The Analysis of a Generic Air-to-Air Missile Simulation Model

Joseph A. Kaplan
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

Alan R. Chappell
Lockheed Engineering & Sciences Company
Hampton, Virginia

John W. McManus
Langley Research Center
Hampton, Virginia

June 1994

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23681-0001
Abstract

A generic missile model was developed to evaluate the benefits of using a dynamic missile fly-out simulation system versus a static missile launch envelope system for air-to-air combat simulation. This paper examines the performance of a launch envelope model and a missile fly-out model. The launch envelope model bases its probability of killing the target aircraft on the target aircraft’s position at the launch time of the weapon. The missile’s probability of kill is calculated by using either a mathematical function or a look-up table. This method presents several problems. The method does not account for any evasive maneuvers that the target aircraft attempts in order to maximize the miss distance of the missile to the aircraft. Most launch envelope implementations do not include the opponent’s closing rate and Line-of-Sight (LOS) rate in the probability of kill computation. The benefits gained from a launch envelope model are the simplicity of implementation and the minimal computational overhead required. The missile’s probability of kill is based strictly on the target’s position at launch time. A missile fly-out model takes into account the physical characteristics of the missile as it simulates the guidance, propulsion, and movement of the missile. The missile’s probability of kill is based on the missile miss distance (or the minimum distance between the missile and the target aircraft). This technique of modeling weapon launches rewards the target aircraft for any evasive maneuvers that it attempts in order to maximize the missile miss distance. The problems associated with this method of modeling are a larger computational overhead, the additional complexity required to determine the missile miss distance, and the additional complexity of determining the reason(s) the missile missed the target. This paper evaluates the two methods and compares the results of running each method on a comprehensive set of test conditions.

Nomenclature

- \( t \) \: Current time (seconds)
- \( t_{\text{BURN}} \) \: Time before fuel exhaustion (seconds)
- \( \Delta t \) \: Simulation time interval (seconds)
- \( t_{\text{impact}} \) \: Estimated time of missile’s closest approach to target (seconds)
- \( X_{\text{Target}} \) \: Target position on the X axis (meters)
- \( X_{\text{Missile}} \) \: Missile position on the X axis (meters)
- \( Y_{\text{Target}} \) \: Target position on the Y axis (meters)
- \( Y_{\text{Missile}} \) \: Missile position on the Y axis (meters)
- \( Z_{\text{Target}} \) \: Target position on the Z axis (meters)
- \( Z_{\text{Missile}} \) \: Missile position on the Z axis (meters)
- \( \Delta X \) \: Distance between \( X_{\text{Target}} \) and \( X_{\text{Missile}} \) (meters)
- \( \Delta Y \) \: Distance between \( Y_{\text{Target}} \) and \( Y_{\text{Missile}} \) (meters)
- \( \Delta Z \) \: Distance between \( Z_{\text{Target}} \) and \( Z_{\text{Missile}} \) (meters)
- RANGE \: Distance between the target and the missile (meters)
- RANGE\text{Estimated} \: Estimated range at \( t_{\text{impact}} \) (meters)
- \( D \) \: Drag affecting missile
- \( V \) \: Velocity (meters/second)
- \( A \) \: Acceleration (meters/second\(^2\))
- \( \Theta \) \: Pitch angle (radians)
\( \psi \)      Yaw angle (radians)
\( \dot{\phi} \)     Pitch Velocity (radians/second)
\( \psi \)      Yaw Velocity (radians/second)
\( q \)     Pitch rate (radians/second)
\( r \)     Yaw rate (radians/second)
\( q \) Pitch acceleration (radians/second\(^2\))
\( r \) Yaw acceleration (radians/second\(^2\))
\( \rho \) Density of air
\( s \) Surface area of missile
\( C_{D0} \) Coefficient of Drag
\( K \) Constant of Drag
\( w \) Weight of missile used in drag calculation (kilograms)
\( T \) Thrust of missile (newtons)
\( G \) Acceleration due to gravity (meters/second\(^2\))
\( M \) Mass of missile (kilograms)
\( M_{MAX} \) Mass of missile at launch time (kilograms)
\( M_{MIN} \) Mass of missile when fuel is expended (kilograms)
\( S_{\phi} \) Pitch Signal
\( S_{\psi} \) Yaw Signal
\( C_{MT} \) Missile time constant. Lag in control surfaces (seconds)
\( \text{LOS} \) Line-of-Sight (LOS) angle (radians)
\( \text{LOS}_{X} \) X component of the LOS angle (meters)
\( \text{LOS}_{YZ} \) YZ component of the LOS angle (meters)
\( C_{R} \) Closing rate between missile and target (meters/second)
\( C_{RX} \) Closing rate between missile and target on the x axis (meters/second)
\( C_{RY} \) Closing rate between missile and target on the y axis (meters/second)
\( C_{RZ} \) Closing rate between missile and target on the z axis (meters/second)
\( C_{V} \) Closing velocity between missile and target (meters/second)
\( V_{X_{\text{Desired}}} \) Desired velocity in the X axis to close with target (meters/second)
\( V_{Y_{\text{Desired}}} \) Desired velocity in the Y axis to close with target (meters/second)
\( V_{Z_{\text{Desired}}} \) Desired velocity in the Z axis to close with target (meters/second)
\( \alpha_{\text{Desired}} \) Desired pitch rate to close with target (radians/second)
\( \tau_{\text{Desired}} \) Desired yaw rate to close with target (radians/second)
Introduction

Modern air combat simulations must perform in a greatly expanded and rapidly changing tactical environment. Such a simulation system must be able to model new aircraft and their advanced capabilities. The system requires a modular software structure so that new weapon systems or aircraft subsystems (e.g., sensors or propulsion systems), modifications to aircraft control systems, or changes to the aircraft configuration can be easily incorporated. In support of the study of aircraft with enhanced maneuverability at Langley Research Center (LaRC), a Tactical Guidance Research and Evaluation System (TiGRES) was developed. The design and development of TiGRES as well as its relationship to past and current air combat simulation systems are described in detail in reference 1.

The TiGRES system allows researchers to develop and evaluate aircraft systems in a tactical environment. The three main components of TiGRES are a Tactical Decision Generator (TDG), the Tactical Maneuver Simulator (TMS), and the Differential Maneuvering Simulator (DMS). A TDG is an intelligent system that selects the combat maneuvers to perform throughout an air combat engagement. Both the TMS and the DMS use a TDG as the automated opponent. The Paladin TDG was developed specifically for the TiGRES research.

This paper presents two methods for modeling weapons systems within TiGRES. The first method evaluated utilizes a missile launch envelope model. The missile launch envelope model utilizes the range and LOS angle between the target aircraft and firing aircraft. The missile’s probability of kill is based on the range and LOS angle between the missile and the target aircraft at the time of missile launch. This technique of modeling weapon launches does not consider any evasive maneuvers that the target aircraft may attempt in order to maximize the missile miss distance. The second method evaluated is a missile fly-out model. The missile fly-out model includes the physical characteristics of the missile and simulates the guidance, propulsion, and movement of the missile. The missile’s probability of kill is based on the missile miss distance, the minimum distance between the missile and the target aircraft. This technique of modeling weapon launches rewards the target aircraft for any evasive maneuvers that it attempts while trying to maximize the missile miss distance.

The Paladin System

Paladin is a knowledge-based TDG implemented using Artificial Intelligence (AI) techniques and a large amount of information about aircraft dynamics, flight control, and air combat. The system provides insight into both the tactical benefits and the costs of enhanced maneuverability. Paladin uses an object-oriented programming approach to represent each aircraft in the simulation. Each aircraft object includes information on the current state of the aircraft’s offensive systems (e.g., guns, missile systems, fire control radar, etc.), defensive systems (e.g., electronic countermeasures, chaff, etc.), and propulsion system. This state information is used to help guide Paladin’s reasoning process.

Paladin models a combat engagement as a series of discrete decisions. Hence, at temporally regular decision points, the system must choose the “best” tactical maneuver to follow until the next decision point. To make this choice, Paladin uses information about its own state and estimated data about the opponent to calculate the relative geometry between the two aircraft. This relative geometry is used to perform a situation assessment and to select a new throttle position. After extrapolating the opponent’s state a short time into the future, Paladin generates a situationally dependent set of trial maneuvers and predicts a future engagement state for each trial maneuver. These future engagement states are passed through a group of scoring functions that
evaluate various aspects of the tactical situation. The results of the scoring functions are weighted, based on the mode of operation, to compute the current best maneuver. Paladin executes the selected maneuver to direct the aircraft until the next decision interval.

