Optimizing Terminal Conditions Using Geometric Guidance for Low-Control Authority Munitions

by

Paul C. Tisa

B.S., Astronautical Engineering
United States Air Force Academy, 2006

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Signature of the Author ....
Department of Aeronautics and Astronautics
May 9, 2008

Certified by .....
Marc McConley
Charles Stark Draper Laboratory
Thesis Supervisor

Certified by ....
Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics
Thesis Advisor

Accepted by ....
David L Darmofal
Associate Professor of Aeronautics and Astronautics
Chairman, Departmental Graduate Committee

[This Page Intentionally Left Blank]

Optimizing Terminal Conditions Using Geometric Guidance for Low-Control Authority Munitions

by

Paul C. Tisa

Submitted to the Department of Aeronautics and Astronautics on May 9, 2008 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Aeronautical Engineering

Abstract

Small munition effectiveness is a function of miss distance from the target and ability to achieve a steep flight path angle at the target. Many small guided munitions have limited control authority to achieve these competing objectives due to system hardware trade-offs. This thesis develops guidance algorithm modifications that demonstrate consistent improvement in achieving these objectives over previously used methods with changes only to the flight software and not the hardware or system concept of operations. Most modifications attempt to intelligently incorporate post-launch information into the guidance system, however there is an investigation into better using pre-launch information through dynamic programming. Dynamic programming is an off-line approach to optimize the guidance parameters applied in flight, based on measurable flight characteristics.

All investigated methods demonstrate varying abilities to improve performance for this munition system. While dynamic programming is computationally intensive, it produces an efficient look up table which is easily implemented in real time with minimal additional memory requirements. The thesis further shows that performance improvements are gained by altering the rocket ignition time in flight, by tightening the tolerances on some key sources of modeling error, and by developing a highly accurate time to impact estimation algorithm. Regardless of the particular modification, better utilizing pre- and post-launch information improves the munition's performance and utility for the user. While not tested, simultaneously implementing several of these improvements could further increase performance.

Thesis Supervisor: Marc McConley
Title: Charles Stark Draper Laboratory

Thesis Advisor: Emilio Frazzoli
Title: Associate Professor, Department of Aeronautics and Astronautics

[This Page Intentionally Left Blank]

Acknowledgments

Above all, I have to thank Marc McConley and Laurent Duchesne, my primary supervisors, for their seemingly endless supply of encouragement, support, and knowledge. My work would not have been possible without them. I would also like to extend my deepest gratitude to Professor Emilio Frazzoli. Despite his more than full schedule, Professor Frazzoli assistance enabled this project to reach a new level. Also to Brent Appleby for his enthusiasm and wisdom that always helped me maintain proper perspective.

In general, I want to extend my appreciation to everyone at The Charles Stark Draper Laboratory, specifically the Education Office, Massachusetts Institute of Technology, in particular the Aeronautics and Astronautics Department, and the United States Air Force. Without the support of any one of these three my life changing experience in Boston would not have been possible.

Finally, I need to thank my family and fiancé for their encouragement and endless ability to feign interest in a subject, which for them, was relatively tedious and overly-technical.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc. under Contract CON01071-1, Contract CON01745-1, and Draper Independent Research and Development.

Paul C. Tisa
2nd Lieutenant, USAF

[**This Page Intentionally Left Blank**]

Contents

List of Figures

[This Page Intentionally Left Blank]

List of Tables

[This Page Intentionally Left Blank]

Nomenclature

α	Angle of attack
ΔCP_δ	Change in center of pressure due to canards with no deflection
δ_{sat}	Saturated Canard Deflection
Δt	Time step
$\dot{\theta}$	Time rate of change of pitch angle
$\dot{\varepsilon}$	Time rate of change in angle between velocity and line of sight vector
$\dot{\vec{V}}$	Acceleration vector
$\dot{m}$	Time rate of change in mass
$\frac{\delta}{g}$	Canard deflection per unit of acceleration
$\frac{\delta}{g_{eff}}$	Maximum canard deflection per unit of acceleration weight
$\frac{\delta}{g_{max}}$	Maximum canard deflection per unit of acceleration constant
$\hat{v}$	Velocity unit vector
λ	Geodetic longitude
μ	Gravitational parameter constant, $3.986005 \times 10^{14}\ \frac{\mathrm{m}^3}{\mathrm{s}^2}$
ϕ	Roll body orientation angle
ψ	Yaw body orientation angle
ρ	Air density
θ	Pitch body orientation angle
ε	Angle between velocity and line of sight vector
φ	Geodetic latitude

ϑ Geometric relationship in transformation matrix calculation

$\vec{\Omega}$ Vector sum of aerodynamic accelerations

$\vec{a}^{cmd}$ Acceleration command vector

$\vec{g}$ Gravitational acceleration vector

$\vec{V}$ Velocity Vector

$\widehat{LOS}$ Line of sight unit vector

ξ Specific energy

ζ Dynamic programming *While* loop limiter indication the steady state solution is reached

a Ellipsoid equatorial radius, Semi-major axis

b Ellipsoid polar radius, Semi-minor axis

c Speed of sound

C_i^j Transformation matrix from coordinate frame i to j

C_d Coefficient of drag

C_{m_δ} Control moment slope due to canard deflection

C_{N_α} Control force slope due to angle of angle

C_{N_δ} Control force slope due to canard deflection

cg Center of gravity from nose

cg_f Final munition center of gravity

cg_i Initial munition center of gravity

cg_{fuel_i} Initial fuel center of gravity

cg_{fuel} Fuel center of gravity

D Drag force magnitude

d Munition diameter

D_ϵ Drag error

e Flattening factor geometric eccentricity

e' Ellipsoid radii geometric eccentricity

f Flattening factor, 0

g_c OTS Gravity Constant, 9.8066 $\frac{\mathrm{m}}{\mathrm{s}^2}$

$gain_{h_T}$ Inverse of the canard deflection per unit of acceleration transition height constant

h Altitude above ground

H_a Aiming point height

$h_{min_{\frac{\delta}{g}}}$ Lowest altitude allowable for maximum canard deflection per unit of acceleration constant

$h_{T_{\frac{\delta}{g}}}$ Canard deflection per unit of acceleration transition height constant

$i_{D_{sw}}$ Drag switch logic bit

$i_{L_{sw}}$ Lift switch logic bit

I_{y_f} Final moment of inertia about y_A

I_{y_i} Initial moment of inertia about y_A

J^R_{end} Downrange miss cost

J^T_{end} Terminal angle cost

K_1 ProNav and ProNav GB Homing gain

K_2 ProNav and ProNav GB Tunable gain

L Lift force magnitude

L_ϵ Lift error

l_{fuel} Nominal initial length of rocket motor fuel

LOS_t Line of sight angle from target

M Mach number

m Total munition mass (rocket and munition)

m_i Total initial munition mass without fuel

m_{fuel} Total mass of rocket fuel

N Total number of points in dynamic programming

n State combination n in the state matrix of size N

$N(\varphi)$ Radius of curvature in prime vertical

N_i Number of independent variable discretizations

N_x^κ Number of quantized states of dimension κ

OTS_{gain} OTS tunable gain

p Geometric relationship in transformation matrix calculation

q Dynamic pressure

R Range magnitude from current position to target

$R^2_{tar_{max}}$ Terminal boundary constant

R_{tar} Range to target

RE Earth radius constant, 6.378137×10^6 m

S Aerodynamic surface area of munition

T Rocket Thrust

t Current Time

t_b Rocket burn duration

T_ϵ Rocket thrust factor error

t_{b_ϵ} Rocket burn duration error

t_{cmd_ϵ} Commanded rocket ignition time error

t_{cmd} Commanded rocket ignition time

t_{go_ϵ} Time until impact error

$t_{go_{act}}$ Actual time until impact

$t_{go_{est}}$ Estimate time until impact

t_{go} Actual time until impact

T_{max} Maximum rocket thrust

T_{min} Minimum rocket thrust

V Velocity magnitude

$V(x)$ Dynamic programming value matrix

$X_{cp_{bt}}$ Center of pressure from nose

[This Page Intentionally Left Blank]

Chapter 1

Introduction

In the past several decades, the ability to put long-range projectiles precisely where desired has greatly improved, augmented by many technologies such as the Global Positioning System (GPS). Many missile programs, such as the Tomahawk Land Attack Missile (TLAM) seen in Figure 1.1, rely on high lift to drag ratios, large control surfaces, and multiple sensors to basically fly into the target [15]. Over the years, these programs have developed advanced systems simultaneously able to achieve unprecedented accuracy, time-of-flight (TOF) control, and high terminal angles. However, these systems are extremely complicated and expensive, many in excess of half a million dollars unit cost [15]. Additionally, the combination of their relatively slow glide speed and long TOFs makes these weapon systems too unresponsive for many missions in today's battlefield [18].

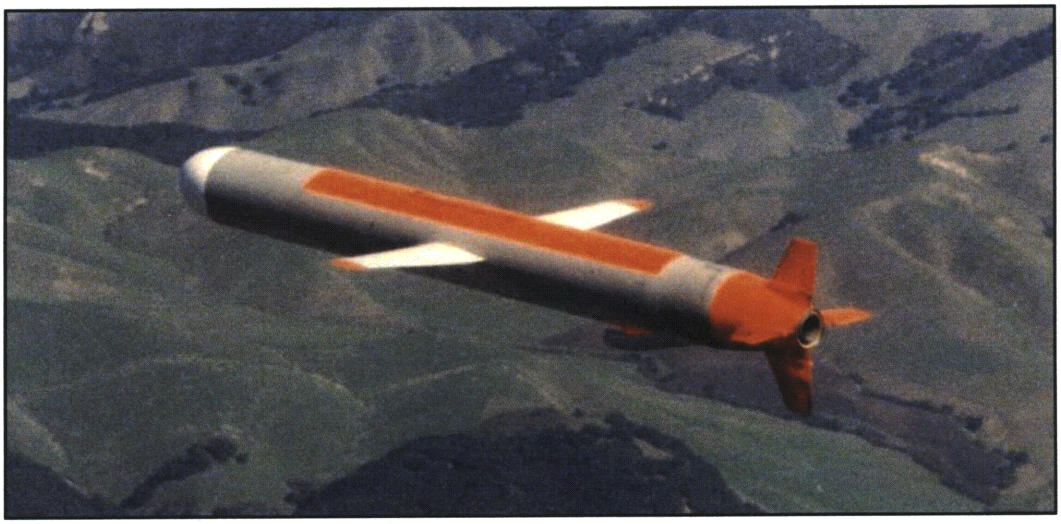


Figure 1.1: Tomahawk Land Attack Missile (TLAM) [28]

Serving a quick response need, traditional dumb munitions serve as a contrast to the long range, smart projectiles [40]. They are more responsive because their short range requires the user to be closer to the target and more cost-effective because their simpler, ballistic trajectory requires no guidance, navigation, and control (GN&C) system [41]. Relative to the TLAM, one of the greatest disadvantages for dumb munitions is there poor accuracy. Improving that accuracy would simultaneously increase the munition's mission envelope and decrease the total number needed to complete a given task [18] [17]. Attempts to increase accuracy with GN&C must balance between overly scanty systems that fail to improve accuracy enough and excessive systems that detract from the munition's original advantages.

1.1 Problem Statement

Traditionally, designers attempted to implement scaled down versions of GN&C systems borrowed from higher control authority missiles. Originally optimized around starkly different systems, these guidance architectures perform well but fail to use the flight characteristics of low-control authority munitions to maximize performance [54]. As a result, they leave room for improvement. For example, Proportional Navigation (ProNav) performs relatively well for most configurations but always fails to maximize a munition's performance when faced with both stringent accuracy and high terminal angle conditions [52]. This guidance system fails to account for gravity, a relatively dominant acceleration when compared to these munitions' control. As a result, ProNav wastes control authority fighting this predictable factor, unnecessarily sacrificing performance. While ProNav can be excellently tuned for specific cases, it lacks the flexibility to function well over a broad mission spectrum [42].

Over the years, engineers and academics have designed other guidance systems that attempt to surpass ProNav [43] [34]. Many studies attempt a two-dimensional engagement, simplifying the equations of motion enough so that the coupled nonlinear ordinary differential equations become solvable [20] [5] [53] [50]. For example, Adler derived a proportional navigation law as a function of geometric curvatures [2], Lin applied linear quadratic Gaussian theory to determine normal acceleration commands for a bank-to-turn missile [35], and Kumar et al. applied optimal control techniques to calculate a three phase feedback guidance law [29]. More generally, recent guidance laws incorporate optimal control theory [12] [44] [36], differential game theory [24] [37] [33], geometric ap-

proaches [11] [49], variable structure control [38], nonlinear guidance laws [32] [47] [51], and/or neural networks [4] [13]. Many times the methods do not completely replace ProNav, but merely modify it, such as Gurfil with his neoclassical guidance [22] [21] and Jalali-Naini with his modified LOS guidance [26].

Two such adaptations that exemplify the effectiveness of simple modifications are ProNav with a gravity bias constant (ProNav GB) and Offset Target Scale (OTS). ProNav GB simply adds a constant to the guidance system's acceleration calculation, in effect hiding gravity from the munition. OTS utilizes a slightly more complicated method by having a falling aim point. While not perfect, these simple improvements have caused ProNav GB and OTS to become quite popular. However, they both suffer from the same major flaw: they fail to realize a lot of the benefit in post-launch information.

Also tested is dynamic programming, an optimal control theory not new to this problem but novel in the details of its attempted implementation [16]. In contrast to improving the utilization of post-launch information, dynamic programming attempts to optimize pre-launch information. The demarcation is made because dynamic programming's output, an optimal control policy, is entirely a function of pre-launch information and cannot be changed after launch.

While it optimally uses the pre-launch information within given constraints, the effectiveness of dynamic programming diminishes as the perturbations between each launch grows. This suggests that a new guidance system, able to beneficially use both pre- and post-launch modifications, has the potential to drastically improve performance.

1.2 Objectives

The objective of this thesis is to demonstrate that better optimizing hard coded decisions early in the trajectory and programming the flexibility to intelligently use post-launch information can produce significant performance improvement in the industry standard guidance systems. This information can contain but is not limited to rocket characteristics and state updates from GPS. If properly done, this new guidance system should simultaneously maintain simplicity and low-cost, while improving the munition's ability to meet terminal constraints throughout its entire mission envelope.

For the user, this means the objective is to improve the capabilities of the munition in many directions of measurable performance through minor modifications to the software and system architecture. This would translate to creating a munition with the ability to simultaneously hit the desired target more accurately and at a higher terminal angle. The higher accuracy should require fewer shots to destroy or incapacitate a target, and the higher terminal angle would broaden the target spectrum of the munition, allowing more hardened targets. Ultimately, the objective is to give the user a better performing, more useful munition.

Within this thesis, solutions are designed for a low-control authority, short TOF munition. However, the simplicity of such a system allows for similar adaptations to be utilized by other projectile programs. Although the object of many works in this field, the target for this thesis does not move and is always in a known, stationary position [31] [10].

1.3 Overview

The outline of this thesis is as follows: Chapter 2 provides some background information on the theoretical munition system this thesis is based around. Chapter 3 details the software simulation all the results were obtained from. Some more information on the industry standard guidance systems mentioned previously is presented in Chapter 4 and their performance in the simulation is contained in Chapter 5. In Chapter 6, the new guidance system attempts are revealed and explained. The stage is then set to compare the old to the new in Chapter 7. Chapter 8 and 9 close the thesis with final conclusions and future work, respectively.

Chapter 2

Physical System Background

Defendable assumptions and realistic parameters are critical for the hardware. Establishing any guidance system, no matter how successful, would be useless if it only worked on fictional munitions. As such, the projectile system models historically established munition characteristics and constraints. Figure 2.1 shows a publicly released picture of and some statistics on one such program, the Extended Range Guided Munition (ERGM) built by Raytheon. Another publicly known project with a similar mission, shown in Figure 2.2, is the Ballistic Trajectory Extended Range Munition (BTERM) developed by Alliant Techsystems (ATK) and The Charles Stark Draper Laboratory [17].

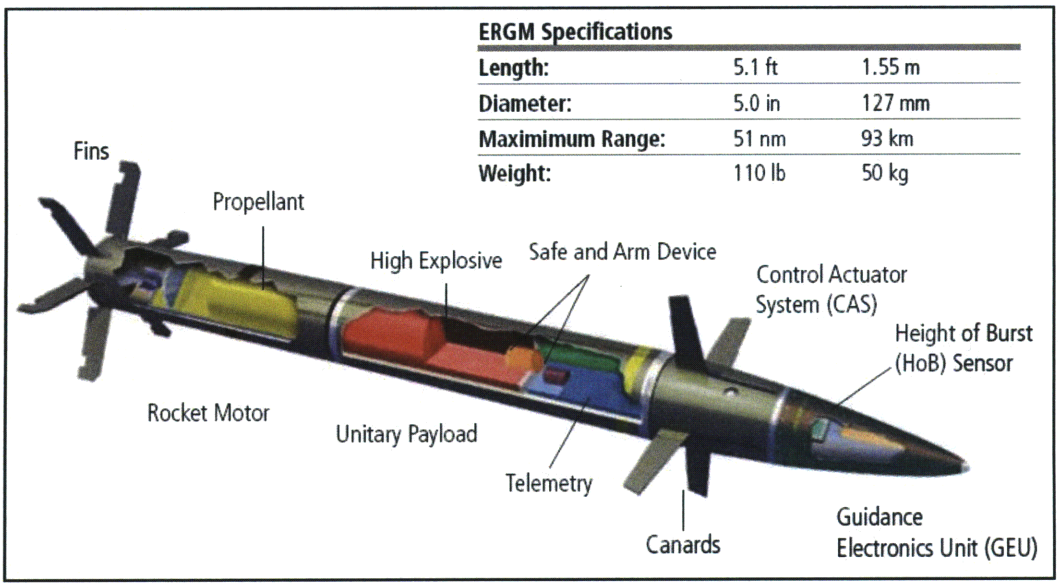


Figure 2.1: Extended Range Guided Munition (ERGM) [1]

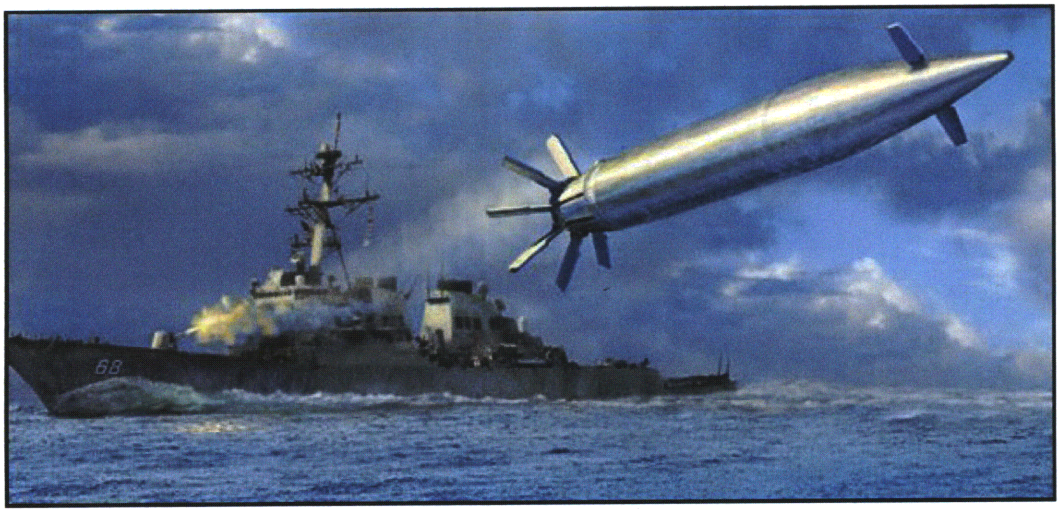


Figure 2.2: Ballistic Trajectory Extended Range Munition (BTERM) [3]

According to what is publicly reported, both of these systems have similar mission plans and munitions with alike characteristics. Figure 2.3 shows how Raytheon represents their system's life cycle. The simulated system's life cycle is qualitatively the same. Table 2.1 summarizes the quantitative parameters used. Because of available firing platforms, the munition must be less than five feet in length and 110 pounds in weight [14]. It must fulfill a support role balancing the responsiveness of traditional artillery with the accuracy of new long-range missiles. As such, this system needs to have an operational range of at least fifteen to forty-one nautical miles from launch point and still meet the accuracy and terminal angle conditions [14]. Accuracy must be no more than twenty meters circular error probable (CEP) and the higher the terminal angle, the better [17]. Terminal angle is the pitch body orientation angle at the point of impact. It must also be able to reach any target within that range window in less than five minutes from time of fire [17]. The launch platform can supply the munition with some preflight information, such as target location and local meteorological data, but the processing time will not exceed ten seconds for assumed rate of fire considerations.

Table 2.1: System Parameters

Length	< 5 ft	Accuracy	< 20 m CEP
Weight	< 110 lb	TOF	< 5 min
Range	≥ 15 & ≤ 41 nm	Pre-launch Time	< 10 sec

There are also less quantitative system requirements. To simultaneously meet the

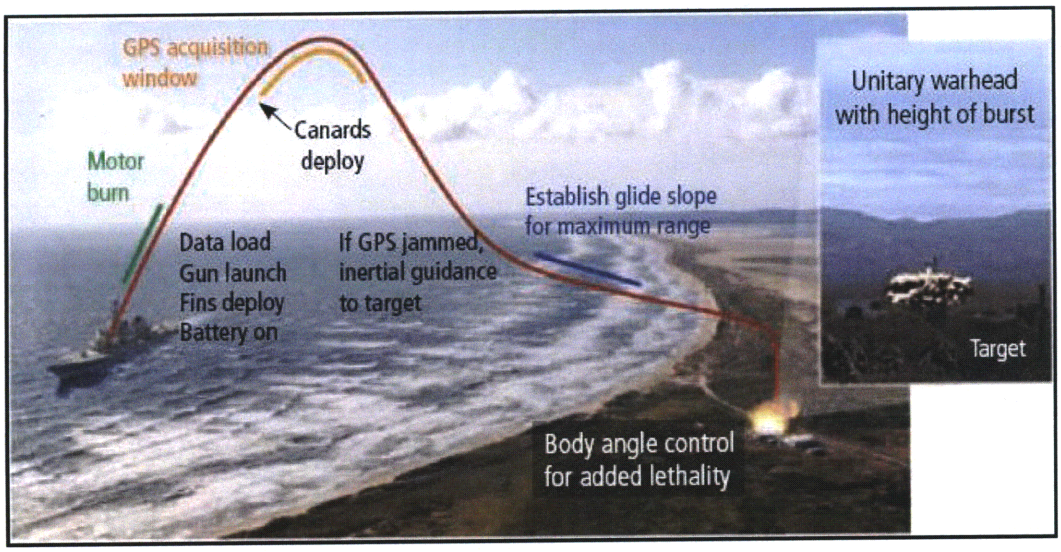


Figure 2.3: ERGM Mission Concept [1]

desired range window and size constraints, this munition must be equipped with a rocket engine and have a guidance system flexible enough to maintain performance in spite of the unknown variabilities of the engine, such as total thrust, ignition start time, and total burn time. While not necessary intuitive, firing the rocket earlier increases downrange, while the opposite is true for a later rocket ignition. The GN&C actuators are typically angle of attack variable canards fixed to the munition body. Using the canards to pull up increases downrange but decreases terminal angle, while pulling down decreases downrange but increases terminal angle. For cost considerations, the only available onboard sensors are a GPS receiver and an Inertial Navigation System (INS). As a whole, the system must also be robust enough to function in all normal weather and atmospheric conditions at any time of day. The characteristics of the rocket, such as burn duration, thrust profile, and mass, are also roughly based on the available information of similar systems.

[This Page Intentionally Left Blank]

Chapter 3

Simulation

The simulation, built in MATLAB and Simulink, is critical to all this work. The simulation's design focuses on replaceable parts and the author's finite knowledge. The first allows the simulation to conduct very different runs without any modifications between runs. The input script can specify different guidance systems, rocket characteristics, and many others in a long series. The simulation takes one setup at a time without any necessary modifications to the code or model. The second focus is not meant as a flamboyant display of humility but just as a matter of fact. Incorporating the vast array of factors separating this thesis' simulation and a high fidelity one would have reduced the available time to work on the subject of importance, leaving little time to investigate the real subject. As with many projects, there is a balance between accuracy and complexity. The simulation needs to be accurate enough so that the trends in data, not the precise values, can translate to the real world. Despite any issues with the particular implementation discussed in this chapter, this simulation's final behavior was compared to the behavior of a vastly more elaborate six degree of freedom (DOF) industry produced model. Their behaviors were satisfactorily similar, with explainable differences. Figure 3.1 compares a few attributes between a commercial-level six DOF simulation, and the Simulink model. The figure specifically shows Mach number, speed of sound, velocity magnitude, and mass versus time.

