND WHETHER IT IS THE LAST ONE.  IF
340 'IT IS NOT GET THE DIRECTION ORTHOGONAL TO THE VEHICLE'S LINE OF SIGHT TO
```

```
350 'THE OBSTACLE .
360 GOSUB 3500:IF LAST=1 THEN 400
370 'IF THE NEAREST OBSTACLE IS NOT THE LAST GET THE INCREMENT FACTOR FOR THE
380 'VEHICLE MOTION ORTHOGONAL TO THAT DIRECTION.
390 GOSUB 4000
400 'CALCULATE THE NEW VEHICLE POSITION AND SET A FLAG INDICATING THAT IT IS NO
410 'LONGER THE INITIAL POSITION.
420 GOSUB 4500:IF FIRST=1 THEN FIRST=0
430 'UPDATE THE SCREEN AND SAVE THE LAST VEHICLE POSITION AND DIRECTION.
440 GOSUB 8010:XX=XV:YY=YV
450 GOSUB 6000:GOSUB 6500:LXV=XV:LYV=YV:LALPH=ALPHA:LBET=BETA
460 IF FIRST=0 THEN 500
470 'FREEZE THE SCREEN CONFIGURATION BEFORE THE VEHICLE STARTS TO MOVE.
480 IF INKEY$="" THEN 480
490 'IF IT IS NOT AT THE DESTINATION GET THE VEHICLE POSITION UPDATE.
500 IF DEST$<>"DEST" THEN 280
510 IF INKEY$="" THEN 510
520 SCREEN 2:SCREEN 0
530 END
540 '
1000 'GET THE OBSTACLE PARAMETERS FROM THE FILE 'OBSTACLE.DAT'.
1010 I=0:INPUT #1,XV0,YV0,ZV0,RV
1020 INPUT #1,XD,YD,ZD
1030 WHILE NOT(EOF(1))
1040 I=I+1
1050 INPUT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
1060 WEND
1070 NO=I
1080 RETURN
1090 '
1500 'GET THE MEMBERSHIP FUNCTION PARAMETERS FROM THE FILE 'MF.DAT'.
1510 OPEN "I",2,"MF.DAT"
1520 FOR I=1 TO 3
1530 FOR J=1 TO 3
1540 INPUT #2,RAD(I,J)
1550 NEXT
1560 NEXT
1570 FOR I=1 TO 3
1580 FOR J=1 TO 3
1590 INPUT #2,DIS(I,J)
1600 NEXT
1610 NEXT
1620 FOR I=1 TO 3
1630 FOR J=1 TO 3
```

```
1640 FOR K=1 TO 3
1650 INPUT #2,RAT(I,J,K)
1660 NEXT
1670 NEXT
1680 NEXT
1690 RETURN
1700 '
2000 'GET THE INITIAL VEHICLE DATA FROM THE KEYBOARD.
2010 INPUT "VEHICLE INITIAL X,Y,Z COORDINATES";XV0,YV0,ZV0
2020 INPUT "VEHICLE RADIUS";RV
2030 INPUT "DESTINATION X,Y,Z COORDINATES";XD,YD,ZD
2040 'RECORD THE DATA IN THE FILE 'OBSTACLE.DAT'.
2050 PRINT #1,XV0,YV0,ZV0,RV:PRINT #1,XD,YD,ZD
2060 RETURN
2070 '
2500 'GET THE INITIAL OBSTACLE DATA.
2510 INPUT "NUMBER OF OBSTACLES";NO
2520 WHILE I<NO
2530 I=I+1
2540 PRINT "OBSTACLE #";I
2550 INPUT "X,Y,Z COORDINATES";X(I),Y(I),Z(I)
2560 INPUT "RADIUS";R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
2570 'RECORD THE DATA IN THE FILE 'OBSTACLE.DAT.
2580 PRINT #1,X(I),Y(I),Z(I),R(I)
2590 WEND
2600 RETURN
2610 '
2620 'GET FUZZY SET PARAMETERS FOR DEFINING A 'SMALL', 'MEDIUM', OR 'LARGE'
2630 'RADIUS OF AN OBSTACLE AND DISTANCE OF AN OBSTACLE FROM A VEHICLE.  ALSO,
2640 'GET THE PARAMETERS ASSOCIATED WITH THE MEMBERSHIP FUNCTIONS OF THE MATRIX
2650 'ELEMENTS IN THE MATRIX DETERMINED BY THE RULES OF THE FORM 'IF THE
2660 'OBSTACLE RADIUS IS * AND THE OBSTACLE DISTANCE IS * THEN THE RATIO IS *'.