The weapons model used has a direct effect on the air-to-air combat results produced by TiGRES. At each simulation execution interval Paladin computes the relative geometry between the opposing aircraft. The weapons model then uses the relative geometry information to compute the current weapons solutions.

Engagement Scoring Data

Evaluating an engagement requires information on aircraft relative geometry and Paladin’s system status. This information is available in the form of participant-specific data maintained by Paladin. All data relating to the Paladin aircraft as well as Paladin sensor data (e.g., the opponent’s relative position) are assumed to be known exactly. Other data required about the opponent must be estimated.

The engagement scoring module uses quantities which are based on exactly known data specific to the Paladin aircraft or relative values from the Paladin aircraft’s point of view. Paladin’s current throttle position and altitude are parameters taken directly from the current state. Range is the magnitude of a vector connecting the centers of gravity of the aircraft. The LOS angle is defined as the angle between the LOS vector and the ownship body x-axis (see figure 1); the deviation angle is defined as the angle between the LOS vector and the ownship velocity vector; and the LOS angle off is defined as the angle between the LOS vector and the opponent’s body x-axis.

![Figure 1. Angle Definitions](image)

Figure 1. Angle Definitions
The deviation angle is calculated as the inverse cosine of the magnitude of the projection of the range onto the velocity vector divided by the range. In equation form,

\[
\text{deviation angle} = \arccos \left( \frac{\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}}{\sqrt{(\text{Range})^2 + (\text{Velocity})^2}} \right)
\]

(1)

The LOS angle is the inverse cosine of the magnitude of the projection of the range onto the x-body axis divided by the range, or,

\[
\text{LOS angle} = \arccos \left( \frac{D(i, 1)\Delta x + D(i, 2)\Delta y + D(i, 3)\Delta z}{\text{Range}} \right)
\]

(2)

where \(\Delta x\), \(\Delta y\), and \(\Delta z\) represent the difference between the two aircraft positions. \(D(i, j)\) is the i, j element of the Paladin body axis direction cosine matrix. Then the LOS elevation is taken to be the inverse sine of minus the opponent's z-coordinate in the Paladin body axis system divided by the range, or,

\[
\text{LOS elevation} = \arcsin \left( -\frac{z_{\text{opponent in Paladin body axis system}}}{\text{Range}} \right)
\]

(3)

The LOS azimuth is the inverse tangent of the opponent's y-coordinate divided by the opponent's x-coordinate, both in the Paladin body axis system.

\[
\text{LOS azimuth} = \arctan \left( \frac{y_{\text{opponent in Paladin body axis system}}}{x_{\text{opponent in Paladin body axis system}}} \right)
\]

(4)

The velocity, acceleration, and orientation of the opponent are estimated, since this data would not be available from sensors. The required data values are estimated using a three point time history of the known position data and several assumptions about the opponent aircraft (weight, wing surface area, and flight characteristics). The current position of the opponent and the opponent's position at the preceding two decision cycles are used to find a quadratic curve fit for the position as a function of time. The first and second derivatives of this function at the current time yield an estimation of the opponent's instantaneous velocity and acceleration. By assuming aerodynamic characteristics of the opposing aircraft, and using the velocity and acceleration estimates, an estimated body-axis orientation for the opponent can be found.

The quantities used by the engagement scoring module which are based on estimated data are largely relative values from the opponent aircraft's point of view. Each of these quantities has some error introduced by the estimation process. The range rate is the magnitude of the projection of the relative velocity onto the range axis (all in the inertial axis system).
The opponent's deviation angle and LOS angle are calculated similarly to the Paladin aircraft parameters (equations 1 and 2), using the opponent's velocity and x-body axis. Paladin's LOS angle off is defined as 180° - opponent's LOS angle. References X and X1 present an evaluation of the error magnitudes (absolute value of the actual value minus the estimated value) during the course of a typical engagement. If the aerodynamics of the opponent aircraft are not well known, the error in the LOS angle should increase, since this error is strongly dependent on the aircraft flight characteristics.

**Engagement Scoring Metrics**

Paladin currently uses four scoring metrics, each computed at the aircraft simulation update rate of 32 times per second, to evaluate each air combat engagement. The first metric consists of the total time that each airplane has its weapons locked on its opponent, the probability that any weapons fired will hit the opponent, the distance between the opponents, the angle-off, and the deviation angle. The results are printed in a table format at the completion of each run.

The second scoring metric computes a Probability of survival ($P_s$) using the data computed by the first metric. The probability to hit for an all-aspect missile and for the cannon are computed using the range and LOS angle to the opponent. The probability to hit for a tail-aspect missile is computed using the range, the LOS angle to the opponent, and the LOS angle off. Aircraft missiles are treated as limited resources and a probability to hit of 0.65 is required to launch the first missile. The probability to hit threshold increases by 0.05 for each missile launched. An estimated fly-out time (the time it will take a missile to reach its target) for each missile is computed based on the launch parameters, and another missile cannot be fired until the fly-out time has passed. The $P_s$ for an aircraft is:

$$P_s = 1.0 - \sum \text{[probability to hit} \times P_s(f)]$$

summing over each weapon fired by the opposing aircraft. $P_s(f)$ represents the $P_s$ of the aircraft firing the weapon at the time the weapon was fired.

The third scoring metric attempts to determine a Lethal Time (LT) advantage for each engagement. LT advantage attempts to weigh the lethality of each distinct type of weapons lock time.

$$LT = \frac{\text{Paladin Gun - Opponent Gun}}{2} + (2 \times (\text{Paladin Tail-Aspect - Opponent Tail-Aspect})) + (\text{Paladin All-Aspect - Opponent All-Aspect})$$

A positive LT value shows Paladin with a lethal time advantage, and a negative LT shows the opponent with an advantage.
Figure 2 - Graphical Representation of A Launch Envelope Model
The fourth metric is Time on Offense (TOF).

\[ \text{TOF} = (\text{Gun time} + \text{All-aspect time} + \text{Tail-aspect time}) \]

(8)

\( \Delta \text{TOF} \) is computed as Paladin’s TOF minus the opponent’s TOF. A positive \( \Delta \text{TOF} \) value shows Paladin with an time on offense advantage, and a negative \( \Delta \text{TOF} \) shows the opponent with a time on offense advantage.

**Missile Launch Envelope Method**

The missile launch envelope model considers the range and LOS angle between the target and firing aircraft at launch. The missile’s probability of kill is based on the range and LOS angle between the missile and the target aircraft at the time of missile launch. This technique of modeling weapon launches does not consider any evasive maneuvers that the target aircraft may attempt in order to maximize the missile miss distance. The problems associated with this method of modeling are the use of simplified scoring functions and the use of a “time of launch” probability of kill computation.

**Missile Fly-out Method**

The missile fly-out model includes the physical characteristics of the missile and simulates the guidance, propulsion, and movement of the missile. The missile’s probability of kill is based on the missile miss distance, the minimum distance between the missile and the target aircraft. This technique of modeling weapon launches rewards the target aircraft for any evasive maneuvers that it may attempt in order to maximize the missile miss distance. Some of the problems associated with this method of modeling are a larger computational overhead, additional complexity in determining the missile’s miss distance, and determining the reason(s) the missile missed the target. Missile miss distance may be caused by any combination of three reasons:

1) the missile failed to reach the target because of physical constraints (i.e., range, inability to quickly maneuver, etc.)
2) the missile’s line of sight limitations were exceeded
3) the target aircraft performed a maneuver that generated an optimal miss distance

**Model Characteristics**

Several constraints were placed upon the implementation of the missile model. These constraints included:

1) the model must exhibit realistic physical properties (e.g., account for gravity, drag, etc.)
2) the guidance and control system must operate realistically
3) the model must require minimal computational intensity
4) the software must possess cross-platform capabilities
5) the subroutine must be re-entrant so multiple missiles may be in-flight against multiple targets

A point-mass missile model was developed based on aerodynamic equations provided in Reference 9. The model is generic so any missile’s characteristic may be incorporated into it. The missile model selected for analysis is a generic radar guided missile with physical characteristics
similar to the AIM9 "Sidewinder." Table 1 presents the characteristics of the missile used for testing. The model takes into account the following physical features of the missile, including:

1) the missile can accelerate faster as the weight is reduced due to the rapid burning of fuel
2) the missile needs to accelerate to a sufficient velocity before it can begin to maneuver (i.e., the missile can not pull as many G's at low speeds as it can at high speeds)
3) the guidance controls in the missile use a proportional navigation system (i.e., they set a path to intercept the target, and not to track it)

<table>
<thead>
<tr>
<th>Table 1 - Physical Characteristics of the Generic Test Missile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight at Launch</td>
</tr>
<tr>
<td>Weight at Burnout</td>
</tr>
<tr>
<td>Thrust</td>
</tr>
<tr>
<td>Time of Motor Burn</td>
</tr>
<tr>
<td>Maximum Acceleration</td>
</tr>
<tr>
<td>Range</td>
</tr>
<tr>
<td>Proportional Navigation</td>
</tr>
</tbody>
</table>

Missile Simulation Computations

After initializing all variables with conditions suitable for launching the weapon, Paladin calls the missile subroutine. The first conditions computed are from the previous iteration. These conditions are used for the current iteration calculations.