The Simulink simulation is a three DOF longitudinal model with many interchangeable parts, most importantly the guidance system. The three DOFs are altitude, downrange, and pitch. Altitude is the vertical distance above or below the chosen origin, downrange is the horizontal distance until or beyond the chosen origin, and pitch is the angle between the downrange and altitude component of velocity. Altitude and downrange are positive when the object is above and beyond its origin, respectively. Pitch has the sign

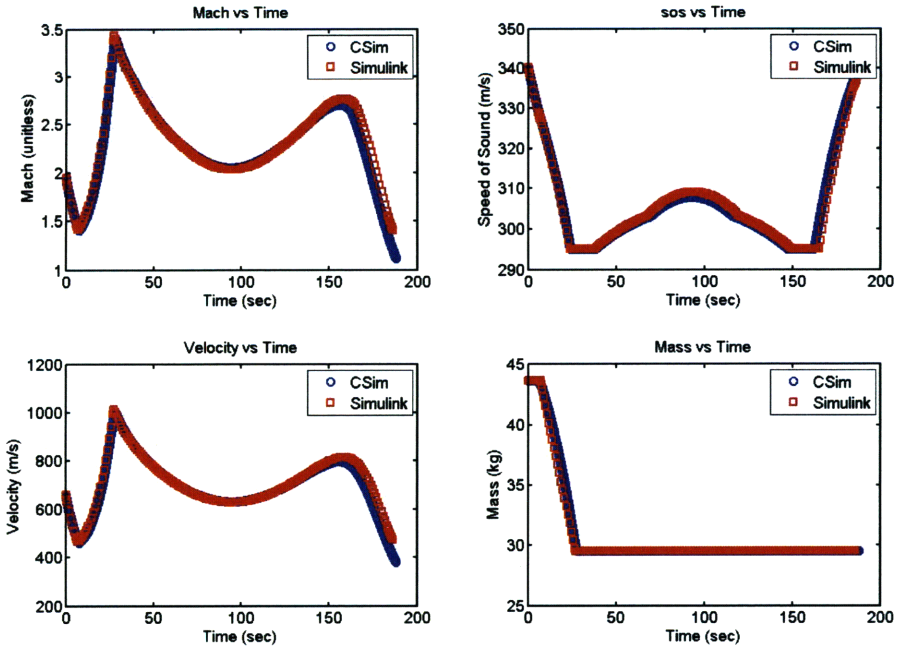


Figure 3.1: Comparison of Csim and Simulink Model

convention such that a munition fired straight up would have a pitch of $+90°$ on the way up and a pitch of $-90°$ after reaching its maximum altitude, apogee, and beginning the fall back to the ground. This example assumes the munition instantly turns $180°$ after gravity starts to bring it back to the surface. Pitch at the time of impact is referred to as the terminal angle. While desirable terminal angles are negative, figures in this thesis show them as positive so that the desired position in the figures progresses from left to right. Because of this "increasing" terminal angle, a desirable improvement, refers to a negative number increasing in magnitude and moving further from zero.

For purposes of clarity, defining the difference between the terms "actual" and "estimated" is important. "Actual" pertains to any information that could be regarded as the truth, in the sense that it is the simulation's closet representation of the truth. "Estimated" describes the onboard flight software's best guess of that particular characteristic's value at any given time. While presently, it is easier and possible to use one set of values, this is meant to mimic the realistic difference between the real world value of some property and the system's approximation. As such, the simulation is simultaneously running two seemingly identical subsystems in several locations, where the only difference is that one is using "actual" values while the other utilizes "estimated".

While others may be discussed within their pertinent subsystems, the simulation at least makes the following assumptions:

1. The munition can be represented as a point mass
2. The angle of attack and its effect on munition aerodynamics can be estimated
3. The munition can be approximated as a rigid body
4. The munition is symmetric about the longitudinal axis
5. The rocket burns fuel at a linear rate
6. The thrust profile is linear
7. The information supplied by the onboard navigation system has no errors
8. The control system is always able to generate the exact amount of canard deflection requested by the guidance system

The entire simulation is designed as a series of routines that are grouped into the six blocks seen in Figure 3.2.

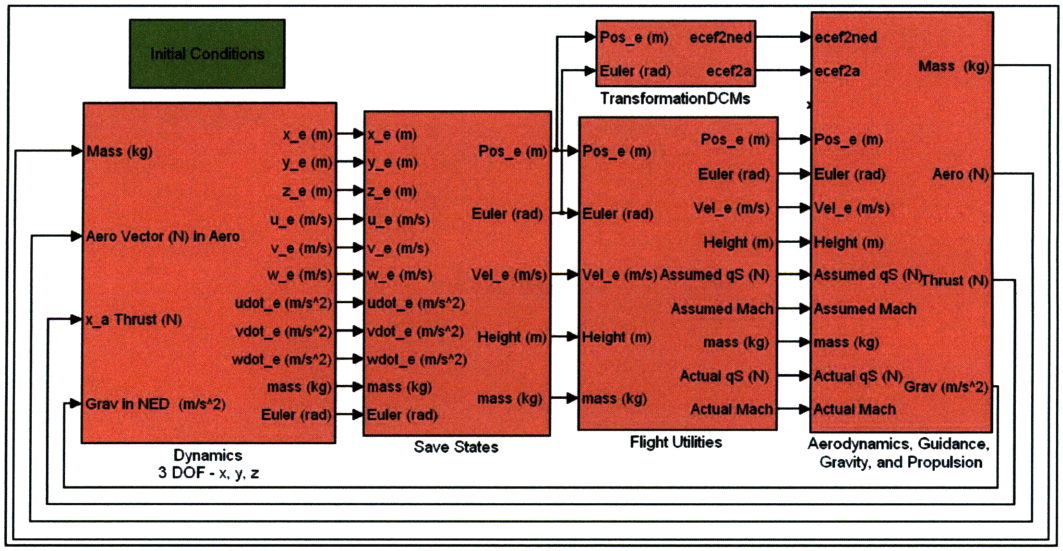


Figure 3.2: Basic Simulation Diagram

3.1 Initializations

Initializations is a simple masked block that allows the user to externally specify those variables held constant throughout an entire test campaign, such as launch date, munition mass, fuel mass, rocket burn duration, maximum and minimum rocket thrust, and munition diameter. Internally, it correctly formats the input data script, where the user specifies the frequently changed variables shown in Table 3.1.

Table 3.1: Input Data Script Initializations, Structure, and Units

Rocket Start Time	scalar	s
Launch Gun Elevation	scalar	deg
Launch Gun Velocity	scalar	$\frac{\text{m}}{\text{s}}$
Guidance System Choice	scalar	unitless
Canard Limit	scalar	deg
Guidance System Gain	scalar	unitless
Gravity Bias Value	scalar	$\frac{\text{m}}{\text{s}^2}$
Gravity Estimate	scalar	$\frac{\text{m}}{\text{s}^2}$
Launch Position	1 by 3 vector $[\varphi \lambda h]$	[deg deg m]
Target Position	1 by 3 vector $[\varphi \lambda h]$	[deg deg m]
Starting Position	1 by 4 vector $[\lvert\vec{R}\rvert$ LOS $\lvert\vec{V}\rvert$ pitch]	[m deg $\frac{\text{m}}{\text{s}}$ deg]
Drag Switch	scalar logical [0 or 1]	unitless
Lift Switch	scalar logical [0 or 1]	unitless
Rocket Switch	scalar logical [0 or 1]	unitless
Gravity Switch	scalar logical [0 or 1]	unitless
Hardwire Actuators Switch	scalar logical [0 or 1]	unitless
Control Table Switch	scalar logical [0 or 1]	unitless
Non-Launch Point Start Switch	scalar logical [0 or 1]	unitless
Error Switches[3.2]	1 by 34 vector logical	unitless

The Monte Carlo test runs have thirty-four errors incorporated in them. A single Monte Carlo run is actually 500 simulation calls. Each simulation call has randomly determined, zero-mean, normally distributed sigma values for its thirty-four errors. Table 3.2 shows the default 1σ values for the errors.

3.2 Dynamics

Equations relating both forces and torques to accelerations and time derivatives of angular momentum, respectfully, are necessary for a true six DOF simulation. In this simulation, the munition is modeled as a point mass, therefore eliminating any torque

Table 3.2: Errors and 1σ Distribution

Rocket Burn Time	$\frac{2}{3}$ s	Mass	0.15 lb
Rocket Ignition Time	$\frac{1}{3}$ sec	Center of Gravity	0.05 in
Rocket Thrust Factor	1.5%	Standard Pressure	1%
Gun Velocity	$\frac{1}{3}$%	Standard Temperature	2 °K
Gun Elevation	0.1°	Wind X-Axis	16 ft/s
6 Aerodynamic Coefficients	1%	Drag	1%
16 Aerodynamic Mach Related	1%	Lift	$\frac{5}{3}$%

and half of the equations. Within the simulation's *Dynamics* block, forces are converted to accelerations. The linear equations of motion could be written as:

$$\begin{aligned} \dot{V}_I &= \Omega_x + \frac{T_x}{m} - |\vec{g}|\sin(\theta) \\ \dot{V}_K &= \Omega_z + |\vec{g}|\cos(\theta) \\ \dot{\theta} &= \arctan\left(\frac{-w_N}{u_N}\right) \end{aligned} \tag{3.1}$$

Above, $\dot{V}$ is the acceleration, $\vec{\Omega}$ is the summation of all the actual aerodynamic accelerations modeled, T is the force from the munition's rocket, m is the current mass, and $\vec{g}$ is the current gravitational acceleration vector, and θ is the current pitch. The subscript for each variable represents the component and coordinate frame the data enters *Dynamics*. I, J, and K are the three components for the Earth-Centered, Earth-Fixed frame, x, y, and z are for the Aerodynamic frame, and N, E, and D are for the North-East-Down frame. See Appendix B for details. All information is converted to Earth-Centered, Earth-Fixed prior to summation. Additionally, the rocket thrust is always entirely along the x_A-axis, so the subscript is dropped for the remainder of this thesis. These equations are numerically integrated to determine the current state. Obviously, any inertial coordinate frame can be used as long as consistency is practiced. While the simulation integrates the equation after each component has been converted to Earth-Centered, Earth-Fixed (E) and summed, each force or acceleration is usually more easily visualized in another coordinate frame. The third equation approximates pitch as a function of velocity. It is an approximation because the previous decision to model the system as a point mass makes the concept of any body angle technically impossible.

3.3 Save States

Everything done within *Save States* is for post-simulation analysis.

3.4 Flight Utilities

Overall, *Flight Utilities* has three major tasks. First, it takes in state information to determine certain aerodynamic properties or characteristics, such as dynamic pressure and Mach number. Second, it performs that first task in two separate subsystems, one used by the actual model and the other by the estimated model. Lastly, *Flight Utilities* stops the simulation when the munition hits the ground.

Initially, speed of sound and air density are calculated as functions of the current height. This is where actual and estimated first deviate. The estimated system assumes standard day air temperature and pressure, while the actual obtains the randomly deviated "true" values. Then, speed of sound and air density, along with the height and velocity, determine the dynamic pressure and Mach number. The calculations are based on the 1976 U .S . Standard Atmosphere. Supplying optional temperature and pressure at sea level biases the temperature versus altitude matrix. The pressure and density matrices are then recalculated with the supplied sea level pressure and the ideal gas law.

These aerodynamic properties determine the dynamic pressure (q) and Mach number (M).

$$q = \frac{1}{2}\rho|\vec{V}|^2 \tag{3.2}$$

$$M = \frac{|\vec{V}|}{c} \tag{3.3}$$

Above, ρ is the current air density in $\frac{\text{kg}}{\text{m}^3}$, $|\vec{V}|$ is the magnitude of the current velocity vector in $\frac{\text{m}}{\text{s}}$, and c is the current speed of sound in $\frac{\text{m}}{\text{s}}$. Using the listed units, dynamic pressure is calculated in Newtons and Mach number is unitless.

The last step in *Flight Utilities* is to calculate the aerodynamic surface area from the user specified munition diameter. The simple geometric function is [48]:

$$S = \pi \frac{d^2}{4} \tag{3.4}$$

The default diameter, 5 in or around 0.127 m, is a function of the system parameters and based on real world systems.

3.5 Transformation Direction Cosine Matrices (DCMs)

Within this block, pitch and position determine the current transformation matrices between the coordinate frames. Pitch determines $C^{\frac{NED}{A}}$, the direction cosine matrix (DCM) from the Aerodynamic to the North-East-Down frame, through the following equations:

$$\mathbf{C}^{\frac{\mathbf{NED}}{\mathbf{A}}} = \begin{pmatrix} \cos(\theta)\cos(\psi) & \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) \\ \cos(\theta)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{pmatrix} \tag{3.5}$$

In Equation 3.5, ϕ, θ, and ψ are the body orientation angles roll, pitch, and yaw, respectively.

In another part of the block, position information determines $C^{\frac{N}{E}}$, by first calculating the geodetic latitude (φ), longitude (λ), and altitude (h) as:

$$\varphi = \arctan\left(\frac{Z + e'^2 b \sin(\vartheta)^3}{p - e^2 a \cos(\vartheta^3)}\right) \tag{3.6}$$

$$\lambda = \arctan\left(\frac{Y}{X}\right) \tag{3.7}$$

$$h = \frac{p}{\cos(\varphi)} - N(\varphi) \tag{3.8}$$

Above X, Y, and Z are the current position components in the Earth-Centered, Earth-Fixed frame, e and e' are both geometric eccentricities, a is the ellipsoid equatorial radius, b is the ellipsoid polar radius, and $N(\varphi)$ is the radius of curvature in prime vertical. These are calculated as:

$$e^2 = 2f - f^2 \tag{3.9}$$

$$e'^2 = \frac{a^2 - b^2}{b^2} \tag{3.10}$$

$$b = \sqrt{a^2(1 - e^2)} \tag{3.11}$$

$$N(\varphi) = \frac{a}{\sqrt{1 - e^2 \sin(\varphi)^2}} \tag{3.12}$$

$$p = \sqrt{X^2 + Y^2} \tag{3.13}$$

$$\vartheta = \arctan\left(\frac{aZ}{bp}\right) \tag{3.14}$$

f is Earth's flattening factor and is a function of the equatorial and polar radii [9].

$$f = \frac{a - b}{b} \tag{3.15}$$

As a final step, the algorithm pulls factors of 2π from the longitude until it is between zero and 2π. The subroutine also corrects for the numerical instability in the altitude calculation that occurs near the poles. The error is around 2 mm with this correction, which approximately matches the numerical precision of the entire function. The function uses the WGS84 model's semi-major axis for both its semi-major and semi-minor variables, resulting in a flattening factor of zero. Accordingly, both geometric eccentricities are zero. As a result, the algorithm could be simplified significantly, but was implemented as described to allow an easy transition to a more accurate model. Currently, the spherical earth assumption is of little consequence as all testing originates at the equator.

Then another routine uses that information to determine the transformation matrix as:

$$\mathbf{C}^{\frac{\mathbf{NED}}{\mathbf{E}}} = \begin{pmatrix} -\cos(\lambda)\sin(\varphi) & -\sin(\lambda) & -\cos(\lambda)\cos(\varphi) \\ -\sin(\lambda)\sin(\varphi) & \cos(\lambda) & -\sin(\lambda)\cos(\varphi) \\ \cos(\varphi) & 0 & -\sin(\varphi) \end{pmatrix} \tag{3.16}$$

Additionally, enough information exists to determine the DCM to convert to and from the Earth-Centered-Inertial Frame. However since the simulation only shoots North along the Prime Meridian for the results displayed, this frame is always the same as the Earth-Centered, Earth-Fixed frame.

3.6 Aerodynamics, Guidance, Gravity, and Propulsion

As the name suggests, this block contains four sub-blocks. *Gravity* is simple enough to be explained here, while the other three are large enough to necessitate separate sections. The sections for those larger blocks are below.

Gravity simply takes in the current altitude, which is in the Launch Centered Earth Fixed (LCEF) frame, converts it to Earth-Centered, Earth-Fixed, and uses the following equation to determine gravitational acceleration:

$$|\vec{g}| = \frac{\mu}{(h + RE)^2} \tag{3.17}$$

In Equation 3.17, μ is the gravitational parameter constant, h is the current altitude, and RE is Earth's radius.

3.6.1 Aerodynamics

The end goal in *Aerodynamics* is to determine actual lift potential, a force, per degree of canard deflection. It does this by using information determined in *Flight Utilities*. The system employs tabled, first-order aerodynamic coefficients to calculate the drag and lift potential. These tables were derived from experimentally determined tables for a real system with similar physical characteristics and system requirements. Table 3.3 shows all the aerodynamic tables used in the simulation.

Table 3.3: Aerodynamic Tables, Definitions, and Units

C_d	Coefficient of drag	Unitless
C_{N_α}	Control force slope due to angle of angle	Unitless
C_{N_δ}	Control force slope due to canard deflection	Unitless
C_{m_δ}	Control moment slope due to canard deflection	Unitless
$X_{cp_{bl}}$	Center of pressure from nose	Meters
ΔCP_δ	Change in center of pressure due to canards with no deflection	Meters

The aerodynamic tables are a function of Mach number. *Guidance* looks up the necessary aerodynamic value with assumed Mach number. *Aerodynamics* uses actual Mach number and deviates the table value by a random deviation specified in the seed. The seed is a large table specifying all the deviations for all of the variables with an error

term. Both sub-blocks linearly interpolate when necessary. Within *Aerodynamics*, there is error deviation both as a function of the coefficient and Mach number. The coefficient error is specific for each coefficient in each scenario, while the Mach number error is specific to a particular Mach number but is applied equally to every coefficient looked up at that Mach number in each scenario. While the error values are randomly determined with a normal distribution mean equal to zero, the same table is repeatedly used to allow for a direct comparison of results. At any point, a new seed can be generated if there are concerns that a particular guidance system is overly adjusted to the old seed.

Aerodynamics receives the current actual dynamic pressure, the aerodynamic surface, the current actual Mach number, and the munition diameter from previous routines. After looking up the appropriate values in the aerodynamic tables, the block calculates the drag and lift per degree of canard deflection as:

$$D = -C_d q S \cdot i_{D_{sw}}(1 + D_\epsilon) \tag{3.18}$$

$$\frac{L}{\delta} = \frac{-qS}{2}\left(-\frac{dC_{m_\delta}}{X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2}} + C_{N_\delta}\right) \cdot i_{L_{sw}}(1 + L_\epsilon) \tag{3.19}$$

The only unexplained variables in these equations are D_ϵ the drag error, L_ϵ the lift error, and $i_{D_{sw}}$ and $i_{L_{sw}}$ switch variables set either to 0 or 1. With these switches, the user can easily run a simulation without either or both of these forces for debugging or testing purposes.

3.6.2 Guidance

Guidance has a job similar to *Aerodynamics*, but, because the process is significantly more complicated, *Guidance* is divided into three subroutines. The first two, Homing and Guidance Command, are guidance system specific, while the third, Lift Command, is not. Because of this, details for Homing and Guidance Command are saved for each guidance system's section later in the thesis. However, there are some helpful generalizations worth mentioning because their overall function is guidance system independent. Homing Command uses the known state information to calculate an acceleration command in the inertial frame to reacquire the "optimal" path. Obviously, "optimal" is dependent on the guidance algorithm implemented. Guidance Command transforms that acceleration

vector from the inertial to the aerodynamic coordinate frame. Most also bias or scale the transformed acceleration command by some factor, once again depending on the utilized algorithm.

Lift Command is the same for all guidance systems used in this work. It takes the acceleration command in the aerodynamic frame and turns it into an amount of canard deflection. Initially, Lift Command looks up the tabulated values for the same variables as *Aerodynamics* with the exception that these tables have not been altered by either the variable specific error or the Mach number correlated error. Next, the routine calculates drag almost exactly as before:

$$D = -C_d q S \cdot i_{D_{sw}} \tag{3.20}$$

Then, Lift Command approximates a canard deflection per number of "g's" ($\frac{\delta}{g}$). This is a conversion factor from the desired acceleration command to the canard deflection required to obtain it.

if *Unstable* $\left(X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2} < 0\right)$ **then**
  $\frac{\delta}{g} = \frac{2m\left(X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2}\right)}{qSC_{m_\delta}d}$;
else
  $\frac{\delta}{g} = \frac{2m\left(X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2}\right)}{qS\left[C_{N_\delta}\left(X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2}\right) + C_{m_\delta}d\right]}$;
  if *Dynamic pressure is low saturate* $\left(\frac{\delta}{g} > \frac{\delta}{g}_{max}\right)$ **then**
    if *Too near ground do not saturate* $\left(h < h_{min_{\frac{\delta}{q}}}\right)$ **then**
      if *Ensure smooth transition between saturated and unsaturated* $\left(h < h_{min_{\frac{\delta}{q}}} + h_{T_{\frac{\delta}{g}}}\right)$ **then**
        $\frac{\delta}{g}_{eff} = \left(h - h_{min_{\frac{\delta}{q}}}\right) gain_T$;
      else
        $\frac{\delta}{g}_{eff} = 1$;
      end
      $\frac{\delta}{g} = \left(1 - \frac{\delta}{g}_{eff}\right)\frac{\delta}{g} + \frac{\frac{\delta}{q}_{eff}\frac{\delta}{g}^2_{max}}{\frac{\delta}{q}}$;
    end
  end
end

Algorithm 1: Calculating $\frac{\delta}{g}$

Besides $\frac{\delta}{g}_{eff}$, all the unexplained variables in Algorithm 1 are constants set by the user. $\frac{\delta}{g}_{max}$ sets a limit on the canard deflection per unit of acceleration, $h_{min_{\frac{\delta}{g}}}$ ensures more conservative canard deflection commands as the munition approaches the ground, $h_{T_{\frac{\delta}{g}}}$ is the transition height, and $gain_{h_T}$ is equal to the inverse of the transition height. These constants are designed to allow the user to shape the $\frac{\delta}{g}$ curve. As it relies on many variables, the $\frac{\delta}{g}$ for any given simulation run differs quantitatively to another run. However, the qualitative trend is strongly correlated to the initial downrange to target. Figure 3.3 depicts the usual appearance of $\frac{\delta}{g}$ for short and long range scenarios. The figure shows some time where $\frac{\delta}{g}$ is equal to zero for both scenarios. This is a result of the simulation. $\frac{\delta}{g}$ is only calculated after the guidance system turns on, which is initiated by the pitch passing through local level or zero degrees. Internally, the simulation uses $\frac{\text{rad}}{\text{m}/\text{s}^2}$ for the units of $\frac{\delta}{g}$. The figures are displayed in a more intuitive unit, $^{\text{deg}}/_{\text{g}}$, where "g" is the average acceleration experienced on Earth at sea level.

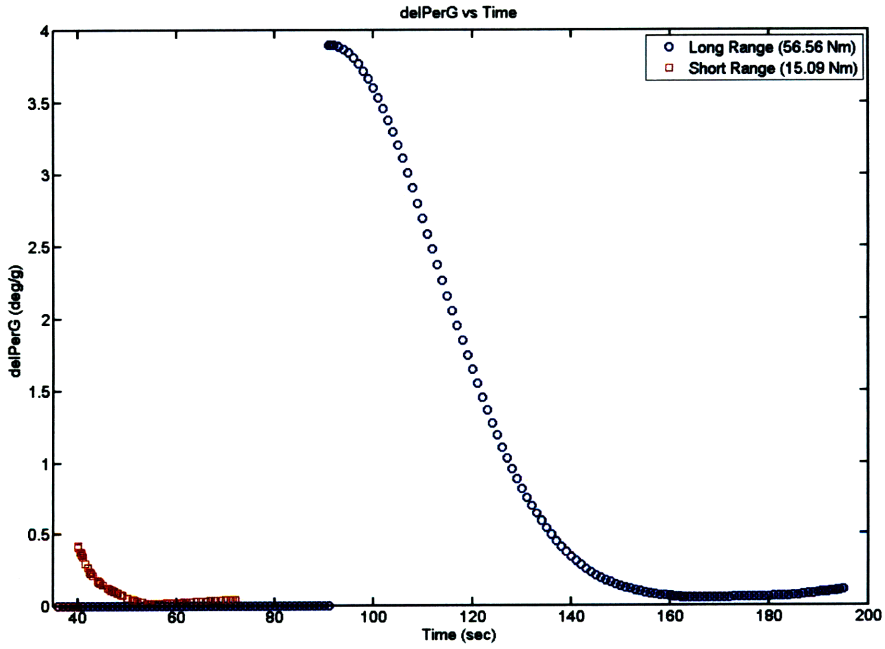


Figure 3.3: $\frac{\delta}{g}$ vs Time for Two Scenarios

Then it is simply multiplied by the acceleration command passed in from Guidance Command to calculate the necessary canard deflection. Before leaving the block, there is a check to ensure the canard deflection is within the physical constraints of the system.