2670 PRINT "3 PARAMETERS DEFINING THE MEMBERSHIP FUNCTIONS:":PRINT
2680 PRINT "'SMALL OBSTACLE RADIUS'--":INPUT
     "SRAD1,SRAD2,SRAD3";RAD(1,1),RAD(1,2),RAD(1,3)
2690 WRITE #2,RAD(1,1),RAD(1,2),RAD(1,3)
2700 PRINT "'MEDIUM OBSTACLE RADIUS'--":INPUT
     "MRAD1,MRAD2,MRAD3";RAD(2,1),RAD(2,2),RAD(2,3)
2710 WRITE #2,RAD(2,1),RAD(2,2),RAD(2,3)
2720 PRINT "'LARGE OBSTACLE. RADIUS'--":INPUT
     "LRAD1,LRAD2,LRAD3";RAD(3,1),RAD(3,2),RAD(3,3)
2730 WRITE #2,RAD(3,1),RAD(3,2),RAD(3,3)
2740 PRINT "'SMALL OBST. DISTANCE'--":INPUT "SDIS1,SDIS2,SDIS3";DIS(1,1),DIS(1,2),DIS(1,3)
2750 WRITE #2,DIS(1,1),DIS(1,2),DIS(1,3)
```

```
2760 PRINT "MEDIUM OBST. DISTANCE'--":INPUT "MDIS1,MDIS2,MDIS3";DIS(2,1),DIS(2,2),DIS(2,3)
2770 WRITE #2,DIS(2,1),DIS(2,2),DIS(2,3)
2780 PRINT "LARGE OBST. DISTANCE'--":INPUT "LDIS1,LDIS2,LDIS3";DIS(3,1),DIS(3,2),DIS(3,3)
2790 WRITE #2,DIS(3,1),DIS(3,2),DIS(3,3)
2800 PRINT "RATIO IF 'SMALL OBSTACLE DISTANCE' AND:"
2810 PRINT "SMALL OBSTACLE RADIUS'--":INPUT
        "SDSR1,SDSR2,SDSR3";RAT(1,1,1),RAT(1,1,2),RAT(1,1,3)
2820 WRITE #2,RAT(1,1,1),RAT(1,1,2),RAT(1,1,3)
2830 PRINT "MEDIUM OBSTACLE RADIUS'--":INPUT
        "SDMR1,SDMR2,SDMR3";RAT(1,2,1),RAT(1,2,2),RAT(1,2,3)
2840 WRITE #2,RAT(1,2,1),RAT(1,2,2),RAT(1,2,3)
2850 PRINT "LARGE OBSTACLE RADIUS'--":INPUT
        "SDLR1,SDLR2,SDLR3";RAT(1,3,1),RAT(1,3,2),RAT(1,3,3)
2860 WRITE #2,RAT(1,3,1),RAT(1,3,2),RAT(1,3,3)
2870 PRINT "RATIO IF 'MEDIUM OBSTACLE DISTANCE' AND:"
2880 PRINT "SMALL OBSTACLE RADIUS'--":INPUT
        "MDSR1,MDSR2,MDSR3";RAT(2,1,1),RAT(2,1,2),RAT(2,1,3)
2890 WRITE #2,RAT(2,1,1),RAT(2,1,2),RAT(2,1,3)
2900 PRINT "MEDIUM OBSTACLE RADIUS'--":INPUT
        "MDMR1,MDMR2,MDMR3";RAT(2,2,1),RAT(2,2,2),RAT(2,2,3)
2910 WRITE #2,RAT(2,2,1),RAT(2,2,2),RAT(2,2,3)
2920 PRINT "LARGE OBSTACLE RADIUS'--":INPUT
        "MDLR1,MDLR2,MDLR3";RAT(2,3,1),RAT(2,3,2),RAT(2,3,3)
2930 WRITE #2,RAT(2,3,1),RAT(2,3,2),RAT(2,3,3)
2940 PRINT "RATIO IF 'LARGE OBSTACLE DISTANCE' AND:"
2950 PRINT "SMALL OBSTACLE RADIUS'--":INPUT
        "LDSR1,LDSR2,LDSR3";RAT(3,1,1),RAT(3,1,2),RAT(3,1,3)
2960 WRITE #2,RAT(3,1,1),RAT(3,1,2),RAT(3,1,3)
2970 PRINT "MEDIUM OBSTACLE RADIUS'--":INPUT
        "LDMR1,LDMR2,LDMR3";RAT(3,2,1),RAT(3,2,2),RAT(3,2,3)
2980 WRITE #2,RAT(3,2,1),RAT(3,2,2),RAT(3,2,3)
2990 PRINT "LARGE OBSTACLE RADIUS'--":INPUT
        "LDLR1,LDLR2,LDLR3";RAT(3,3,1),RAT(3,3,2),RAT(3,3,3)
3000 WRITE #2,RAT(3,3,1),RAT(3,3,2),RAT(3,3,3)
3010 RETURN
3020 '
3500 'GET THE DIRECTION ORTHOGONAL TO THE LINE OF SIGHT FROM THE VEHICLE TO THE
3510 'NEAREST OBSTACLE. FIRST GET THE NEAREST OBSTACLE.
