Computers Math. Applic. Vol. 26, No. 6, pp. 97–124, 1993
Printed in Great Britain. All rights reserved

0898-1221/93 $6.00 + 0.00
Copyright© 1993 Pergamon Press Ltd

Application of Semantic Control to a Class of Pursuer-Evader Problems

A. Garcia-Ortiz and J. Wootton
Advanced Development Center, Electronics and Space Corp.
8100 W. Florissant Ave., St. Louis, MO 63136, U.S.A.

E. Y. Rodin, S. M. Amin, M. Meusey, C. Ruan,
A. Y. Wu, P. De and J. Revetta
Center For Optimization and Semantic Control
Department of Systems Science and Mathematics, Washington University
One Brookings Drive, St. Louis, MO 63130-4899, U.S.A.

Abstract—In this article, we describe our work in developing a comprehensive software system for tactical decision aiding for an evader faced with multiple pursuers. The objective is to provide the evader with defensive maneuver decisions that maximize its chances of survival.

We have developed a hierarchical semantic controller consisting of a System Identifier, a Goal Selector and an Adapter. This system is implemented on a 386DX personal computer via object-oriented programming, knowledge-based systems, Analytical Hierarchy Process, optimal control, and differential game methodologies.

The viability of our Semantic Control approach to the evasive action selection problem has been shown and its operation has been tested against pursuers which follow either pure pursuit or proportional guidance strategies. The user is included in the decision process via approval of setpoints. Displays of the engagement and effect of coverage by countermeasures provide a visual reinforcement of the recommendation made. Use of semantic controllers as TDA support systems for man-in-the-loop, pursuer-evader problems on a PC with current software and hardware technology is feasible. The ability to deploy the system on a portable PC permits the use of the technology in a wide variety of applications.

1. BACKGROUND

The subject of the study here presented is the development of a viable methodology, which the pilot of an unarmed plane (but one equipped with so-called countermeasures) can employ, so as to evade one or more missile equipped pursuers. The problem is obviously very complex: it may be appropriate, therefore, to devote a few paragraphs to an explanation of why we are employing the philosophy and the techniques presented in the next several sections.

To begin with, one would obviously try to set up some sort of mathematical and/or computer model of such a scenario. In that connection, the mere statement of the problem would probably bring to one's mind some simple obvious associations, such as:

(1) "Airplane" $\mapsto$ *Dynamics*
(2) "Evader and Pursuer" $\mapsto$ *Differential Games*
(3) "Viable Methodology" $\mapsto$ *Fast, On-line Computation*
(4) "One or More" $\mapsto$ *Complexity*
(5) "Countermeasures" $\mapsto$?????????????

The work of all the members affiliated with the Center for Optimization and Semantic Control was supported in part by AFOSR under Grant number 890158.

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX

On the one hand, the first two of these items would clearly indicate to attempt to model this dynamic conflict situation as a differential game: namely, one in which the objective functions are those of the pursuers and of the evader, while the constraints would have to describe the dynamics of all the players. The principal reason for wanting to do so is that Differential Game theory seems to be the only one available today which—at least in principle—is capable of modelling such a situation more or less exactly.

On the other hand, taking the requirement of having to produce a "viable methodology" into consideration, it is well-known that there exists only a very small number of differential games, for which exact solutions are available. There are a few more for which numerical solutions can be computed, under rather restrictive conditions, in a reasonable amount of time. However, each one of these games is one-on-one (which generally is not the case here), and they are also of the zero-sum type. This last point is also important, because in the present situation, an encounter may result not only in the possibility of one side winning and the other one losing, but—for example—a mutual disengagement is possible too.

Item 4 seems to cause even more head scratching. After all, how complex can the situation become? Does one have to deal with a three-dimensional problem, or is a two-dimensional approximation sufficient? How many "players" will participate in the "game"? Are their lifetimes the same? Can a new "player" join the "game" while it is already being played?

The answer to all these questions seems to be that, in order to develop something halfway realistic, all of the above things, and many more, will definitely have to be taken into consideration.

Finally, how does one handle countermeasures mathematically? There seem to be so many kinds: active and passive, highly mobile and stationary, those with short and with long lifetimes, etc. On top of this, there do not seem to exist any extant mathematical models describing them. So, the question of how to handle them quantitatively seems to warrant the multiple question marks above ...

Having arrived at the rather pessimistic set of thoughts so far described, let us try to take a more formal approach to the problem, and define for ourselves what it is that we need to do. A proper recipe seems to be the following:

(a) formulate a properly descriptive and well-posed mathematical model;
(b) solve the model in real time;
(c) interpret the solution for the pilot;
(d) provide the pilot with alternative possible courses of action, while indicating their relative qualities in terms of deviation from optimality.

Assuming that we have somehow succeeded in obtaining an appropriate, differential game type mathematical model, encompassing all of the relevant aspects of such encounters (which, incidentally, is rather doubtful for a situation as complex as this one), we have to examine point b: would such a model be amenable to a real time solution, given the time constraints of a pilot in a combat situation? For an answer to this question, we quote the observations of J. Shinar [1]:

> "The difficulties in solving a 'well-formulated' differential game have the following origin. The classical solution of a differential game is based on the simultaneous backward integration of the state and adjoint equations, starting at the target set (terminal manifold) of the game, in order to fill the entire game space by the ensemble of optimal trajectories. The backward integration is a rather direct operation as long as no singular surface of the game, implying a discontinuity of the adjoint vector, is encountered. Experience has shown that there are several different singular surfaces which are frequently encountered in differential games solutions. The game solution requires, once a singular surface is reached, to determine the type of singularity and to continue the backwards integration accordingly. As the first step, the existence of the singular surface has to be identified by observing the intersection of retrograde trajectories belonging to different

> end conditions. In a simple game, with no more than two state variables, such intersections can be easily visualized. For a differential game of three independent variables, the same process becomes very cumbersome and, for dynamic models of higher dimension, it is virtually impossible."

Our discussion so far seems to indicate clearly that the theory of Differential Games alone is insufficient in order to provide an appropriate solution to the problem at hand. Fortunately, however, there exists a relatively new theory, that of Semantic Control [2], which may provide a way out of these difficulties. Semantic Control is a theory which attempts to combine classical mathematical approaches (such as Differential Game theory) with Artificial Intelligence paradigms, in order to obtain "reasonable" solutions to highly complex, and otherwise intractable, problems. The question that remains then, is this: can the merger of these two approaches be applied successfully here? For a possible answer, we quote again Shinar:

> "Every dynamic conflict is essentially a multi-stage decision process, and can be represented as such by a 'tree.' Such representation is indeed identical to a game in its extensive form. In fact, a differential game is also a game in extensive form and equivalent to a 'decision tree' of infinite nodes. By discretizing time, a finite 'game tree' is obtained. The nodes of the tree represent the states of the game, where the players can select their controls for a given period of time. The branches of the tree are the moves in the game space. The pruning of such a tree is one of the basic tasks of heuristic search techniques used in most AI programs. In the case where the players do not change their decisions at every node (selected by arbitrary discretization of the time scale), some moves can be aggregated to a single branch and the resulting 'game tree' becomes simplified. It is also possible that the players do not make their decisions simultaneously and in this case, the respective moves of the two sides can be easily distinguished."

The remainder of this paper provides, then, an illustration of how one can indeed surmount the formidable difficulties we mentioned earlier, by a judicious blending of classical mathematical methodologies (i.e., Differential Games and other techniques) with aspects of Artificial Intelligence. In particular, we are demonstrating our ability to do each of the following:

(i) take into account the dynamics of each player;
(ii) utilize "piecewise" Differential Games;
(iii) perform the necessary calculations in real time, on line;
(iv) take into account large varieties of countermeasures;
(v) provide an easy interpretation of the solution for the pilot;
(vi) provide alternative policies, with a relative measure of their respective qualities.

2. INTRODUCTION

The objective of this work has been to develop a tactical decision aid (TDA) which, in the presence of a real or potential threat, aids its user in the selection of evasive maneuvers. The selection and activation of evasive maneuvers may seem to be a straightforward process; however, defensive actions may prove threatening to the evader. For example, a terrain following, low-flying aircraft that activates its radar jamming equipment to render ineffective its opponent's anti-aircraft batteries is effectively revealing its presence in the area. The opponent(s) may then take advantage of other techniques to pinpoint the aircraft's location and engage the aircraft. In order to effectively implement evasive maneuvers, one must recognize that defensive actions affect all participants in a combat situation.

Recent work in the field of Semantic Control Theory [2–6] provides an appropriate framework for addressing this problem—a three-level, hierarchical control structure (see Figures 1 and 2a) consisting of:

(a) an Identifier, which collects and interprets the available information;
(b) a Goal Selector, which generates and evaluates several plans; and
(c) an Adapter, which implements the optimal plan.

This system mirrors the human decision-making process; its mathematical structure facilitates computer implementation. The implication that many more alternative actions can be evaluated in a given amount of time, which enhances the probability of survival.

We apply Semantic Control Theory to the development of a TDA for multiple-pursuer, single-evader problems in which each side possesses "countermeasure devices" (e.g., radar jamming equipment) for obscuring its movements from the other side. The evader's objective is to avoid detection and engagement by the pursuers while traveling to a specified destination. The Identifier portion of the TDA collects all available data and uses it to calculate a "mission score" for the evader based on his proximity to his destination and the threat posed by pursuers. The Goal Selector uses a rule-based system to generate various evasive maneuver strategies (including the use of countermeasure devices) for maximizing the mission score. The user, who acts as the Adapter, selects one of the strategies. The process is repeated each time new data on pursuers is received, and continues until the evader reaches his destination or is neutralized.

The TDA is currently implemented on a Dell System 333D^{TM} personal computer, using KAPPA-PC^{TM} and Toolbook^{TM} under the Windows^{TM} environment (see Figures 2b and 2c). A "Scenario Generator" was developed to simulate the mission environment. This Scenario Generator controls the number and behavior of pursuers and provides the TDA with sensory data of their movements. Through these means, the viability of our Semantic Control approach to the defensive countermeasure selection problem has been shown. Its operation has been tested against pursuers which follow either pure pursuit or proportional guidance strategies.

The sections that follow describe our work in greater detail. Section 3 describes how the Identifier collects and maintains sensory data, and how it quantifies this data into a mission score. Section 4 lists the rules used by the Goal Selector to generate and evaluate various evasive maneuvers. Section 5 describes the man-machine interface screens available to the user in his role as Adapter. Section 6 describes the Scenario Generator. Finally, in Section 7, we present results.

3. IDENTIFIER

The requirements for the Identifier are:

(1) to collect all available sensory data,
(2) to maintain this data in a global database, and
(3) to quantify this data into a mission score for the evader.

An object oriented formulation was selected for the first two tasks because it allows incremental refinement of the situation assessment, knowledge of the pursuer's capability, and evasion strategies. The mission score is calculated from the sensory data using the Analytic Hierarchy Process.

Player Database

Structure

Kappa-PC was used to create a semantic network type class hierarchy for storing the player data. A diagram of part of this hierarchy is shown in Figure 3. Any entity encountered during the mission is referred to as a Player. The entity is further classified as being Known if it falls

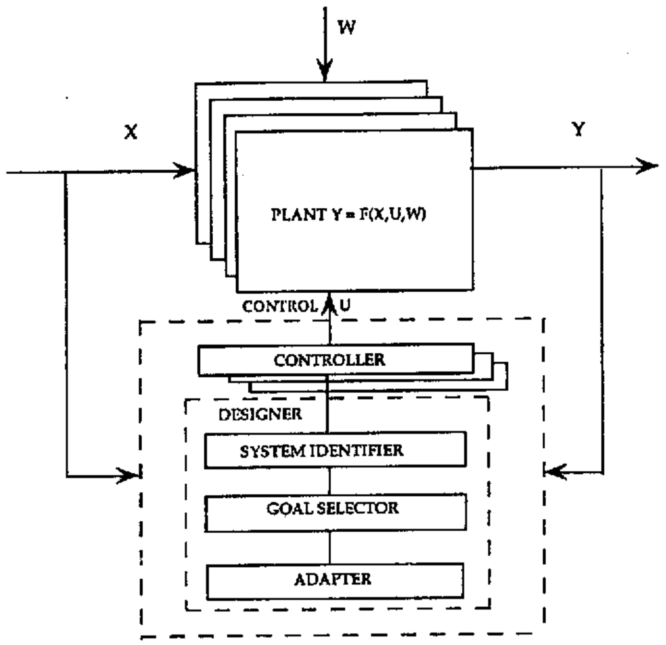


Figure 1. Semantic control procedure.