The currently assumed deflection limit is $\pm 12°$. As a final step, the lift and angle of attack (α) are estimated and saved for debugging.

$$L = -\frac{qS}{2}\left(-\frac{dC_{m_\delta}}{X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2}} + C_{N_\delta}\right)\delta_{sat} \cdot i_{L_{sw}} \tag{3.21}$$

$$\alpha = -\frac{1}{2C_{N_\alpha}}\left(\frac{dC_{m_\delta}}{X_{cp_{bt}} - cg + \frac{\Delta CP_\delta}{2}}\right)\delta_{sat} \tag{3.22}$$

This deflection command then exits *Guidance* to be multiplied with *Aerodynamics'* computed lift per degree of canard deflection to determine the actual force generated by the canards. As this lift per degree of canard deflection is determined with errors, there is a difference between what acceleration the guidance system wanted and what acceleration the munition actually receives. Assuming the guidance system is stable, this problem is corrected by the system's closed loop feedback.

3.6.3 Propulsion

Propulsion basically decides which of three possible events is currently taking place. Either the rocket has yet to start, the rocket is firing, or the rocket has expired. If it has not started, then the rocket's thrust is zero, the munition's change in mass is zero, and its center of gravity is located at the initial location, about 0.83 meters from the nose. If it has expired, then the rocket's thrust is zero, the munition's change in mass is equal to the total fuel mass, and the center of gravity is located at the final location, around 0.78 meters from the nose. *Propulsion* only performs calculations when the simulation is between the former and the latter stage. As the rocket fires, thrust (T), mass change ($\dot{m}$), and center of gravity (cg) are represented by the following functions:

$$T = \left\{\frac{T_{max} - T_{min}}{t_b + t_{b_\epsilon}}[t - (t_{cmd} + t_{cmd_\epsilon})] + T_{min}\right\}(1 + T_\epsilon) \tag{3.23}$$

$$\dot{m} = \frac{m_{fuel}}{t_b + t_{b_\epsilon}}[t - (t_{cmd} + t_{cmd_\epsilon})] \tag{3.24}$$

$$cg = \frac{m_i cg_f + (m - m_i)cg_{fuel}}{m} \tag{3.25}$$

In the thrust equation, T_{max} is the rocket's maximum thrust, T_{min} the rocket's min-

imum thrust, t_b the burn duration of the rocket, t_{b_ϵ} the error in that burn duration, t the current time, t_{cmd} the commanded ignition time of the rocket, t_{cmd_ϵ} the error in that commanded start time, and T_ϵ the thrust factor error. In the mass change calculation, m_{fuel} is the total mass of the fuel. In the center of gravity equation, m_i is the munition's initial mass without fuel, cg_f is the final center of gravity, m is the current total mass, and cg_{fuel}, the fuel center of gravity, is determined by

$$cg_{fuel} = cg_{fuel_i} - l_{fuel}\frac{m_i + m_{fuel} - m}{2m_{fuel}} \tag{3.26}$$

$$cg_{fuel_i} = \frac{(m_i + m_{fuel})cg_i - m_i cg_f}{m_{fuel}} \tag{3.27}$$

$$l_{fuel} = \frac{\sqrt{12\left[I_{y_i} - I_{y_f} - m_i(cg_i - cg_f)^2 - m_{fuel}(cg_i - x_b)^2\right]}}{m_{fuel}} \tag{3.28}$$

Above, cg_i is the initial center of gravity, and I_{y_i} and I_{y_f} are the munition's initial and final moment of inertia about the y_A axis.

The first two calculations are linear, first-order approximations of the unknown real functions. Figure 3.4 shows how this approximation compares to the experimental data it is based on. "Simulink" is the three-DOF model, while "Csim" is the higher DOF, commercial grade simulation.

While thrust at any one time may not perfectly match or even decently resemble the data, the total thrust profile area was adjusted to fit the data with a difference of less than five milliNewton-seconds or about a $1.25 \times 10^{-5}\%$ error. The mass change function is designed in the same way.

The center of gravity equation is mathematically derived to represent the empirically derived data of the center of gravity as the rocket burned. This data was collected for a real world munition with similar mission and physical characteristics to the one simulated. The results of comparing the simulation's representation and that of the higher accuracy model are shown in Figure 3.5. The figure depicts the simulation's assumed center of gravity. The actual center of gravity is determined by adding in an error term.

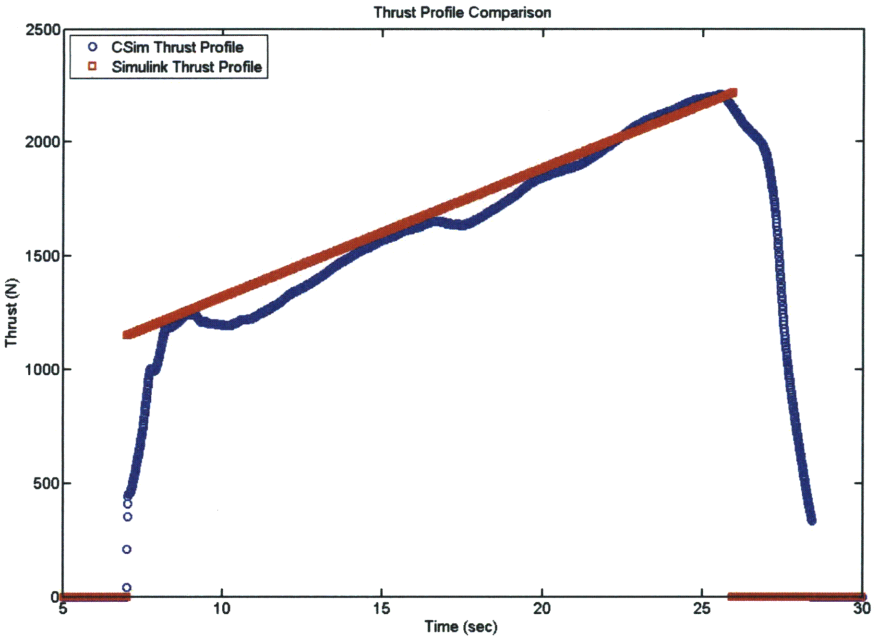


Figure 3.4: Thrust Comparison

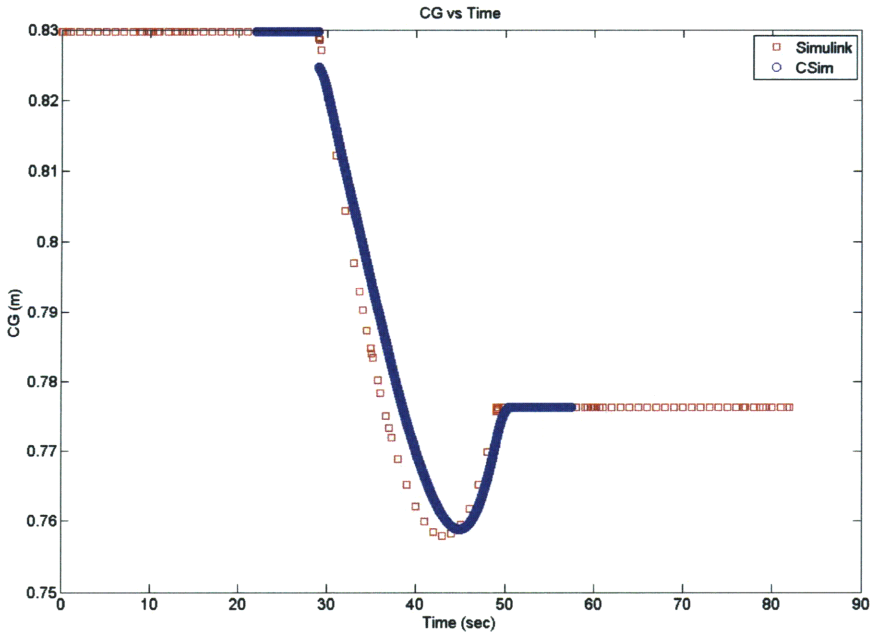


Figure 3.5: Center of Gravity Comparison

[This Page Intentionally Left Blank]

Chapter 4

Industry Standard Guidance Systems

While countless other academics and engineers have tested ProNav, ProNav GB, and OTS, the performance of these guidance systems on the simulation used in this thesis is critical for establishing a baseline. It is necessary to directly compare the older guidance systems with any methods attempted in this work.

4.1 Background

4.1.1 Proportional Navigation

ProNav is the simplest guidance algorithm presented in this work. The primary value of discussing ProNav is that the other guidance systems build directly from it. Simply stated, ProNav tries to align the munition's velocity vector with the line-of-sight (LOS) vector. Figure 4.1 illustrates this geometry for a two dimensional example, where $\vec{v}$ is the velocity vector, $\overrightarrow{LOS}$ is the LOS vector, and ε is the angle between the two vectors. Represented in this notation, it could also be said that ProNav looks for a command force to keep $\dot{\varepsilon}$, the time rate of change of ε, negative [39]. For this two dimensional example and the simulation, $\dot{\varepsilon}$ can be approximated as the following, where L is the lift produced by canard deflection, m is the current mass, and $|\vec{V}|$ is the velocity magnitude:

$$\Delta\varepsilon = -\frac{L}{m|\vec{V}|} \tag{4.1}$$

Assuming ProNav can maintain this and has enough time, the approximation ensures

the two vectors converge. As a reminder, the navigation information is treated as perfect, resulting in an errorless LOS vector [23].

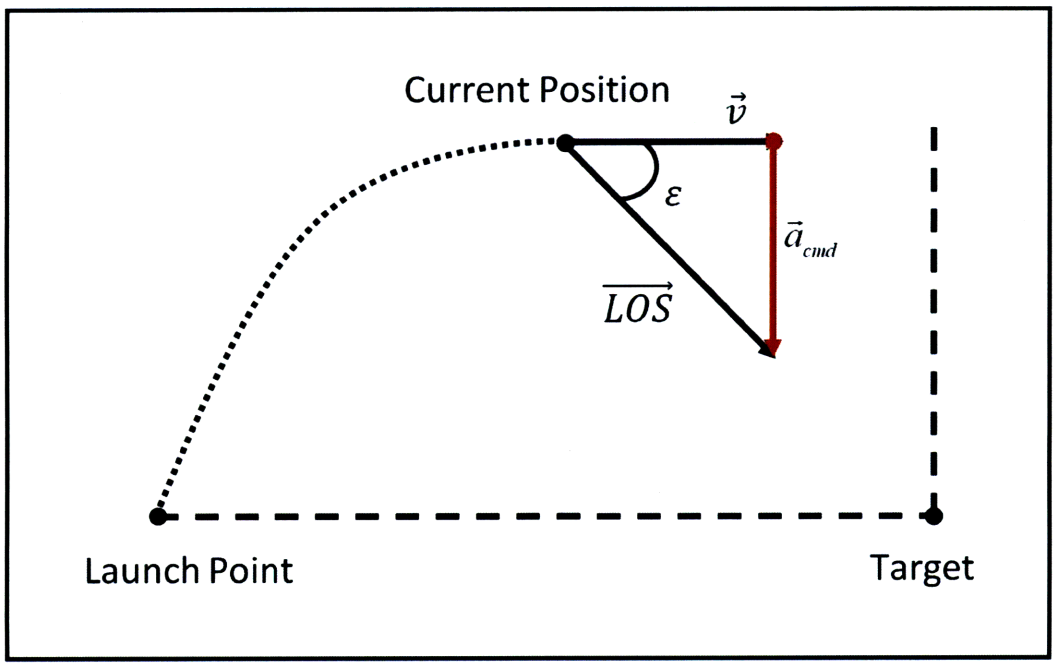


Figure 4.1: ProNav Geometry

In actual implementation, ProNav requires the current position and velocity, target location, and transformation DCM from Earth-Centered, Earth-Fixed to North-East-Down. Within *homing_cmd.m*, that information is used to calculate $\vec{\epsilon}$ and the acceleration command, $\vec{a}_E^{cmd}$, as:

$$\vec{\varepsilon} = \widehat{LOS}_E - \left(\widehat{LOS}_E \cdot \hat{v}_E\right) \hat{v}_E \tag{4.2}$$

$$\vec{a}_E^{cmd} = K_1 K_2 \vec{\varepsilon} \tag{4.3}$$

In the previous equations, $\hat{}$ represents a unit vector, $\cdot$ signifies a dot product, K_1 is the homing gain, and K_2 is a user tunable gain. The homing gain is inversely proportional to the range to target. However, to prevent this gain from exploding as the munition enters the terminal phase of flight, ProNav uses a second, decreasing function as follows:

$$K_1 = \min \left\{ \begin{array}{c} \frac{\vec{v}^2}{R_{tar}} \\ \frac{\vec{v}^2 R_{tar}}{R^2_{tar_{max}}} \end{array} \right\} \tag{4.4}$$

Above, R_{tar} is the current range to the target in meters and $R^2_{tar_{max}}$ is a user selected constant that sets the boundary between the normal and terminal flight phase.

The simulation passes the calculated acceleration command and the DCM from Earth to aerodynamic frame to *guidance_cmd.m.* For ProNav, the acceleration command simply undergoes a coordinate transformation from the Earth-Centered, Earth-Fixed to the Aerodynamic frame:

$$\vec{a}^{cmd}_A = C^{\frac{A}{E}} \vec{a}^{cmd}_E \tag{4.5}$$

The limitations of ProNav are well known. For this problem, the most significant issue is that gravity is neglected during the derivation. Unfortunately, for this munition, gravity is a significant force in comparison to the force the canards can generate. In fact for this system, except for in the densest control authority states, gravity cannot be directly countered. The would-be ballistic point of impact, decided mostly by gravity, can only be moved within a relatively small window even if all control authority is utilized in a single direction.

As a result, one of the most identifiable behaviors of ProNav in a system like this is spending most of its control trying to fight gravity. Beginning at apogee, when the guidance system turns on, ProNav would immediately nose down, trying to align its velocity vector with the target's LOS vector. Without knowledge of the future, this makes sense to ProNav as the munition's velocity vector is usually around horizontal to the ground and the target is far below, as depicted in Figure 4.1. The guidance system fails to realize gravity would eventually do this to the munition without any canard deflection. For an instant, the two vectors are aligned and ProNav commands no deflection from the canards. However, the guidance system soon realizes that the velocity vector has not stopped moving and continues its fall through the LOS vector towards the ground even though the canards were neutralized. For the remainder of the flight, ProNav slowly increases the canard deflection fruitlessly trying to pull the velocity vector back into alignment with LOS vector. As a result, even if the munition was close enough ballistically to hit the target despite this misuse of control authority, the terminal angle is still

needlessly sacrificed. Running a long range simulation with the ProNav guidance system easily demonstrates this behavior, illustrated in Figure 4.3.

Accordingly, ProNav can only practically be used as a terminal guidance system, where its assumptions do not have enough time to so detrimentally affect performance. The other two guidance systems try to improve on ProNav's model, keeping its simplicity as much as possible and removing its weaknesses.

4.1.2 Proportional Navigational with Gravity Bias

ProNav GB attempts to solve ProNav's greatest weakness for this type of system, an inability to account for gravity. While it calculates everything in *homing_cmd.m* exactly the same as ProNav, ProNav GB adds a constant approximation of the gravitational acceleration in *guidance_cmd.m* before determining the canard deflection necessary to generate the commanded acceleration, illustrated in Figure 4.2.

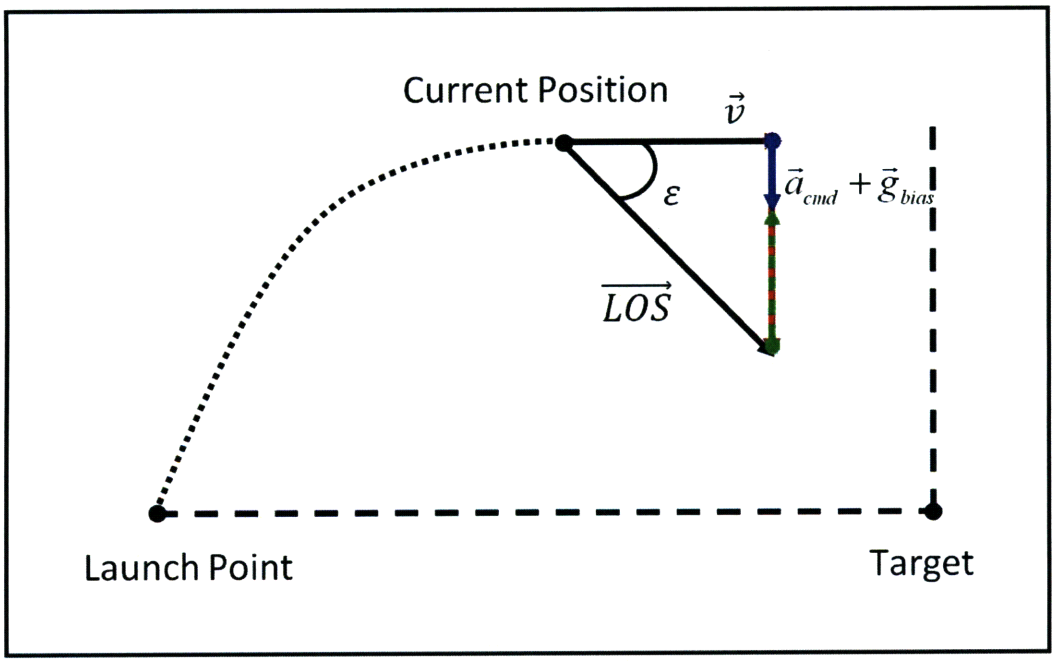
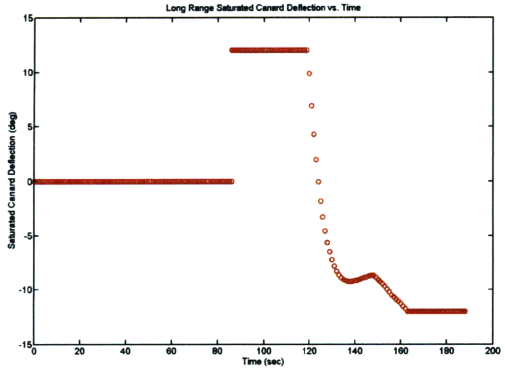


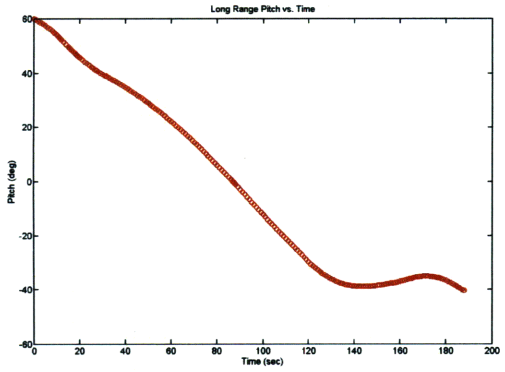
Figure 4.2: ProNav with Gravity Bias Geometry

Despite its simplicity, this small change drastically reshapes the average trajectory and greatly improves performance. Figure 4.4 shows the results of the simulation running the exact same initial conditions as seen in Figure 4.3 except the guidance system

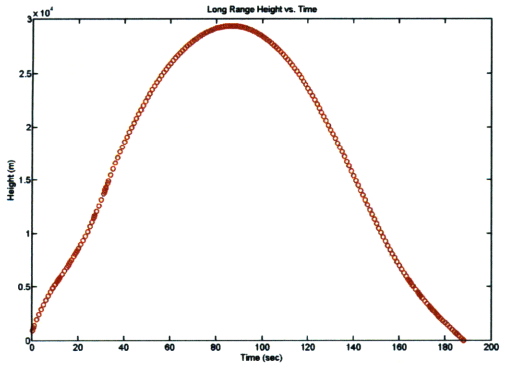
is ProNav GB. It is easy to see that control authority is not wasted at apogee, the final terminal angle is more negative, and altitude is straighter. In fact, ProNav is not just wasting control authority but pulling in the wrong direction. ProNav GB discovers that it must pull up or suffer the same fate as ProNav and miss the target. However, note that Figure 4.4 shows the canard deflection gradually relax then spike as the target approaches. Without this correction at the end, ProNav GB would have missed. Ideally, it would make this correction as early in the trajectory as possible, reducing the detrimental effect on terminal angle. ProNav GB is certainly not ideal, just better.



(a) Saturated Canard Deflection vs Time

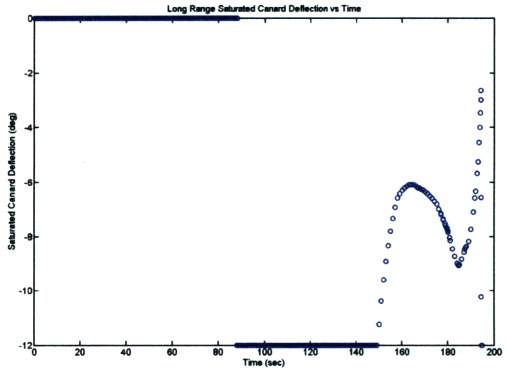


(b) Pitch vs Time

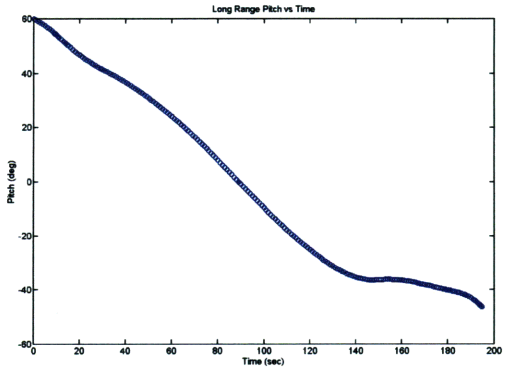


(c) Height vs Time

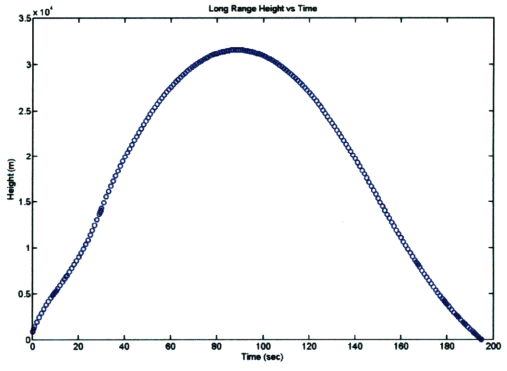
Figure 4.3: ProNav Misusing Control Authority



(a) Saturated Canard Deflection vs Time



(b) Pitch vs Time



(c) Height vs Time

Figure 4.4: ProNav GB Using Control Authority

4.1.3 Offset Target Scale

OTS continues this escalation, adding a few more steps to the guidance process with the hope of significantly bettering the munition's performance. This method utilizes a falling aim point, starting at the intersection of the target's downrange value and the munitions maximum altitude, see Figure 4.5. It falls as the following function:

$$H_a = \frac{1}{2} g_c OTS_{gain} t_{go_{est}}^2 \tag{4.6}$$

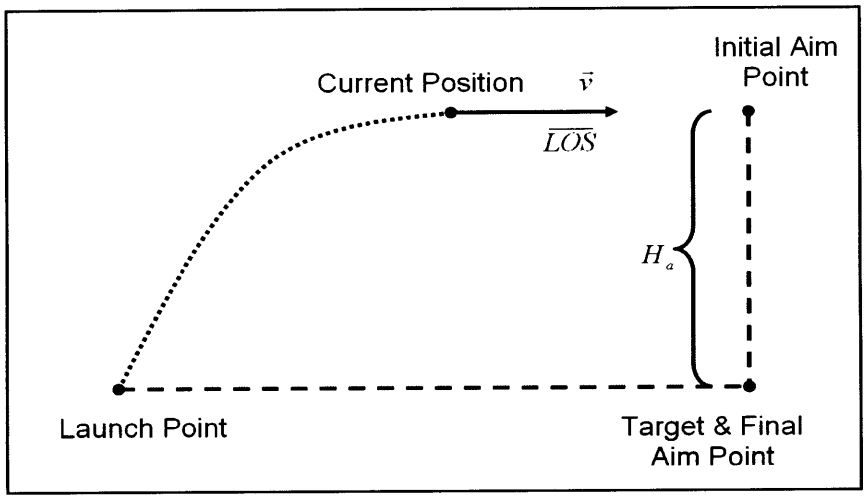


Figure 4.5: OTS Geometry

In Equation 4.6, H_a is the aiming point height, g is a constant approximation of gravity, OTS_{gain} is a user defined constant gain, and $t_{go_{est}}$ the estimated time until impact. Looking at the equation, it is obvious that the units of the aiming point are dependent on the length units used in the gravitational approximation. Depending of the gain used, this falling aim point can keep OTS from ProNav's tendency to "belly-flop" into the target.