The first variables to be computed are the differences in the X, Y, and Z axis. The ΔX, ΔY, and ΔZ all represent the difference between the missile's position and the target's position. Having computed the differences along each axis, the range is computed.

The difference in the X, Y, and Z axis, respectively, for the previous iteration in equation form is

\[ \Delta X_{t-1} = X_{\text{Target}_{t-1}} - X_{\text{Missile}_{t-1}} \]  
\[ \Delta Y_{t-1} = Y_{\text{Target}_{t-1}} - Y_{\text{Missile}_{t-1}} \]  
\[ \Delta Z_{t-1} = Z_{\text{Target}_{t-1}} - Z_{\text{Missile}_{t-1}} \]  

The range between the missile and the target for the previous iteration in equation form is

\[ \text{Range}_{t-1} = \sqrt{\Delta X_{t-1}^2 + \Delta Y_{t-1}^2 + \Delta Z_{t-1}^2} \]  

The subroutine then computes the forces which will effect the missile's flight path.

The missile's drag for the current iteration is computed by combining the profile drag and the drag due to lift drag. The drag, in equation form, is

\[ D_t = (K_1 \cdot V_{t-1}) + \frac{K_2 \cdot (q_{t-1}^2 + r_{t-1}^2)}{V_{t-1}^2} \]  

9
where

\[ K_1 = \frac{1}{2} \rho \cdot s \cdot C_{D_0} \]  
\[ K_2 = \frac{2K \cdot w^2}{\rho \cdot s} \] 

\( p, s, C_{D_0}, \) and \( w \) are all assumed to be constant in the drag computation. When the weight has changed significantly enough to alter the drag, the missile’s fuel will be exhausted. Having lost thrust, the missile will begin to quickly decelerate. This assumption was made to simplify the drag computations.

The missile’s acceleration for the current iteration is computed by subtracting the drag from the thrust and dividing by the mass. Acceleration due to gravity must then be subtracted from the result.

The computation of the acceleration, in equation form, is

\[ a_t = \frac{(T_{t-1} - D_{t-1})}{M_{t-1}} - G \cdot \sin(\Theta_{t-1}) \]  

The pitch and yaw accelerations are computed based upon the signals for the control surfaces minus pitch and yaw rates of the last iterations. These are then divided by the missile time constant representing the time necessary to move the missile’s control surfaces.

The pitch and yaw accelerations, in equation form, respectively, are

\[ \dot{\phi}_t = \frac{S_{\theta_{t-1}} - \dot{\theta}_{t-1}}{C_{MT}} \]  
\[ \dot{\psi}_t = \frac{S_{\psi_{t-1}} - \dot{\psi}_{t-1}}{C_{MT}} \] 

The pitch velocity is computed based upon the previous iteration’s pitch rate minus the cosine of the pitch angle. The result is then divided by the velocity to produce the pitch velocity.

The pitch velocity, in equation form, is

\[ \dot{\theta}_t = \frac{\dot{\phi}_{t-1} - \cos(\phi_{t-1})}{V_{t-1}} \] 

The yaw velocity is computed dividing the previous iteration’s yaw rate by the velocity multiplied by the cosine of the pitch.

The yaw velocity, in equation form, is

\[ \dot{\psi}_t = \frac{\dot{\psi}_{t-1}}{V_{t-1} \cdot \cos(\phi_{t-1})} \]
The velocities for the X, Y, and Z axis are computed. This is done by multiplying the velocity by the components of the pitch and yaw angles that contribute to motion in the particular axis.

The computation for velocity in the X, Y, and Z axis respectively, in equation form, are

\[ \dot{X}_t = V_{t-1} \cdot \cos(\Theta_{t-1}) \cdot \cos(\Psi_{t-1}) \]  
\[ \dot{Y}_t = V_{t-1} \cdot \cos(\Theta_{t-1}) \cdot \sin(\Psi_{t-1}) \]  
\[ \dot{Z}_t = V_{t-1} \cdot \sin(\Theta_{t-1}) \]  

The missile's velocity is computed by multiplying the missile's acceleration by the iteration rate and adding the result to the previous iteration's velocity.

The computation of the velocity, in equation form, is

\[ V_t = V_{t-1} + A_t \cdot \Delta t \]  

The pitch and yaw rates for the current iteration are computed by multiplying the angle rate by the iteration rate and adding the result to the previous iteration's rate.

The computation of the pitch and yaw rates, respectively, in equation form, are

\[ \dot{q}_t = \dot{q}_{t-1} + \dot{q}_t \cdot \Delta t \]  
\[ \dot{r}_t = \dot{r}_{t-1} + \dot{r}_t \cdot \Delta t \]  

The pitch and yaw angles for the current iteration are computed by multiplying the angle velocity by the iteration rate and adding the result to the previous iteration's angle.

The computation of the pitch and yaw angle, respectively, in equation form, are

\[ \Theta_t = \Theta_{t-1} + \Theta_t \cdot \Delta t \]  
\[ \Psi_t = \Psi_{t-1} + \Psi_t \cdot \Delta t \]  

The X, Y, and Z position of the missile are computed for the current iteration by multiplying the axis' velocity by the iteration rate and adding the result to the previous iteration's position.

The computation of the X, Y, and Z position, respectively, in equation form, are

\[ X_t = X_{t-1} + \dot{X}_t \cdot \Delta t \]  
\[ Y_t = Y_{t-1} + \dot{Y}_t \cdot \Delta t \]  
\[ Z_t = Z_{t-1} + \dot{Z}_t \cdot \Delta t \]
If the fuel of the missile has not been exhausted, the mass is updated to reflect the mass lost due to oxidation. This is done by subtracting the mass at fuel exhaustion from the mass at missile launch and dividing the result by the time required to reach fuel exhaustion. This value represents the amount of mass lost per second. The amount of mass lost per second is then multiplied by the iteration rate to give the amount of mass lost per iteration. This is then subtracted from the previous iteration's mass. The resultant value is the current iteration's mass.

The computation of the mass, in equation form, is

\[ M_t = M_{t-1} - \left( \frac{M_{\text{MAX}} - M_{\text{MIN}}}{t_{\text{BURN}}} \right) \Delta t \]  

(32)

The distances between the target and the missile along the axis are computed for the current iteration. This is done by subtracting the missile's position from the target's position.

The computation for the X, Y, and Z position, respectively, in equation form, are

\[ \Delta X_t = X_{\text{Target}} - X_{\text{Missile}} \]  

(33)

\[ \Delta Y_t = Y_{\text{Target}} - Y_{\text{Missile}} \]  

(34)

\[ \Delta Z_t = Z_{\text{Target}} - Z_{\text{Missile}} \]  

(35)

The range between the missile and the target for the current iteration in equation form is

\[ R_{\text{range}} = \sqrt{\Delta X_t^2 + \Delta Y_t^2 + \Delta Z_t^2} \]  

(36)

Having computed the new distances between the missile and the target, the LOS components are computed. This is done to determine whether the target has exceeded the limitation of the seeker's radar.

The computation for the X component of the LOS, in equation form, is

\[ \text{LOS}_X = |\Delta X_t| \]  

(37)

The computation for the YZ component of the LOS, in equation form, is

\[ \text{LOS}_{YZ} = \sqrt{\Delta Y_t^2 + \Delta Z_t^2} \]  

(38)

The LOS is computed by taking the inverse tangent of the YZ component of the LOS divided by the X component of the LOS.