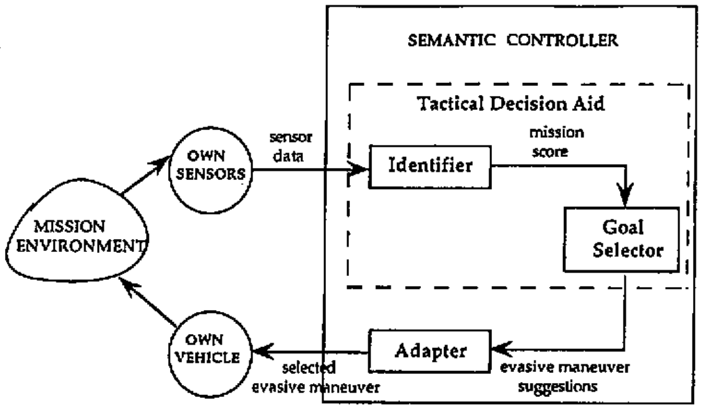


Figure 2a. Structure diagram with semantic control.

into one of four subclasses: Evader, Pursuer, Shadowing Player, or Neutral. If data on the entity is insufficient to classify it as one of these four types, it is put into the class Unknown until sufficient data becomes available.

The two main classes of Known players in the game are Evader and Pursuer. Evader represents the "own vehicle" controlled by the user. Pursuers are subdivided into two classes: Primary and Secondary. Primary Pursuers represent the enemy vehicles, and Secondary Pursuers represent the weapons they launch (e.g., missiles). Shadowing Players are the countermeasure devices used to obscure the perception of the opponent. A Shadowing Player is classified as being either an Evader Shadowing Player or a Pursuer Shadowing Player, according to which side launched it. Neutral players are entities which are encountered but are not involved in the conflict and pose no threat to either side.

Each of the classes above has characteristics or "slots" which are particular to it. These slots are listed in Figure 4. Each class inherits all the slots of all its ancestors in the hierarchy. Notice that not every class in the hierarchy has slots which are unique to it. For instance, Primary and Secondary Pursuers have all the same slots, but they need to be distinguished in the hierarchy because of the qualitative differences between the two.

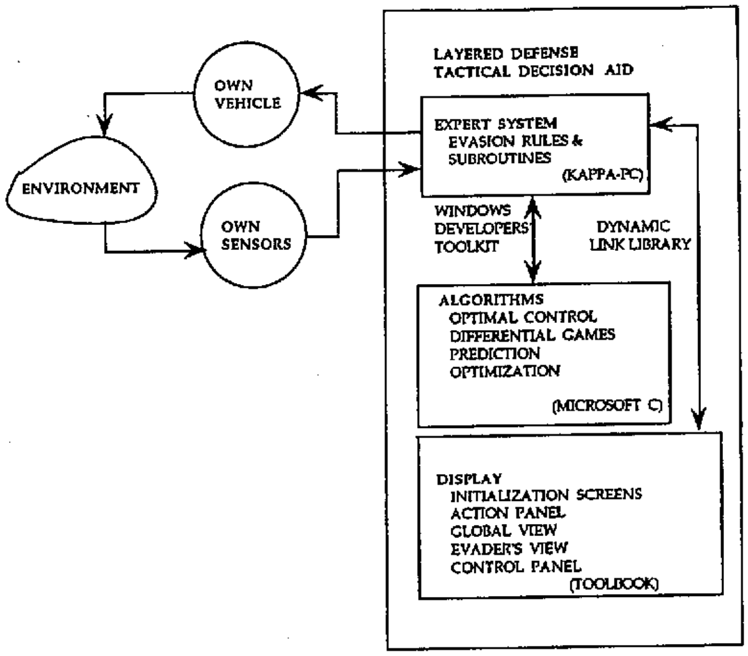

LAYERED DEFENSE TACTICAL DECISION AID
OWN VEHICLE
ENVIRONMENT
OWN SENSORS
EXPERT SYSTEM EVASION RULES & SUBROUTINES (KAPPA-PC)
WINDOWS DEVELOPERS' TOOLKIT
DYNAMIC LINK LIBRARY
ALGORITHMS OPTIMAL CONTROL DIFFERENTIAL GAMES PREDICTION OPTIMIZATION (MICROSOFT C)
DISPLAY INITIALIZATION SCREENS ACTION PANEL GLOBAL VIEW EVADER'S VIEW CONTROL PANEL (TOOLBOOK)

Figure 2b. Functional diagram of the system.

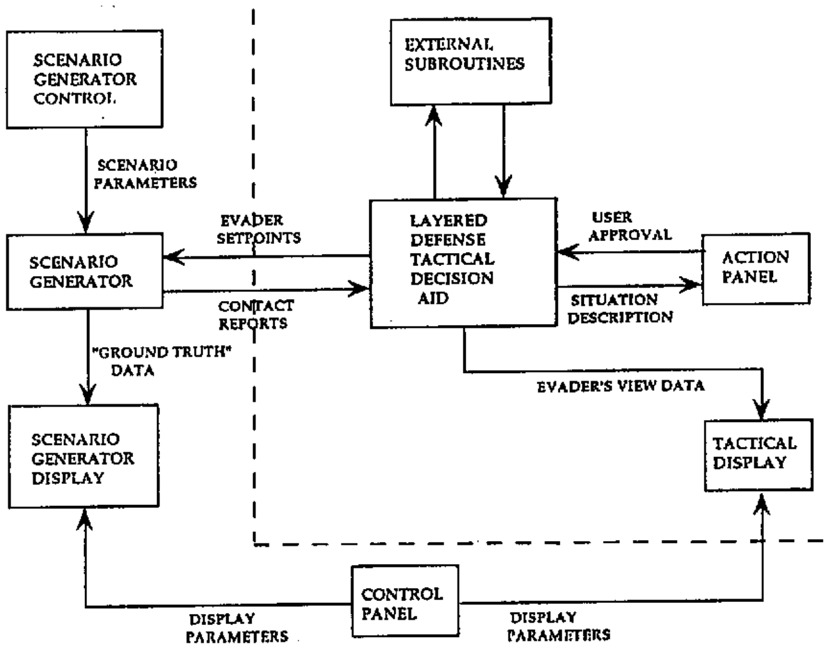


Figure 2c. Simulation flowchart.

The *a priori* data slots reflect knowledge gained previously about a player's behavior and abilities. The sensory data slots store the dynamic information received by the Identifier. This datum can be either "spatial" or "descriptive." Spatial data relate to the location and movements of a player. Descriptive data pertain to a player's dynamic behavior and capabilities.

Further details about the meanings of the slots are deferred to the following section, which describes how entities encountered during a mission are put into the hierarchy and how the information on them is maintained.

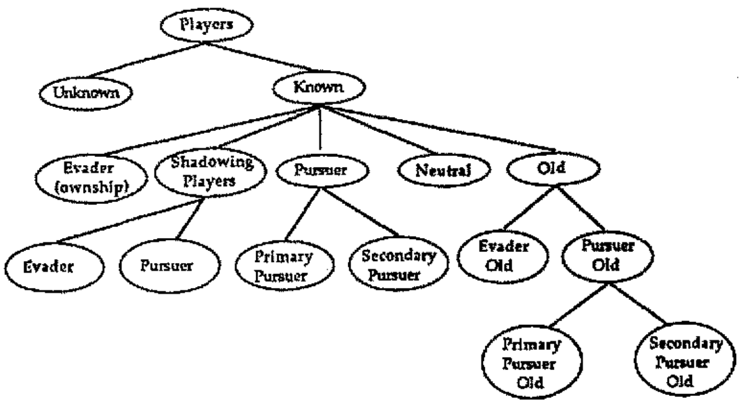


Figure 3. Semantic hierarchy of players. This diagram shows the inheritance relationship among the players. Every player inherits the characteristics of all its ancestors.

Type of Data		Class				
		Player	Known	Evader	Pursuer	Shadowing Player
Sensory Data	Spatial	Position Speed Range Bearing Heading Event		Home Destination		
	Descriptive	Contact ID Classification Confidence		Inventory	Effectiveness Inventory	Effectiveness
A priori Data			Lifetime Max Speed Perception	Risk Level Panic Distance Max Safe Distance		Delay Duration Spatial Effectiveness

Figure 4. Table of slots.

Contact Report Data	
Spatial	Range Bearing Heading Speed Event
Descriptive	ContactID Time Stamp Classification Confidence

Figure 5. Contact report data.

Maintenance

Sensory data are received by the Identifier in the form of "contact reports." A contact report is given for each entity detected by the evader's sensors. The data contained in these contact reports are shown in Figure 5.

Upon receiving the contact report, the Identifier first looks at the "Classification" slot, which contains the class to which the player belongs. The Identifier uses the name provided in "ContactID" to see whether this class has an instance by this name. If not, a new instance is created in the appropriate class.

We first assign a position to every player. To accomplish this, it is necessary to define a coordinate frame. The TDA keeps track of a player's position in two coordinate frames: an inertial one and the body frame of the evader. The inertial frame is defined with the positive x direction pointing to the front of the body frame, the positive y direction pointing to the right and the positive z direction pointing down. Three slots are then defined for each player: one for the x coordinate, one for the y coordinate and one for the z coordinate. The body frame of the evader is defined in the usual manner: the positive x direction points in the direction of motion of the evader, the positive y direction is $+90°$ from the positive x direction (i.e., if the evader is traveling due north, the positive x direction is north and the positive y direction is east) and the positive z direction points to the bottom of the vehicle and is orthogonal to the x-y plane. Again, three slots are defined to store the x, y and z coordinates of a player in this frame.

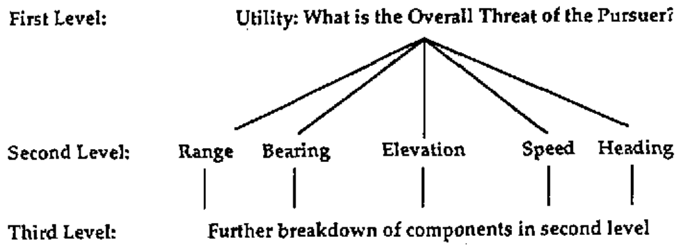
To define the motion of a player, slots are defined to keep track of speed, range, bearing, and heading. Speed is self explanatory; range represents the distance from player to evader. The heading angle indicates in what direction the contact player is traveling relative to north. This angle is measured clockwise from 0 to 360°, with north being defined as a heading of 0°. The bearing angle is the angle between the evader's direction of motion and the evader's line of sight to the pursuer. This angle ranges from -180 to 180°, with positive angles corresponding to clockwise displacement from the body frame (x axis).

Four additional slots are defined to record other data received in the contact reports. The first is called "contact ID," which gives the identification code assigned to the contact in the contact report. Another slot contains the classification of the contact: primary pursuer, etc. The third is an "event" slot, which describes a derived feature of the contact. For instance, a secondary pursuer could be in runout, search, or terminal homing mode. The final slot is called "confidence," which is a number between 0 and 1 indicating the accuracy of the data received in the contact report for that player.

All known players are given slots called "Lifetime," "Maxspeed," and "Perception" range. "Lifetime" gives the approximate useful life span of a player. This is particularly important for secondary pursuers and shadowing players, both of which remain effective for relatively short periods of time. "Maxspeed" is used to store the top speed a player can achieve. This is useful in determining whether or not the evader can outrun a potential threat. "Perception" gives the maximum range at which one player can detect another.

All pursuers have an additional slot called "effectiveness." This number assigns the player a "score" which represents the degree of threat posed to the evader. This score is useful in evading multiple pursuers, as it gives a measure of how dangerous one pursuer is relative to another. Different methods must be used to score primary and secondary pursuers.