Specifically, besides the moving aim point and some additional stability checks, the difference between OTS and ProNav GB boils down to one additional step. OTS estimates the time until impact as portrayed in Equation 4.7. Note that this is an approximation of the time until impact. The validity and affect of this assumption on the munition's performance is tested later in this work.

$$t_{go_{est}} = \frac{\overrightarrow{LOS} \cdot \vec{v}}{|\vec{v}|^2} \tag{4.7}$$

These few differences all occur within *homing_cmd.m*. In *guidance_cmd.m*, OTS acts identically to ProNav GB.

Chapter 5

Establishing The Baseline Performance

5.1 Software Performance Envelope

The first step in establishing a baseline is to determine an appropriate way to quantify and compare performance. The chosen metric looks at probability of kill versus minimum terminal angle. Probability of kill is an attempt to quantify the munition's lethality in a single number. Lethality is assumed to be a function of terminal angle and accuracy. Accuracy is intuitively a part of measuring the munition's effectiveness as is terminal angle, especially against hardened targets. Without further information, these two terms are weight equally. Accordingly, probability of kill is defined as the percentage of scenarios within the seed that hit within five meters of the target with at least a certain amount of terminal angle. Figure 5.1 shows the performance of each one of the 500 Monte Carlo scenarios for one simulation configuration. During baseline determination, a configuration refers to a particular guidance gain and rocket ignition time (RIT) setting that is held for all the Monte Carlo scenarios. While both ProNav GB and OTS have a couple tunable gains, only one is adjusted while the others are held constant. It is also important to bear in mind that, while it is possible to specify an ignition time for the rocket, each scenario has some random error. Figure 5.1 demonstrates a few scenarios that failed to hit and that, even within the portion that hit the target, some landed with more desirable terminal angles than others. Note that this figure and all subsequent performance metric figures display terminal angle as a positive number. All favorable terminal angles are negative within the simulation. However, the sign is switched so that the more preferred conditions are up and/or right in the figures. Besides creating a more intuitive figure, eliminating the negative signs also removes redundant data as all

scenarios, even uncontrolled ballistic firings, impact the ground with a negative terminal angle.

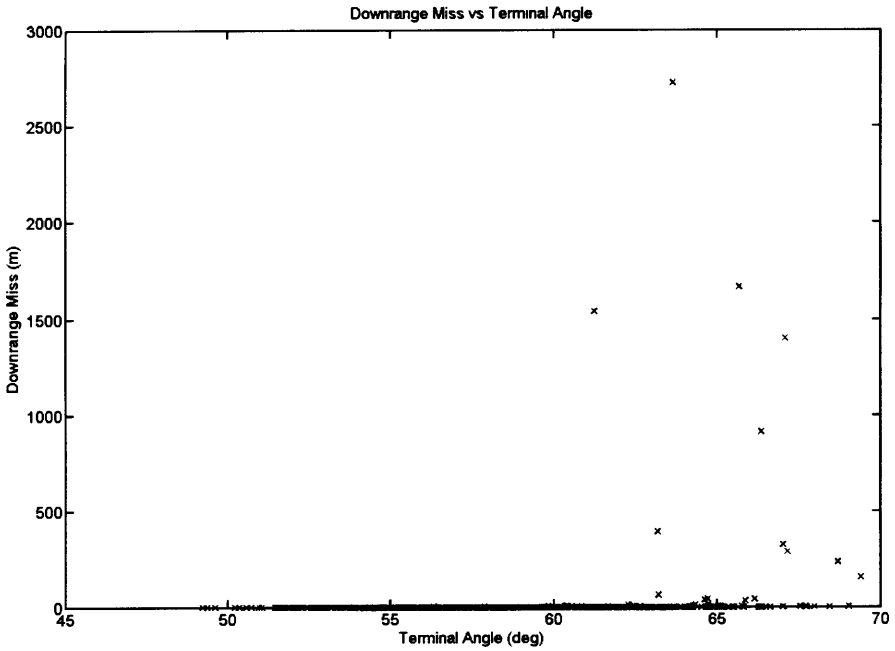


Figure 5.1: Downrange Miss vs Terminal Angle for Single Monte Carlo Run

Figure 5.2 illustrates how the data in Figure 5.1 appears in the performance metric. Each of the "x"'s represents a minimum terminal angle bin. After the data is generated, a file runs through and determines for the run how many of the 500 scenarios in the seed hit the target with at least the terminal angle specified by that bin. It performs this for all the bins, from 30° to 80° in 0.2° increments. The "x"'s are not connected. The metric has many practical applications besides making large amounts of data easier to evaluate. It portrays the trade off between terminal angle and probability of kill, quickly showing the cost/benefit of changing the munition's configuration to increase the former or latter performance factor. This is very useful when configuring the munition to have the desired performance against a target given one or several mission constraints. For example, assume an identified target, such as a hardened facility, requires a certain minimum terminal angle for the munition to cause any lasting effect. A computer or table identifies the configuration that guarantees at least that terminal angle, according to the simulation. The user receives this information along with the corresponding probability of kill. Taking in to account issues such as proximity of civilian people and buildings

and target importance, the user decides if the munition configuration is acceptable. If it is, the user fires the munition. If not, the user requests a configuration with a higher probability of kill, sacrificing the minimum terminal angle. Altering the configuration in this way increases the chance of hitting the target but lowers the minimum terminal angle. If reducing collateral damage is the pressing priority, several shots could be fired until one hits with the terminal angle necessary to damage the facility.

For comparison, an earlier iteration of the metric was to graph probability of hit versus 50% median terminal angle, where a "hit" was still defined as within five meters of the target. There are two large problems with this metric. First, hitting a target alone has no guarantee of achieving the goal since the definition has no inclusion of terminal angle, and terminal angle is apart of effectiveness. Second, 50% median angle is not a very useful statistic for this application. The graph should ultimately tell the user what values to set the guidance and ignition system at to obtain a certain performance. However, median angle only tells the user that after firing several of the munitions with those settings specified on the graph, one could expect 50% to have the terminal angle indicated on the graph. So even after sacrificing probability of hit, the user still has no assurance of the expected terminal angle for a single shot.

Filling out this graph to make it useful is a simple but time consuming task. Determining when Monte Carlo testing has converged to a solution is more qualitative than quantitative. Ultimately making that determination becomes a balance between time and utility. For this work, simulation runs ceased once ProNav GB and OTS formed discernible and fairly smooth envelopes. Figure 5.3 shows the final satisfactory curve. It represents the results of varying the gain in ProNav GB from 0 to 4.3 g's, the gain in OTS from -0.5 to 2.6, and the RIT from 29 to 39.5s for both the ProNav GB and OTS gain configurations. The simulation also ran cases where ProNav GB and OTS ran simultaneously. The line on the graph connecting the circles, represents the limit of software performance or the software envelope. The envelope represents the best discovered performance for the range of configurations tested. Table 5.1 matches the points on the envelope with the guidance settings used and performance achieved. Given the tested configuration range and fairly smooth transition between configurations, the software envelope is much closer to defining the "optimal" rather than just "good" performance. OTS always outperformed ProNav GB and, as such, generates all the points on the software envelope. The table only shows configurations resulting in a probability of kill over 50%. This is done to save space, and because decreasing benefits in minimum terminal

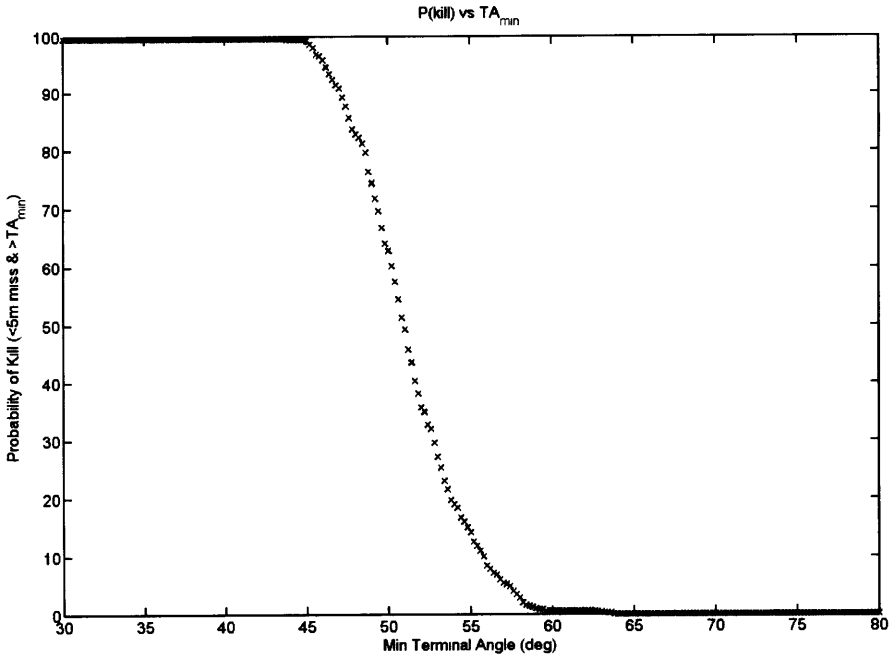


Figure 5.2: Single Monte Carlo Run

angle come at increasing costs to probability of kill. It is difficult to envision situations justifying these high increases in the probability of missing the target for only a few extra degrees of terminal angle. Because all attention is on system performance and none on flexibility, it is impractical to program a real munition with this graph. The envelope is used to simplify comparison with new guidance systems in the next chapter. Performance can be compared to performance without having to quantify and compare flexibility.

5.2 Hardware Performance Envelope

While the software envelope is established, there is no method yet discussed to determine how "good" that performance is relative to the physical potential of the munition. For the sake of discussion, imagine a performance line that quantifies the hardware limit, a line that represents the aerodynamic limits of the munition reachable only when a perfect guidance system had perfect a priori knowledge available. Assuming it is determinable, this line has great value in illustrating how much improvement potential exists, an item of tremendous interest especially after a successful alteration to the guidance system. Once a method of improvement is discovered, pushing the envelope up and/or

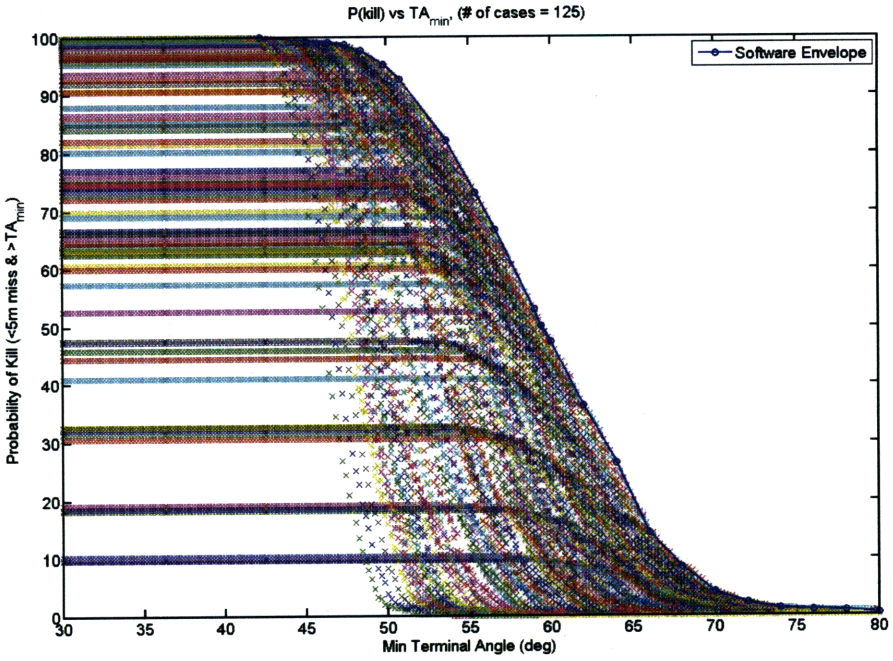


Figure 5.3: Baseline Performance and Optimal Envelope

Table 5.1: Optimal Performance and Guidance Identity

Number	TA_{min} (deg)	P(Kill) (%)	Guidance	Gain	Rckt Start (sec)
1	59.4	50.1	OTS	2.0	30.50
2	59.0	52.9	OTS	2.0	30.50
3	56.6	66.7	OTS	1.9	30.75
4	55.4	73.1	OTS	1.9	30.75
5	53.6	82.2	OTS	1.9	31.00
6	50.8	92.6	OTS	1.8	31.25
7	49.8	95.2	OTS	1.8	31.25
8	48.4	97.6	OTS	1.7	31.50
9	47.4	98.6	OTS	1.7	32.25
10	46.4	99.0	OTS	1.6	32.00
11	45.0	99.4	OTS	1.6	33.00
12	42.2	100.0	OTS	1.4	34.50

to the right, the hardware limit could easily show how much potential improvement remains. In other words, the designer could use some amount of cost/benefit analysis to determine when the next alteration, depending on its complexity, is no longer worth the potential gain in performance.

While extremely useful in theory, the actual determination of this line proves difficult. All attempts produce mixed successes, where the resulting line has to be interpreted carefully. The final attempt represents the best balance between portraying an achievable but unsurpassable hardware envelope. It assumes the optimal controller resembles a bang-bang controller. A search algorithm attempts to find a RIT, the "ideal" RIT, that allows the munition to hit the target if the guidance system pulls towards the ground from the time it turns on until impact. If obtainable, the "ideal" RIT maximizes performance as all the control authority focuses on increasing the terminal angle. When determining this ignition time, the RIT errors are hidden from the search algorithm. This prevents it from unknowingly adjusting to the errors, making any results impractical without implementing a system to eliminate the RIT errors. Another reason to hide the RIT errors is that the information is unobtainable when it is most desirable to know, prior to rocket ignition. The INS could determine, within limits, how much error took place on a given test only after the rocket started firing. The other errors are left visible for the search algorithm because the hardware performance loop represents a perfect guidance system with perfect pre-launch and post-launch data. This is necessary so that when modifications are made to the guidance system to adapt to pre- and post-launch information, its performance can never surpass this established hardware envelope. It would serve as a poor measure of the system's ultimate performance limit if a software system change causes the munition performance to exceed the hardware envelope.

With the errors back in place, a 3σ, 1 second, safety margin is added to the ideal RIT. For example, if a RIT of 29.8 seconds puts scenario 186 within 5 meters of the target, the safety margin changes that ignition time to 30.8 seconds. Recalling that a later ignition time causes the munition to ballistically travel a shorter distance, requiring that the guidance system pull up for some amount of time before directing its effort to pulling down and increasing terminal angle. This hardware performance attempt then takes that rocket ignition time with the safety margin and uses a search algorithm to find the minimum time to pull up. It pulls up immediately after the guidance first turns on to increase downrange just enough so that it can spend the rest of its control authority pulling down. Table 5.2 shows the statistics and distributions generated by the "ideal"

controller up-time search algorithm. The first column shows the attribute of interest. The difference between the two "up time" attributes is that **Ctrl Up Time** is the time to pull up requested by the search algorithm. This is not identical to **Actual Up Time** because the simulation runs at a variable time step so guidance may not be able to change the canard deflection at the exact time of the command. **Scope** identifies whether the statistics shown are for all the data or only the filtered out successes, hitting within five meters of the target. The last two columns show the spread of the "ideal" control up time. Demonstrating the factor's sensitivity, there is only around a four second difference between the shortest and longest downrange scenario in the Monte Carlo seed. Figure 5.4 represents Figure 5.3 with the generated hardware performance limit. The unfiltered hardware performance envelope has no implementable value and is not shown in any following performance figures. It is merely presented to visually demonstrate that the lift per unit of canard deflection when the guidance first runs on for the short range scenario is too sensitive for the simulation to converge to a solution for all scenarios with its current tolerance.

Table 5.2: "Ideal" Control Up Time Statistics and Distribution

	Scope	Mean	50% Median	Std Dev.	Earliest	Latest
DR Miss (m)	Filtered	0.15	0.13	2.60	NA	NA
	All	-185.20	-0.64	741.02	NA	NA
Ctrl Up Time (sec)	Filtered	3.13	3.27	0.89	0.33	4.50
	All	3.34	3.53	0.99	0.00	4.50
Actual Up Time (sec)	Filtered	3.13	3.27	0.89	0.33	4.50
	All	3.33	3.51	0.99	0.00	4.50
Terminal Angle (deg)	Filtered	-72.82	-72.20	5.10	NA	NA
	All	-73.19	-72.34	6.09	NA	NA

There are many important points to keep in mind when interpreting this hardware envelope. Because the rocket ignition time and time to pull up are hard coded, any unaccounted or imperfectly accounted force that drives the munition towards the target puts the munition in an unrecoverable state. The flexibility gained by adding the safety margin to the ideal RIT is eliminated by determining the minimum control up time. After pulling up for the determined amount of time, the munition has just enough control authority left to hit the target. Any miscalculation in such factors as control authority, wind, and air density, doom the munition to miss the target. As a result, the performance represented in Figure 5.4 is not obtainable since this guidance could not be implemented on real world system even if the guidance system had perfect knowledge

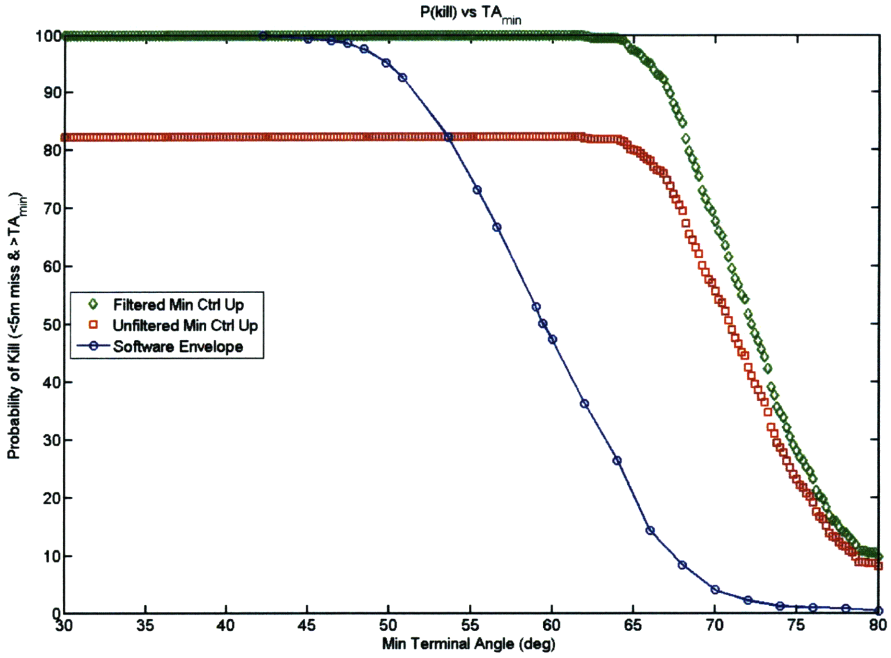


Figure 5.4: Software and Hardware Performance Envelope

and could change the RIT after launch.

In some ways this is a benefit, and in others, a detriment. For example, this hardware envelope is a good ceiling to have as it can never be out done, one of the aims. Solutions will just approach it. On the other hand, this boundary does not give a great idea how much improvement is possible on a system that needs to balance performance and flexibility. For other works, adjusting the methodology of the attempt could create a more realistic hardware boundary that incorporates flexibility is a more practical manner. Staying in line with the idea of the software envelope, this is actually desirable for this thesis work and allows easier comparison of pure performance to performance. As such, this attempt suites the needs of this work well.

Chapter 6

Industry Standard Guidance Algorithm Characteristics and Issues

In addition to any issue mentioned for a particular guidance system, all of these models thus far share some important disadvantages. Understanding these problems are critical to designing a new guidance system that can match these older algorithms when they are at their best and beat them outright when at their worst.

As is seen, all of the standard guidance systems have at least one user-defined, unitless gain. Usually these gains are just a tuning knob with no link to the physical system. Determining appropriate settings for this gain normally comes from simply running the simulation a very large number of times. During these runs various parameters are systematically altered so that eventually enough of the system's state space is examined. Even within this relatively basic simulation there are too many parameters to directly examine every scenario, or possible combination of parameters, so a method known as Monte Carlo testing is utilized. In Monte Carlo testing, many of the parameters are normally distributed with statistically determined sigma values. A large number of runs are accomplished, this work used 500, constituting a seed. Within this seed, the standard deviation is randomly determined for every run's normally distributed parameters . The seed is held constant for all simulation runs so that each run is faced with the same, unknown errors. In the end, if enough scenarios are run, it is assumed that all of the most likely scenarios are examined. So if the guidance system performs well within the Monte Carlo runs then it should perform well for the vast majority of real world scenarios. There are many debatable assertions made within this argument. To name just a few:

1. How many runs constitutes a "large number"?

2. How were the standard deviations determined for each parameter?
3. How well do those standard deviations encapsulate the system's real environment?
4. In the end, is the modified guidance system now adjusted for real world or just the seed?

Additionally, these systems are by design rather inflexible. Anytime a significant parameter in the rest of the simulation or real system is altered, the test campaigns have to be rerun to ensure the old gain has not made the guidance too unreliable. To help compensate, the designer is forced to sub-optimize performance for the sake of robustness. This is a necessity because the unitless gain is based on simulation runs, where both the run settings and simulation itself are imperfect matches to the real world. So after all the Monte Carlo simulation runs necessary to define the optimal gain, a sensible implementer for this kind of empirical guidance system must sacrifice for a factor of safety.

6.1 Correlation

There are several other weaknesses in the mentioned guidance systems. Most importantly, all of the above guidance systems, as described, are largely fixed from the time of launch to the end of the flight. Vast amounts of new information becomes available through GPS and the system's INS, yet little of it is used in the guidance system. Also, while it accounts for gravity, OTS does not account for other significant forces that could be approximated relatively easily, such as drag and lift. However, attempting to augment the current guidance system to beneficially use all the new information obtained after launch would be difficult and greatly increase the computational demand. Understanding which parameters the performance most heavily depends upon could give an idea of where to focus attention.

The simulation conducted several correlation tests to try to filter the most important information from the rest. Figures 6.1 and 6.2 visually represent the correlation coefficients of the performance parameters and errors to each other. Table 6.1 holds the name of the performance parameter or error to the corresponding row and column in the figures. For example, the most significant negative point in Figure 6.1, the darker blue point in row 2 column 12 of Figure 6.2, is the correlation of the aerodynamic center of pressure modeling error, $X_{cp_{bt}}$ error, to terminal angle. Also note that the matrix is symmetric, as the correlation of $X_{cp_{bt}}$ error to terminal angle is exactly the same as the correlation of terminal angle to $X_{cp_{bt}}$ error. The MATLAB color scheme "JET" is used,

meaning dark red corresponds to a perfect proportional correlation of 1, while dark blue represents a perfect inverse correlation of -1. Colors in between correspond accordingly. As seen in Figures 6.1 and 6.2, each factor perfectly correlates with itself, represented by the high, dark red peaks on the diagonal. There also is some significant correlation in the 12^{th} row/column, the $X_{cp_{bt}}$ error, and near the origin, the rocket errors. The majority, in between light yellow and light blue, is noise. Logically, this makes sense as a randomly generated error should have no significant correlation to another randomly generated error.