3520 FOR I=1 TO NO
3530 DX(I)=X(I)-XV:DY(I)=Y(I)-YV
3540 DXY(I)=DX(I)*DX(I)+DY(I)*DY(I):DRR(I)=(R(I)+RV)^2
3550 IF DXY(I)<MIND THEN MINI=I:MIND=DXY(I)
3560 'IF IT IS KNOWN THAT THE LAST OBSTACLE IS AT HAND SKIP THE DIRECTION TEST.
3570 IF LAST=1 THEN 3620
```

```
3580 'IF NOT ALREADY KNOWN TO BE THE LAST OBSTACLE, CHECK WHETHER THE OBSTACLE
3590 'IS OR NOT THE LAST BY VIRTUE OF ITS DIRECTION AND SET THE -1 FLAG IF IT
3600 'IS NOT.
3610 DDOT(I)= DX(I)*DXD+DY(I)*DYD:IF DDOT(I)>0 THEN LAST=-1
3620 NEXT
3630 'CALCULATE THE DISTANCE BETWEEN THE VEHICLE AND THE NEAREST OBSTACLE.
3640 ODIS=SQR(DXY(MINI))
3650 'IF THE -1 FLAG IS NOT SET THEN THE OBSTACLE MUST BE THE LAST ONE.
3660 IF LAST=-1 THEN LAST=0 ELSE LAST=1
3670 'CHECK IF THE NEAREST OBSTACLE IS THE LAST BECAUSE THE VEHICLE IS CLOSER
3680 'TO THE DESTINATION THAN TO THE OBSTACLE AND QUIT IF IT IS.
3690 IF DXY(MINI)>DXYD2 THEN LAST=1:GOTO 3780
3700 'GET THE UNIT VECTOR (ALPHA,BETA) IN THE DIRECTION ORTHOGONAL TO THE PATH
3710 'FROM THE VEHICLE TO THE NEAREST OBSTACLE CENTER, SLANTED TOWARD THE
3720 'DESTINATION.
3730 ALPHA=-DY(MINI)/ODIS:BETA=DX(MINI)/ODIS
3740 'CHECK TO DETERMINE WHETHER THE SLANT OF THE UNIT VECTOR IS CORRECT.
3750 DOTP=ALPHA*DXD+BETA*DYD
3760 'IF NOT REVERSE ITS SIGN.
3770 IF DOTP<0 THEN ALPHA=-ALPHA:BETA=-BETA
3780 RETURN
3790 '
4000 'GET THE RATIO OF THE MAGNITUDE OF THE VECTOR IN THE (ALPHA,BETA)
4010 'DIRECTION TO THE INCREMENT DD OF THE VEHICLE MOVEMENT.  FIRST GET THE
4020 'RADIUS ORAD OF THE NEAREST OBSTACLE.
4030 ORAD=R(MINI)
4040 'THEN GET THE FUZZY SET 'RATIO' MEMBERSHIP VALUES AND DEFUZZIFY THE RATIO.
4050 RATI=0:NSUM=0:DSUM=0
4060 WHILE RATI<=RAT(1,3,2)
4070 GOSUB 7000
4080 NSUM=NSUM+RATI*MAXMF:DSUM=DSUM+MAXMF
4090 RATI=RATI+DRAT
4100 WEND
4110 IF NSUM=0 THEN RATIO=0 ELSE RATIO=NSUM/DSUM
4120 RETURN
4130 '
4500 'GET THE NEW VEHICLE POSITION.
4510 IF LAST=1 THEN 4610
4520 'CALCULATE THE SIZE OF THE CORRECTION TO THE VEHICLE COORDINATES.
4530 DD1=RATIO*DD
4540 'IF THE NEAREST OBSTACLE CHANGES, A NEW CALCULATION OF THE CORRECTIVE
4550 'DIRECTION VECTOR (ALPHA,BETA) IS NEEDED.
4560 IF MINI<>LMINI AND FIRST=0 THEN GOSUB 5000
4570 'INCREMENT THE VEHICLE COORDINATES BY THE RESULTING CORRECTIONS.