The computation of the LOS, in equation form, is

\[ \text{LOS}_t = \text{ARCTAN}\left( \frac{\text{LOS}_{YZ}}{\text{LOS}_X} \right) \]  

(39)
Having computed the LOS angle between the missile and the target, the subroutine computes the closing rates as the missile tracks the target. This is done by subtracting the current iteration's distance between the missile and the target from the previous iteration's distance. The result is then multiplied by the iteration rate to give the closing rate for the current iteration.

The computation for the closing rates in X, Y, and Z, respectively, in equation form, are

\[ CR_X = (\Delta X_{t-1} - \Delta X_t) \times \Delta t \]  
(40)

\[ CR_Y = (\Delta Y_{t-1} - \Delta Y_t) \times \Delta t \]  
(41)

\[ CR_Z = (\Delta Z_{t-1} - \Delta Z_t) \times \Delta t \]  
(42)

The relative closing rate computation, in equation form, is

\[ CR = (RANGE_{t-1} - RANGE_t) \times \Delta t \]  
(43)

The closing velocity is the negation of the relative closing rate.

\[ CV = -CR \]  
(44)

Having computed all of the relative positions, the guidance algorithm of the missile can now be employed. The first computed variables are the desired velocities in the body axis that will lead to an intercept.

The computation of the desired velocities in the X, Y, and Z axis, respectively, presented in equation form, are

\[ V_{X_{\text{Desired}}} = \frac{(\Delta Y_t \times CR_Z) - (\Delta Z_t \times CR_Y)}{\sqrt{\Delta X_t^2 + \Delta Y_t^2 + \Delta Z_t^2}} \]  
(45)

\[ V_{Y_{\text{Desired}}} = \frac{(\Delta Z_t \times CR_X) - (\Delta X_t \times CR_Z)}{\sqrt{\Delta X_t^2 + \Delta Y_t^2 + \Delta Z_t^2}} \]  
(46)

\[ V_{Z_{\text{Desired}}} = \frac{(\Delta X_t \times CR_Y) - (\Delta Y_t \times CR_X)}{\sqrt{\Delta X_t^2 + \Delta Y_t^2 + \Delta Z_t^2}} \]  
(47)

With the desired velocities in the body axis computed, these are then transformed into equivalent desired pitch and yaw rates.

\[ q_{\text{Desired}} = -\sin(\Psi) \times V_{X_{\text{Desired}}} + \cos(\Psi) \times V_{Y_{\text{Desired}}} \]  
(48)

\[ r_{\text{Desired}} = (\cos(\Theta) \times V_{Y_{\text{Desired}}}) + \left[ \sin(\Theta) \times \left( \cos(\Psi) \times V_{X_{\text{Desired}}} + (\sin(\Psi) \times V_{Y_{\text{Desired}}}) \right) \right] \]  
(49)
The pitch and yaw signals are computed by multiplying the desired rates by the closing velocity and the navigational constant. The navigation constant adjust the signal so the missile will intercept the target instead of tracking it.

The computation of the pitch and yaw signals, respectively, in equation form, are

\[ S_\theta = C_{Nav} \ast CV \ast q_{Desired} \]  
\[ S_\psi = C_{Nav} \ast CV \ast r_{Desired} \]  

(50)  
(51)

**Missile Warhead Detonation Computations.**

The missile's estimated time of closest approach is calculated, along with the estimated range at the time of closest approach.

\[ t_{impact} = \frac{-(\Delta X \ast CV_X) + (\Delta Y \ast CV_Y) + (\Delta Z \ast CV_Z)}{CV_X^2 + CV_Y^2 + CV_Z^2} \]  

(52)

\[ \text{RANGE}_{\text{Estimated}} = \sqrt{(CV_X \ast t_{impact} + \Delta X_t)^2 + (CV_Y \ast t_{impact} + \Delta Y_t)^2 + (CV_Z \ast t_{impact} + \Delta Z_t)^2} \]  

(53)

By using the missile's detonation logic (see Appendix D), weapon detonation distance from the aircraft can be determined. One of three possible results are returned from the missile's detonation logic. The missile either missed the target, struck the target, or is still actively engaged in tracking the target.

**Missile Simulation Testing.**

To evaluate the model, extensive testing was performed. The launching aircraft was positioned at an altitude of 6,000 meters. A grid was set up in front of the launching aircraft that extended 5,000 meters down range, 4,000 meters to both sides of the aircraft, and 6,000 meters above and below the launching aircraft. Each of the axis were then broken down into 25 meter increments. Each point in the grid indicated a starting position for the target aircraft. The launching aircraft traveled down range in straight and level flight at a constant velocity of Mach 0.7. The target aircraft was also in straight and level flight at Mach 0.7. Mach 0.7 was chosen as the test velocity because the majority of tactical encounters occur in the subsonic range. The direction of travel for the target aircraft was away from the launching aircraft.

\[ \text{number of possible starting conditions} = \frac{5000}{25} \ast \frac{8000}{25} \ast \frac{12000}{25} \]

This works out to 30,720,000 possible starting positions for the target aircraft. The target aircraft was put into each of these starting position and the simulation was allowed to continue until one of three possible conditions occurred:

1) the seeker's line of sight limitation was exceed  
2) a non-negative closure rate was achieved between the missile and the target aircraft  
3) the missile scored a "hit" upon the target aircraft (a hit was denoted by the missile passing within 10 meters of the target aircraft)
Most modern air-to-air missiles inflict lethal damage upon their target by showering it with shrapnel. Detonation is achieved by passing within a predetermined distance of the target. Based upon various methods of producing shrapnel, 10 meters was judged to be an adequate distance to produce a mission kill. If a hit was scored, the starting position of the target aircraft was printed out to a file.

Several different simulations were run. The first denoted what areas the missile could hit by giving the seeker a perfect radar. This was done to determine the range and physical properties of the missile model. In Figure 3, a vertical slice was taken from the center of the volume generated by the simulation run. Several distinct features of this slice stand out. Each mark on the plot shows a starting position of the target aircraft that the missile was able to hit. The center of the slice forms a "peak" that leads to the position of the launching aircraft at zero X, zero Y, and 6,000 meters of altitude. This "peak" feature is primarily due to the fact that the missile cannot maneuver sufficiently at low speeds to hit the target aircraft. There exists a larger concentration of hits in the lower portion of the slice due to the effects of gravity on the missile.

Figure 3 - Vertical Slice of Missile Fly-Out Simulation
The simulations that followed incorporated placing a LOS limitation upon the seeker. If the target was beyond this LOS limitation, then the seeker lost sight of the target and failed to track it, thus ending the simulation. Since TiGRES currently implements a 30° launch parameter, a 15° LOS limitation was placed upon the seeker (i.e., the missile’s radar cone only extends 15° in any direction from a line projected out of the center of the seeker at the front of the missile). If this LOS limitation is rotated 360° in the X, Y, and Z axis, a 30° seeker cone in front of the missile is produced. This was done to see if TiGRES was over estimating the missile launch parameters. Running this simulation with a 15° LOS limitation produced a trumpet shaped cone (see Figure 3). The cone in Figure 4 was produced by running the simulation at an iteration rate of 32 times a second and taking a horizontal slice from the generated volume at the altitude of the launching aircraft. This cone closely resembles the shape of the launch envelope model. The narrow beginning of the cone was due to the missile’s low speed at launch. In Figure 5 and Figure 6, the simulation was run at iteration rates of 64 and 128 times per second, respectively. The higher iteration rates allow the model to predict more accurately the missile’s closest approach to the target aircraft. If the model estimates the missile will pass or overtake the target before the next iteration, it will detonate the warhead at the point of closest approach. If the model is run at lower iteration rates, it cannot predict accurately the closest approach distance. To determine whether a larger LOS limitation would affect performance, a 20 and 30° LOS limitation were tested. These limitations produced 40° and 60° cones, respectively. These simulations continued to reveal wider, yet similar, trumpet shaped cones as shown in Appendix E. Since the missile was tested against a non-maneuvering target, evasive maneuvering will raise the chance that the pilot can escape the kill range of the missile. The use of a non-maneuvering target in testing was done to insure the robustness of the missile model.

The missile’s seeker relies on the target’s relative position (X, Y, and Z). This is the only information that the launching aircraft is given about the target aircraft, so it is realistic to assume the missile will have access to similar information. The model is implemented as a re-entrant FORTRAN subroutine, so multiple missiles can be in-flight against multiple targets.