Shadowing players are also given an "effectiveness" slot, which reflects their usefulness in hiding the evader from pursuers. This measure can be used to determine whether or not a shadowing player should be released. Shadowing players are also given slots called "delay" and "duration." The first gives the length of time after launch before a shadowing player becomes active, while the second gives the length of time the shadowing player will remain active before "burning out." There is also a slot called spatial effectiveness which specifies the radius of cover provided by the shadowing player while it is active.


First Level:
Utility: What is the Overall Threat of the Pursuer?
Second Level:
Range
Bearing
Elevation
Speed
Heading
Third Level:
Further breakdown of components in second level
Figure 6. Hierarchy for threat level evaluation.

The mission of the evader is to safely travel to a predetermined destination and then safely return to the starting point. It is therefore necessary to define slots for the evader which contain the coordinates of these two points. Slots are created for storing the x, y and z coordinates of both points, in the inertial and body coordinate frames.

Since safety is a factor in the evader's mission, he is given additional slots pertaining to it. The first is called "risk level," which is a value determined *a priori* to reflect how daring the evader is willing to be to complete his mission. There are two slots defined as the "maximum safe distance" and the "panic distance." The first represents the range at which the evader first considers a pursuer to be an immediate threat, and the second represents the range at which the evader temporarily abandons the goal of reaching his destination and gives full priority to escape. These two distances will vary according to the degree of risk the evader is willing to assume.

Finally, the evader has a slot which keeps track of the number of remaining shadowing players. This can be a factor in determining whether or not a shadowing player should be released.

Threat Level Evaluation

The System Identifier must also assess the current game situation. It does this by assigning a threat level to all pursuers currently chasing the evader. This assignment is made whenever new contact reports are received. The first step taken by the Identifier is to call a function which processes the contact reports. If a contact report identifies a player not previously detected, then a new instance is created in the appropriate location within the hierarchy. If the contact report has new information about a player previously identified, then two actions are taken: the information from the previous report is moved to an instance of the "Old" class, and the new information is used to update the player.

When this information has been processed, the TDA makes a preliminary assessment of the game situation. From the information given in the contact report, a methodology is developed to incorporate the characteristics of the pursuer and its relationship to the evader, in order to develop a threat level in the form of a threat level function.

The position of the pursuer can be thought of as one measure of the pursuer's threat to the evader. The threat level is the utility function of the pursuer relative to the evader. The closer the threat level value to zero, the lesser the threat of the pursuer to the evader. The closer the threat level value to one, the greater the threat. This threat level-function approach is very useful when the evader is operating in a multiple-pursuer environment; the utilities of all the pursuers are evaluated to determine which poses the greatest threat. The threat levels also aid in choosing an evasive action to handle the appropriate threat(s) of the pursuer(s).

We refined our current approach to combine the quantifiable characteristics of the pursuers in relation to the evader. Basically, this approach is an implementation of the Analytic Hierarchy Process (AHP) [7–13] developed by Saaty in the 1970's, which implements a hierarchy of a decision process. Here, through pairwise comparisons of the elements of the decision's characteristics, one is led to a calculation of the threat level function. The threat level of the pursuer is the first level of the hierarchy. It is divided into five components: range, bearing, elevation, speed, and heading. These become the second level of the hierarchy. They feed back to the first level, which is the threat value of the evader in terms of pursuer characteristics (Figure 6).

The situation now becomes one of determining appropriate weights for these five components. If the five components all contribute equally to the threat level function, each component would have a weight of 0.20. This is rarely the case, so a decision matrix must be formed to evaluate the components through pairwise comparisons. The decision matrix is a five-by-five matrix with each ij^{th} element of the matrix being the pairwise comparison of component i to component j. The pairwise comparisons are made on the following scale with component i's relation to component j evaluated according to the following criteria (Figures 7 and 8).

Scale of Measurement for AHP	
Numerical Values	Definition
1	Equally Important
3	Slightly More Important
5	Strongly More Important
7	Very Strongly More Important
9	Extremely More Important
2, 4, 6, 8	Intermediate Values to Reflect Compromise

Figure 7. Scale of measurements for the AHP. Reciprocals are used to reflect dominance of the second alternative as compared with the first.

5 x 5 Pairwise Decision Matrix for Second Level					
	Range	Bearing	Elevation	Speed	Heading
Range	1	a_{ij}	a_{ij}	a_{ij}	a_{ij}
Bearing	$1/a_{ij}$	1	a_{ij}	a_{ij}	a_{ij}
Elevation	$1/a_{ij}$	$1/a_{ij}$	1	a_{ij}	a_{ij}
Speed	$1/a_{ij}$	$1/a_{ij}$	$1/a_{ij}$	1	a_{ij}
Heading	$1/a_{ij}$	$1/a_{ij}$	$1/a_{ij}$	$1/a_{ij}$	1

Figure 8. Pairwise comparison matrix for the second level of the hierarchy.

It follows that the diagonal of the matrix consists of ones since each element is of equal preference to itself. Also, the lower diagonal of the matrix will simply be the inverse of the upper diagonal ($a_{ji} = 1/a_{ij}$). This reduces the number of pairwise comparisons to $n(n-1)/2$ and greatly speeds the somewhat tedious process of making pairwise comparisons. The eigenvalues of this matrix (Perron roots) can be used as the solutions to the set of simultaneous equations represented by the matrix equaling one. In other words, the eigenvalues can be used as the weights for the five components of the hierarchical level, with $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 = 1.00$.

From the contact report, quantifiable values for the five components of the second level of the hierarchy are received. The contribution of each of these contact values to the components discussed above is determined through appropriate pairwise comparisons. Once the weights of the components of the second level of the hierarchy have been determined, we know the contribution of each component to the threat level function value.

However, the job is not complete since each component of the second level can be decomposed further. A third hierarchical level is developed under each component of the second level. Five additional decision matrices for pairwise comparisons of subcomponents determine the contribution to the overall weight of the main component. The subcomponents are based on the range of values accessible to the components and on the position of the pursuer relative to the evader.

When the decision matrices are formed for this third level, the weights for all decision variables of the component matrices can be calculated as above and scaled from zero to one.

We thus calculated the value of the threat level function, TL, through a synthesis of the five components of the second level of the hierarchy and the corresponding values of the third level obtained from the information in the contact report. The threat level function is written as

$$\mathrm{TL} = f(x_1, x_2, x_3, x_4, x_5),$$

where

$$x_1 = \text{range}; \quad x_2 = \text{bearing}; \quad x_3 = \text{elevation}; \quad x_4 = \text{speed}; \quad x_5 = \text{heading}.$$

Each component under the x_i's, called x'_{ij}, is synthesized to calculate the threat level function by

$$\mathrm{TL} = (x_1)(x'_{1j}) + (x_2)(x'_{2j}) + (x_3)(x'_{3j}) + (x_4)(x'_{4j}) + (x_5)(x'_{5j}).$$

The threat level function is scaled from zero to one by normalizing the product above. The threat level function assigns to each pursuer a threat level value from zero to one in relation to its position to the evader. However, some additional information must be added to the above hierarchy: effectiveness and perception. If the pursuer is primary, it has an effectiveness. If it is secondary, it has an effectiveness and a perception. These are both under the component of range; the second level of the hierarchy is augmented to include these two pieces of information. Decision matrices are formed for both effectiveness and perception, and the component range is augmented into a function:

$$\text{Range} = g(\text{range}, \text{effectiveness}, \text{perception}).$$

Effectiveness and range become the indicator variables $(0, 1)$; if the pursuer is primary, only range and effectiveness will contribute to the overall weight of range. If the pursuer is secondary, range, effectiveness and perception will contribute to the overall weight of range.

However, this methodology (in its current structure) has a limitation. To obtain a measure of threat within the threat level function framework, a calculation must be made at every step in the time domain of the pursuer relative to the evader. Additional calculations must be made to determine the evader's reactions to this threat, if any. This calculation intensity is not within the framework of quick, efficient computer reaction time. Another strategy is to identify all possible threat values in the state space of the game and arrange them in a suitable look-up table (some large $n \times m$ matrix). This has the advantage of making all calculations off-line, but has the severe limitation of being unsuitable within the current KAPPA-PC$^{\mathrm{TM}}$ environment. As an alternative (but in keeping with the spirit of the AHP), a revised hierarchy is structured where the highest level, threat is evaluated as a function of only two factors: effectiveness and motion (see Figure 9).

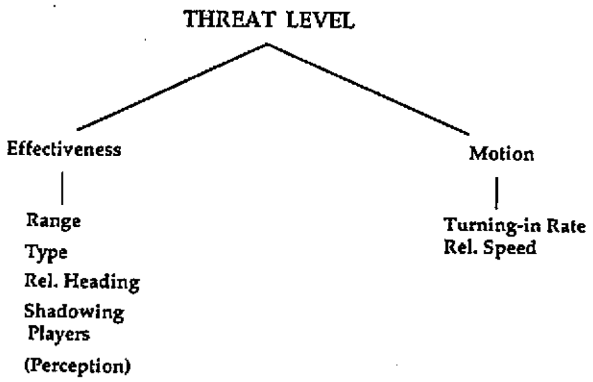


Figure 9. Revised hierarchy for contribution of effectiveness and motion to threat level.

The effectiveness of the pursuer is reflected by:

(a) its distance (range) from the evader;
(b) its potential to do damage to the evader;
(c) its heading relative to the evader's heading; and
(d) the "effectiveness" of the shadowing players spawned by the evader to hide its location from the pursuer.

Effectiveness is depicted graphically with the x-axis as a normalized range [0–1] based on the evader's perception range. The effectiveness of the pursuer (shown by its value in the y-direction from [0–1]) simply constitutes the pursuer's ability to harm the evader, with effectiveness increasing in inverse proportion to the distance between the pursuer and the evader. For simplicity, a strictly linear function was utilized and refined through curve-fitting as knowledge of scenarios was obtained.

The second factor, motion, is described as the pursuer's dynamics relative to the evader for a given scenario. Motion is seen as a function of the turning-in rate of a given pursuer and its speed relative to that of the evader.

These two factors, effectiveness and motion (see Figure 9), are compared through the AHP to determine their individual weights (α_1 and α_2). Threat is then calculated as a linear combination of effectiveness and motion. Within each factor, the criteria range, type, relative heading, and shadowing players (all under "effectiveness") and turning-in rate and relative speed (both under "motion"), are also compared to determine their individual contributions. This is continuously improved as more insight and knowledge is obtained of the individual pursuer's characteristics and the scenarios within which the pursuer and the evader compete. The overall threat evaluation can be calculated as:

$$\text{Threat} = (\alpha_1)(\text{effectiveness}) + (\alpha_2)(\text{motion}),$$

where

$$\text{effectiveness} = (\beta_1)(\text{range}) + (\beta_2)(\text{type}) + (\beta_3)(\text{relative heading}) + (\beta_4)(\text{shadowing players}),$$

and

$$\text{motion} = (\gamma_1)(\text{turning-in rate}) + (\gamma_2)(\text{relative speed}),$$

and where scalars $\alpha_i, \beta_i, \gamma_i \, \varepsilon (0,1)$ are appropriate weights determined through the pairwise comparisons of the AHP.

4. GOAL SELECTOR

Evasion Tactics

The rules used to govern the movement of the evader change with the game situation. Situations include no pursuers within a specified range, one primary pursuer within a specified range, one secondary pursuer within a specified range, or both a primary and secondary pursuer within a specified range. Each of these situations has an associated specific set of rules. The rule set appropriate to the situation is activated and used to make recommendations for safely moving the evader.

The "No Threat" rule set is used either when there are no pursuers present or when the pursuers are too far away to be considered a threat. In the first case, the evader should logically travel on a straight line toward the destination. A function called "DestHeading" calculates the heading necessary to reach the destination, and recommends this as the heading setpoint. The speed setpoint is recommended to be the value of "CruiseSpeed," which is a constant that can be set as the chosen speed when the evader is not pursued.

When pursuers are present, two cases are identified. When the pursuer is behind the evader, i.e., when the magnitude of the pursuer's bearing is more than 90°, the "Run Far" rule will fire and the evader will simply try to outrun the pursuer to his destination. When the pursuer is in front of the evader, the "Common Normal" rule, explained in detail later, is chosen.