```

```
4580 XV=XV+DD1*ALPHA:YV=YV+DD1*BETA
4590 IF DIRCH=1 THEN 4670
4600 'GET THE DESTINATION DIRECTION RELATIVE TO THE VEHICLE.
4610 DXD=XD-XV:DYD=YD-YV:DXYD=SQR(DXD*DXD+DYD*DYD):DXD0=DXD/DXYD:DYD0=DYD/DX
     YD
4620 'INCREMENT THE VEHICLE POSITION ALONG THAT DIRECTION BY THE INITIALLY
4630 'SELECTED MAGNITUDE DD.
4640 XV=XV+DD*DXD0:YV=YV+DD*DYD0
4650 'GET THE DISTANCE OF THE VEHICLE IN THE NEW POSITION FROM THE NEAREST
4660 'OBSTACLE.
4670 DXY(MINI)=(X(MINI)-XV)^2+(Y(MINI)-YV)^2
4680 'RECORD THIS DATA IN THE VEHICLE PATH FILE ALONG WITH THE SUM OF THE
4690 'VEHICLE AND OBSTACLE RADII.  IF THE DISTANCE BETWEEN THE VEHICLE AND THE
4700 'NEAREST OBSTACLE IS GREATER THAN THE SUM OF THE RADII THEN THERE IS NO
4710 'COLLISION.
4720 IF DXY(MINI)>DRR(MINI) THEN GOTO 4760
4730 'OTHERWISE THERE IS A COLLISION THAT MUST BE AVOIDED SO REVERSE THE
4740 'DIRECTION OF THE VEHICLE MOTION AND TRY AGAIN.
4750 XV=LXV:YV=LYV:ALPHA=-LALPH:BETA=-LBET:GOTO 4580
4760 PRINT #3,XV,YV,DXY(MINI),DRR(MINI)
4770 RETURN
4780 '
5000 'IF NORMAL DIRECTION BRINGS VEHICLE CLOSER TO FORMER NEAREST OBSTACLE
5010 'CHANGE DIRECTION UNLESS LAST OBSTACLE IS NOT TOO CLOSE TO PRESENT ONE
5020 IF LMINI=0 THEN 5130
5030 OBSEP=(R(LMINI)+R(MINI)+2*RV)^2
5040 DMINI=(X(LMINI)-X(MINI))^2+(Y(LMINI)-Y(MINI))^2
5050 'IF THERE IS ROOM TO SPARE THEN CONTINUE
5060 IF OBSEP<DMINI THEN 5130
5070 'IF NOT THEN REVERSE DIRECTION AND TRY AGAIN
5080 TXV=XV:TYV=YV
5090 TXV=TXV+DD1*ALPHA:TYV=TYV+DD1*BETA:TDXD=XD-TXV:TDYD=YD-TYV
5100 TDXYD=SQR(TDXD*TDXD+TDYD*TDYD):TDXD0=TDXD/TDXYD:TDYD0=TDYD/TDXYD
5110 TXV=TXV+DD*TDXD0:TYV=TYV+DD*TDYD0
5120 IF (TXV-X(LMINI))^2+(TYV-Y(LMINI))^2<(XV-X(LMINI))^2+(YV-Y(LMINI))^2 THEN
       ALPHA=-ALPHA:BETA=-BETA:DIRCH=1:GOTO 5140
5130 LMINI=MINI:DIRCH=0
5140 RETURN
5150 '
5500 'A COLLISION OCCURRED, SO ANNOUNCE IT ON THE SCREEN AND QUIT.
5510 SCREEN 2:SCREEN 0:LOCATE 12,30:PRINT "COLLISION!!!"
5520 PRINT #3,-1,-1,DXY(I),DRR(I) :PRINT #3,I,X(I),Y(I):PRINT #3,"COLLISION!!!"
5530 IF INKEY$="" THEN 5530
```

```
5540 XX=XV:YY=YV:GOSUB 6000:GOSUB 6500
5550  IF INKEY$="" THEN 5550
5560 SCREEN 2:SCREEN 0:STOP
5570 '
6000 'DRAW THE OBSTACLES, ORIGINAL VEHICLE PATH AND DESTINATION ON SCREEN.
6010 CLS:SCREEN 1
6020 LINE(XV0,200-YV0)-(XD,200-YD):CIRCLE(XD,200-YD),RV,1
6030 FOR C=1 TO NO
6040 CIRCLE(X(C),200-Y(C)),R(C),,,,1
6050 NEXT
6060 RETURN
6070 '
6500 'DRAW THE VEHICLE POSITION ON SCREEN.