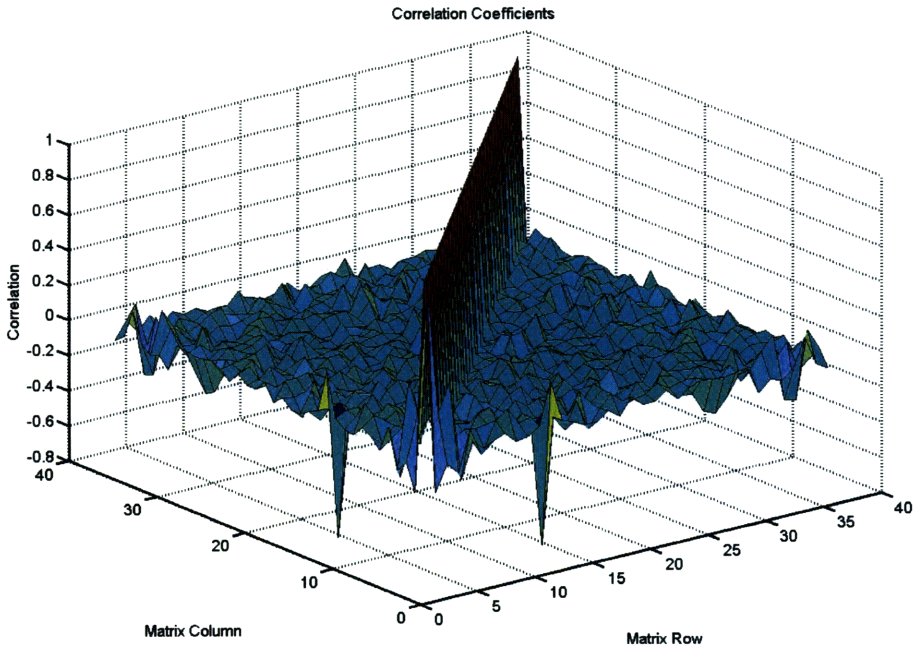


Figure 6.1: 3D Correlation Factors for OTS 1.7 and Rocket Ignition 31 sec

The level of noise is difficult to accurately quantify. However, looking at the maximum absolute value of correlation between the errors for several runs indicates the noise very rarely exceeds a correlation of 0.2. Therefore, any correlation value at or below that value is regarded as noise and ignored. Figure 6.3 shows a stem plot of the correlation factors for the particular run used in this example. The figure views the correlation matrix along the correlation-matrix row axis, more clearly illustrating the noise level.

There is also concern that the correlation coefficients do not represent the true trend in a data set, but are the function of just a few large outliers within each Monte Carlo

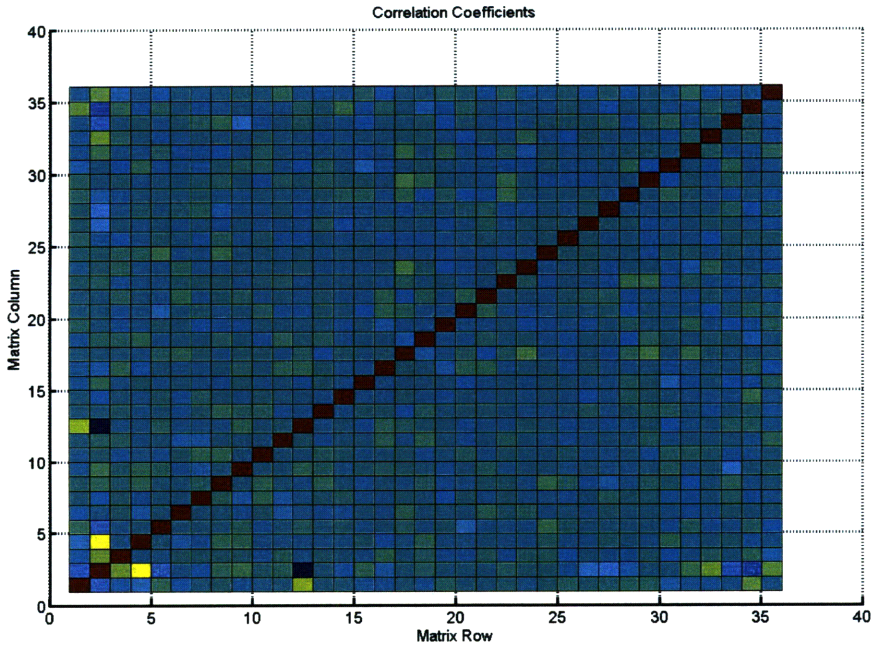


Figure 6.2: 2D Correlation Factors for OTS 1.7 and Rocket Ignition 31 sec

Table 6.1: Performance Parameter and Error Row/Column (i/j)

i/j	Label	i/j	Label	i/j	Label
1	Downrange Miss	13	ΔCP_δ Error	25	Mach 2.00 Error
2	Terminal Angle	14	Mach 0.00 Error	26	Mach 2.50 Error
3	Burn Time Error	15	Mach 0.60 Error	27	Mach 3.00 Error
4	Ignition Time Error	16	Mach 0.80 Error	28	Mach 3.50 Error
5	Gun Velocity Error	17	Mach 0.85 Error	29	Mach 4.00 Error
6	Thrust Error	18	Mach 0.93 Error	30	Mass Error
7	Gun Elevation Error	19	Mach 0.98 Error	31	CG Error
8	C_{N_α} Error	20	Mach 1.05 Error	32	Standard Pressure Error
9	C_D Error	21	Mach 1.10 Error	33	Standard Temp Error
10	C_{N_δ} Error	22	Mach 1.20 Error	34	Wind x_A Error
11	C_{M_δ} Error	23	Mach 1.40 Error	35	Drag Error
12	$X_{cp_{bt}}$ Error	24	Mach 1.60 Error	36	Lift Error

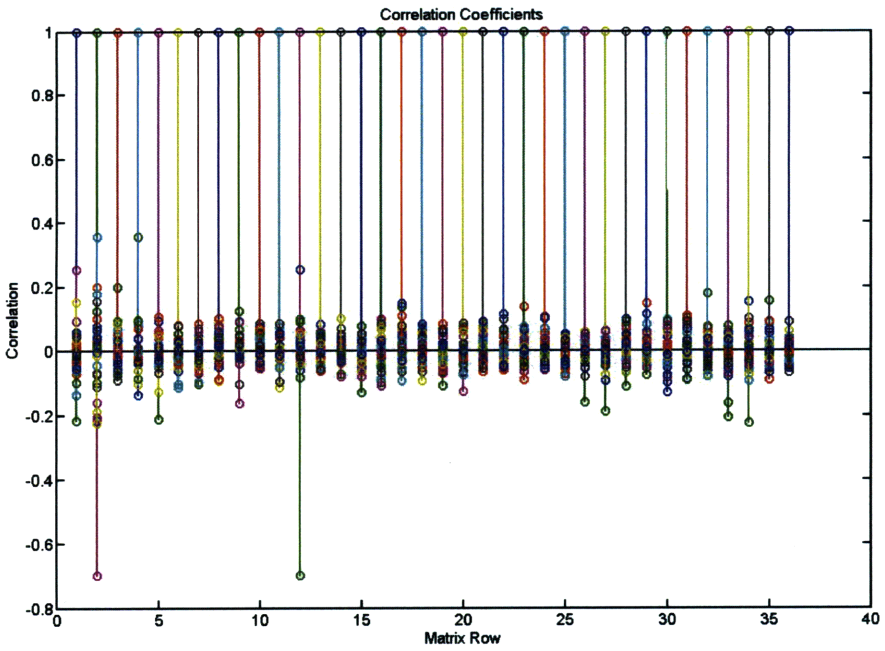
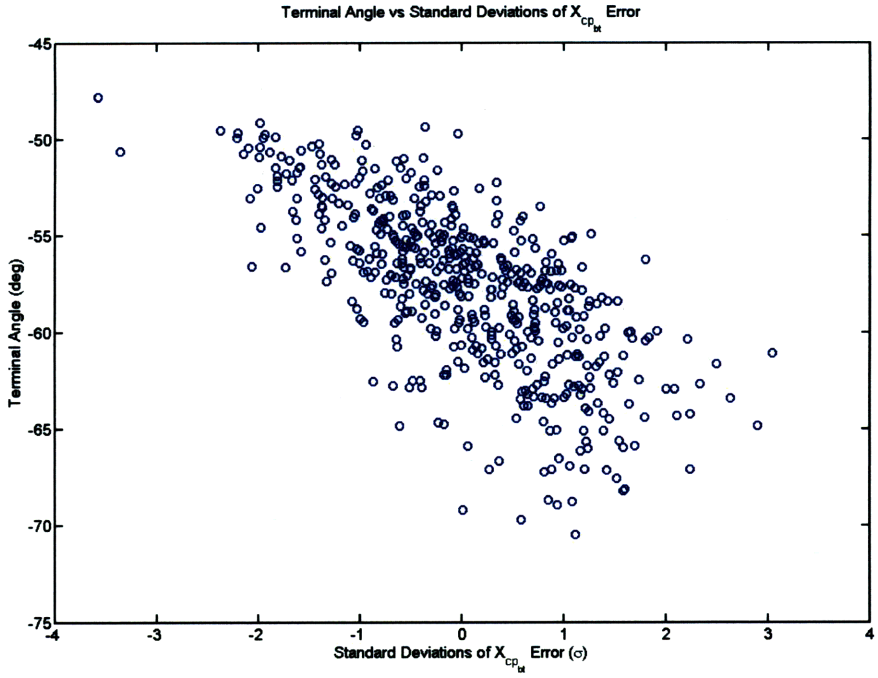


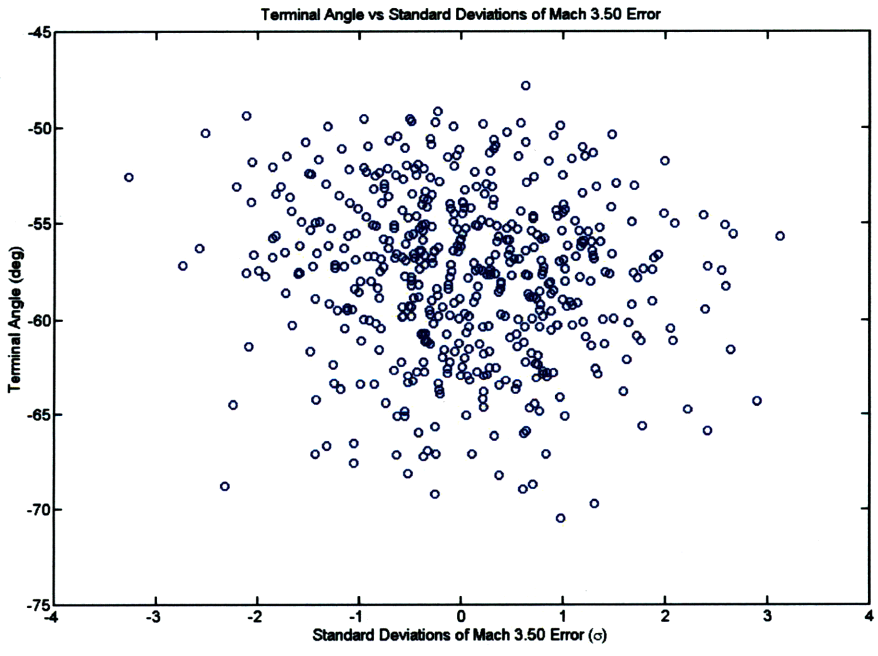
Figure 6.3: Stem Plot of Correlation Factors for OTS 1.7 and Rocket Ignition 31 sec

run. For several cases, all the coefficients are checked to confirm this is not occurring. Figure 6.4 shows terminal angle versus the standard deviations of two errors for each of the 500 scenarios in the seed. Figure 6.4a demonstrates the appearance of such a comparison when the error, $X_{cp_{bt}}$, had a relatively strong inverse correlation coefficient. Figure 6.4b illustrates the same comparison but against an error, Mach 3.50, with a negligible, noise level correlation coefficient. As demonstrated, the correlation coefficients properly represent the strength and direction of trends in the data.

To create a view of the correlation independent of the particular guidance gain and rocket ignition time of a single run, the correlation coefficients of many runs are averaged together. Table 6.2 shows these average correlation coefficients for the simulation runs in Figure 5.1, which represent an OTS gain range of 1.2 to 2.2 and a rocket ignition start range of 29 to 35.5 seconds. Only those coefficients corresponding to the performance parameters are shown. There are only two errors with correlation coefficients above the noise level. The data indicates that more accurately quantifying the $\mathrm{X_{cp_{bt}}}$ error should help both the downrange miss and terminal angle, while improved knowledge of the true ignition time should help only the terminal angle. If post-launch information is intelligently used to improve the guidance system's knowledge of these two errors, the



(a) $X_{cp_{bt}}$



(b) Short Range $\frac{\delta}{g}$

Figure 6.4: Terminal Angle vs. Standard Deviations of Two Errors

munition's performance may improve. Implementing a way to reduce the other errors appears ineffectual.

Table 6.2: Performance Parameter Average Correlation to Simulation Errors

Error	DR Miss	TA	Error	DR Miss	TA
Burn Time	-0.109	0.180	Mach 1.05	0.007	0.011
Ignition Time	-0.183	**0.280**	Mach 1.10	-0.022	0.016
Gun Velocity	0.075	-0.189	Mach 1.20	0.024	0.051
Thrust	-0.030	0.011	Mach 1.40	-0.029	0.024
Gun Elevation	0.001	-0.047	Mach 1.60	-0.006	0.041
C_{N_α}	0.001	0.047	Mach 2.00	0.056	-0.035
C_D	-0.056	0.122	Mach 2.50	0.051	-0.141
C_{N_δ}	-0.014	0.014	Mach 3.00	0.056	-0.172
C_{M_δ}	-0.029	0.028	Mach 3.50	0.032	-0.103
$X_{cp_{bt}}$	**0.248**	**-0.594**	Mach 4.00	0.032	-0.009
ΔCP_δ	0.008	0.017	Mass	-0.073	0.015
Mach 0.00	-0.003	-0.06	CG	-0.042	0.089
Mach 0.60	-0.056	0.074	Standard Pressure	-0.080	0.164
Mach 0.80	-0.020	-0.075	Standard Temp	0.031	-0.190
Mach 0.85	-0.001	-0.008	Wind x_A	0.141	-0.187
Mach 0.93	-0.047	0.049	Drag	-0.033	0.143
Mach 0.98	0.020	0.005	Lift	-0.047	0.035

6.2 Time-to-Go Estimate

As described earlier, there is a time until impact estimate within OTS, called $t_{go_{est}}$, that is used to adjust the aiming point. Given earlier, equations 4.6 and 4.7 estimate both the $t_{go_{est}}$ and aiming point height. If shown to be poor approximations and a better method of determination is discovered, performance might improve. Figures 6.5, 6.6, and 6.7 show the aiming height as a function of both the actual and the estimate time until impact. Each figure used the guidance gain and rocket ignition time specified in the caption, and only represent four of the 500 scenarios for each run, randomly chosen. The true time until impact is determined after the simulation run, where the time data is re-centered about the actual time of impact. The aiming point is then calculated. The assumed time until impact is saved during runs, and the aiming height is calculated after the simulation run is complete. Saving the latter aiming height directly is easier but would require a revision in the simulation.

These figures, and others not shown, indicate several behaviors of the estimation, some

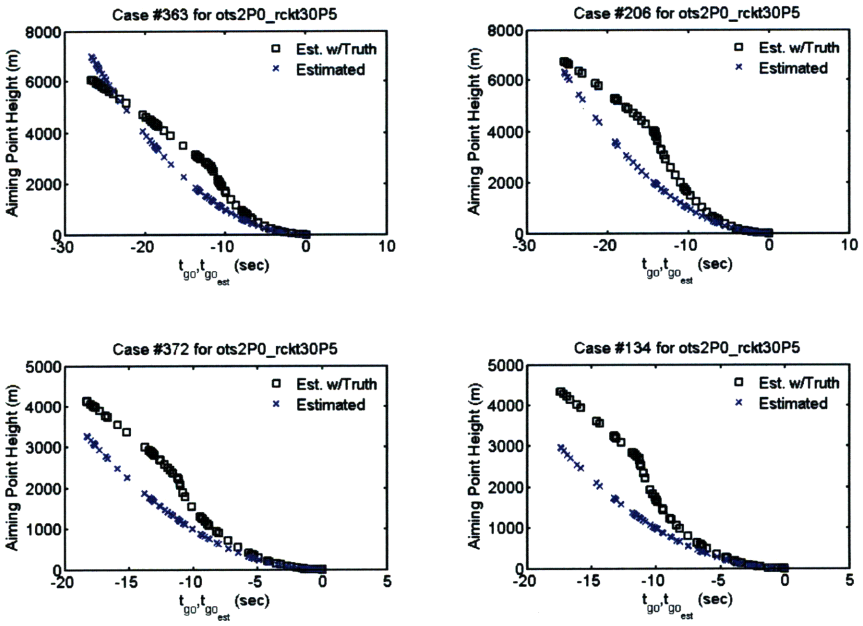


Figure 6.5: Aiming Height vs $t_{go}, t_{go_{est}}$ for OTS gain 2.0 and Rocket Ign of 30.5 sec

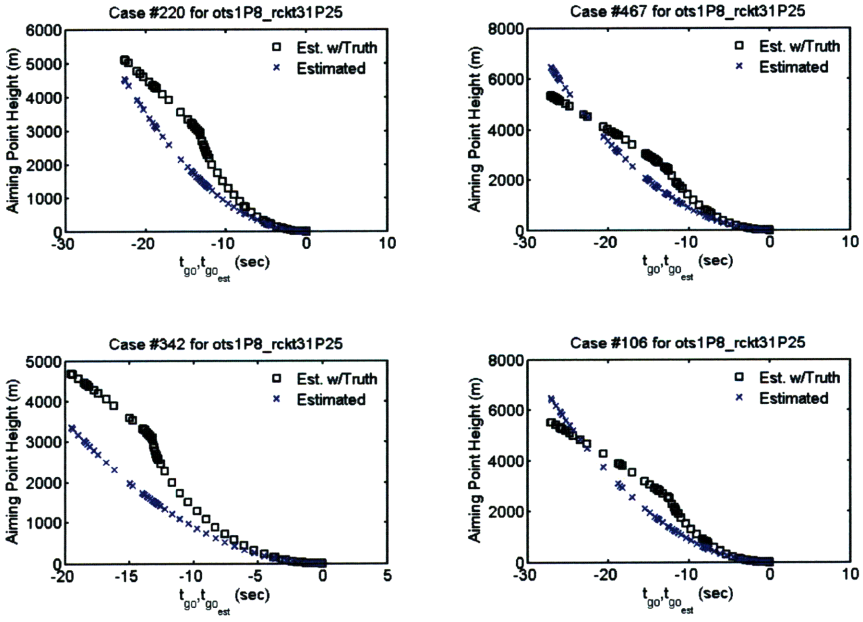


Figure 6.6: Aiming Height vs $t_{go}, t_{go_{est}}$ for OTS gain 1.8 and Rocket Ign of 31.25 sec

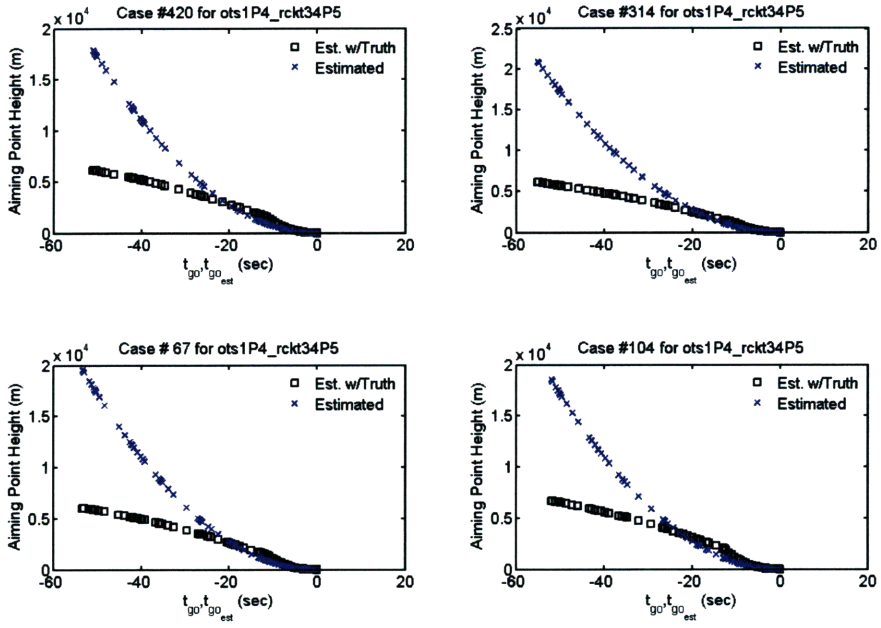


Figure 6.7: Aiming Height vs $t_{go}, t_{go_{est}}$ for OTS gain 1.4 and Rocket Ign of 34.5 sec

which may be corrected to improve performance. The current estimation converges to the truth as time until impact approaches zero. This means any correction made should be engineered to have the greatest affect when the guidance system first turns on and gradually fade to nothing. Comparing the three figures, the estimated aiming height is not always greater than or less than the actual. However, once initialized as either greater than or less than, the estimated aiming height tends to maintain that relationship as it converges. Additionally, the estimated aiming height's tendency to initialize greater than the actual increases as the rocket ignition start time occurs later. The data gathered shows that an equal tendency between estimated aiming height and OTS gain value does not exist. This indicates that trends exist between the error in aiming height and available post-launch information. So if the guidance system were altered to include such information as actual rocket ignition time, perhaps error in the aiming height could be reduced, ultimately improving performance.

However, before determining if such an alteration is possible, the benefit of reducing estimated aiming height error needs analyzed. The best possible outcome is to reduce that error to zero. By running the simulation with the true time until impact gathered from previous runs, the estimated aiming height error is nearly negated. It is not entirely

eliminated as the same scenario with every parameter unchanged, except the estimated time until impact, flies a slightly different flight path. Modifying the flight path changes the actual time until impact, so as the time until impact drops below ten seconds, it returns to the old method of estimation. This should not be a big concern as the estimated aiming height converges to the actual near the ground. If completely eliminating the $t_{go_{est}}$ error improves performance, a more implementable solution to do so could be attempted. However, if the performance does not improve with nearly perfect time until impact knowledge, it is a waste to attempt a more realistic solution.

6.3 Dynamic Programming and Varying Guidance Gain

6.3.1 Background

As described, the industry standard guidance systems have a user defined gain that remains constant for a particular run. Prior to fielding, designers use simulations and real world runs to determine the proper gain setting as a function of physical parameters. This "proper" gain setting function is not an optimization process. It is a somewhat subjective attempt by the designer to balance robustness and performance given the characteristics of the munition and the requirements of the user. The purpose is so the actual end user does not have to intimately know the munition to effectively use it. As a simple example, the user might just specify the distance to the target, information available to someone on a ship, and the system chooses the preselected designer gain to exhibit the desired performance for that scenario. Therefore, while it is not hard coded from scenario to scenario, the guidance gain is constant for a particular firing. While more complicated, a varying gain guidance system may perform better.