When a pursuer becomes a potential threat to the evader, the rule set changes. The Pursuer Threat rule set determines the level of threat posed by the pursuer. The threat level is determined by two factors: the range and the bearing of the pursuer. The range is checked to see whether the pursuer is inside or outside the panic distance, and the bearing is checked to determine whether the pursuer is in front of, behind, or beside the evader.

Six cases are identified when a pursuer is outside the panic distance. The pursuer's location is determined relative to the evader's direction of motion. This is accomplished by checking the pursuers bearing and using the following criteria:

- if the magnitude of the pursuer's bearing is less than a specified number of degrees, then he is considered to be "in front of" the evader;
- if the magnitude of the pursuer's bearing is between two specified degrees, he is considered to be "beside" the evader;
- if the magnitude of the pursuer's bearing is greater than a specified number of degrees, he is considered to be "behind" the evader.

Next, it is determined whether or not the pursuer is closing on the evader. The pursuer's heading is compared with the heading along the line of sight to the evader. If the current heading is within a specified number of degrees of the line of sight heading, the pursuer is said to be closing.

There are six possible situations when these criteria are combined. The rule base contains one rule for each case. The rules will instruct the evader to move in one of two ways: either perpendicularly to the heading of the pursuer, or along the line of sight, away from the pursuer. The option more likely to immediately reduce the threat to the evader is the one chosen as the recommendation for the heading setpoint.

The above rules require minimal calculations and can quickly provide good recommendations. This is useful if the time allocated to recommendations is too short for more complex calculations.

A pursuer within the panic distance is considered enough of a threat that shadowing players are needed to evade him. Three rules arise: DangerRear, DangerSide, and DangerFront. If the pursuer is behind or to the side of the evader, the corresponding rule recommends that the evader spawn a shadowing player and set the heading setpoint to the line-of-sight heading. The shadowing player will then be in the line of sight when activated and will block the pursuer's view of the evader.

When the pursuer is in front of the evader, it is more difficult to spawn a shadower and move it between the two players when activated. Therefore, the DangerFront rule instructs the evader to move perpendicularly to the heading of the pursuer. The objective is to move the pursuer to a more favorable position (either beside or behind the evader) so that a shadowing player may be used effectively.

Vehicle Heading Computation

In addition to the rules described, we implement optimal control methodology based on minimum terminal time and use of the Hamiltonian equation. The optimal control algorithm is utilized when the pursuer is within a user-specified distance (e.g., the pursuer is within evasion distance and/or the panic distance of the evader). To obtain the optimal solution over a large or small time interval, we model the dynamics of the evader and pursuer(s), and set an appropriate performance functional, which is to be minimized.

Because it is difficult to completely formulate and solve such an optimal control problem, we formulated a simpler version which delivers a local solution. When the pursuer is within a certain distance from the evader, the goal is to achieve a pre-specified distance in a minimum time. The optimal control algorithm computes a trajectory for the evader which accomplishes this goal.

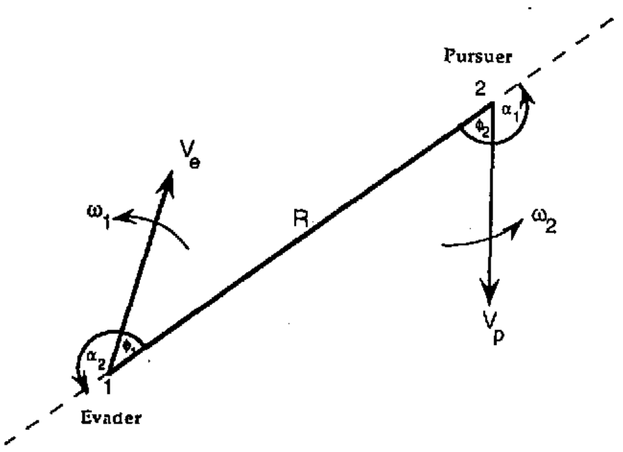


Figure 10. Line-of-sight coordinate system.

To make the optimal control problem rapidly implementable, a line-of-sight coordinate model was chosen [14]. In differential game theory [14–18], line-of-sight coordinates have been used to study the two-target game problem. As in Figure 10, the line-of-sight coordinates are described by the following equations:

$$\dot{R} = -(v_e \cos\phi_1 + v_p \cos\phi_2), \tag{1}$$

$$\dot{\phi}_1 = \frac{(v_e \sin\phi_1 + v_p \sin\phi_2)}{R} + \sigma_1, \tag{2}$$

$$\dot{\phi}_2 = \frac{(v_e \sin\phi_1 + v_p \sin\phi_2)}{R} + \sigma_2, \tag{3}$$

where R is the range (the magnitude of the line of sight vector), ϕ_1 is the "off-boresight" angle (the firing direction relative to the evader's line of sight) and ϕ_2 is the corresponding angle for the pursuer.

The problem is: Given σ_2, a control strategy for the pursuer, find an optimal control for the evader, i.e., $\sigma_{1\text{optimal}}$, where $|\sigma_i| \le 1$, $i = 1, 2$.

The necessary condition to be satisfied at every point of the barrier surface is:

$$\min_{\sigma_1} \max_{\sigma_2} \left[\lambda_R \dot{R} + \lambda_1 \dot{\phi}_1 + \lambda_2 \dot{\phi}_2\right] = 0, \tag{4}$$

where λ_R, λ_1, λ_2 are the respective components of the gradient vector satisfying the adjoint equations:

$$\dot{\lambda}_R = (v_e \sin\phi_1 + v_p \sin\phi_2)\left(\frac{\lambda_1 + \lambda_2}{R}\right), \tag{5}$$

$$\dot{\lambda}_1 = \left(-\lambda_R \sin\phi_1 - \left(\frac{\lambda_1 + \lambda_2}{R}\right)\cos\phi_1\right)v_e, \tag{6}$$

$$\dot{\lambda}_2 = \left(-\lambda_R \sin\phi_1 - \left(\frac{\lambda_1 + \lambda_2}{R}\right)\cos\phi_1\right)v_p. \tag{7}$$

The optimal control for (4) is (cf. Appendix A for the derivations):

$$\sigma_{1\text{optimal}} = -\operatorname{sgn}\lambda_1(t). \tag{8}$$

In Appendix A, equations (A9a), (A12) and (A13) together give an evaluation of the optimal control based on the choice of λ_R and λ_1. By taking advantage of the line-of-sight coordinate model, we derived a closed-form solution.

Many optimal control problems are formulated as optimization problems, usually requiring large computation times for each iteration. To overcome such problems, Shinar [15] used a near optimal feedback control for the interception problem, and Järmark [16] used differential dynamic programming methodology. Although these methods have many advantages, the computation time is still significantly large for our PC-based application.

Computation of the optimal control law is highly dependent upon the choice of $[\lambda_{R0}, \lambda_{10}]$ (cf. Appendix A); different sets of values can cause very different values of final times. The use of an optimization algorithm for selecting these values is therefore essential. We used the so-called Box algorithm [19] for this purpose, which requires no derivatives (cf. Appendix B).

Another evasion tactic we implemented was an avoidance strategy based on Potential Field Theory [20,21]. The "common normal" vector, a perpendicular line which intersects both trajectories and represents the closest point of approach, T_e for evader, and T_p for the pursuer, is computed, i.e.,

$$h_{cn} = h_{\text{common normal}} = T_e \times T_p.$$

The norm of h_{cn} is

$$|h_{cn}|^2 = [(x_e - x_p)^2 + (y_e - y_p)^2 + (t_e - t_p)^2],$$

where (x_e, y_e, t_e) = point of intersection of T_e and the h_{cn}, and anticipated time of intersection, and (x_p, y_p, t_p) = point of intersection of T_p and the h_{cn}, and anticipated time of intersection.

Once computed, the common normal is transformed into a "repulsive force" used to alter the course of vehicle motion. This force transformation is modeled after the work of Khatib [20] and Steele [21], in which the vehicle is attracted to the destination by a normalized force, and repulsed from pursuers, using an inverse-cube law. A vector sum is then used to compute the resultant direction and speed of the vehicle. The magnitude and direction of the repulsive force used to guide the evader away from the pursuer is computed:

$$F_{rf} = \frac{(h_{cn})(k_{rf})}{(|h_{cn}|)^4},$$

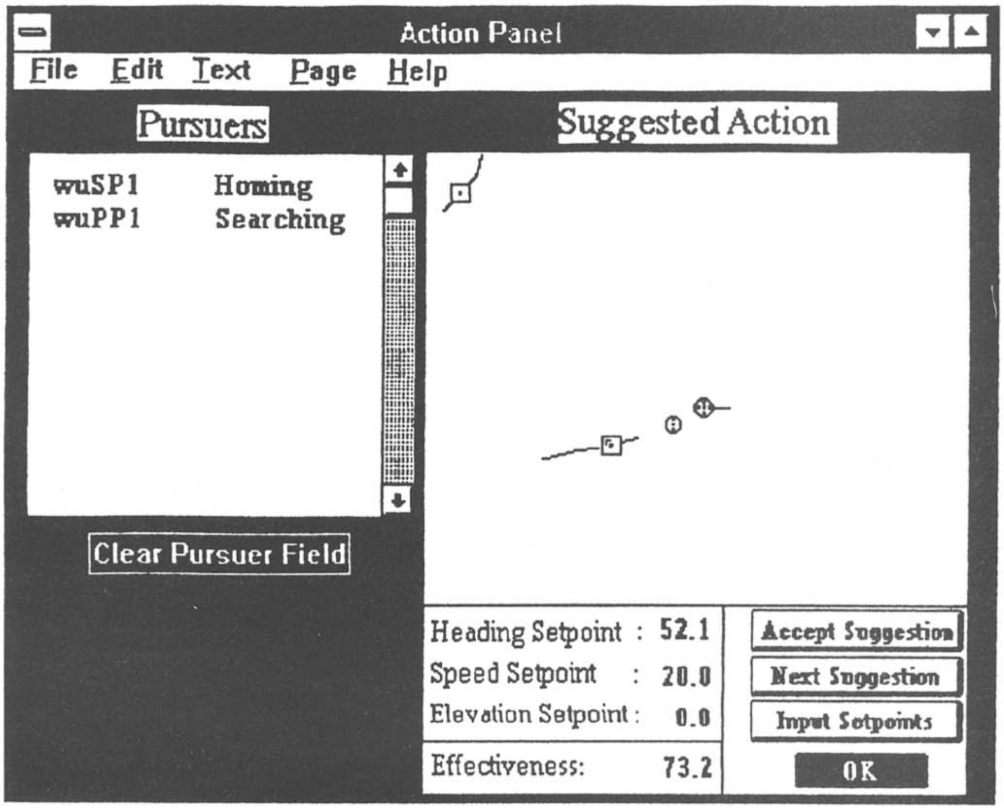
where k_{rf} is a variable gain used to provide the desired avoidance characteristic versus distance of separation.

5. ADAPTER

Man-Machine Interface

In order to increase situational awareness, a graphical interface exists which provides the user with the information generated by the game. This graphical interface was added to the existing program in a modular fashion, so that different display formats are used without affecting the remaining program structure. This modularity is accomplished by AssessSituation method, which calls a function named "UpdateScreen." This function contains all the commands used for updating the graphic display. The interface can be updated merely by loading in different versions of the function "UpdateScreen."

The Toolbook$^{\text{TM}}$ software package provides the necessary graphical interface for our development. Five different Toolbook$^{\text{TM}}$ screens are used to display various types of information. At each step of the game, the "UpdateScreen" function is called and displays the new information in each of the screens. The screens and the information provided by each are described below, and illustrated in Figures 12–20. The Action Panel (Figure 11) has a display window which shows a view of all the players in the body frame of the evader. A field also exists which lists pursuers and their current behavior (search, homing, etc.). The set points suggested by the rule base at each step are shown in the display window. Three buttons are located under this window which allow the user to accept or reject a suggested set of set points, or to input his own set points.



wuSP1 Homing
wuPP1 Searching

Heading Setpoint : 52.1
Speed Setpoint : 20.0
Elevation Setpoint : 0.0
Effectiveness: 73.2

Figure 11. Action panel.