Conclusions

The results of this study show that a robust missile launch model envelope model will produce realistic results based on a non-maneuvering target. Simulations that include an active target require the additional features of a missile fly-out model to produce accurate results. The missile fly-out model considers the physical characteristics of the missile as it simulates its guidance system, propulsion, and movement. The missile’s probability of kill is based on the missile miss distance, not the missile launch conditions. This technique of modeling weapon launches rewards the target aircraft for attempting evasive maneuvers to maximize the missile miss distance. The problems associated with this method of modeling are a larger computational overhead, the additional complexity required to determine the miss distance, and reason(s) why the missile missed the target. Achieving the maximum benefit of the fly-out model requires providing the target aircraft with visual or sensor information when the opponent launches a missile. This information allows the target to actively perform evasive maneuvers. Further research in this area requires adding missile avoidance logic to Paladin.
References


Appendix A:

Format for Generic Missile Model

The model is implemented as a re-entrant subroutine. It has no concept of which missile it is calculating, the missile's target, or time. All these things must be passed to it. The model is just a single iteration calculator. By giving it the previous iteration's data, it can continue to calculate a course for the missile. The model calculates everything in metric units. If your program uses English units, several variables must be converted at missile initialization. The target's relative position (X, Y, and Z), old and new, must be converted every time.

Definition of Variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VELCTY</td>
<td>REAL</td>
<td>Velocity meters/second</td>
</tr>
<tr>
<td>PITCHR</td>
<td>REAL</td>
<td>Pitch rate radians/second</td>
</tr>
<tr>
<td>PTHSIG</td>
<td>REAL</td>
<td>Pitch signal radians/second**2</td>
</tr>
<tr>
<td>YAWR</td>
<td>REAL</td>
<td>Yaw rate radians/second</td>
</tr>
<tr>
<td>YAWSIG</td>
<td>REAL</td>
<td>Yaw signal radians/second**2</td>
</tr>
<tr>
<td>PITCH</td>
<td>REAL</td>
<td>Pitch radians</td>
</tr>
<tr>
<td>YAW</td>
<td>REAL</td>
<td>Yaw radians</td>
</tr>
<tr>
<td>X</td>
<td>REAL</td>
<td>Earth-based x in meters</td>
</tr>
<tr>
<td>Y</td>
<td>REAL</td>
<td>Earth-based y in meters</td>
</tr>
<tr>
<td>Z</td>
<td>REAL</td>
<td>Earth-based z in meters</td>
</tr>
<tr>
<td>MASS</td>
<td>REAL</td>
<td>Weight pounds</td>
</tr>
<tr>
<td>TARX</td>
<td>REAL</td>
<td>Target aircraft's x position</td>
</tr>
<tr>
<td>TARY</td>
<td>REAL</td>
<td>Target aircraft's y position</td>
</tr>
<tr>
<td>TARZ</td>
<td>REAL</td>
<td>Target aircraft's z position</td>
</tr>
<tr>
<td>SEC</td>
<td>INTEGER</td>
<td>Iteration # in current second</td>
</tr>
<tr>
<td>ITS</td>
<td>INTEGER</td>
<td># of iterations per second</td>
</tr>
<tr>
<td>ITMIN</td>
<td>INTEGER</td>
<td>Iteration # of iterations per second</td>
</tr>
<tr>
<td>HIT</td>
<td>INTEGER</td>
<td>1 or 0</td>
</tr>
<tr>
<td>MISS</td>
<td>INTEGER</td>
<td>1 or 0</td>
</tr>
<tr>
<td>OTARX</td>
<td>REAL</td>
<td>Target aircraft's x position</td>
</tr>
<tr>
<td>OTARY</td>
<td>REAL</td>
<td>Target aircraft's y position</td>
</tr>
<tr>
<td>OTARZ</td>
<td>REAL</td>
<td>Target aircraft's z position</td>
</tr>
<tr>
<td>RANGE</td>
<td>REAL</td>
<td>Meters</td>
</tr>
<tr>
<td>RANGE</td>
<td>REAL</td>
<td>Course meters</td>
</tr>
</tbody>
</table>

Initialization of Variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VELCTY</td>
<td>Initialize to launching aircraft’s velocity</td>
</tr>
<tr>
<td>PITCHR</td>
<td>Initialize to 0.0</td>
</tr>
<tr>
<td>PTHSIG</td>
<td>Initialize to 0.0</td>
</tr>
<tr>
<td>YAWR</td>
<td>Initialize to 0.0</td>
</tr>
<tr>
<td>YAWSIG</td>
<td>Initialize to 0.0</td>
</tr>
<tr>
<td>PITCH</td>
<td>Initialize to launching aircraft’s alpha</td>
</tr>
<tr>
<td>YAW</td>
<td>Initialize to launching aircraft’s heading</td>
</tr>
<tr>
<td>X</td>
<td>Initialize to launching aircraft’s X position</td>
</tr>
<tr>
<td>Y</td>
<td>Initialize to launching aircraft’s y position</td>
</tr>
<tr>
<td>Z</td>
<td>Initialize to launching aircraft’s z position</td>
</tr>
<tr>
<td>MASS</td>
<td>Initialize to 56.7 kilograms</td>
</tr>
<tr>
<td>TARX</td>
<td>Initialize to target aircraft’s X position</td>
</tr>
<tr>
<td>TARY</td>
<td>Initialize to target aircraft’s Y position</td>
</tr>
<tr>
<td>TARZ</td>
<td>Initialize to target aircraft’s Z position</td>
</tr>
<tr>
<td>SEC</td>
<td>Initialize to 0</td>
</tr>
</tbody>
</table>

21
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITS</td>
<td># of iterations in current sec</td>
<td>initialize to 0</td>
</tr>
<tr>
<td>ITMIN</td>
<td># of iterations per sec</td>
<td>initialize to 32 (if you are running at 32/sec)</td>
</tr>
<tr>
<td>HIT</td>
<td>hit determiner</td>
<td>initialize to 0 (will change to 1 if hit scored)</td>
</tr>
<tr>
<td>MISS</td>
<td>miss determiner</td>
<td>initialize to 0 (will change to 1 if miss scored)</td>
</tr>
<tr>
<td>OTARX</td>
<td>X pos of target last iteration</td>
<td>initialize to target's old X pos</td>
</tr>
<tr>
<td>OTARY</td>
<td>Y pos of target last iteration</td>
<td>initialize to target's old Y pos</td>
</tr>
<tr>
<td>OTARZ</td>
<td>Z pos of target last iteration</td>
<td>initialize to target's old Z pos</td>
</tr>
<tr>
<td>RANGE</td>
<td>distance of missile &amp; target</td>
<td>initialize to -1.0</td>
</tr>
</tbody>
</table>

After all the variables have been initialized, the following variables must be updated every iteration:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARX</td>
<td>X position of target</td>
<td>set to target aircraft's X position</td>
</tr>
<tr>
<td>TARY</td>
<td>Y position of target</td>
<td>set to target aircraft's Y position</td>
</tr>
<tr>
<td>TARZ</td>
<td>Z position of target</td>
<td>set to target aircraft's Z position</td>
</tr>
<tr>
<td>OTARX</td>
<td>X pos of target last iteration</td>
<td>set to target's X pos from previous iteration</td>
</tr>
<tr>
<td>OTARY</td>
<td>Y pos of target last iteration</td>
<td>set to target's Y pos from previous iteration</td>
</tr>
<tr>
<td>OTARZ</td>
<td>Z pos of target last iteration</td>
<td>set to target's Z pos from previous iteration</td>
</tr>
<tr>
<td>SEC</td>
<td>Seconds of missile flight</td>
<td>set to target's old X pos, Y pos, Z pos</td>
</tr>
<tr>
<td>ITS</td>
<td>Number of iterations in current second</td>
<td>set to target's old X pos, Y pos, Z pos</td>
</tr>
</tbody>
</table>