6510 CIRCLE(XX,200-YY),RV,2,,,1
6520 RETURN
6530 '
7000 'PERFORM THE COMPOSITION OF RULES FOR CALCULATING THE RATIO.  GET
7010 'THE MEMBERSHIP FUNCTION OF THE INTERSECTION OF 'OBSTACLE RADIUS' AND
7020 "OBSTACLE DISTANCE FROM THE VEHICLE' FOR ALL COMBINATIONS OF FUZZY RADIUS
7030 'AND DISTANCE SIZES.
7040 MAXMF=0
7050 FOR DSIZE=1 TO 3
7060 'GET THE 'OBSTACLE DISTANCE FROM THE VEHICLE' MEMBERSHIP FUNCTION VALUE
7070 'MF1.
7080 P1=DIS(DSIZE,1):P2=DIS(DSIZE,2):P3=DIS(DSIZE,3):P=ODIS:GOSUB 7500
7090 MF1=MF
7100 'GET THE 'OBSTACLE RADIUS' MEMBERSHIP FUNCTION VALUE MF2.
7110 FOR RSIZE=1 TO 3
7120 P1=RAD(RSIZE,1):P2=RAD(RSIZE,2):P3=RAD(RSIZE,3):P=ORAD:GOSUB 7500
7130 MF2=MF
7140 'GET THE INTERSECTION MEMBERSHIP FUNCTION VALUE MFDR.
7150 IF MF1<MF2 THEN MF=MF1 ELSE MF=MF2
7160 MFDR=MF
7170 'GET THE MEMBERSHIP FUNCTION VALUE FOR 'RATIO' AS THE CARTESIAN PRODUCT OF
7180 'OBSTACLE DISTANCE FROM THE VEHICLE' AND 'OBSTACLE RADIUS' ACCORDING TO
7190 'EACH RULE AND THEN THE UNION OF ALL RULES.
7200 P1=RAT(DSIZE,RSIZE,1):P2=RAT(DSIZE,RSIZE,2):P3=RAT(DSIZE,RSIZE,3):P=RATI
7210 GOSUB 7500
7220 IF MF<MFDR THEN MFRDR=MF ELSE MFRDR=MFDR
7230 IF MFRDR>MAXMF THEN MAXMF=MFRDR
7240 NEXT
7250 NEXT
7260 RETURN
7270 '
```

```
7500 'THE GENERIC MEMBERSHIP FUNCTION MF(P1,P2,P3).
7510 IF P1<P2 THEN 7530
7520 IF P<=P2 THEN MF=1:GOTO 7580 ELSE GOTO 7550
7530 IF P<=P1 THEN MF=0:GOTO 7580
7540 IF P<P2 THEN MF=(P-P1)/(P2-P1):GOTO 7580
7550 IF P<P3 THEN MF=(P3-P)/(P3-P2):GOTO 7580
7560 IF P3<P2 THEN MF=1:GOTO 7580
7570 IF P>=P3 THEN MF=0
7580 RETURN
7590 '
8000 'CHECK FOR A COLLISION WITH ANY OBSTACLE.
8010 FOR I=1 TO NO
8020 IF DXY(I)<DRR(I) THEN 5500
8030 NEXT
8040 RETURN
```

```
0 'PROGRAM TO AVOID CYLINDRICAL OBSTACLES USING THE SAFOR CRISP ALGORITHM
10 INPUT "USE OBSTACLE PARAMETERS IN FILE Y(N)";ANS$
20 IF ANS$="Y" OR ANS$="y" THEN OPEN "I",1,"OBSTACLE.DAT" ELSE GOTO 60
30 'YES, SO GET PARAMETERS FROM FILE.
40 GOSUB 500:GOTO 110
50 'NO, SO GET PARAMETERS DIRECTLY FROM KEYBOARD.
60 OPEN "O",1,"OBSTACLE.DAT"
70 'GET THE VEHICLE RADIUS AND ITS INITIAL AND DESTINATION (X,Y) LOCATIONS.
80 GOSUB 1000
90 'GET THE NUMBER OF OBSTACLES, THEIR RADII, AND THEIR (X,Y) LOCATIONS.
100 GOSUB 1500
110 FOR I=1 TO NO
120 PRINT "OBSTACLE #";I;": X =";X1(I),"Y =";Y1(I)
130 NEXT
140 IF INKEY$="" THEN 140
150 'START WITH THE VEHICLE IN ITS INITIAL POSITION, INITIALIZE THE
160 'CONFIGURATION STATE, AND OPEN THE FILE TO RECORD THE VEHICLE PATH.