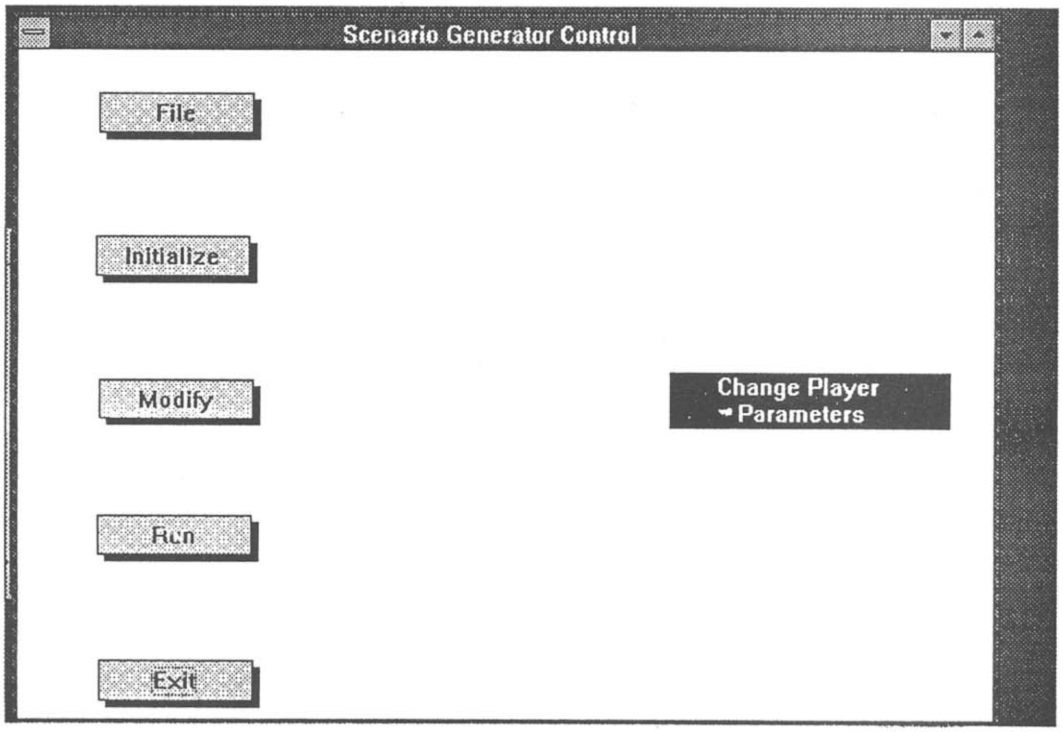
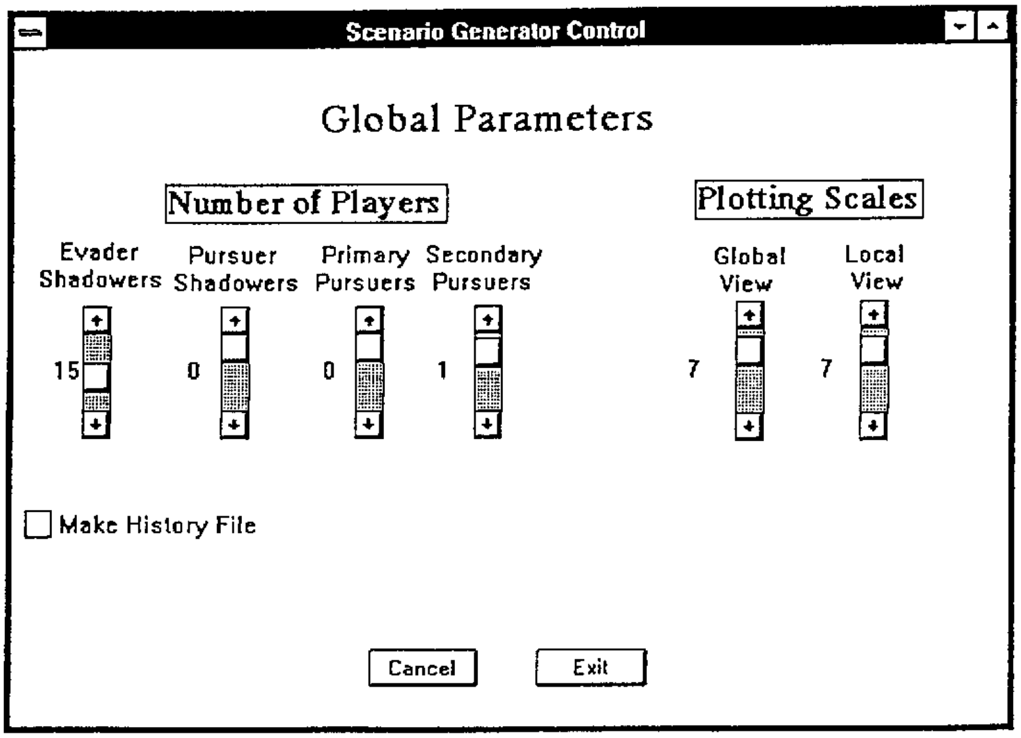


Figure 12. Initialization screen.


Scenario Generator Control

Global Parameters

Number of Players

Evader Shadowers 15

Pursuer Shadowers 0

Primary Pursuers 0

Secondary Pursuers 1

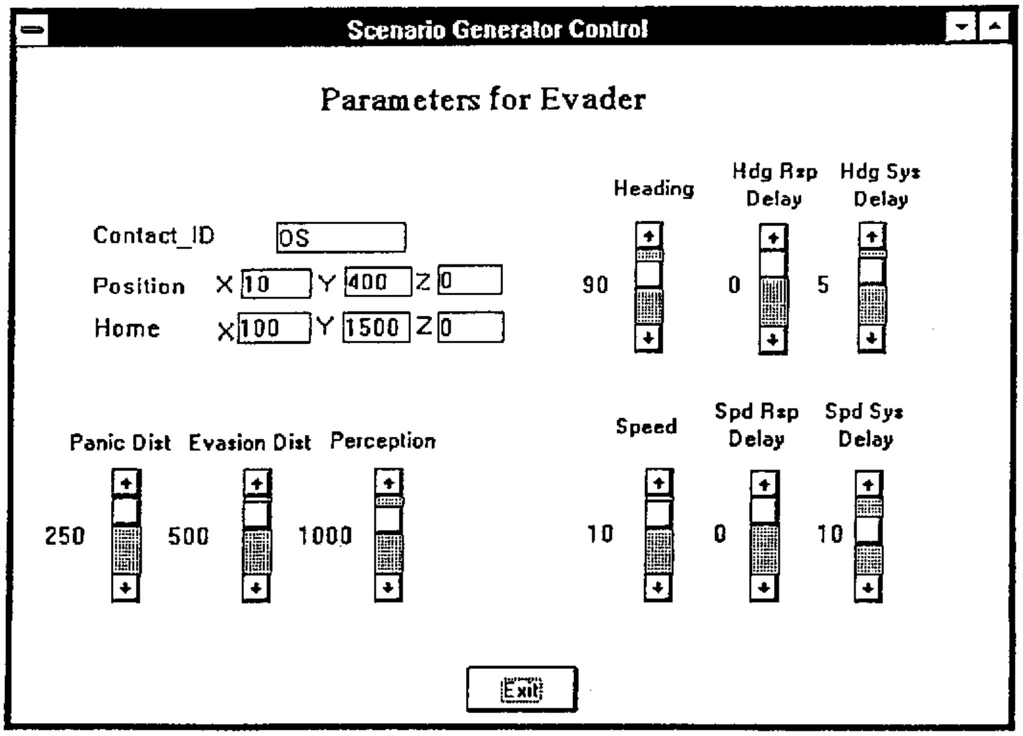
Plotting Scales

Global View 7

Local View 7

Make History File

Figure 13. Global parameters.


Scenario Generator Control

Parameters for Evader

Contact_ID OS

Position X 10 Y 400 Z 0

Home X 100 Y 1500 Z 0

Heading 90

Hdg Rsp Delay 0

Hdg Sys Delay 5

Panic Dist 250

Evasion Dist 500

Perception 1000

Speed 10

Spd Rsp Delay 0

Spd Sys Delay 10

Figure 14. Parameters for evader.

6. SCENARIO GENERATOR

Man-Machine Interface

The Initialization screen is used to initialize the parameters for the simulation. A picture of this screen is shown in Figure 12. The "Initialize" button takes the user to a screen called "Global Parameters" which allows him to determine how the simulation will be run (Figure 13). After these values are initialized and the user exits the screen, Toolbook^{TM} runs the KAPPA-PC^{TM} function "Initialize Game" which uses these values to initialize the proper values and create the appropriate players.

The "Modify" button can then be used to set the parameters for each created player. Clicking this button activates a menu which asks which type of player is to be modified. The user is then taken to the initialization screen for the type of player he selected. This screen contains all parameters pertinent to this type of player (Figures 14–16). After all parameters have been initialized as desired, the user exits the screen and Toolbook^{TM} sends a command to KAPPA-PC^{TM} to change the appropriate values. This process is repeated for each of the players the user wants to initialize.

Once all players have been initialized, the game is ready to be run. The "Run" button is used to start the simulation. Clicking this button initializes the display screens and sends a command to Kappa to run the "PlayGame" function. This function starts the simulation. In Toolbook^{TM}, the user is sent to the control panel window, where he can control the behavior of the simulation.

The Control Panel screen controls the running and appearance of the simulation. There are buttons to pause, restart, and halt the simulation. There is also a button for reinitializing the simulation once it has been halted. The user also chooses whether or not the perception, maximum safe distance, and panic distance circles are shown. Each of these options can be clicked on or off separately at any point during the simulation. Finally, this screen has two fields: one that displays all the data on the pursuers, and one that displays all the data on the evader and any shadowing players he has spawned. This information is updated at each step and sent to a scrollable field, so it can be reviewed at any time later (Figure 17).

The Local and Global View windows track the progress of the players in the body and inertial frame, respectively. In the global view, all pursuers within the evader's perception at a given time are displayed, along with a line segment which represents their past trajectory. For the evader, the current heading and heading setpoint are also displayed (Figure 18). If any active shadowing players exist, they are displayed and the region of protection they provide for the evader is shown as a gray area (Figure 19). The local view displays the same information in the body frame of the evader. The evader is always displayed in the center of the screen and all other items are moved relative to him (Figure 20).

Engagement Simulation

To simulate the interaction between the various players, we use an object oriented paradigm. This approach allows us to define a player class hierarchy through which attributes and behavior can be inherited by each player. The attributes identify static and time-varying data belonging to each player (see Table 1).

The pursuer-evader engagement is simulated by a time-driven, iterative process (Figure 21). There are one outer loop and two inner loops; the outer loop updates the state of the engagement, determines the players' reaction to the current state of the engagement, and updates the simulation time. The loop is controlled by a logical variable. This variable is initialized to FALSE at the beginning of the simulation. Later, it is set to TRUE if (1) the specified maximum simulation time is exceeded, (2) the evader reaches its destination, or (3) the pursuer scores a kill.