Constants in program:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAVC</td>
<td>navigational constant</td>
<td>4</td>
</tr>
<tr>
<td>PTHMAX</td>
<td>maximum pitch acceleration</td>
<td>30 G</td>
</tr>
<tr>
<td>YAWMAX</td>
<td>maximum yaw acceleration</td>
<td>30 G</td>
</tr>
<tr>
<td>TIMRNG</td>
<td>distance to fuse warhead at 100 meters</td>
<td></td>
</tr>
<tr>
<td>DETRNG</td>
<td>detonation range (hit score)</td>
<td>.261798 radians (15 degrees)</td>
</tr>
<tr>
<td>SEEKER</td>
<td>seeker limitation</td>
<td></td>
</tr>
<tr>
<td>MXTIME</td>
<td>time of missile burn</td>
<td>8 seconds</td>
</tr>
<tr>
<td>MINWGT</td>
<td>minimum weight of missile</td>
<td>22.7 kilograms</td>
</tr>
<tr>
<td>MAXWGT</td>
<td>maximum weight of missile</td>
<td>56.7 kilograms</td>
</tr>
<tr>
<td>MAXTRT</td>
<td>Thrust during motor burn</td>
<td>6800 N</td>
</tr>
<tr>
<td>MINTRT</td>
<td>Thrust after motor burnout</td>
<td>0 N</td>
</tr>
<tr>
<td>G</td>
<td>Gravitational constant</td>
<td>9.8 meters per second</td>
</tr>
<tr>
<td>OPVEL</td>
<td>Optimal velocity of missile</td>
<td>700 meter per second</td>
</tr>
</tbody>
</table>

The managing software must keep track of the missiles in flight. It will also be responsible for updating the time of flight and the target for each missile. At every iteration, add one to the ITS variable until ITMIN is reached, then reset it to zero and add one to the SEC variable.
Appendix B - Test Program

PROGRAM TMS_MISS_CALC
IMPLICIT NONE

REAL VELCTY,PITCHR,YAWR,PITCH,YAW
REAL X,Y,Z,MASS,RANGE

INTEGER LOOP1,LOOP2
REAL TARX,TARY,TARZ

INTEGER ITMIN
PARAMETER (ITMIN = 128)

INTEGER STEP
REAL TARGTX,TARGTY,TARGTZ
REAL PTHSIG,YAWSIG
INTEGER TOPX,TOPY, TOPZ
INTEGER BOTX,BOTY,BOTZ
PARAMETER (TOPX = 4500)
PARAMETER (TOPY = 4000)
PARAMETER (TOPZ = 12000)
PARAMETER (BOTX = 0)
PARAMETER (BOTY = -4000)
PARAMETER (BOTZ = 2000)
PARAMETER (STEP = 25)
INTEGER HIT,MISS
REAL OTARX,OTARY,OTARZ

OPEN(UNIT = 10,FILE = 'TESTCONE',STATUS = 'NEW')
DO 3000 TARGTY=BOTY,TOPY,STEP
   DO 2000 TARGTZ=-BOTZ,-TOPZ,-STEP
      PRINT *,'TARGTZ = ',TARGTZ,'TARGTX = ',TARGTX
      DO 1000 TARGTX=BOTX,TOPX,STEP
         VELCTY = 200.0
         PITCHR = 0.0
         YAWR = 0.0
         PTHSIG = 0.0
         YAWSIG = 0.0
         PITCH = 0.0
         YAW = 0.0
         X = 0.0
         Y = 0.0
         Z = -6000.0
         MASS = 56.7
         TARX = TARGTX
         TARY = TARGTY
         TARZ = TARGTZ
         HIT = 0
         MISS = 0
         RANGE = -1

23
DO 200 LOOP1 = 1,30  
DO 100 LOOP2 = 1,ITMIN  
OTARX = TARX  
OTARY = TARY  
OTARZ = TARZ  
CALL GETTAR(TARX,TARY,TARZ,ITMIN)  
CALL CALCULATIONS(VELCTY,PITCHR,THSIG,YAWR,YAWSIG,  
+ PITCH,YAW,X,Y,Z,  
+ MASS,TARX,TARY,TARZ,LOOP1,LOOP2,ITMIN,  
+ HIT,MISS,OTARX,OTARY,OTARZ,RANGE)  
75 FORMAT(F8.2,'',F8.2,'',F8.2)  
IF (HIT .EQ. 1) THEN  
WRITE(10,75) TARGTX,TARGTY,-TARGTZ  
HIT = 0  
MISS = 0  
GOTO 1000  
END IF  
IF (MISS .EQ. 1) THEN  
MISS = 0  
HIT = 0  
GOTO 1000  
END IF  
100 CONTINUE  
200 CONTINUE  

1000 CONTINUE  
2000 CONTINUE  
3000 CONTINUE  

END
**Appendix C - Missile Model**

```plaintext
SUBROUTINE CALCULATIONS(VELCTY,PITCH,PTHSIG,YAWR,YAWSIG,
  + PITCH,YAW,
  + X,Y,Z,MASS,TARX,TARY,TARZ,SEC,
  + ITS,ITMIN,HIT,MISS,OTARX,OTARY,OTARZ,
  + RANGE)

C* Name - Subroutine Calculations
C* Purpose - Implements a missile simulation for the
  * Paladin tactical decision generator.
C* Variables -
C* NAME TYPE     UNITS
C*---------------------------------------------------------
C* VELCTY  REAL  meters/second
C* PITCHR REAL   radians/second
C* PTHSIG REAL   radians/second**2
C* YAWR   REAL   radians/second
C* YAWSIG REAL   radians/second**2
C* PITCH  REAL   radians
C* YAW    REAL   radians
C* X      REAL   earth-based x in meters
C* Y      REAL   earth-based y in meters
C* Z      REAL   earth-based z in meters
C* MASS   REAL   pounds
C* TARX   REAL   earth-based x in meters
C* TARY   REAL  earth-based y in meters
C* TARZ   REAL  earth-based z in meters
C* SEC    INTEGER   seconds
C* ITS    INTEGER   iteration # in current second
C* ITMIN  INTEGER   # of iterations per second
C* HIT    INTEGER   1 or 0
C* MISS   INTEGER   1 or 0
C* OTARX  REAL   earth-based x in meters
C* OTARY  REAL   earth-based y in meters
C* OTARZ  REAL   earth-based z in meters
C* RANGE  REAL   meters

IMPLICIT NONE
REAL VELCTY,PITCHR,YAWR,PITCH,YAW
REAL THRUST,DRAG,YAWSIG,PTHSIG
REAL X,Y,Z
REAL ACCELR,PTCHRD,YAWRD,PITCHD,YAWD
REAL XD,YD,ZD,MASS
REAL CMPDIS,CMPSIG
REAL CMPDRG,CMPTHT,ACCEL
REAL PTHACC,YAWACC,CNGPTH,CNGYAW
REAL CNGX,CNGY,CNGZ

REAL TARX,TARY,TARZ
REAL OTARX,OTARY,OTARZ
```

25
REAL CURRR, CURRX, CURRY, CURRZ
REAL NEWR, NEWX, NEWY, NEWZ
REAL RESR, RESX, RESY, RESZ
REAL CLSVEL, CMPOP, CMPOY

REAL LINEX, LINEY, LINEZ
REAL PITCHO, YAWO
REAL RANGE, IMPTME, INTRVL
INTEGER ITMIN, SEC, ITS, HIT, MISS
REAL NAVC
PARAMETER (NAVC = 4)
REAL G
PARAMETER (G = 9.8)
REAL PTHMAX, YAWMAX
PARAMETER (PTHMAX = 30.0)
PARAMETER (YAWMAX = 30.0)
REAL TIMRNG, DETRNG
PARAMETER (TIMRNG = 100.0)
PARAMETER (DETRNG = 10.0)
REAL OPVEL, MATTRT, MINTRT
PARAMETER (OPVEL = 700.0)
PARAMETER (MATTRT = 6800.0)
PARAMETER (MINTRT = 0.0)
REAL BRNTME, MAXWGT, MINWGT
PARAMETER (BRNTME = 8.0)
PARAMETER (MAXWGT = 56.7)
PARAMETER (MINWGT = 22.7)

REAL SEEKER
C degrees * .017453293 = radians
C 15 deg = .261798 radians
C PARAMETER (SEEKER = .261798)
C 20 deg = .349064 radians
C PARAMETER (SEEKER = .349064)
C 30 deg = .523596 radians
PARAMETER (SEEKER = .523596)

REAL XCOM, YZCOM
REAL LNANG, ATAN

INTRVL = 1/(ITMIN * 1.0)
CURRX = (OTARX - X)
CURRY = (OTARY - Y)
CURRZ = (OTARZ - Z)

CURRR = CMPDIS(CURRX, CURRY, CURRZ)