Dynamic programming tests this theory. It is an optimal control method that attempts to either minimize a cost or maximize a reward. This implementation of dynamic programming utilizes the idea of a cost rather than a reward. The form of the cost is user defined and mathematically represents the undesirable outcomes [7]. Despite it being a function of the system under investigation, there is some latitude in defining the cost. However, improperly defining it can create serious problems as dynamic programming attempts to minimize a cost poorly related to a system's performance metric. Debatably the most crucial aspect of it, dynamic programming is able to balance the preference for small present costs and the dislike of high future costs. Dynamic programming sums both the present and expected optimal future costs for subsequent stages when choosing

the current best action. The optimal control is defined as being a closed-loop optimal control of the form shown in Equation 6.1 [27].

$$u^\star = f(x(y), y) \tag{6.1}$$

Above, x are dependent states, y is the independent state, u is the control, and a "star" superscript represents optimal. The relationship between the states and the optimal control is known as the optimal control policy, f. A concept known as the "Principle of Optimality" is used to find the optimal control policy [27]. The principle is easily explained with an example. As seen in Figure 6.8, there exists three possible states, a, b, and c, in a multistage decision process. If the minimum cost from state a to b is $J^\star_{ab}$ and the minimum cost from state b to c is $J^\star_{bc}$, then the minimum cost from a to c has to be $J^\star_{ac}$, where:

$$J^\star_{ac} = J^\star_{ab} + J^\star_{bc} \tag{6.2}$$

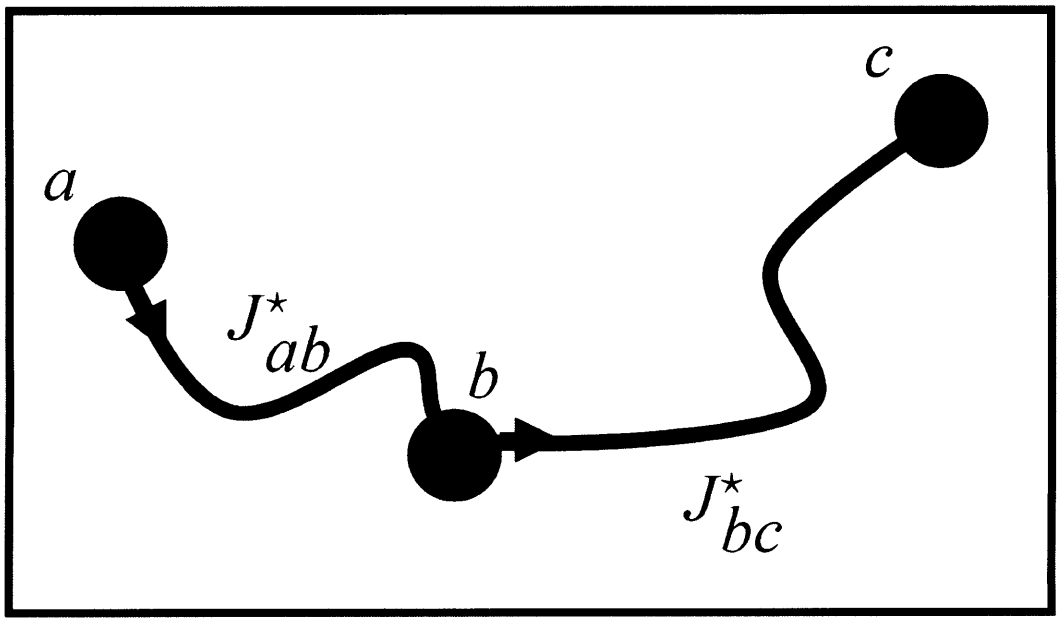


Figure 6.8: Principle of Optimality

The principle is easily proven through contradiction. The consequences of the Principle of Optimality on dynamic programming are best demonstrated by another simple example based on an illustration in *Optimal Control Theory: An Introduction* [27], shown

in Figure 6.9. Assume the goal is to transition from state a to f with the minimal cost. The direction and cost of transit between states is shown in the figure. Starting at a, suppose the minimum costs from b to f, $J^{\star}_{bf}$, and from c to f, $J^{\star}_{cf}$, are known. The decision is made by simply finding the lesser of the two costs:

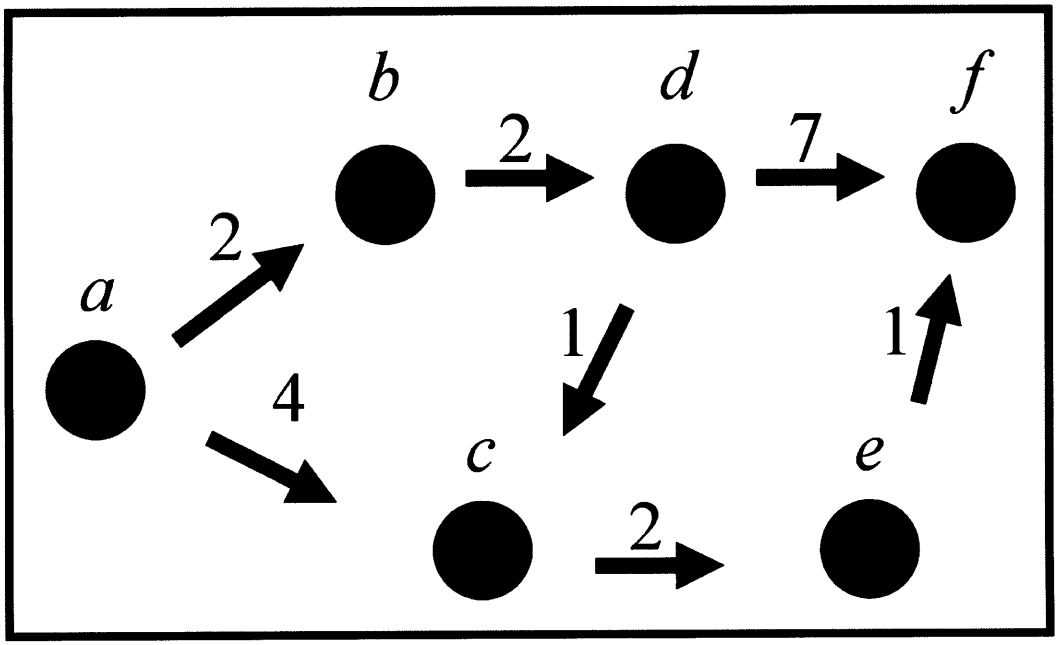


Figure 6.9: Principle of Optimality

$$
\begin{aligned}
J^{\star}_{af} &= \min\left\{J^{\star}_{abf},\ J^{\star}_{acf}\right\} \\
J^{\star}_{af} &= \min\left\{J_{ab} + J^{\star}_{bf},\ J_{ac} + J^{\star}_{cf}\right\} \\
J^{\star}_{af} &= \min\left\{2 + 6,\ 4 + 3\right\} \\
J^{\star}_{af} &= 7
\end{aligned}
\tag{6.3}
$$

As demonstrated, the optimal first step when traveling from state a to f is to go to state c. On a simple level, the example demonstrates how dynamic programming works, and how it balances between present and future costs. The large future costs of b to f outweigh the small present cost of a to b, so dynamic programming chooses the other path.

This leaves just one more issue to discuss, determining $J^{\star}_{bf}$ and $J^{\star}_{cf}$. Dynamic programming calculates these optimal costs starting at the end, state f, and working backwards. For example, as there is only one way to travel from state e to f:

$$J^{\star}_{ef} = 1 \tag{6.4}$$

In this simple example, optimally moving from d to f is only slightly harder. There are two choices, the lower cost establishing the optimal control. Just as before:

$$\begin{aligned} J^{\star}_{df} &= \min\{J_{df},\ J_{dcef}\} \\ J^{\star}_{df} &= \min\{7, 1+2+1\} \\ J^{\star}_{df} &= 4 \end{aligned} \tag{6.5}$$

This process of working backwards from the goal continues until the optimal cost and control policy from a to f are known.

Stepping back from this simple, specific example, the process of determining the optimal cost can be represented as: [25]

$$J^{\star}(x^{\imath}_k, y_k) = \min_{x^{\jmath}_{k+1}}[\Delta J(x^{i}_k, x^{\jmath}_{k+1}) + J^{\star}(x^{\jmath}_{k+1})] \tag{6.6}$$

or

$$J^{\star}(x^{\imath}_k, y_k) = \min_{u^{\imath\jmath}_k}[g(x^{\imath}_k, u^{\imath\jmath}_k, y_k)\Delta y + J^{\star}(x^{\jmath}_{k+1}, y_{k+1})] \tag{6.7}$$

Above i and j delineate between dependent states, k is the step number, and g is an approximation of an integrated cost. Equation 6.7 states that if the best path is know at states $x^{\jmath}_{k+1}$ and y_{k+1}, the control from state i to j at step k that minimizes the cost in between defines the optimal cost at states $x^{\imath}_k$ and y_k. The difference between equations 6.6 and 6.7 is that the first determines the control that moves the state from state i on grid k to state j on grid $k+1$, while the second quantizes the control inputs and evaluates the resulting state from each possible input. The latter method, which better describes the one implemented, develops an optimal control policy, from step $k = end$ to step $k = 1$ that optimally transitions the system from any current state to the state at the next step along the path. For more information on dynamic programming, consult Bertsekas [7], Kirk [27], or Bellman [6]

6.3.2 Cost Function Determination

Properly defining the cost function is a critical aspect to implementing dynamic programming. The coefficients in Equation 6.8 create a cost function with a strong preference for very small downrange miss, 90% of the cost reduction is within $\pm 20\,\mathrm{m}$ of the target. Equation 6.9's coefficients reflect more of the specific attributes of this system. Similar to the downrange miss, this cost function's greatest area of cost reduction is in the center. At a terminal angle of $-60°$, the cost if 0.5, and at a terminal angle $-60° \pm 20°$ the cost is 0.9 and 0.1, respectively. Ballistically, this system impacts the ground with a terminal angle around $-45°$. The terminal angle cost slope is designed to give dynamic programming the strongest incentive to pull the nose down only to a point. After $-65°$ or $-70°$, the cost reduction drops off, which, hopefully, causes dynamic programming to stop focusing as much on reducing the terminal angle. Limitations of this system make it extremely difficult to exceed $-70°$. Without reducing its priority, dynamic programming may waste a lot of control authority attempting to pull the nose down a few degrees instead of focusing on a more accomplishable goal of further reducing the downrange miss. Ultimately, the implemented cost function is merely a summation of the following two cost equations:

$$J_{end}^{R} = 1 - \frac{1}{0.0225\sqrt{x_{TCEF}^2 + y_{TCEF}^2 + z_{TCEF}^2} + 1} \tag{6.8}$$

$$J_{end}^{T} = \arctan\left(\frac{0.1539\,\theta_{end} + 9.234}{\pi} + 0.5\right) \tag{6.9}$$

$$J_{end} = J_{end}^{R} + J_{end}^{T} \tag{6.10}$$

Both functions are bounded between zero and one. Therefore, the total cost is bounded between zero and two. There is no direct cost placed on control. However, since the total control authority is limited, the runs with the least cost and best performance are those runs that best utilized their control. Above, x, y, and z are components of the terminal location relative to the target, Target-Centered, Earth-Fixed (TCEF) frame. There are no specific requirements for the position vector's coordinate frame in Equation 6.8. As long as the coordinate frame is centered at the target, the equation works. Terminal points stop when impacting the ground, leaving altitude equal to zero and cross range is also zero since it is currently not implemented. As a result, regardless of coordinate frame, any magnitude remaining in the position vector is downrange miss. Figures 6.10 and 6.11 respectively illustrate the previously described behaviors of the cost functions.

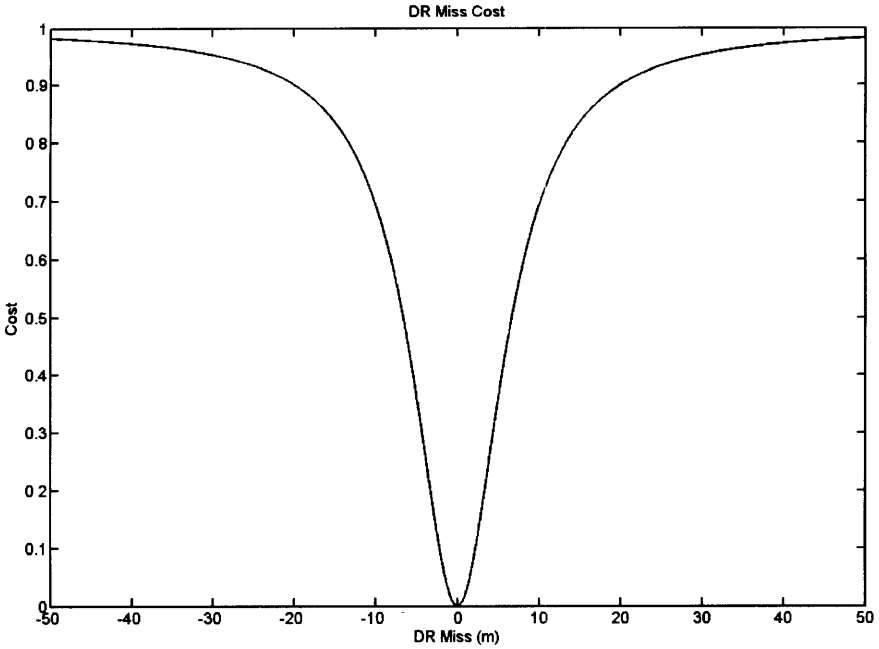


Figure 6.10: Downrange Miss Cost

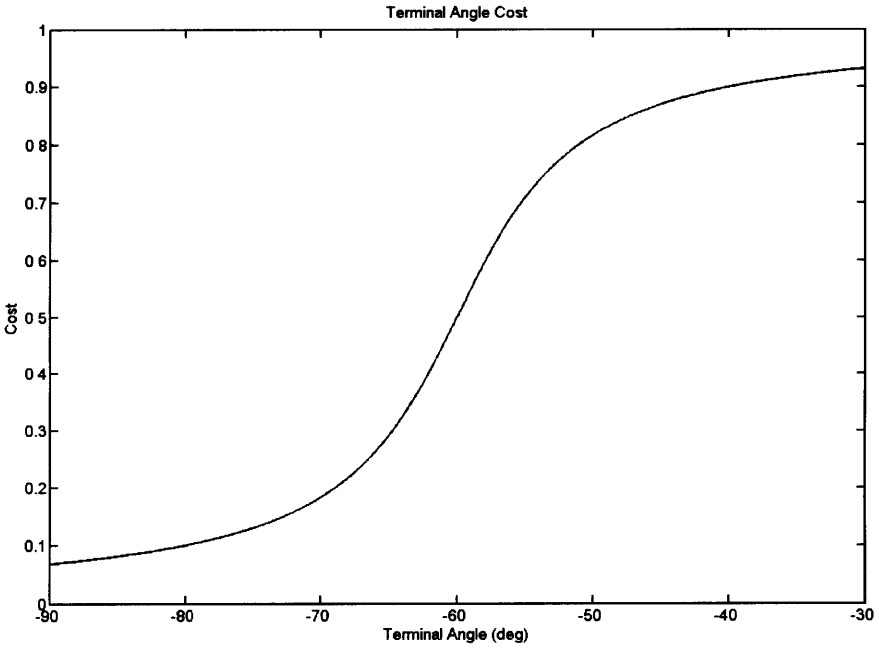


Figure 6.11: Terminal Angle Cost

The relationship of these two cost functions is also critically important to the ultimate performance of dynamic programming. With the described setup, the cost functions are the same as the primary metric, the probability of kill versus minimum terminal angle. The ability of the cost function to generate a high performance control policy relies heavily on the cost reduction rates or slopes within the functions and capabilities of the munition. For example, dynamic programming may find that little control is required to reduce the downrange miss to 6 m. Because of some physical characteristic of the system, attempts to further reduce the downrange miss require the remaining control authority. Instead, dynamic programming may then decide to split the control authority between reducing the terminal angle and the downrange miss because it results in a lower terminal cost. In the end, the optimal control policy causes most of the Monte Carlo scenarios to finish with a downrange miss slightly greater than 5 m and a very good terminal angle. However, when plotted on the probability of kill versus terminal angle metric, performance would appear poor as most scenarios did not hit the target by definition. While a possibility, this situation is unlikely because of the capabilities of the particular system modeled and the slope of downrange miss cost as it approaches zero.

Because of concern for the above behavior, the first cost function attempt more directly connected with probability of kill and minimum terminal angle. Total cost was still a summation of downrange miss and terminal angle cost. However, reductions in terminal angle cost were possible only when the absolute value of downrange miss was less than 5 m. Therefore, the optimal control policy favored scenarios with good terminal angles only once a hit was ensured. Unfortunately, the bandwidth for the downrange miss cost proved too narrow for the grid discretization, and almost every state combination failed to have any reductions in cost. Without any cost reductions, the resulting control policy merely reflected its initialized values. This led to very poor performance compared with a control policy derived with the current cost function. While not ideal, the utilized cost function balances metric performance and bandwidth well.

6.3.3 Discretization Grid

In comparison to simply attempting every possible state and control combination, dynamic programming scales better as:

$$N = N_i N_x^{\kappa} \tag{6.11}$$

In Equation 6.11, N is the total number of points, N_i is the number of discretizations in the independent variable, and N_x is the number of quantized states of dimension κ. While this is an improvement versus blind testing, dynamic programming scales poorly for higher dimension problems when compared to other algorithms, such as A* and Dijkstra's [30]. This scaling issue is so prominent a characteristic that shortly after Bellman introduced his generalization of the classical Hamilton-Jacobi theory [8], he coined the now famous phrase "curse of dimensionality" [6] to describe it. As an example of this, the first version of dynamic programming attempted a uniform, polar, target centered discretization of a simplified two DOF, four variable state space. The four variables were position from the target magnitude (R), line of sight angle from target to munition (LOS_t), velocity magnitude (V), and pitch angle (θ). Limiting the launch, target, and munition states to a two dimensional plane, these four scalar variables are all the information necessary to completely define the problem [19]. The first two states are centered at the target. For the sake of the example, assume each state is only chopped into ten slices, every state combination has only ten control options, each unique state combination and control option combination has to run through the entire Monte Carlo seed, and each individual test takes one second. The resulting estimated total simulation run time is shown below. It is clearly unacceptable.

$$\text{Estimated Run Time} = 10^4 \times 10 \times 500 \times 1 = 5 \times 10^7 \,\text{s} \approx 1.58 \,\text{yr} \tag{6.12}$$

Not only would it have run time problems, the preceding example has extremely poor resolution in necessary parts of the state space and wastes grid points in unnecessary parts. It simultaneously fails to supply satisfactory resolution and finish in a reasonable time.

A low control authority system actuals facilitates the task of limiting the state space. Even with control extremes, the path a successful munition travels from the launch to the target point is fairly well bounded. Better still, the exhibited performance of those few extreme cases is undesirable. As a result, any scenario outside of a nominal trajectory can use hard coded control to return within an acceptable distance of the nominal. This eliminates many areas of the total state space dynamic programming needs to test.

To determine the nominal trajectory, the simulation gathers the state information over the entire trajectory for a range of constant OTS gains. Previous testing establishes the appropriate range of OTS gains, where gains between 1.4 and 2 created the optimal

performance envelope. For additional robustness, runs used gains between -1 and 2.5. Figures 6.12, 6.13, 6.14, and 6.15 show the median nominal trajectories state information versus downrange miss, the independent variable, for all the Monte Carlo scenarios. Each OTS gain state information is displayed in a different color. Also illustrated are the ± 1 standard deviations from the median values, which usually grow as the munition travels from the launch point.

In the figures, the black dots represent the chosen grid points. They are functions of the most extreme minimum and maximum standard deviation values at a specified downrange miss. The distance between specified downrange misses shrinks as the munition approaches the target and the standard deviations grow. The two specified downrange misses furthest from the target occur before any of the scenarios at that distance and the next distance have a low enough pitch for the guidance system to activate. As a result, they are not added to the state grid. Figures 6.16, 6.17, 6.18, and 6.19 demonstrate this method of discretization's ability to cover the state space of all the scenarios. In the end, this method of discretization limits the state grid to 437 unique state combinations. While still very computationally demanding relative to the other methods tested, this is a huge reduction from the total unique states described in the earlier example and still supplies adequate resolution near the target [46].