170 XV=XV0:YV=YV0:FIRST=1:LAST=0:OPEN "O",3,"VCOORDS.DAT"
180 CLS:INPUT "BASIC INCREMENT FOR VEHICLE MOTION ALONG PATH";DD
190 GOTO 360
200 'GET THE DISTANCE OF THE VEHICLE FROM THE DESTINATION.
210 DXD=XD-XV:DYD=YD-YV
220 DXYD2=DXD*DXD+DYD*DYD:MIND=DXYD2:DXYD=SQR(DXYD2)
230 'IF THE VEHICLE IS CLOSER TO THE DESTINATION THAN THE LENGTH OF ITS OWN
240 'RADIUS LET THE VEHICLE GO STRAIGHT TO THE DESTINATION.
250 IF DXYD<RV THEN XV=XD:YV=YD:DEST$="DEST":GOTO 350
260 'IF NOT THEN FIND THE NEAREST OBSTACLE, WHETHER IT IS THE LAST ONE, AND IF
```

```
270 'IT IS NOT GET THE DIRECTION ORTHOGONAL TO THE VEHICLE'S LINE OF SIGHT TO
280 'THE OBSTACLE.
290 GOSUB 2000
300 'IF THE NEAREST OBSTACLE IS NOT THE LAST GET THE INCREMENT FACTOR FOR THE
310 'VEHICLE MOTION ORTHOGONAL TO THE DIRECTION OF THE OBSTACLE FROM THE
320 'VEHICLE. CALCULATE THE NEW VEHICLE POSITION AND SET A FLAG INDICATING
330 'THAT IT IS NO LONGER THE INITIAL POSITION.
340 GOSUB 2500
350 'UPDATE SCREEN AND SAVE LAST VEHICLE POSITION AND DIRECTION
360 XX=XV:YY=YV
370 GOSUB 3500:GOSUB 4000
380 IF FIRST=0 THEN 420
390 'FREEZE THE SCREEN CONFIGURATION BEFORE THE VEHICLE STARTS TO MOVE
400 IF INKEY$="" THEN 400
410 IF FIRST=1 THEN FIRST=0
420 'IF NOT AT DESTINATION GET VEHICLE POSITION UPDATE
430 IF DEST$<>"DEST" THEN 210
440 IF INKEY$="" THEN 440
450 SCREEN 2:SCREEN 0
460 END
470 '
500 'GET OBSTACLE PARAMETERS FROM FILE
510 I=0:INPUT #1,XV0,YV0,ZV0,RV
520 INPUT #1,XD,YD,ZD
530 WHILE NOT(EOF(1))
540 I=I+1
550 INPUT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):R1(I)=R(I)
560 WEND
570 NO=I
580 RETURN
590 '
1000 'GET INITIAL VEHICLE DATA
1010 INPUT "VEHICLE INITIAL X,Y,Z COORDINATES";XV0,YV0,ZV0
1020 INPUT "VEHICLE RADIUS";RV
1030 INPUT "DESTINATION X,Y COORDINATES";XD,YD,ZD
1040 PRINT #1,XV0,YV0,ZV0,RV:PRINT #1,XD,YD,ZD
1050 RETURN
1060 '
1500 'GET INITIAL OBSTACLE DATA
1510 INPUT "NUMBER OF OBSTACLES";NO
1520 WHILE I<NO
1530 I=I+1
1540 PRINT "OBSTACLE #";I
1550 INPUT "X,Y,Z COORDINATES";X(I),Y(I),Z(I)
```

```
1560 INPUT "RADIUS";R(I):X1(I)=X(I):Y1(I)=Y(I):R1(I)=R(I)
1570 'IF FILE IS IN INPUT MODE DON'T WRITE IN IT
1580 'IF FIL$="I" THEN 2100
1590 'NEW DATA AND WRITE ONLY FILE MODE SO PRINT PARAMETERS IN FILE
1600 PRINT #1,X(I),Y(I),Z(I),R(I)
1610 WEND
1620 RETURN
1630 '
2000 'GET DIRECTION ORTHOGONAL TO LINE OF SIGHT FROM VEHICLE TO NEAREST
2010 'OBSTACLE. FIRST GET THE NEAREST OBSTACLE AND CHECK IF IT IS THE LAST.