THRUST = CMPTHT(SEC, MATTRT, BRNTME, MINTRT)
DRAG = CMPDRG(VELCTY, PITCHR, YAWR)
ACCELR = ACCEL(THRUST, DRAG, PITCH, MASS)
PTCHRD = PTHACC(PTHSIG, PITCHR)
YAWRD = YAWACC(YAWSIG, YAWR)
PITCHD = CNGPTH(PITCHR, VELCTY, PITCH)
YAWD = CNGYAW(YAWR, VELCTY, PITCH)
XD = CNGX(VELCTY, PITCH, YAW)
YD = CNGY(VELCTY, PITCH, YAW)
ZD = CNGZ(VELCTY, PITCH)

VELCTY = VELCTY + (ACCELR * 1/ITMIN)
PITCHR = PITCHR + (PTCHR * 1/ITMIN)
YAWR = YAWR + (YAWRD * 1/ITMIN)
PITCH = PITCH + (PITCHD * 1/ITMIN)
YAW = YAW + (YAWD * 1/ITMIN)
X = X + (XD * 1/ITMIN)
Y = Y + (YD * 1/ITMIN)
Z = Z - (ZD * 1/ITMIN)
IF (SEC .LE. BRNTME) THEN
  MASS = MASS - (((MAXWGT-MINWGT)/BRNTME) * 1/ITMIN)
END IF

NEWX = (TARX - X)
NEWY = (TARY - Y)
NEWZ = (TARZ - Z)

NEWR = CMPDIS(NEWX,NEWY,NEWZ)

XCOM = CMPDIS(NEWX,0.0,0.0)
YZCOM = CMPDIS(0.0,NEWY,NEWZ)

LNANG = ATAN(YZCOM/XCOM)

RESR = (CURRR - NEWR)*ITMIN
RESX = (CURRX - NEWX)*ITMIN
RESY = (CURRY - NEWY)*ITMIN
RESZ = (CURRZ - NEWZ)*ITMIN

CLSVEL = -RESR

CALL LNOFST(NEWX,NEWY,NEWZ,RESX,RESY,RESZ, +
              LINEX,LINEX,LINEX)

PITCHO = CMPOP(YAW,LINEX,LINEX)
YAWO = CMPOY(PITCH,YAW,LINEX,LINEX,LINEX)

PTHSIG = G * CMPSIG(NAVC,CLSVEL,PITCHO,PTHMAX,MASS, +
                     VELCTY,MINWGT,OPVEL)
YAWSIG = G * CMPSIG(NAVC,CLSVEL,YAWO,YAWMAX,MASS, +
                     VELCTY,MINWGT,OPVEL)

IF ((Z .GT. 0.0) .OR. (VELCTY .LE. 0.0)) THEN
  MISS = 1
  RETURN
END IF

IMPTME = -((NEWX*RESX)+(NEWY*RESY)+(NEWZ*RESZ))
IMPTME = IMPTME/((RESX**2)+(RESY**2)+(RESZ**2))
RANGE = (RESX*IMPTME + NEWX)**2
RANGE = RANGE + (RESY*IMPTME + NEWY)**2
RANGE = RANGE + (RESZ*IMPTME + NEWZ)**2
RANGE = SQRT(RANGE)
IF (RANGE .LT. DETRNG) THEN
  IF (ABS(IMPTME) .LE. INTRVL) THEN
    HIT = 1
    RETURN
  END IF
END IF

IF (ABS(LNANG) .GT. SEEKER) THEN
  IF (NEWR .LT. DETRNG) THEN
    HIT = 1
  ELSE
    MISS = 1
  END IF
  RETURN
END IF

IF (((CLSVEL .GT. 0) .AND. SEC .GT. BRNTME)) THEN
  IF (NEWR .LT. DETRNG) THEN
    HIT = 1
  ELSE
    MISS = 1
  END IF
  RETURN
END IF

RETURN
END

C******************************************************************************

C******************************************************************************
C SUBROUTINES BEGIN HERE
C******************************************************************************

REAL FUNCTION CMPDRG(VELCTY, PITCHR, YAWR)
C* Name - Function CMPDRG
C* Purpose - Computes drag value for the missile simulation
C* Variables -
C* NAME    TYPE            UNITS
C* VELCTY  REAL            meters/second
C* PITCHR  REAL            radians/second
C* YAWR    REAL            radians/second
C******************************************************************************
IMPLICIT NONE
REAL VELCTY, PITCHR, YAWR
C
REAL K1, K2

28
PARAMETER (K1 = 0.009412)
PARAMETER (K2 = 93850/(9.8**2))

CMPDRG = K1 * VELCTY**2 + (K2 * (PITCH**2 + YAWR**2))/VELCTY**2
RETURN
END

REAL FUNCTION CMPSIG(NAVC,CLSVEL,LINQ,MASK,MAXTN,MASS,
+ VELCTY,VLPSMG,OPVXL)

NAME - Function CMPSIG
Purpose - Computes steering signals for missile
Variables -
NAME TYPE UNITS
NAVC INTEGER Navigational constant. Usually 4
CLSVEL REAL Meters/second
LINE0 REAL radians (line of sight)
MAXTN REAL G's (max number of G's capable of pulling)
VELCTY REAL meters/second
MASS REAL kilograms

IMPLICIT NONE

REAL NAVC,CLSVEL,LINQ,MAXTN,MASS,VELCTY

REAL SIGNAL,VLPSMG,VLPSMG,VLPSMG,OPVXL,VLPSMG
REAL VELSIG

SIGNAL = NAVC * CLSVEL * LINQ

VLPSMG = VELCTY * VELCTY
OPVXL = OPVXL * OPVXL

VELSIG = (VLPSMG/OPVXL)

IF (VELSIG .GT. 1.0) THEN
VELSIG = 1.0
END IF

VLPSMG = (VLPSMG/MASS) * VELSIG * MAXTN

IF (ABS(SIGNAL) .GT. VLPSMG) THEN
SIGNAL = VLPSMG * ABS(SIGNAL)/SIGNAL
END IF

IF (ABS(SIGNAL) .GT. MAXTN) THEN
SIGNAL = MAXTN * ABS(SIGNAL)/SIGNAL
END IF
CMPSIG = SIGNAL

RETURN
END

C*****************************************************************************

C*****************************************************************************

REAL FUNCTION CMPOP(YAW,LINEX,LINEY)
C*****************************************************************************

C* Name - Function CMPOP
C* Purpose - Computes an element used in computing pitch signals from the seeker.
C* Variables -
C* NAME TYPE UNITS
C*---------------------------------------------------------------
C* YAW REAL radians
C* LINEX REAL meters
C* LINEY REAL meters
C*****************************************************************************

IMPLICIT NONE
REAL YAW,LINEX,LINEY

CMPOP = (-1) * SIN(YAW) * LINEX + COS(YAW) * LINEY

RETURN
END

C*****************************************************************************

REAL FUNCTION CMPOY(PITCH,YAW,LINEX,LINEY,LINEZ)
C*****************************************************************************

C* Name - Function CMPOY
C* Purpose - Computes an element used in computing yaw signals from the seeker.
C* Variables -
C* NAME TYPE UNITS
C*---------------------------------------------------------------
C* PITCH REAL radians
C* YAW REAL radians
C* LINEX REAL meters
C* LINEY REAL meters
C* LINEZ REAL meters
C*****************************************************************************

IMPLICIT NONE
REAL PITCH,YAW
REAL LINEX,LINEY,LINEZ

CMPOY = SIN(PITCH) * (COS(YAW)*LINEX + SIN(YAW)*LINEY)
CMPOY = CMPOY + COS(PITCH)*LINEZ

RETURN
END

C*****************************************************************************
SUBROUTINE LNOFST(RX,RY,RZ,RESX,RESY,RESZ,
+                      LINEX,LINEY,LINEZ)

C* Name - Subroutine LNOFST
C* Purpose - Computes line of sight angles
C* Variables -
C*   NAME TYPE UNITS
C* RX REAL meters
C* RY REAL meters
C* RZ REAL meters
C* RESX REAL meters/second
C* RESY REAL meters/second
C* RESZ REAL meters/second
C* LINEX REAL meters
C* LINEY REAL meters
C* LINEZ REAL meters
IMPLICIT NONE
REAL RX,RY,RZ
REAL RESX,RESY,RESZ
REAL LINEX,LINEY,LINEZ
REAL CMPDIS
REAL R,R2