Within the context of the outer loop, the first inner loop is used to release shadowing players and to update the position of each player. The second inner loop is used to allow each player

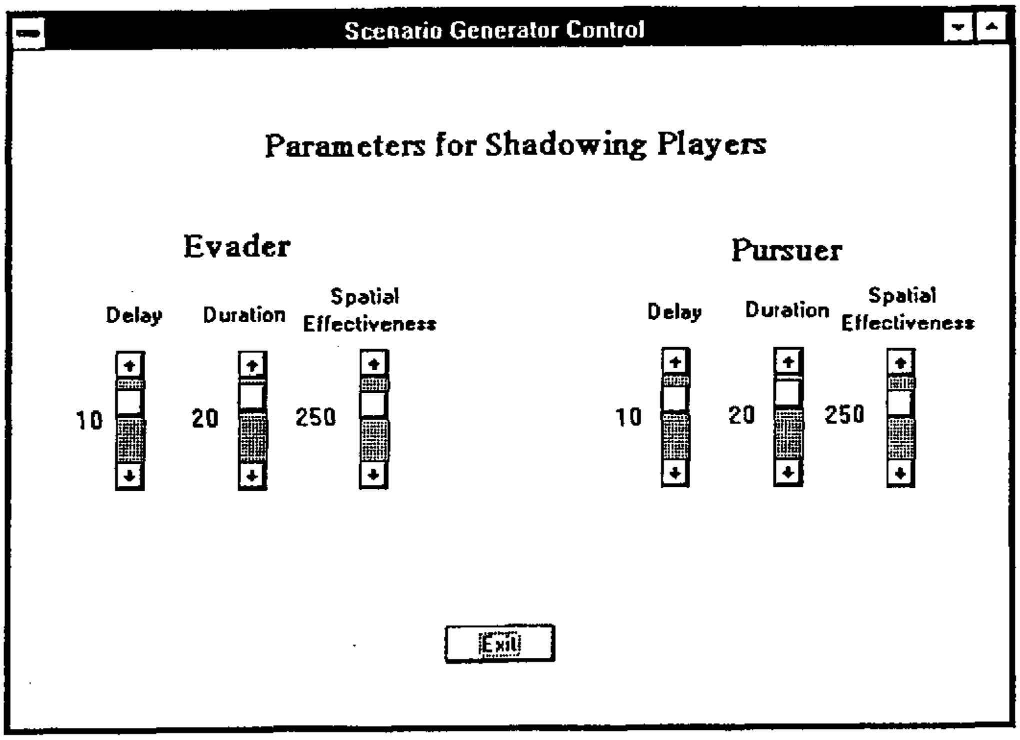


Figure 15. Parameters for shadowing players.

Table 1. Common attributes and behavior ascribed to the players.

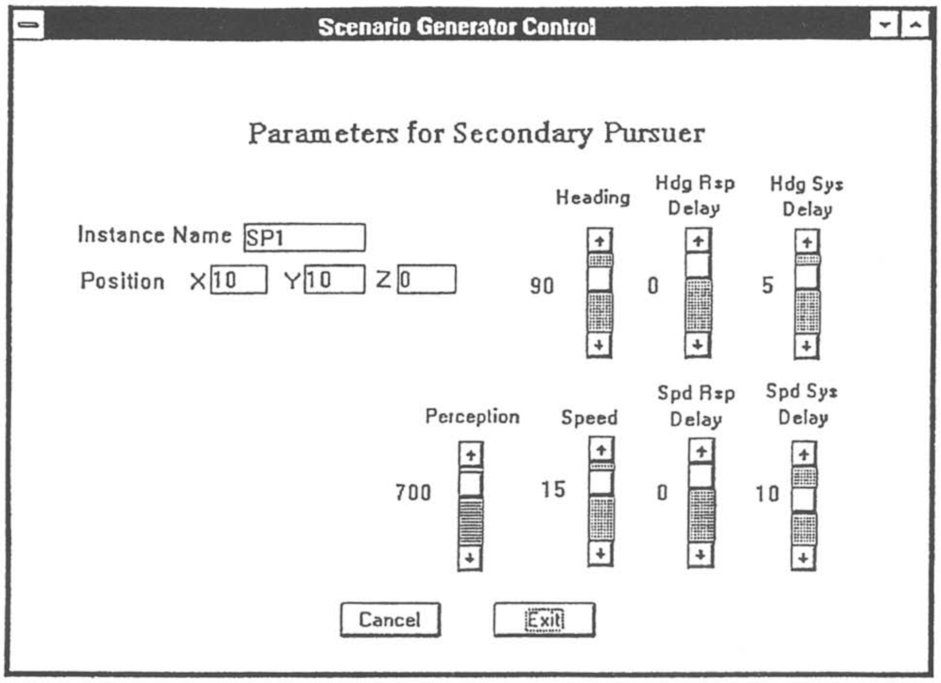
Attribute	Behavior
(N, E, D) Position	Initialize
(N, E, D) Velocity	Move
(N, E, D) Acceleration	Scan
(N, E, D) Destination	AssessSituation
(S, H, E) Setpoint	Spawn
(S, H, E) Position	Course?
(S, H, E) Time Constant	Home?
Maximum Speed	Kill?
Cruise Speed	ReportContact
Scan Interval	Report State
Perception Range	Report Parameters
(R, H, E) To Contact	
Spatial Effectiveness	
Temporal Effectiveness	

Legend: (N, E, D) ≡ (North, East, Down),
(S, H, E) ≡ (Speed, Heading angle, Elevation angle),
(R, H, E) ≡ (Range, Heading angle, Elevation angle).

to scan the scene, make an assessment of the situation, and lay out a new course in response to the assessment. All of these activities are accomplished by sending messages to each player. The particular response to each message is determined by the behavior ascribed to the player, which is inherited from his player class.

7. CONCLUSIONS

A pragmatic solution to differential game problems that involve multiple pursuers and a single evader has been developed using a semantic control approach. While such differential game


Scenario Generator Control

Parameters for Secondary Pursuer

Instance Name SP1

Position X 10 Y 10 Z 0

Heading 90

Hdg Rsp Delay 0

Hdg Sys Delay 5

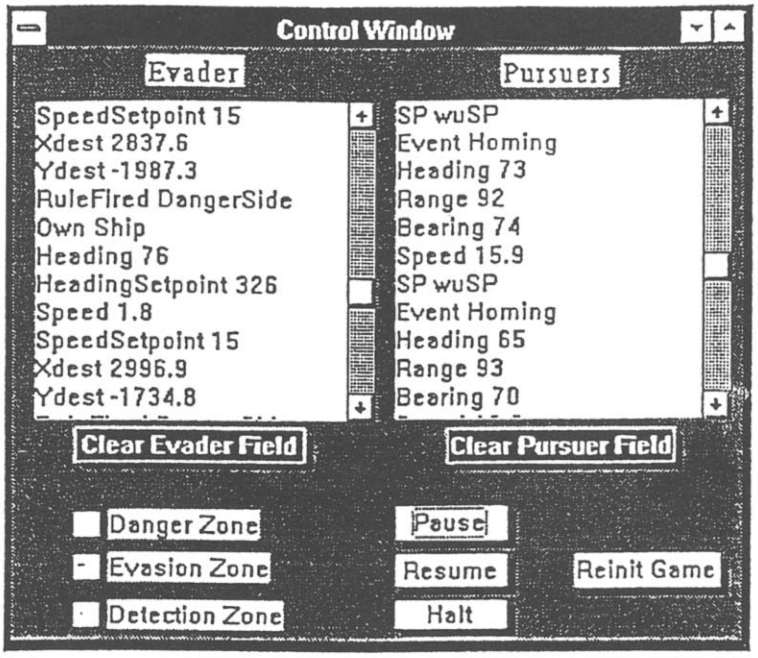
Perception 700

Speed 15

Spd Rsp Delay 0

Spd Sys Delay 10

Figure 16. Parameters for secondary pursuer.


Control Window

Evader

```
SpeedSetpoint 15
Xdest 2837.6
Ydest -1987.3
RuleFired DangerSide
Own Ship
Heading 76
HeadingSetpoint 326
Speed 1.8
SpeedSetpoint 15
Xdest 2996.9
Ydest -1734.8
```

Pursuers

```
SP wuSP
Event Homing
Heading 73
Range 92
Bearing 74
Speed 15.9
SP wuSP
Event Homing
Heading 65
Range 93
Bearing 70
```

Figure 17. Control panel.

problems have been addressed in the past, their solution has always been based on a determinate game, i.e., one in which the number of players does not change during the course of the game. Our approach transcends this earlier work by rendering a solution to dynamic games where the number of players change. The latter class of problems is of preeminent interest because it represents the norm of combat situations encountered by military aircraft, ground vehicles, and ocean-going vessels, where the use of weapons and countermeasures form an intrinsic part of an engagement.

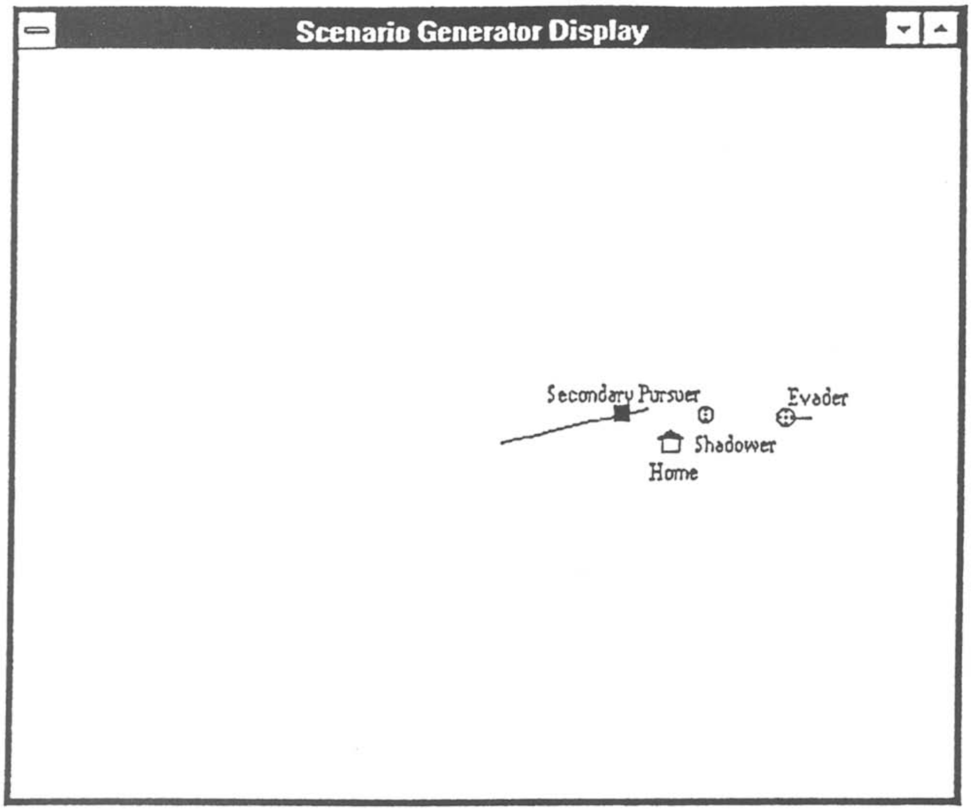


Figure 18. Global view.

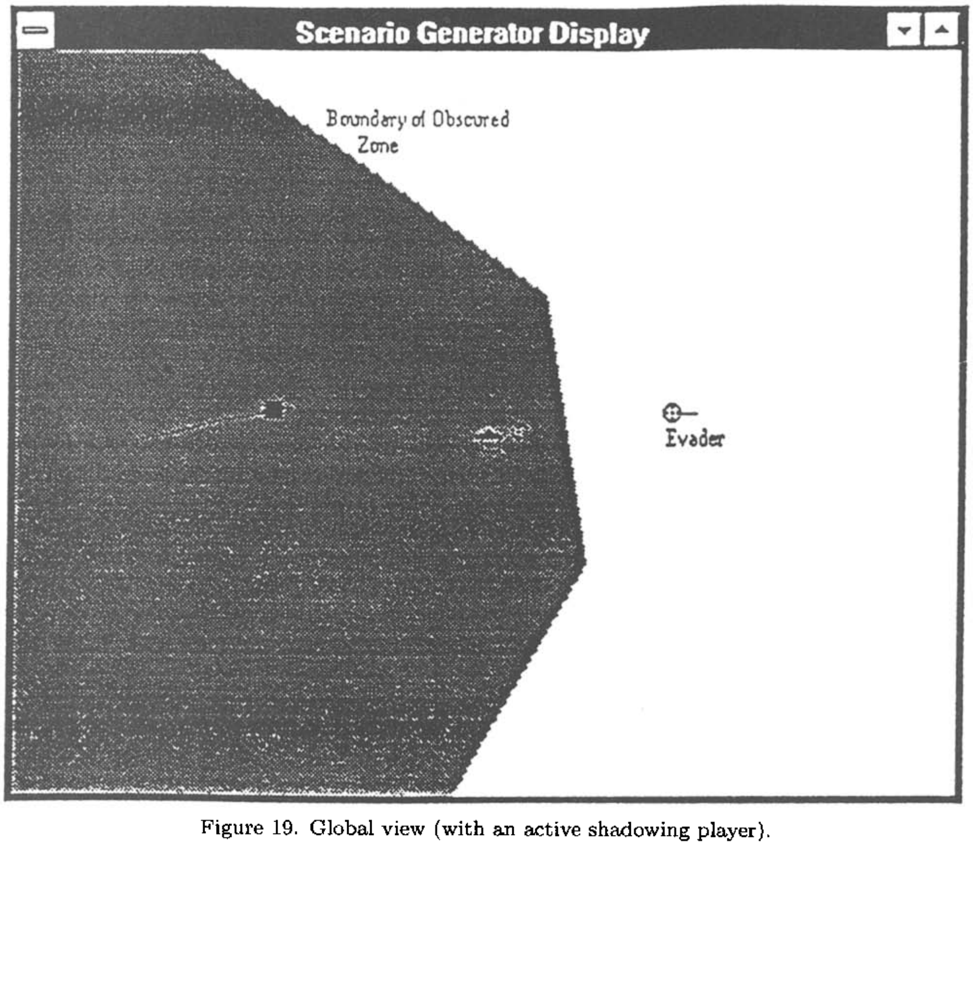


Figure 19. Global view (with an active shadowing player).