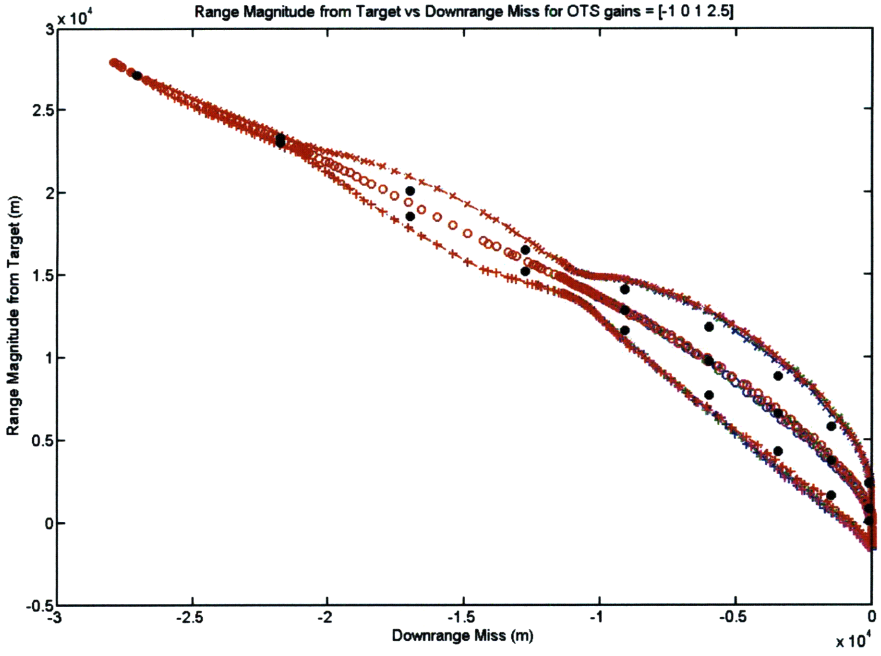


Figure 6.12: Nominal Range Magnitude to Target vs Downrange Miss

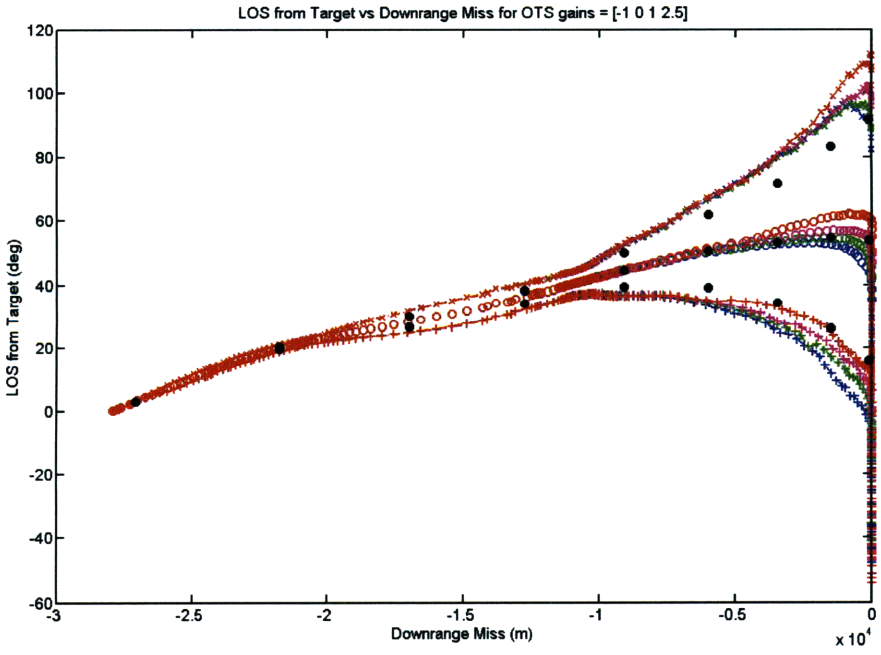


Figure 6.13: Nominal Line of Sight Angle vs Downrange Miss

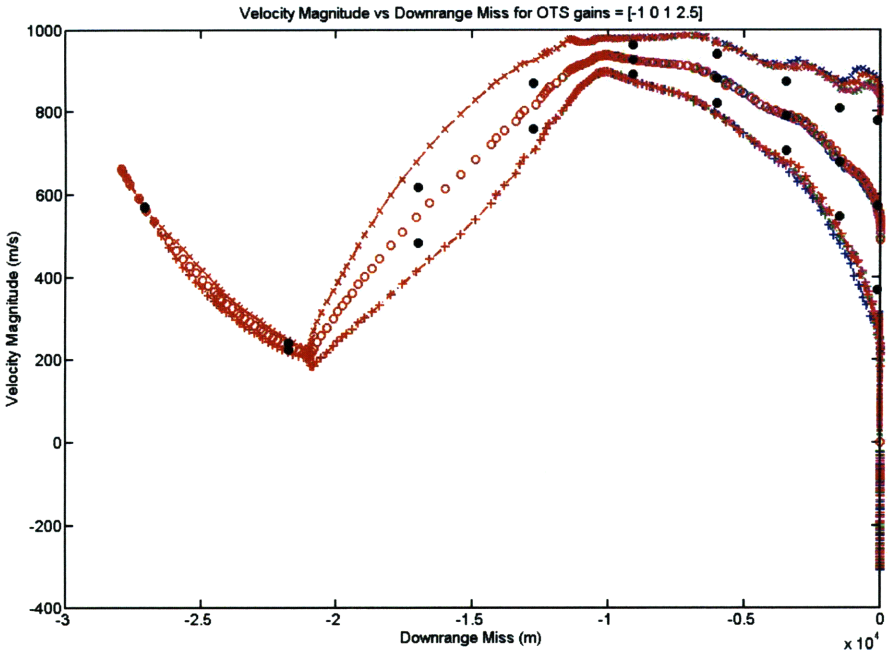


Figure 6.14: Nominal Velocity Magnitude vs Downrange Miss

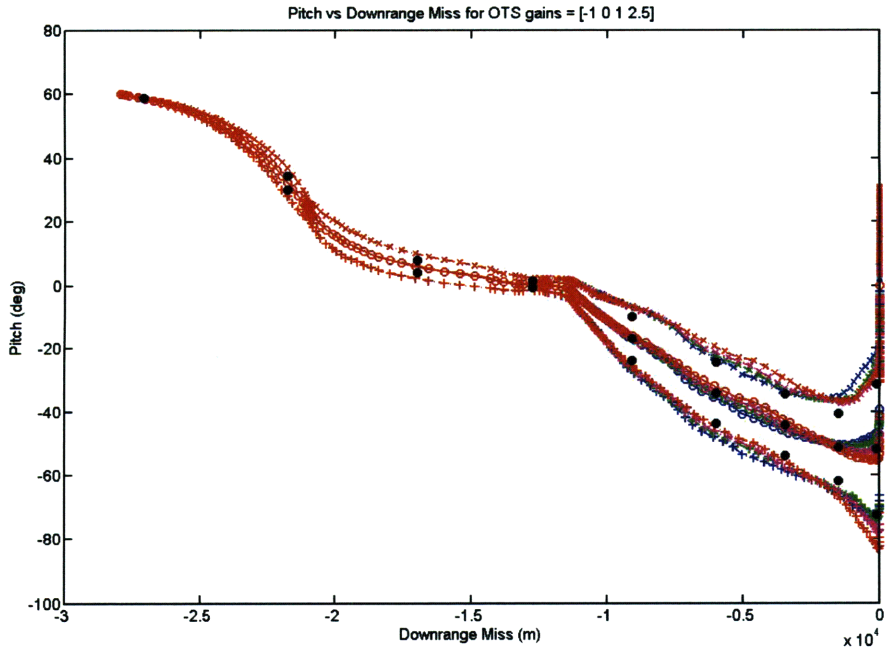


Figure 6.15: Nominal Pitch Angle to Target vs Downrange Miss

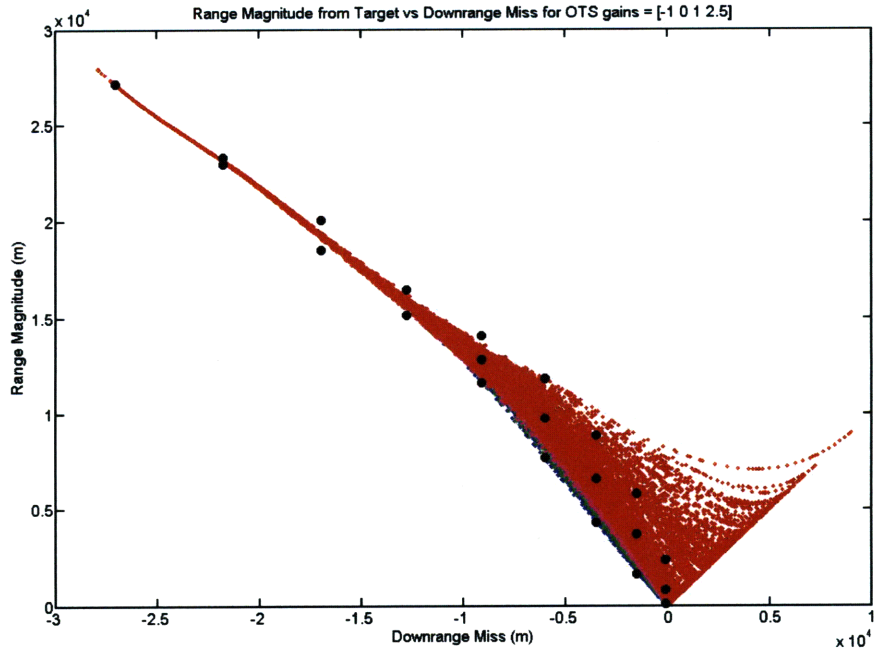


Figure 6.16: Range Magnitude Grid over All Scenario Trajectories

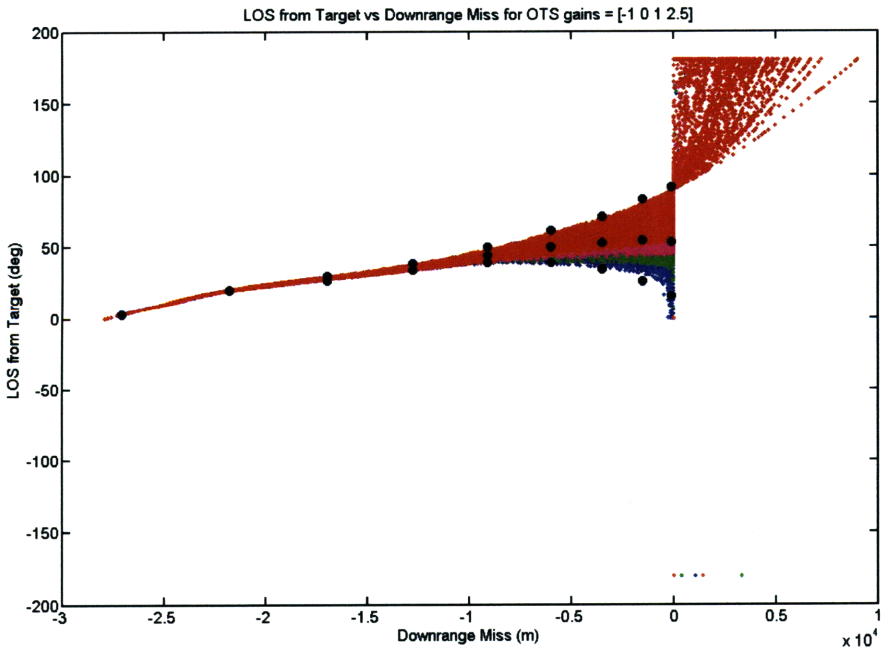


Figure 6.17: Line of Sight Grid over All Scenario Trajectories

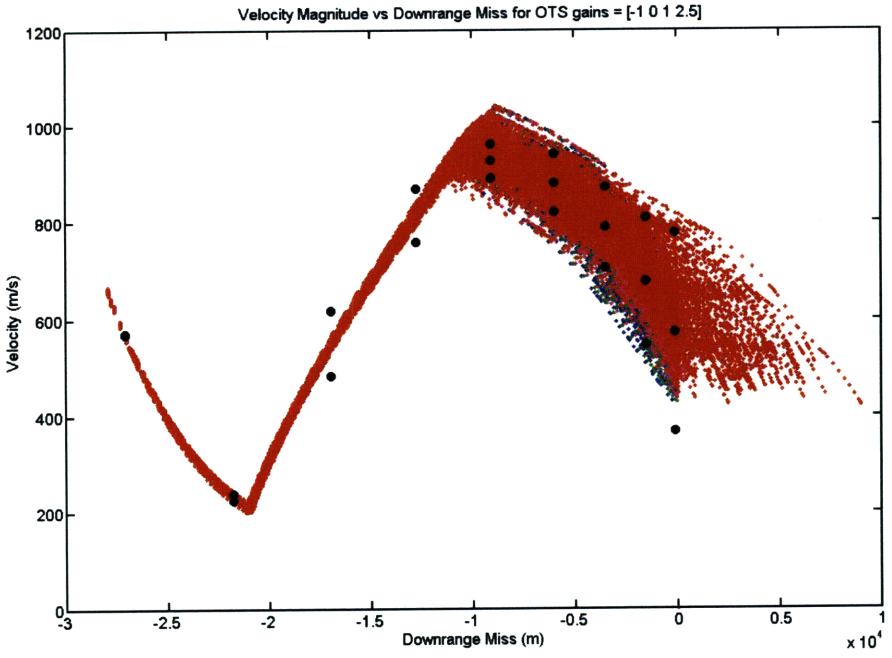


Figure 6.18: Velocity Magnitude Grid over All Scenario Trajectories

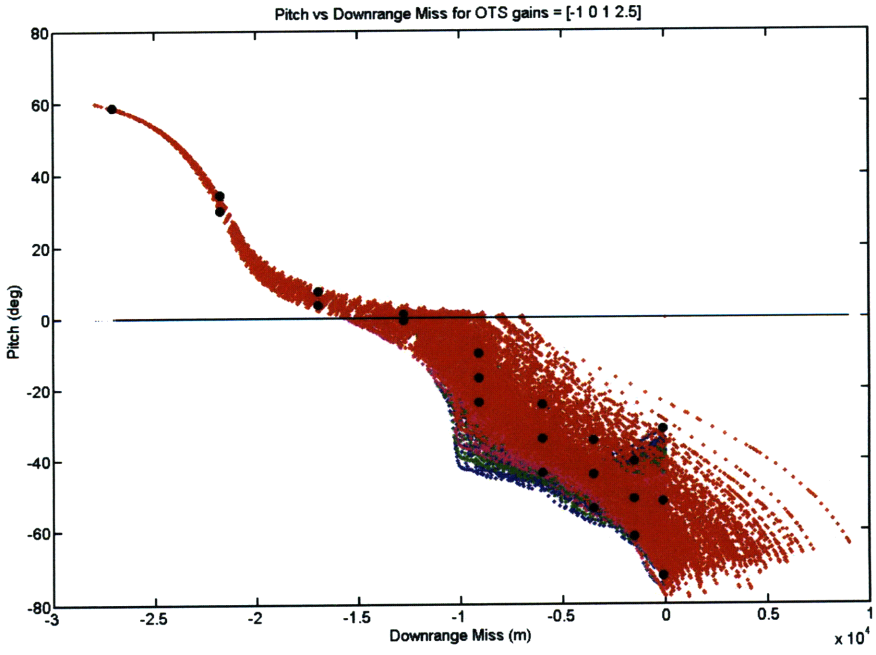


Figure 6.19: Pitch Angle Grid over All Scenario Trajectories

6.3.4 Implementing Dynamic Programming

In general, dynamic programming works as a value iteration process. A value matrix initializes with problem specific values. An iterative process determines the lowest cost action to take for each value matrix entry. Iterations through the matrix continue until the solution converges. This process is summarized in the algorithm below.

INITIALIZE VALUE MATRIX

$$V(x) = \begin{cases} J(x) & \text{If } x \text{ is a terminal state} \\ 3 & \text{Otherwise} \end{cases}$$

$i = 0$

REPEAT for all x

$$V(x)^{i+1} = \min_u V^i(f(x,u))$$

$$\zeta = \max_x |V^i(x) - V^{i+1}(x)|$$

$$i = i + 1$$

UNTIL $|\zeta| < 0.01$

Algorithm 2: Dynamic Programming Algorithm

The value function, $V(x)$, is initialized to a cost of 3 for all non-terminal states. As defined the maximum cost is 2. This impossible cost is always replaced on the first iteration.

At the beginning of the simulation run, a list of control options, OTS gains, is specified. The simulation runs every state combination at the smallest specified downrange miss to target with every available OTS gain until each scenario hits the ground. Initially, each state combination with each OTS gain only tests through the first 25 Monte Carlo scenarios. The simulation then determines the cost according to Equation 6.10 for each state/control combination. The cost of that state combination and OTS gain is the 50% median of those 25 scenarios' terminal conditions. For each state combination, the OTS gain producing the lowest cost is saved as the optimal cost along with its gain. Once the optimal cost and gain are determined for all state combinations nearest the target, the simulation tests all state combinations at the next specified downrange miss nearest the target. This time, the simulation runs each state combination using each OTS gain until its current downrange miss is equal to the nearest specified downrange miss. Its ending

state information is compared to the starting state combinations of the nearest specified downrange miss. The simulation determines the two closest by minimizing the vector norm and interpolates the cost. The interpolation process is summarized below.

$$
\begin{aligned}
J_{211} &= \frac{(J_2^\star - J_1^\star) r_{n1}}{r_{21}} + J_1^\star \\
J_{122} &= \frac{(J_1^\star - J_2^\star) r_{n2}}{r_{21}} + J_2^\star \\
J_{121} &= J_1^\star - \frac{(J_2^\star - J_1^\star) r_{n1}}{r_{21}} \\
J_{212} &= J_2^\star - \frac{(J_1^\star - J_2^\star) r_{n2}}{r_{21}} \\
J_n &= \frac{J_{211} + J_{122} + J_{121} + J_{212}}{4}
\end{aligned}
\tag{6.13}
$$

The process persists until the simulation determines the lowest interpolated cost for every state combination at the second nearest specified downrange miss. It then continues to work from all the state combinations at the next nearest specified downrange miss until it reaches the furthest downrange miss states. The whole process starts back at the first states tested until the largest change in optimal cost for all state combination falls below an established value.

The resulting control policy is a table of state combinations and each one's minimum cost guidance gain. During a performance run, the simulation simply stores the table for reference. When called, *homing_cmd.m* takes the current state information and looks up the two closest state combinations in the table, determined by minimizing the vector norm. Combined with the two closest table values and their minimum cost guidance gain, guidance linearly interpolates the gain to use. Other than the interpolated gain replacing the constant gain, the guidance system continues the process described in the previous chapters that ultimately results in a canard deflection command.

The assumptions to use the 50% median to determine cost and to calculate the optimal control policy with only 25 of the 500 Monte Carlo scenarios are not blindly accepted. They both raise a multitude of issues. Some are addressed later, while others are left to future work.

Chapter 7

New Guidance Attempts and Results

Overall, dynamic programming expanded the software envelope along the entire metric and was the only attempt nearing an implementable stage. Table 7.1 gives an initial impression of each attempts' effect on the software envelope (**SE**) performance, given an arbitrary constraint. The constraints are just two of many possible constraints placed on the munition by the user. The two arbitrarily chosen constraints give an idea of the performance improvement gained by implementing each attempt. The two constraints are configuring for the largest minimum terminal angle (TA_{min}) given a required probability of kill (**P(Kill)**), and configuring for maximum probability of kill given a required minimum terminal angle. For the probability of kill constraint, improvement is measured as the gain in minimum terminal angle. For the minimum terminal angle constraint, improvement is measured as the gain in probability of kill. The new attempts are listed in the order this chapter presents them: dynamic programming (**DP**), predictive rocket ignition time (**PRIT**), eliminating the $X_{cp_{bt}}$ error (**No** $X_{cp_{bt}}$ **Err.**), eliminating the rocket ignition time error (**No RIT Err.**), having a near perfect time until impact estimate (**Perf.** $t_{go_{est}}$), and the first attempt at an implementable way to reduce the time until impact estimate error (**New** $t_{go_{est}}$). Given the selected constraints, Table 7.1 shows that dynamic programming does not result in the greatest performance improvements compared to the other attempts. However, dynamic programming's gains in performance are realized improvements within the simulation, while the other attempts' improvements merely represent their greatest potential, except for **New** $t_{go_{est}}$. This thesis does not answer the question "how achievable are those potentials?" for most of the attempts. While not an objective of this work, that is a critical aspect to keep in mind when viewing Table 7.1.

Table 7.1: New Attempts' Performance Improvement Given Arbitrary Constraints

	Constraint: P(Kill) = 90%					
SE TA_{min}	51.5 deg					
New Attempt	DP	PRIT	No $X_{cp_{bt}}$ Err.	No RIT Err.	Perf. $t_{go_{est}}$	New $t_{go_{est}}$
TA_{min}	54.84 deg	55.96 deg	57.97 deg	56.08 deg	56.07 deg	51.90 deg
ΔTA_{min}	3.34 deg	4.46 deg	6.47 deg	4.58 deg	4.57 deg	0.40 deg
% ΔTA_{min}	6.48%	8.67%	12.56%	8.89%	8.87%	0.78%
	Constraint: TA_{min} = 55 deg					
SE P(Kill)	75.16%					
New Attempt	DP	PRIT	No $X_{cp_{bt}}$ Err.	No RIT Err.	Perf. $t_{go_{est}}$	New $t_{go_{est}}$
P(Kill)	89.47%	95.4%	98.54%	92.84%	93.20%	76.00%
ΔP(Kill)	14.31%	20.24%	23.38%	17.68%	18.04%	0.84%
% ΔP(Kill)	19.03%	26.93%	31.11%	23.53%	24.00%	1.12%

Additionally, the current implementation of dynamic programming requires a specified rocket ignition time. Even though only two rocket ignition time runs converged in time for this thesis, the envelope they created outperformed the established software envelope throughout the entire metric, except at the unusably low probability of kill configurations. Figure 7.1 summarizes the results and shows the three critical envelopes: the software envelope, the hardware envelope, and the dynamic programming envelope. The dynamic programming envelope is a result of two runs, with RITs of 30 and 33s and outperforming a software envelope generated with over 100 runs that had RITs between 29 and 39.5s and OTS gains between -0.5 and 2.6. Additionally, the pains taken to carefully discretize the state space and verify certain assumptions reduced the average run time to just over 4 days. After discovering the negligible effect of cutting the Monte Carlo scenario size from 25 to 5, the run time was reduced by another factor of 5 to just over 20 hours. This makes dynamic programming much more useful than originally projected by the first time estimate determined in Chapter 6 with Equation 6.12.

The rest of this chapter first details the process that led to Figure 7.1. Then it delves into the results of the other guidance modifications focused on using post-launch information to beneficially change the rocket ignition time, reduce the strongly correlated errors, and decrease the time until impact estimate error.

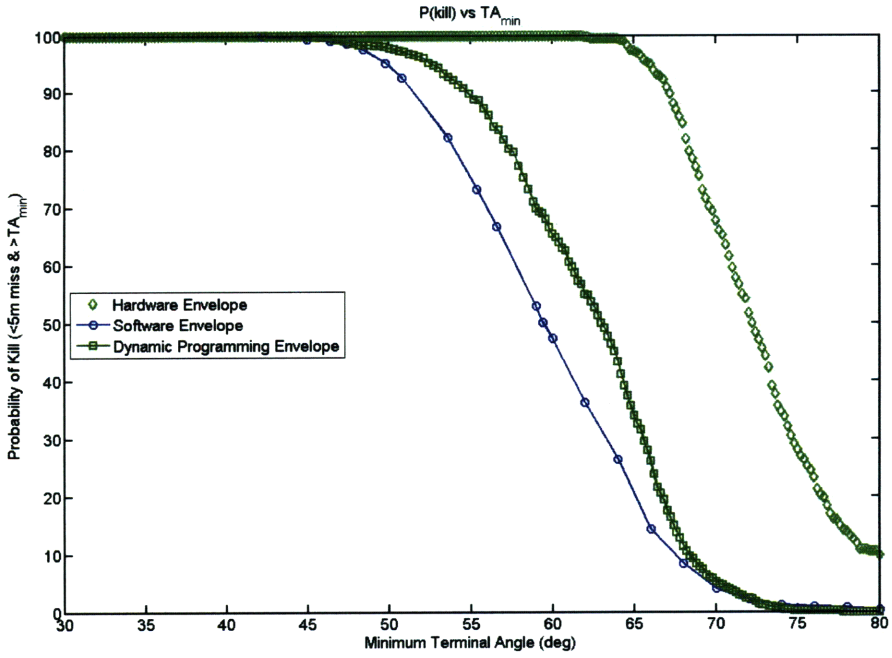


Figure 7.1: Final Performance Comparison

7.1 Dynamic Programming

This is the one modification that attempts to better utilize pre-launch information to improve performance. For the first run, dynamic programming had an OTS gain range from 0.8 to 2.4. It had a state grid with 469 unique state combinations and only used the first 25 Monte Carlo scenarios to generate the optimal control policy. The assumption is that the error dispersions between individual scenarios in the Monte Carlo seed are relatively insignificant when compared to the spacing of the grid points. While the previous figures of the grid point spacing and individual run dispersion, Figures 6.16, 6.17, 6.18, and 6.19, visually support this, the assumption is tested more rigorously in the following runs. Figure 7.2 shows the first results of the dynamic programming derived optimal control policy. Regardless of how many Monte Carlo scenarios it is calculated from, the optimal control policy's performance is always determined with the full seed. At first, the data results demonstrate a performance loss compared to the baseline envelope. The hardware performance limit on the right side of the figure are merely present for reference.

Figure 7.3, a histogram of the OTS gain frequency in the optimal control policy, reveals the problem. The subplot entitled "Normal Gain" shows the frequency of the gains

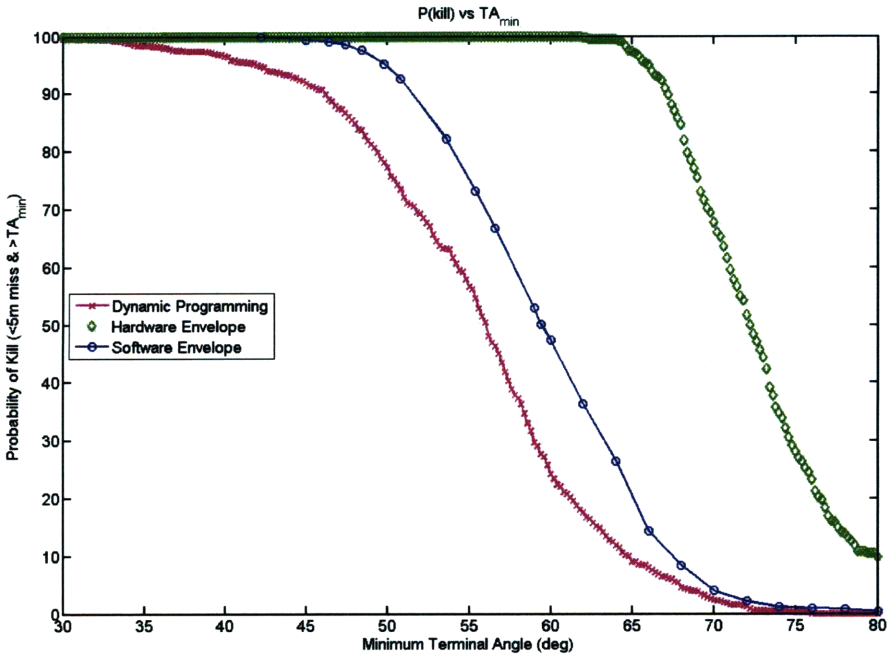


Figure 7.2: Dynamic Programming Performance for First Run

in the optimal control policy that created the performance in Figure 7.2. While both the smallest and largest gains are never chosen, 0.9 is overly represented. For several reasons, dynamic programming determines that for most states the optimal gain is not unique. In fact for this setup, dynamic programming had 417 state combinations with several gains that had the exact same minimum cost. This means that about 88.91% of the total state combinations had redundant minimum costs. As a function of the software setup, the lowest gain with the minimum cost is chosen. This means that if gains 0.9 to 1.1 are projected to have the same cost and that cost is the minimum for that particular state, then 0.9 is saved in the control policy table, not 1 or 1.1.

The other subplots in Figure 7.3 help illustrate this behavior. "Filter Gain" shows a histogram for only the state combinations with unique minimum costs. "Max Gain" and "Mid Gain" are the gain frequencies after reexamining the control policy. "Max Gain" replaces the current gain with the largest gain calculated to have the same optimal cost. "Mid Gain" replaces the current gain with the average of the smallest and largest gain determined to have the same optimal cost. Figure 7.4 illustrates the dramatic change of performance when the "Max Gain" and "Mid Gain" control policies are tested. A simple adjustment to the dynamic programming routine ensures that future runs choose

the maximum guidance gain with the minimum cost. This creates the most performance aggressive trajectory that still minimizes the cost.