R = CMPDIS(RX,RY,RZ)
R2 = R * R

LINEX = (RY * RESZ - RZ * RESY) / (R2)
LINEY = (RZ * RESX - RX * RESZ) / (R2)
LINEZ = (RX * RESY - RY * RESX) / (R2)
RETURN
END

REAL FUNCTION CMPDIS(X,Y,Z)

C* Name - Function CMPDIS
C* Purpose - Computes distance
C* Variables -
C*   NAME TYPE UNITS
C* X REAL earth-based x
C* Y REAL earth-based y
C* Z REAL earth-based z
IMPLICIT NONE
REAL X,Y,Z
CMPDIS = SQRT(X*X + Y*Y + Z*Z)

RETURN
END

C******************************************************************************

C SUBROUTINE GETTAR(TARX,TARY,TARZ,ITMIN)
C******************************************************************************
C* Name - Subroutine GETTAR
C* Purpose - Provides a moving target.
C* Variables -
****
C* NAME TYPE UNITS
C**
C* TARX REAL earth-based X
C* TARY REAL earth-based Y
C* TARZ REAL earth-based Z
C* ITMIN INTEGER (# of iterations per second)
****
IMPLICIT NONE
REAL TARX,TARY,TARZ
INTEGER ITMIN

REAL MACH
PARAMETER (MACH = 234.375)

TARX = TARX + (MACH/ITMIN)
TARY = TARY
TARZ = TARZ

RETURN
END

C******************************************************************************

C REAL FUNCTION CMPTHT(T,MAXTRT,MXTME,MINTRT)
C******************************************************************************
C* Name - Function CMPTHT
C* Purpose - Computes thrust for missile simulation
C* Variables -
****
C* NAME TYPE UNITS
C**
C* T INTEGER seconds (# of seconds missile has flown)
****
IMPLICIT NONE
INTEGER T

REAL MAXTRT,MXTME,MINTRT

IF (T .LE. MXTME) THEN
CMPTHT = MAXTRT
ELSE
C**-----------------------------------------------
C** REAL FUNCTION ACCEL(THRUST,DRAG,PITCH,MASS)
C**-----------------------------------------------
C* Name - Function Accel
C* Purpose - Computes acceleration
C* Variables -
C* NAME TYPE UNITS
C* THRUST REAL newtons
C* DRAG REAL newtons
C* PITCH REAL radians
C* MASS REAL kilograms
C**-----------------------------------------------
IMPLICIT NONE

REAL THRUST,DRAG,PITCH
REAL G,MASS

C ACCELERATION DUE TO GRAVITY IS 9.8 meters PER SECOND
PARAMETER (G = 9.8)

ACCEL = (THRUST-DRAG)/MASS - (G*(SIN(PITCH)))
RETURN
END

C**-----------------------------------------------
C** REAL FUNCTION PTHACC(SIGNAL,PITCHR)
C**-----------------------------------------------
C* Name - Function PTHACC
C* Purpose - Computes pitch acceleration
C* Variables -
C* NAME TYPE UNITS
C* SIGNAL REAL radians/sec**2
C* PITCHR REAL radians/sec
C**-----------------------------------------------
IMPLICIT NONE

REAL SIGNAL,PITCHR
REAL MSSCST

C MISSLE TIME CONSTANT IS 0.25 SECONDS
PARAMETER (MSSCST = 0.25)
PTHACC = (SIGNAL - PITCHR)/MSSCST

RETURN
END

REAL FUNCTION YAWACC(SIGNAL, YAWR)

*******************************************************************************
C* Name - Function YAWACC
C* Purpose - Computes Yaw acceleration
C* Variables -
C* NAME TYPE UNITS
C*---------------------------------------------------------------
C* SIGNAL REAL radians/sec**2
C* YAWR REAL radians/sec
C*******************************************************************************

IMPLICIT NONE

REAL SIGNAL, YAWR
REAL MSSCST

C MISSLE TIME CONSTANT IS 0.25 SECONDS
PARAMETER (MSSCST = 0.25)

YAWACC = (SIGNAL - YAWR)/MSSCST

RETURN
END

*******************************************************************************

REAL FUNCTION CNGPTH(PITCHR, VELCTY, PITCH)

*******************************************************************************
C* Name - Function CNGPTH
C* Purpose - Computes changes in the pitch angle
C* Variables -
C* NAME TYPE UNITS
C*---------------------------------------------------------------
C* PITCHR REAL radians/second
C* VELCTY REAL meters/second
C* PITCH REAL radians
C*******************************************************************************

IMPLICIT NONE

REAL PITCHR, VELCTY, PITCH

CNGPTH = ((PITCHR - COS(PITCH))/VELCTY)

RETURN
END

*******************************************************************************
REAL FUNCTION CNGYAW(YAWR, VELCTY, PITCH)

C* Name - Function CNGYAW
C* Purpose - Computes yaw changes
C* Variables -
C* NAME TYPE UNITS
C* YAWR REAL radians/second
C* VELCTY REAL meters/second
C* PITCH REAL radians

IMPLICIT NONE

REAL YAWR, VELCTY, PITCH

CNGYAW = (YAWR / (VELCTY * (COS(PITCH))))

RETURN
END

REAL FUNCTION CNGX(VELCTY, PITCH, YAW)

C* Name - Function CNGX
C* Purpose - Computes changes in x direction
C* Variables -
C* NAME TYPE UNITS
C* VELCTY REAL meters/second
C* PITCH REAL radians
C* YAW REAL radians

IMPLICIT NONE

REAL VELCTY, PITCH, YAW

CNGX = VELCTY * COS(PITCH) * COS(YAW)

RETURN
END

REAL FUNCTION CNGY(VELCTY, PITCH, YAW)

C* Name - Function CNGY
C* Purpose - Computes changes in y direction
C* Variables -
C* NAME TYPE UNITS
C* VELCTY REAL meters/second

35
C* PITCH REAL radians
C* YAW REAL radians
C******************************************************************************
IMPLICIT NONE

REAL VELCTY,PITCH,YAW

CNGY = VELCTY * COS(PITCH) * SIN(YAW)

RETURN
END
C******************************************************************************

C******************************************************************************

REAL FUNCTION CNGZ(VELCTY,PITCH)
C******************************************************************************
C* Name - Function CNGZ
C* Purpose - Computes changes in z direction
C* Variables -
C* NAME TYPE UNITS
C******************************************************************************
IMPLICIT NONE

REAL VELCTY,PITCH

CNGZ = VELCTY * SIN(PITCH)

RETURN
END
C******************************************************************************
Appendix D

Missile Detonation Logic

**Tracking**

- **LOS > Seeker Limitation**
  - **No** → Continue
  - **Yes** → **Estimated Range < 10 meters**
    - **No** → Report Miss
    - **Yes** → Report Hit
Impact

- Estimated Range < 10 meters?
  - No: Continue
  - Yes: Missile Impact Time < Interval?
    - No: Continue
    - Yes: Report Hit
Fuel Exhausted

Closing Velocity < 0? No → Continue

Yes → Fuel Exhausted? No → Continue

Yes → Absolute Distance < 10? No → Report Miss

Yes → Report Hit
The Analysis of a Generic Air-to-Air Missile Simulation Model

Joseph A. Kaplan, Alan R. Chappell, and John W. McManus

NASA Langley Research Center
Hampton, VA 23681-0001

National Aeronautics and Space Administration
Washington, DC 20546-0001

Kaplan - Virginia Polytechnic Institute and State University, Blacksburg, VA
Chappell - Lockheed Engineering & Sciences Company, Hampton, VA
McManus - Langley Research Center, Hampton, VA

A generic missile model was developed to evaluate the benefits of using a dynamic missile fly-out simulation system versus a static missile launch envelope system for air-to-air combat simulation. This paper examines the performance of a launch envelope model and a missile fly-out model. The launch envelope model bases its probability of killing the target aircraft on the target aircraft's position at the launch time of the weapon. The benefits gained from a launch envelope model are the simplicity of implementation and the minimal computational overhead required. A missile fly-out model takes into account the physical characteristics of the missile as it simulates the guidance, propulsion, and movement of the missile. The missile's probability of kill is based on the missile miss distance (or the minimum distance between the missile and the target aircraft). The problems associated with this method of modeling are a larger computational overhead, the additional complexity required to determine the missile miss distance, and the additional complexity of determining the reason(s) the missile missed the target. This paper evaluates the two methods and compares the results of running each method on a comprehensive set of test conditions.