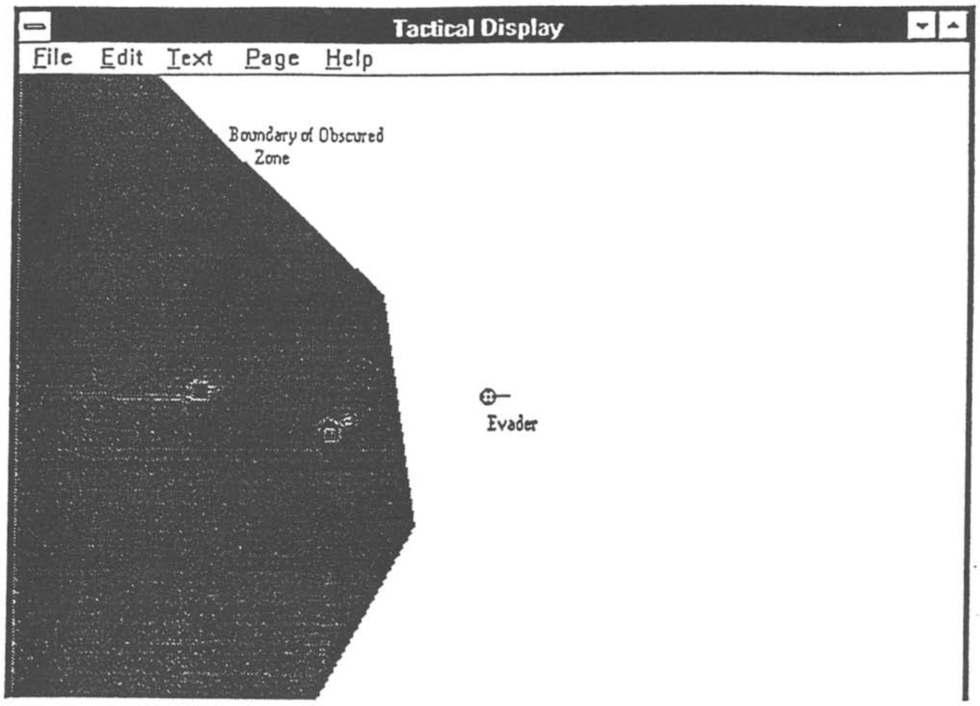


Figure 20. Evader's view (with an active shadowing player).

```
While ( Not( MissionEnded ) )
   {
   ForAll [ player | Players ]
      {
      SendMessage( player, Spawn );
      SendMessage( player, Move );
      };
   ForAll [ player | Players ]
      {
      player:ScanCount += 1;
      If player:ScanCount = player:ScanInterval
      Then
         {
         player:ScanCount = 1;
         SendMessage( player, Scan );
         SendMessage( player, AssessSituation );
         SendMessage( player, Course? );
         };
       };
    Time += Dt;
      };
```

Figure 21. The engagement is simulated as a time-driven, iterative process in which messages are sent to the players to evoke their ascribed behavior.

The objective of our work has been to devise a decision aid that helps an aircraft pilot, a tank commander, or a ship captain to make and affect defensive maneuver decisions that maximize the chances of surviving an encounter with an opponent. In this context, we have described the use of the hierarchical structure of a semantic controller as the basis for the decision aid. We also use object-oriented programming, knowledge-based systems, and the Analytical Hierarchy Process to embed the System Identifier and the Goal Selector into the decision aid. And, we use a graphical user interface to facilitate the decision maker's role as the adapter.

This interface provides a tactical situation display in a (North, East, Down) inertial coordinate frame centered at the own-vehicle's position. The display also shows the recommended vehicle heading and speed. The user can either accept, modify, or reject a recommendation. Every recommendation is reinforced by overlaying, on the tactical situation display, how previous actions, such as spawning a shadowing, affect the engagement.

To evaluate the operation of the decision aid, we built a scenario generator. This tool allows us to set up the initial conditions of the game. It also evolves the game using an object-oriented formulation. In this paper, we describe how the scenario generator serves to close the decision loop by taking a defensive maneuver decision and translating it into player motion and actions that are then fed back to the decision aid as sensor reports.

The scenario generator and the tactical decision aid were both implemented on a 16 MHz, 386DX (Dell 333D$^{\text{TM}}$) personal computer using the KAPPA-PC$^{\text{TM}}$ and ToolBook$^{\text{TM}}$ software packages. These packages operate by interpreting source code written in their respective native language. In its present form, the decision aid implementation would be adequate for vehicles where sensor reports occur at intervals of 10 seconds or more. Translation of the native source code into C code, followed by the traditional compilation and linking steps, is expected to improve the system performance considerably. Our goal is to process reports arriving at 1 second intervals. With the availability of 50MHz, 486DX processors, and array processor boards, we do not anticipate any major obstacles in confronting a system with the processing power needed to achieve this type of performance.

To focus our system enhancement efforts on those areas where major yet affordable improvements are needed, we are working on the definition of a set of performance metrics. Some of these metrics are pertinent only to the performance of the individual evasion algorithms, while others will be pertinent to the decision aid as a whole. The set of metrics and the evaluation results will be the subject of a later publication.

APPENDIX A

PROBLEM. Given σ_2, a control strategy for the pursuer, find an optimal control for the evader, i.e., $\sigma_{1\text{optimal}}$, where $|\sigma_i| \le 1,\ i = 1, 2$.

NECESSARY CONDITION. The necessary condition to be satisfied at every point of the barrier surface is:

$$\min_{\sigma_1} \max_{\sigma_2} \lambda_R \dot{R} + \lambda_1 \dot{\phi}_1 + \lambda_2 \dot{\phi}_2 = 0, \tag{A1}$$

where λ_R, λ_1, λ_2 are the respective components of the gradient vector satisfying the adjoint equations:

$$\dot{\lambda}_R = (v_e \sin\phi_1 + v_p \sin\phi_2)\left(\frac{\lambda_1 + \lambda_2}{R}\right), \tag{A2}$$

$$\dot{\lambda}_1 = \left(-\lambda_R \sin\phi_1 - \frac{\lambda_1 + \lambda_2}{R}\cos\phi_1\right) v_e, \tag{A3}$$

$$\dot{\lambda}_2 = \left(-\lambda_R \sin\phi_1 - \frac{\lambda_1 + \lambda_2}{R}\cos\phi_1\right) v_p. \tag{A4a}$$

It is easy to show that equations (A2)–(A4a) can be combined with equation (1) to obtain

$$\frac{d}{dt}\left[\lambda_R^2 + \left(\frac{\lambda_1 + \lambda_2}{R}\right)^2\right] = 0, \tag{A4b}$$

yielding an integral

$$\lambda_R^2 + \left(\frac{\lambda_1+\lambda_2}{R}\right)^2 = 1. \tag{A5}$$

Using (A5), equation (A2) can be written as

$$\begin{aligned}\dot\lambda_R &= (\sin\phi_1 + \sin\phi_2)\left(\frac{\lambda_1+\lambda_2}{R^2}\right) = (\dot\phi_1 - \sigma_1)\,R\left(\frac{\lambda_1+\lambda_2}{R^2}\right) = (\dot\phi_1 - \sigma_1)\,R\left(\frac{\lambda_1+\lambda_2}{R^2}\right)\\ &= (\dot\phi_1 - \sigma_1)(1-\lambda_R^2)^{1/2}.\end{aligned} \tag{A6a}$$

Likewise,

$$\dot\lambda_R = (1-\lambda_R^2)^{1/2}\,(\dot\phi_2 - \sigma_2). \tag{A6b}$$

Define in $0 \le t \le t_1$ for $\theta_1(t)$ as:

$$\theta_1(t) = -\phi_{10} - \int_0^t \sigma_1^*(\tau)\,d\tau, \tag{A6c}$$

where $0 \le t \le t_1$ and $\phi_{10} = \phi_1(0)$.

Assuming $\sigma_1^*(t) = \sigma_1^* =$ constant for $0 \le t \le t_1$, we have

$$\theta_1(t) = -\phi_{10} - \sigma_1^* t, \tag{A6d}$$

or

$$\dot\theta_1 = -\sigma_1^*. \tag{A7a}$$

Therefore,

$$\frac{\dot\lambda_R}{\sqrt{1-\lambda_R^2}} = \dot\phi_1 + \dot\theta_1, \tag{A7b}$$

and thus,

$$\arcsin\lambda_R = \phi_1 + \theta_1 + C_2, \tag{A8a}$$

where C_2 is an arbitrary constant.

Equation (A8a) becomes

$$\lambda_R = \sin(\phi_1 + \theta_1 + C_2). \tag{A8b}$$

Now for $t = 0$, we have

$$\lambda_{Rt} = \lambda_R(0) = \sin(\phi_1(0) + \theta_1(0) + C_2) = \sin(\phi_{10} - \phi_{10} + C_2) = \sin C_2. \tag{A8c}$$

Thus,

$$\lambda_{R0} = \sin C_2, \tag{A9a}$$

now

$$\sqrt{1-\lambda_R^2} = \cos(\phi_1 + \theta_1 + C_2). \tag{A9b}$$

Therefore,

$$\begin{aligned}\dot\lambda_1 &= -\lambda_R\sin\lambda_1 - \frac{\lambda_1+\lambda_2}{R}\cos\phi_1 = -\sin(\phi_1+\theta_1+C_2)\sin\phi_1 - \cos(\phi_1+\theta_1+C_2)\cos\phi_1\\ &= -\cos(\phi_1+\theta_1+C_2-\phi_1) = -\cos(\theta_1+C_2),\end{aligned} \tag{A10}$$

and

$$\lambda_1(t) = \frac{1}{\sigma_1^*}\sin(\theta_1(t) + C_2) = \frac{1}{\sigma_1^*}\sin(-\phi_{10} - \sigma_1^* t + C_2), \tag{A11}$$

where

$$\sigma_1^* = -\operatorname{sgn}[\lambda_{10}]. \tag{A12}$$

The optimal control for (A1) is

$$\sigma_{1\text{optimal}} = -\operatorname{sgn}\lambda_1(t). \tag{A13}$$

Equations (A9a), (A12), (A13) together give an evaluation of the optimal control based on the choice of λ_R and λ_1.

ALPHA-BETA TRACKER [22]

We must still guarantee that σ_2 can only be derived from a contact report (CR). However, this only gives the σ_2 at the instant of the CR. To compute the trajectory of the system, we need σ_2 for the duration of our computation, after the CR. We have used an alpha-beta tracker to predict the pursuer's control. This tracker is given by:

$$\begin{aligned} \sigma_2^s(k) &= \sigma_2^p(k) + \alpha[\sigma_2^0(k) - \sigma_2^p(k)], \\ v_s(k) &= v_s(k-1) + \frac{\beta}{T}[\sigma_2^0(k) - \sigma_2^p(k)], \\ \sigma_2^p(k+1) &= \sigma_2^s(k) + Tv_s(k), \end{aligned} \tag{A14}$$

where $\sigma_2^0(k)$ is the observation received at k, T is the CR period and α, β are the fixed-coefficient filter parameters.