2020 FOR I=1 TO NO
2030 DX(I)=X(I)-XV:DY(I)=Y(I)-YV:DRR(I)=(RV+R(I))^2
2040 DXY2(I)=DX(I)*DX(I)+DY(I)*DY(I):DXY(I)=SQR(DXY2(I)):DIST(I)=DXY(I)-R(I)
2050 IF (DIST(I)<MIND) THEN MINI=I:MIND=DIST(I)
2060 DCPA(I)=DX(I)*DXD+DY(I)*DYD
2070 IF LAST=1 THEN 2110
2080 'IF NOT ALREADY KNOWN TO BE THE LAST OBSTACLE CHECK WHETHER IT IS OR NOT
2090 'AND SET -1 FLAG IF IT IS NOT BY VIRTUE OF ITS DIRECTION.
2100 IF DCPA(I)>0 THEN LAST=-1
2110 NEXT
2120 'IF THE FLAG NOT SET THEN OBSTACLE MUST BE THE LAST.
2130 IF LAST=-1 THEN LAST=0 ELSE LAST=1
2140 'CHECK IF THE NEAREST OBSTACLE IS LAST BECAUSE THE VEHICLE IS CLOSER TO
2150 'THE DESTINATION THAN TO THE OBSTACLE AND QUIT IF IT IS.
2160 IF DXY2(MINI)>DXYD2 THEN LAST=1:GOTO 2270
2170 'GET UNIT VECTOR IN THE DIRECTION ORTHOGONAL TO THE VEHICLE PATH DIRECTION
2180 'AND POINTING AWAY FROM THE OBSTACLE CENTER. GET THE DISTANCE DACPA TO
2190 'THE OBSTACLE CENTER AT THE CLOSEST POINT OF APPROACH, THE DISTANCE DTCPA
2200 'TO THE CLOSEST POINT OF APPROACH, AND THE ALLOWED PASSING DISTANCE PD.
2210 DACPA=(DXD*DY(MINI)-DYD*DX(MINI))/DXYD
2220 DTCPA=DCPA(MINI)/DXYD:PD=SQR(2*DRR(MINI))
2230 'NOW CALCULATE THE SCALE FACTOR SCF AND DETERMINE THE VEHICLE COORDINATE
2240 'INCREMENTS INCX, INCY.
2250 SCF=SGN(DACPA)*(PD-ABS(DACPA))*DD/ABS(DTCPA)
2260 INCX=DY(MINI)*SCF/DXY(MINI):INCY=-DX(MINI)*SCF/DXY(MINI)
2270 RETURN
2280 '
2500 'GET THE NEW VEHICLE POSITION. IF THE LAST OBSTACLE IS PASSED SKIP THE
2510 'FIRST INCREMENT. OTHERWISE, INCREMENT THE VEHICLE POSITION ORTHOGONAL
2520 'TO THE DIRECTION OF THE NEAREST OBSTACLE RELATIVE TO THE VEHICLE.
2530 IF LAST=1 THEN 2550
2540 IF DCPA(MINI)>0 THEN XV=XV+INCX:YV=YV+INCY
2550 'GET THE DIRECTION OF THE DESTINATION FROM THE VEHICLE AND THE VEHICLE
2560 'POSITION AFTER INCREMENTING ITS COORDINATES IN THAT DIRECTION.
```

```
2570 DXD=XD-XV:DYD=YD-YV:DXYD=SQR(DXD*DXD+DYD*DYD):DXD0=DXD/DXYD:DYD0=DYD/DX
     YD
2580 XV1=XV+DD*DXD0:YV1=YV+DD*DYD0
2590 'GET THE SQUARED DISTANCE OF THE VEHICLE IN THE NEW POSITION FROM THE
2600 'NEAREST OBSTACLE.
2610 DXY21(MINI)=(X(MINI)-XV1)^2+(Y(MINI)-YV1)^2
2620 'IF THE DISTANCE IS LARGE ENOUGH ACCEPT THE LAST INCREMENT IN THE VEHICLE
2630 'POSITION.
2640 IF DXY21(MINI)>DRR(MINI) THEN XV=XV1:YV=YV1
2650 'RECORD THIS DATA IN THE VEHICLE PATH FILE ALONG WITH THE SQUARED SUM OF
2660 'THE VEHICLE AND OBSTACLE RADII.
2670 PRINT #3,XV,YV,DXY2(MINI),DRR(MINI)
2680 'CHECK FOR A COLLISION.
2690 GOSUB 4500
2700 RETURN
2710 '
3000 'A COLLISION OCCURRED, SO ANNOUNCE IT ON THE SCREEN AND QUIT.