The initialization here is

$$\begin{aligned} \sigma_2^s(1) &= \sigma_2^p(0) \\ &= \sigma_2^0(1), \end{aligned} \qquad v_s(1) = 0, \quad v_s(2) = \frac{\sigma_2^0(2) - \sigma_2^0(1)}{T}. \tag{A15}$$

The value of $\alpha \leq 0.6$ is chosen. The value of β is $\alpha^2/(2-\alpha)$. The optimal value of α is to be found by some trials. The line-of-sight coordinate system has one disadvantage. As the three state variables R, ϕ_1, ϕ_2 evolve, the system does not reflect the players' position in the inertial coordinate frame at any time t. But the desired output is the set-point heading, where north is fixed as the x-axis (Figure 22). The variables here are available as,

R = CR Range,	v_e = OS Speed,
ϕ_1 = CR Bearing,	v_p = CR Speed,
θ_E = Ownship (OS) Heading,	x_e = OS x-position,
θ_p = CR Heading,	y_e = OS y-position.

For convenience, θ_E and θ_p are converted to the scale $[-\pi, \pi]$ instead of a $[0, 2\pi]$ scale. The variables $\phi_2, (x_p, y_p)$ and θ_R are not supplied by the CR and OS data. In fact, ϕ_2 is one of the state variables.

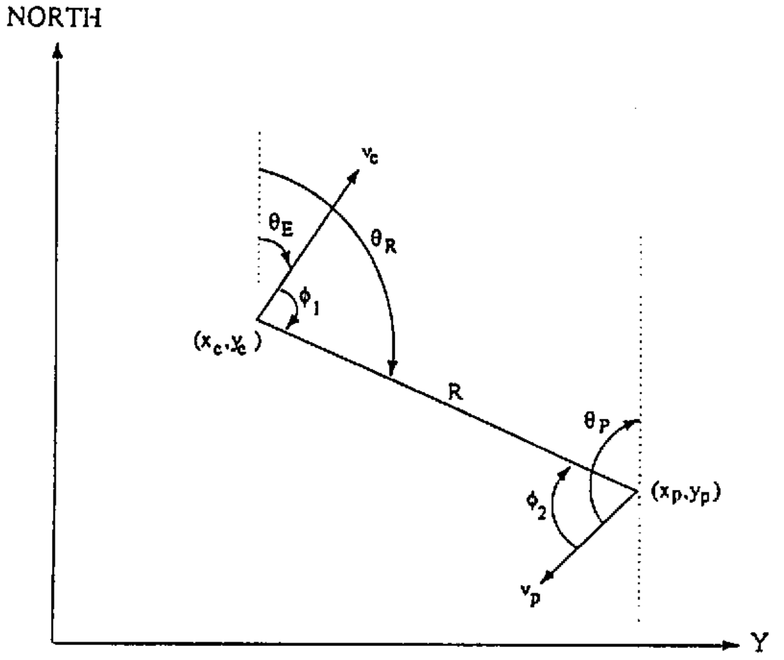


Figure 22. Two-player LOS coordinates.

Depending on the situation, ϕ_2 and θ_E are given by:

$$\begin{aligned} \phi_2 &= \theta_E + \phi_1 + \pi - \theta_p, & & \\ \theta_R &= 2\pi + \theta_E + \phi_1, & &\text{if } \theta_E < 0 \text{ and } \phi_1 < 0. \quad \text{(A16)} \\ \phi_2 &= \theta_E + \phi_1 - \pi - \theta_p, & & \\ \theta_R &= \theta_E + \phi_1, & &\text{otherwise.} \quad \text{(A17)} \end{aligned}$$

Then the pursuer's position is given by:

$$x_p = x_e + R\cos\theta_R, \quad \text{(A18)}$$
$$y_p = y_e + R\sin\theta_R. \quad \text{(A19)}$$

Next, the trajectory is updated in xy coordinates as:

$$x_e = x_e + hv_e\cos\theta_E, \quad \text{(A20)}$$
$$y_e = y_e\, hv_e\sin\theta_E, \quad \text{(A21)}$$
$$x_p = x_p + hv_p\cos\theta_p, \quad \text{(A22)}$$
$$y_p = y_p + hv_p\sin\theta_p. \quad \text{(A23)}$$

In Figure 22, the evader-pursuer's position can give rise to the line-of-sight representation in different quadrants of the xy plane, the new evader's heading is found as

$$\theta_E^{(\text{new})} = \tilde{\theta}_E - \phi_1^{(\text{new})}, \quad \text{(A24)}$$

where $\phi_1^{(\text{new})}$ is the state trajectory at the next time interval after integration of the state equation using the 4th order Runge-Kutta method and

$$\tilde{\theta}_E = \tan^{-1}\frac{y_p - y_e}{x_p - x_e}, \qquad x_p > x_e \text{ and } y_p > y_e; \quad \text{(A25)}$$
$$\tilde{\theta}_E = \tan^{-1}\frac{x_p - x_e}{y_e - y_p} + \frac{3\pi}{2}, \qquad x_p > x_e \text{ and } y_p < y_e; \quad \text{(A26)}$$
$$\tilde{\theta}_E = \tan^{-1}\frac{x_e - x_p}{y_p - y_e} + \frac{\pi}{2}, \qquad x_p < x_e \text{ and } y_p > y_e; \quad \text{(A27)}$$
$$\tilde{\theta}_E = \tan^{-1}\frac{y_e - y_p}{x_e - x_p} + \pi, \qquad \text{else.} \quad \text{(A28)}$$

IMPLEMENTATION

The code is written in the C language. The optimal control solution is implemented in a function called "game." The input parameters for this function are:

YG_1 = CR Range,	v_e = OS Speed,
YG_2 = CR Bearing,	v_p = CR Speed,
T = CR Period,	x_e = OS x-pos.,
θ_E = OS Heading,	y_e = OS y-pos.,
θ_p = CR Heading,	σ = estimate of pursuer's control from the alpha-beta tracker.

This function returns the heading set point of the evader as an integer (all the input arguments are floating point numbers).

APPENDIX B

BOX'S COMPLEX ALGORITHM [19]

The goal of the algorithm is to find the maximum/minimum of a multivariable nonlinear function subject to nonlinear equality constraints:

$$\begin{aligned} &\text{Maximize/minimize } F(x_1, x_2, \dots, x_N) \\ &\text{Subject to } G_k \le x_k \le H_k, && k = 1, 2, \dots, N. \\ &\qquad G_j \le f_j(x_1, x_2, \dots, x_N) \le H_j, && j = 1, \dots, M - N, \end{aligned}$$

where the variables $x_1, x_2, \dots, x_N$ are the explicit variables while f_j are the dependent functions of the variables $x_1, x_2, \dots, x_N$. The upper and lower constraints H_k and G_k are either constants or functions of the independent variables.

The algorithm proceeds as follows. An original "complex" of $K \ge N + 1$ points is generated consisting of a feasible starting point and $K-1$ additional points generated from random numbers and constraints for each of the independent variables:

$$X_{i,j} = G_i + r_{i,j}(H_i - G_i), \qquad i = 1, 2, \dots, N \quad \text{and} \quad j = 1, 2, \dots, K - 1,$$

where $r_{i,j}$ are random numbers between 0 and 1. The selected points must satisfy both the explicit and implicit constraints. If at any time the explicit constraints are violated, the point is moved a small distance (delta) inside the violated limit. If an implicit constraint is violated, the point is moved one-half the distance to the centroid of the remaining points

$$x_{i,j}^{(\text{new})} = \frac{x_{i,j}^{(\text{old})} + x_{i,c}}{2}, \qquad i = 1, 2, \dots, N.$$

This process is repeated as necessary until all the implicit constraints are satisfied. The objective function is evaluated at each point. The point having the lowest function value is replaced by a point located at a distance alpha times as far from the centroid of the remaining points as the distance of the rejected point on the line joining the rejected point and the centroid. The Box algorithm recommends a value of $\alpha = 1.3$.

If a point continues to give the lowest function value on consecutive trials, it is moved one-half the distance to the centroid of the remaining points. The new point is checked against the constraints and is adjusted as before if the constraints are violated.

Convergence is assumed when the objective function values at each point are within beta units for gamma consecutive iterations. An iteration is defined as the calculations required to select a new point which satisfies the constraints and does not continue to yield the lowest function value.

IMPLEMENTATION

In our problem, we have used the parameters shown below:

G_1	-9	beta	0.0001
H_1	9	alpha	1.3
G_2	-8	gamma	3
H_2	8	delta	0.01
K	4		

We have two independent variables; however, no constraint functions of explicit independent variables exist. A minimum, instead of a maximum, is selected by the algorithm. A problem arose in our implementation: the process was arrested at the centroid of the points other than the rejected point. A modification might be to compare the values at different points with the value at the centroid of all points, before the process begins. If the value at the centroid is the highest, we reselect the complex of starting points until the highest value does not occur at the centroid of all points.

REFERENCES

1. J. Shinar, Analysis of dynamic conflicts by techniques of artificial intelligence, INRIA Report, Sophia Antipolis, (1990).
2. E.Y. Rodin, Semantic control theory, *Appl. Math. Lett.* **1** (1), 73–78 (1988).
3. Y. Lirov, Artificial intelligence methods in decision and control systems, Ph.D. Thesis, Washington University, St. Louis, MO, (August 1987).
4. S.M. Amin, Intelligent prediction methodologies in the navigation of autonomous vehicles, Ph.D. Thesis, Washington University, St. Louis, MO, (January 1990).
5. R.D. Weil, AI methods in utilizing low dimensional models of differential games, Ph.D. Thesis, Washington University, St. Louis, MO, (September 1990).
6. D. Geist, Semantic control in continuous systems applications to aerospace problems, Ph.D. Thesis, Washington University, St. Louis, MO, (December 1990).
7. *Expert Choice*, Expert Choice, Inc., Pittsburgh, PA, (1991).
8. T.L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, New York, (1980).
9. R.W. Saaty, The analytic hierarchy process—What it is and how it is used, *Mathematical Modelling* **9**, 3–5 (1987).
10. P.T. Harker, The art and science of decision making: The analytic hierarchy process, In *The Analytical Hierarchy Process, Applications and Studies*, (Edited by B.L. Golden, E.A. Wasil and P.T. Harker), Springer-Verlag, Berlin, (1989).
11. W.R. Hughes, Deriving utilities using the analytic hierarchy process, *Socio-Economic Plann. Sci.* **20** (6) (1986).
12. F. Zahedi, A utility approach to the analytic hierarchy process, *Mathematical Modelling* **9**, 3–5 (1987).
13. L.G. Vargas and J.J. Dougherty III, The analytic hierarchy process and multicriterion decision making, *American Journal of Mathematical and Management Sciences* **2** (1) (1982).
14. A. Davidovitz and J. Shinar, Eccentric two-target model for qualitative air combat game analysis, *Journal of Guidance, Control and Dynamics* **8** (3), 325–331 (1985).
15. J. Shinar, K.H. Well and B. Järmark, Near-optimal feedback control for three-dimensional interceptions, Preprint of ICAS Paper No. 86-5.1.3, Presented at the *15th ICAS Congress*, London, UK, (1986).
16. B. Järmark, A missile duel between two aircraft, *Journal of Guidance* **8** (4), 508–513 (1985).
17. T.L. Vincent, D.J. Sticht and W.Y. Peng, Aircraft missile avoidance, *Operations Research* **24** (3) (May–June 1976).
18. A. Davidovitz and J. Shinar, Two-target game model of an air combat with fire-and-forget all-aspect missiles, *Journal of Optimization Theory and Applications* **63** (2), 133–165 (November 1989).
19. J.L. Kuester and J.H. Mize, *Optimization Techniques with FORTRAN*, McGraw-Hill, New York, (1973).
20. O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, *Robotics Research* **5** (1), 90–98 (Spring 1986).
21. J.P.H. Steele, Avoiding moving obstacles, Presented at the *IEEE Int'l Conference on Systems, Man, and Cybernetics*, Boston, MA, (November 1989).
22. S.S. Blackman, *Multiple-Target Tracking with Radar Applications*, Artech House, Massachusetts, (1986).