3010 SCREEN 2:SCREEN 0:LOCATE 12,30:PRINT "COLLISION!!!"
3020 PRINT #3,-1,-1,-1,-1:PRINT #3,"COLLISION!!!"
3030 IF INKEY$="" THEN 3030
3040 STOP
3050 '
3500 'DRAW BASIC ORIGINAL OBSTACLES, VEHICLE PATH AND DESTINATION ON SCREEN
3510 CLS:SCREEN 1
3520 LINE(XV0,200-YV0)-(XD,200-YD):CIRCLE(XD,200-YD),RV,1
3530 FOR C=1 TO NO
3540 CIRCLE(X(C),200-Y(C)),R(C),,,,1
3550 NEXT
3560 RETURN
3570 '
4000 'DRAW VEHICLE POSITION ON SCREEN
4010 CIRCLE(XX,200-YY),RV,2,,,1
4020 RETURN
4030 '
4500 'CHECK FOR A COLLISION WITH ANY OBSTACLE.
4510 FOR I=1 TO NO
4520 IF DXY2(I)<DRR(I) THEN 3000
4530 NEXT
4540 RETURN
```

```
0 'PROGRAM TO REPLOT VEHICLE PATH AROUND OBSTACLES
10 LASTX=10:LASTY=100
```

```
20 'INPUT "VEHICLE COORDINATES FILENAME";VF$
30 'INPUT "OBSTACLE DATA FILENAME";OF$
40 OPEN "I",1,"OBSTACLE.DAT"
50 OPEN "I",2,"VCOORDS.DAT"
60 GOSUB 1000:GOSUB 5000
70 WHILE NOT(EOF(2))
80 INPUT #2,X,Y,DUMMY,DUMMY
85 'IF A COLLISION THEN ANNOUNCE IT AND QUIT
90 IF X=-1 AND Y=-1 THEN GOSUB 500:STOP
100 LINE(LASTX,200-LASTY)-(X,200-Y)
110 LASTX=X:LASTY=Y
120 WEND
130 IF INKEY$="" THEN 130
140 SCREEN 2:SCREEN 0
150 END
160 '
500 'COLLISION TO ANNOUNCE
510 IF INKEY$="" THEN 510
520 INPUT #2,C$
530 SCREEN 2:SCREEN 0:LOCATE 12,30:PRINT C$
540 IF INKEY$="" THEN 540
550 RETURN
560 '
1000 'GET OBSTACLE PARAMETERS FROM FILE
1010 I=0:INPUT #1,XV0,YV0,ZV0,RV
1020 INPUT #1,XD,YD,ZD
1030 WHILE NOT(EOF(1))
1040 I=I+1
1050 INPUT #1,X(I),Y(I),Z(I),R(I):X1(I)=X(I):Y1(I)=Y(I):Z1(I)=Z(I):R1(I)=R(I)
1060 WEND
1070 NO=I
1080 RETURN
1090 '
5000 'DRAW BASIC ORIGINAL OBSTACLES, VEHICLE PATH AND DESTINATION ON SCREEN
5010 CLS:SCREEN 1
5020 LINE(XV0,200-YV0)-(XD,200-YD):CIRCLE(10,100),10,2:CIRCLE(310,100),10,1
5030 FOR C=1 TO NO
5040 CIRCLE(X(C),200-Y(C)),R(C),,,,1
5050 NEXT
5060 RETURN
5070 '
```

DATA FILES FOR USE IN THE AVOIDANCE PROGRAMS (TO USE OBSTACLE DATA FOR ANY OBSTACLE CONFIGURATION NUMBER N COPY THE FILE NAMED OBSTACLN.DAT TO OBSTACLE.DAT. WHEN THE PROGRAM ASKS "USE PARAMETERS IN FILE ?" ANSWER "Y").

OBSTACL1.DAT
```
10      100     10      10
310     100     10
100     125     20      20
150     70      25      25
200     70      25      25
```

OBSTACL2.DAT
```
10      100     10      10
310     100     10
100     125     20      20
150     70      25      25
200     70      25      25
250     120     30      30
```

OBSTACL3.DAT
```
10      100     10      10
310     100     10
75      100     25      25
125     120     20      20
125     70      30      30
200     70      25      25
225     120     25      25
270     80      30      30
```

OBSTACL4.DAT
```
10      100     10      10
310     100     10
75      60      25      25
100     110     30      30
125     120     25      25
175     80      30      30
225     120     25      25
```

OBSTACL5.DAT
```
10      100     10      10
310     100     10
50      90      25      25
150     110     30      30
250     120     25      25
```

MF.DAT (THIS IS FUZZY MEMBERSHIP FUNCTION DATA)

```
35,5,20
5,20,35
20,35,5
100,20,50
20,50,100
50,100,20
1,3,5
3,10,5
3,10,5
5,1,3
1,3,5
3,10,5
5,1,3
5,1,3
1,3,5
```