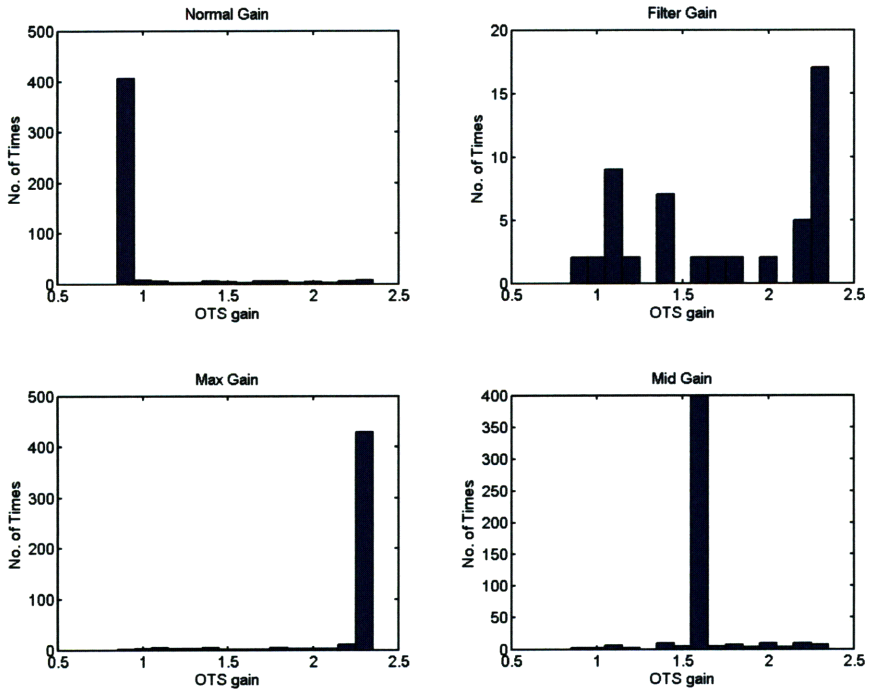


Figure 7.3: Gain Frequency of First Optimal Control Policy

The most obvious reason for these minimum cost redundancies is created by the way guidance is activated in the simulation. Guidance only activates once the pitch has dropped below zero degrees. The first dynamic programming run included the farthest two specified downrange misses in the discretization figures, seen in the grid figures starting on page 79. All state combinations at these specified downrange misses did not have any scenarios with a pitch less than zero throughout their downrange miss window. Even after the state matrix removed all points with these two downrange misses, the third furthest specified downrange miss has some scenarios that do not reduce pitch below zero before the next miss distance. As a result, the guidance system never activates. Logically, the gain has no effect on performance if the guidance and control systems are not turned on. So in the end, dynamic programming thinks all gain options have the same minimum cost for these state combinations.

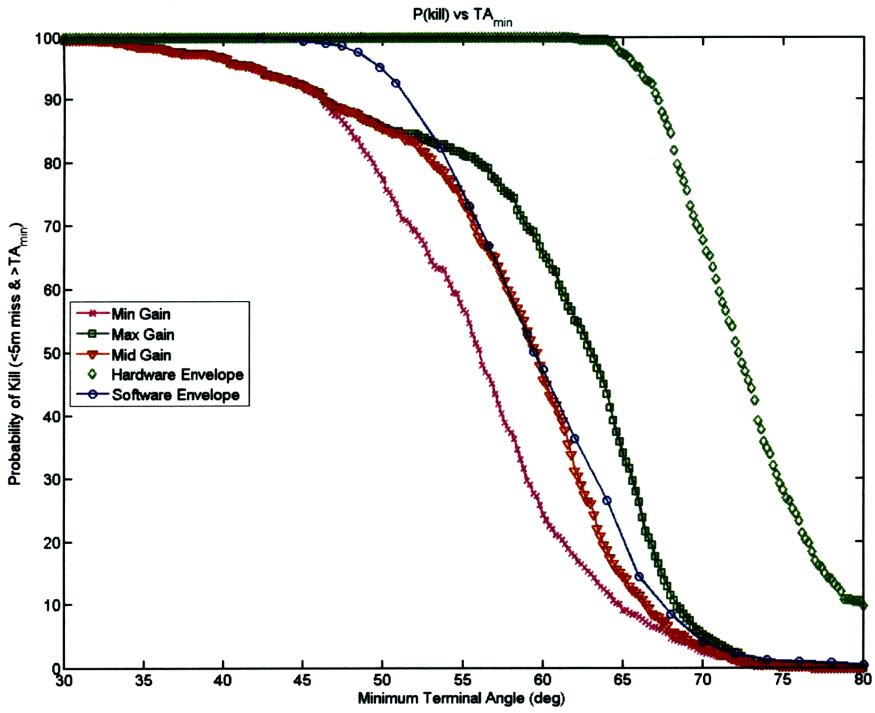


Figure 7.4: Dynamic Programming Performance for First Run with Modified Control Policies

Another cause of redundant minimum costs is related to how the state matrix is formed from all possible state combinations. The histograms paint a fairly detailed picture of the difference between the three control policies. However, those figures portray the frequency of all gains not the frequency of just the utilized gains. State combinations can be so destined to failure that even extreme changes in the gain do not effect the cost. The grid is well defined but not perfect. For example, looking back at the discretization figures, there is a state combination at 100 m downrange miss with a range to target of around 1000 m, a line of sight of 84 °, a velocity magnitude of just under 800 $\frac{\text{m}}{\text{s}}$, and a pitch of -30 °. Figure 7.5 illustrates this particular example. With these initial conditions, the guidance system does not have enough time to positively effect the terminal conditions regardless of the gain. As a result, dynamic programming determines that all gain values have equal costs. This example illustrates one reason why so many redundant optimal costs exist in the optimal control policy.

Figure 7.6 illustrates the gain used over time for the first and last five Monte Carlo scenarios for the "Min", "Max", and "Mid" control policies. This figure indirectly shows

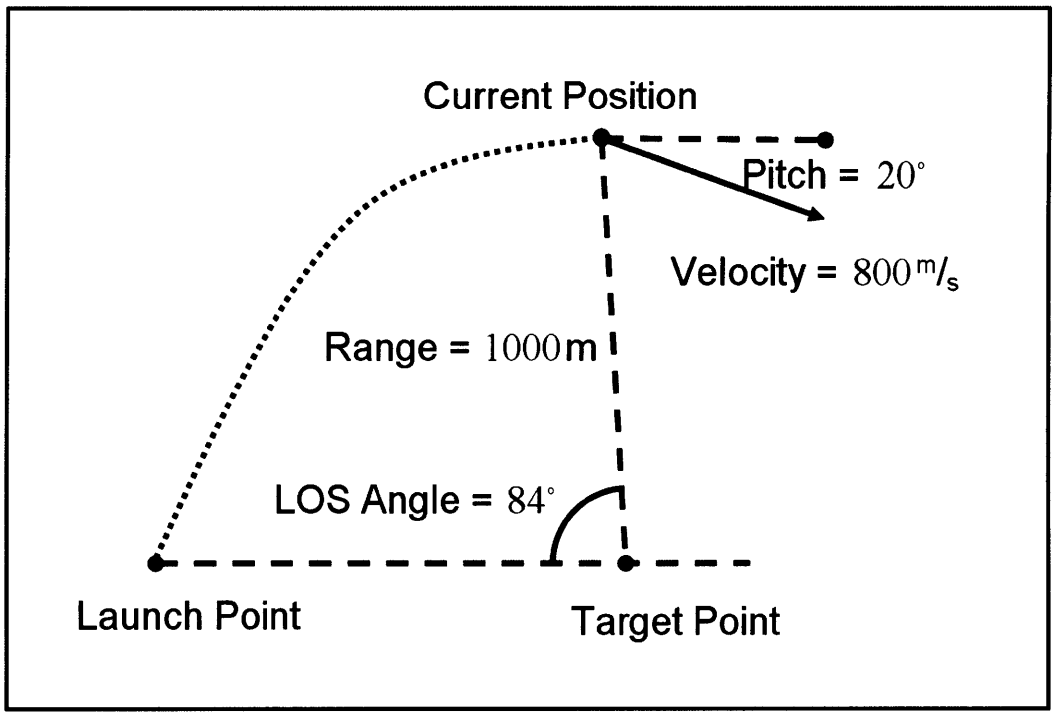


Figure 7.5: Example of Doomed State Combination

the frequency of just the utilized gains within the control policy, insight indeterminable from the histograms. It is also easy to see where the three control policies differ as a function of the simulation time.

In addition to the two issues just addressed, the relative size between specified downrange misses, gain options, and state combinations plays a critical role in either aggravating or alleviating the issue of redundant minimum cost control gains. Adjustments to any three of these discretizations that improves the chances of generating different costs from different initial conditions, increases the percent of unique gains in the optimal control policy. Figure 7.7 shows a histogram of the control policy for a dynamic programming run with only one specified downrange miss, 6000 m short of the target. As the figure shows, the larger spacing between specified downrange misses results in fewer redundant optimal gains. This setup generates a control policy that has 81 redundant minimum costs. This is out of the 256 state combinations. The percent of redundancy, 31.64%, is much lower than that of the first setup, 88.91%. This run fails to demonstrate which issue is more influential on the percent of redundancy, state combinations before guidance activation or specified downrange miss distance. Relative to the first one, this run both eliminated all state combinations before guidance activation and increased the

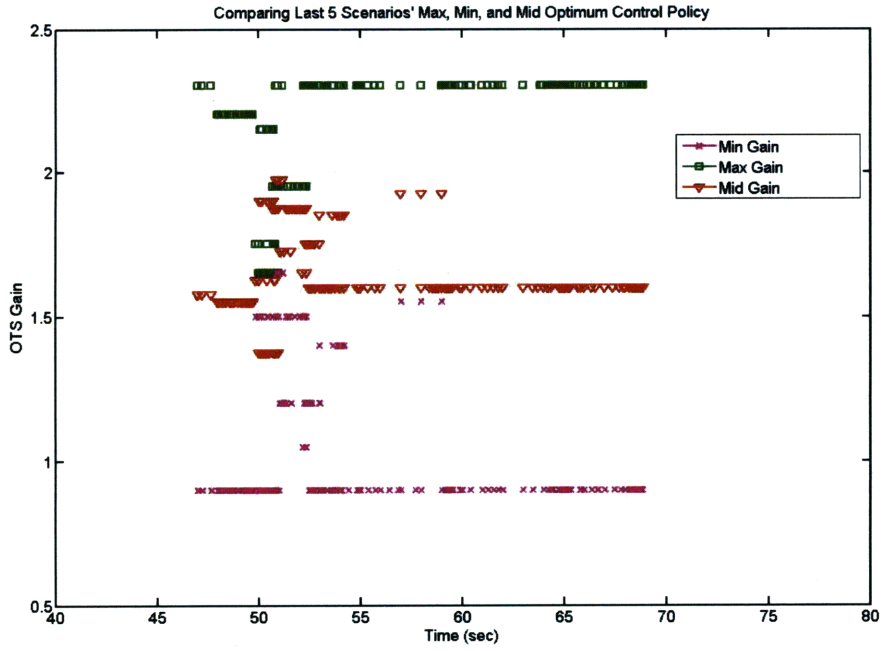


Figure 7.6: OTS Gain versus Time for "Max", "Min", and "Mid" Control Policies

specified downrange miss spacing.

Besides the issue of multiple gains with the same minimum cost for a large percentage of the total state combinations, the first dynamic programming run suffers from the so far inadequately defended assumption that the control policy can be generated from a greatly reduced number of the Monte Carlo scenarios. Figure 7.8 shows the differences between the control policies formed with 5 and 25 Monte Carlo runs. There is clearly a change in the gains chosen over time. However, Figure 7.8 fails to show whether those differences are exhibited by the majority of Monte Carlo scenarios or just a few, and if they result in significantly altered performance. Figure 7.9 shows the performance comparison of those two control policies. The figure demonstrates that performance is not significantly effected by the reducing the Monte Carlo scenarios from 25 to 5. However, the 5 fold decrease in computational demand has other costs to consider. Reducing the Monte Carlo scenarios does change the optimal cost for each state. While most of the time, this change in cost does not effect the optimal gain, sometimes it does. Additionally, the percentage of redundancy increases from 88.91% to 96.59%. Ultimately, however, these changes do not create a control policy with drastically different performance.

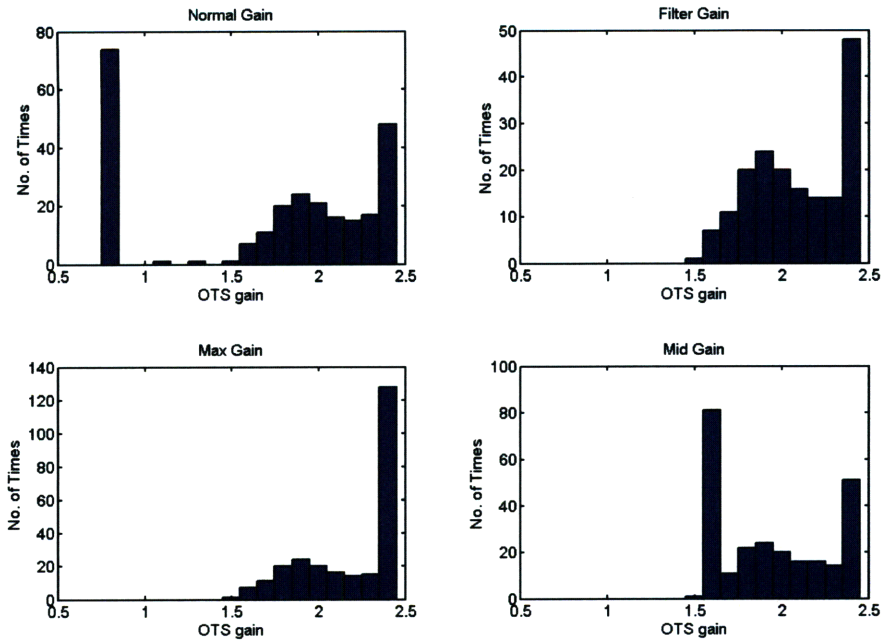


Figure 7.7: Gain Frequency of Optimal Control Policy with One Specified Downrange Miss

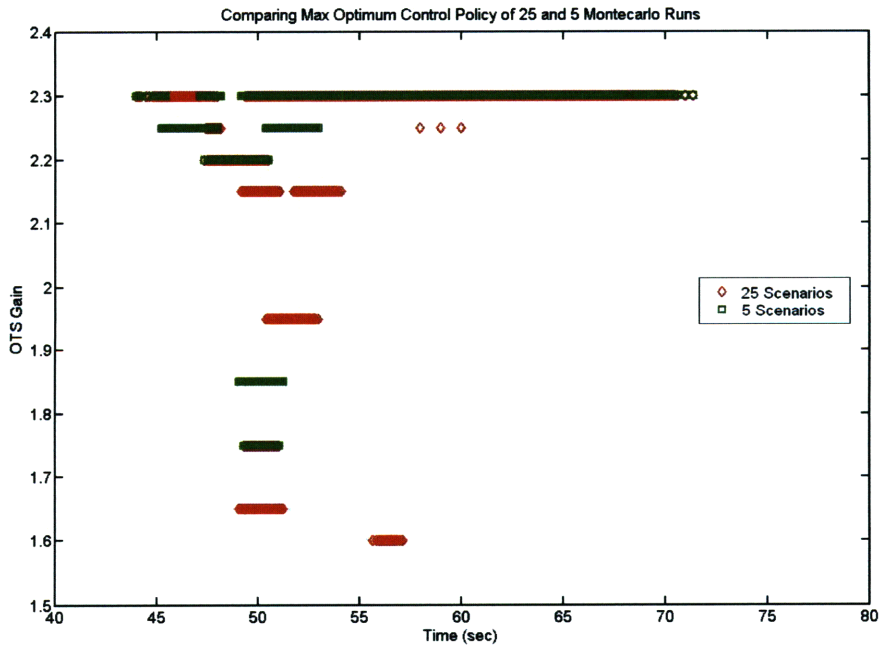


Figure 7.8: OTS Gain versus Time for "Max" Control Policies for 5 and 25 Monte Carlo Runs

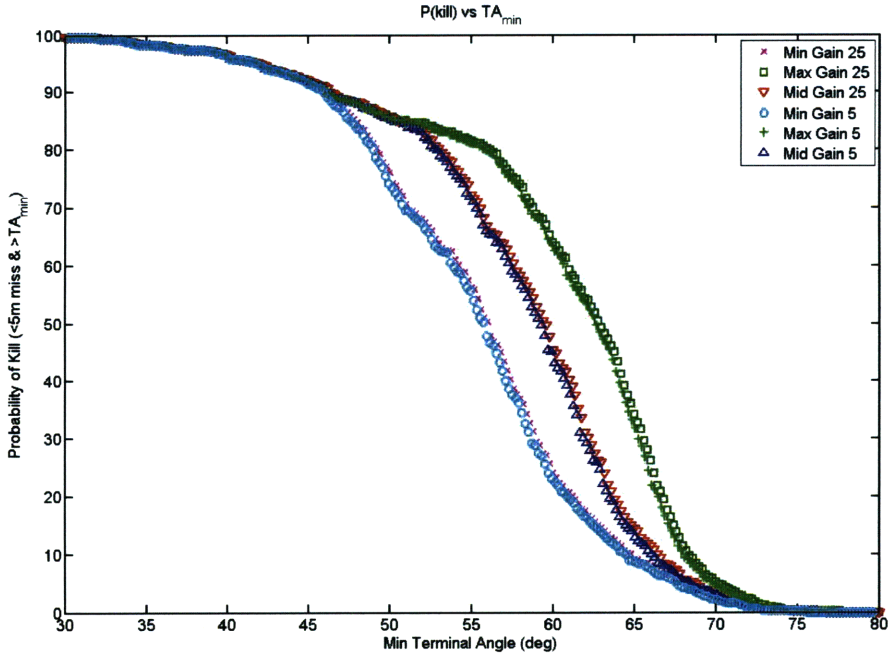


Figure 7.9: Performance Difference Between Optimal Control Policy Generated from 5 and 25 Monte Carlo Scenarios

Having addressed some of the significant considerations, the performance in Figure 7.4 is even better that is it originally looks. In the figure, dynamic programming's performance is compared to the optimal envelope from Chapter 5. As a quick reminder, that envelope represents the best performance for over 100 runs where both the gain in between runs and rocket start time were adjustable variables. When attempting to maximize performance, dynamic programming had the ability to change the gain as a function of the state but could not change the rocket ignition time. As such, dynamic programming's performance should only be compared with the performance of those runs in the baseline that had the same rocket ignition time of $30\,s$, shown in Figure 7.10. In Figure 7.4, dynamic programming outperformed the old envelope for only a part of the metric. In Figure 7.10, dynamic programming surpasses the old data runs throughout the entire metric. This makes sense as the a guidance system that intelligently varies the gain over time should outperform a constant gain guidance.

While the adjusted dynamic programming demonstrated great improvements in performance when compared to the constant gain runs, the software envelope in the higher probability of kill region remained unbeaten. Table 5.1 on page 57 confirms that the

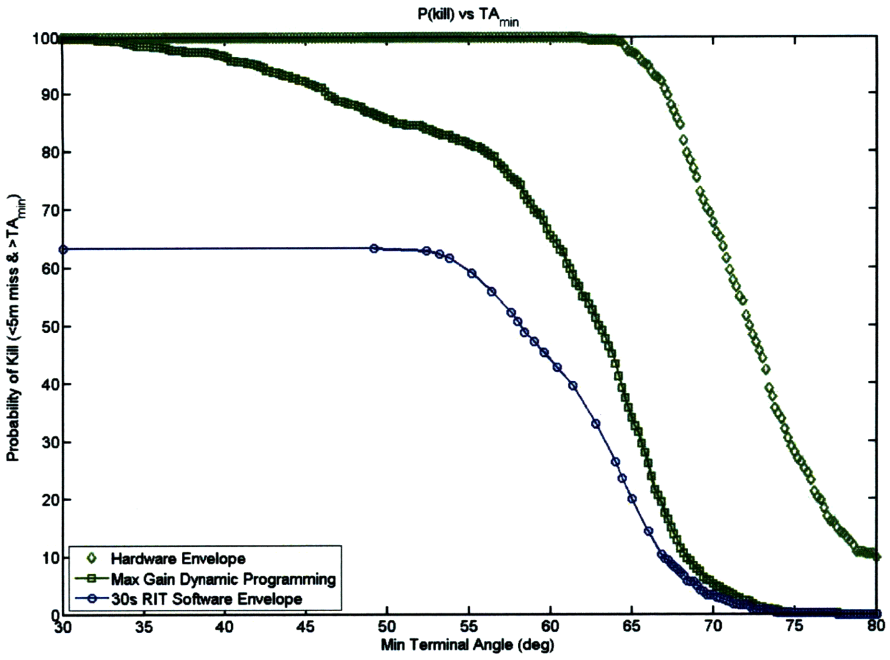


Figure 7.10: Dynamic Programming Performance Compared to Constant Gain Performance with Same Rocket Ignition Time of 30s

higher probability of kill region of the envelope is formed by later rocket ignition times. Figure 7.11 shows the results of dynamic programming when run with a rocket ignition time of 33 s instead of 30 s. For the purposes of comparison, the simulation run used the same state combination matrix, gain range, and number of Monte Carlo scenarios. As expected, dynamic programming produces a control policy that in a single run with a varying gain outperforms a wide range of constant gain runs with the same rocket ignition time.

As mentioned earlier, Figure 7.1 shows the combined performance of the envelopes in Figures 7.10 and 7.11.

7.2 Predictive Rocket Ignition Time Analysis

Predictive Rocket Ignition Time aims to use post-launch information to beneficially change the rocket ignition time. However, as shown earlier, altering the rocket ignition time is a powerful tool for changing the performance of the munition. However, because

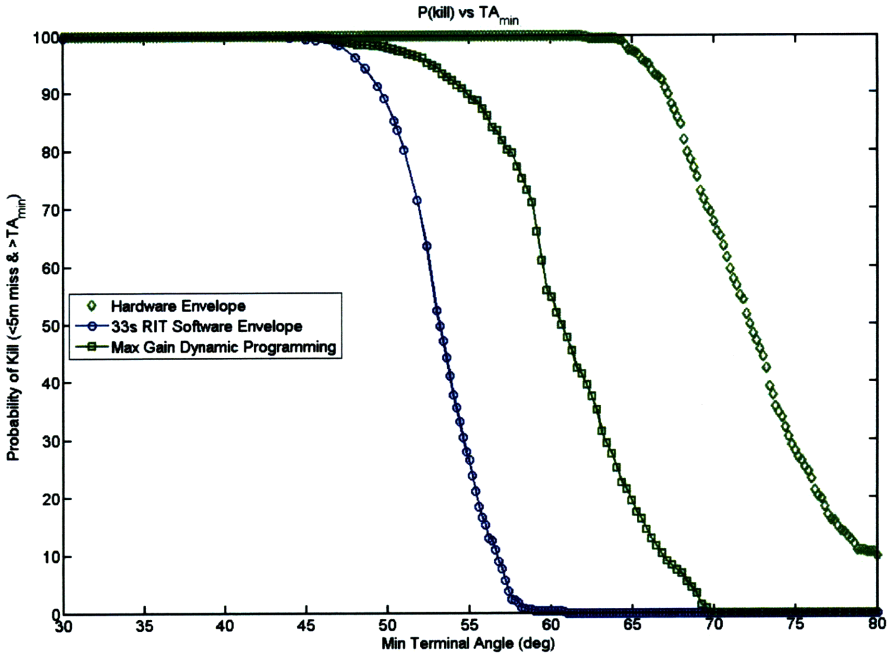


Figure 7.11: Dynamic Programming Performance Compared to Constant Gain Performance with Same Rocket Ignition Time of 33s

it is so powerful and errors exist, any modifications must be both intelligent and include a factor of safety. If made haphazardly or too aggressively, changes to the rocket ignition time can more easily doom a single firing to failure than improve its performance.

For those reasons, investigation begins with a large safety factor, initializing all rocket start times at the latest ideal ignition time found while hiding the rocket ignition errors, 32.31s. While this does not maximize performance, initializing at this value gives all the scenarios in the seed an opportunity to succeed, improving performance in the sense that no scenario is fated to fail. For comparison, starting at the earliest ignition time would cause many scenarios to land well beyond the target, obviously not desired. Figure 7.12 shows the results from this test run. As expected, the overall performance does not globally supersede the already established performance envelope but it is a safe starting place. Interestingly, there are points that do break the envelope.

The next step attempts to predict a rocket ignition time that more aggressively balances between maintaining that established factor of safety and improving performance. Using post-launch information, the munition must differentiate between trajectory cases

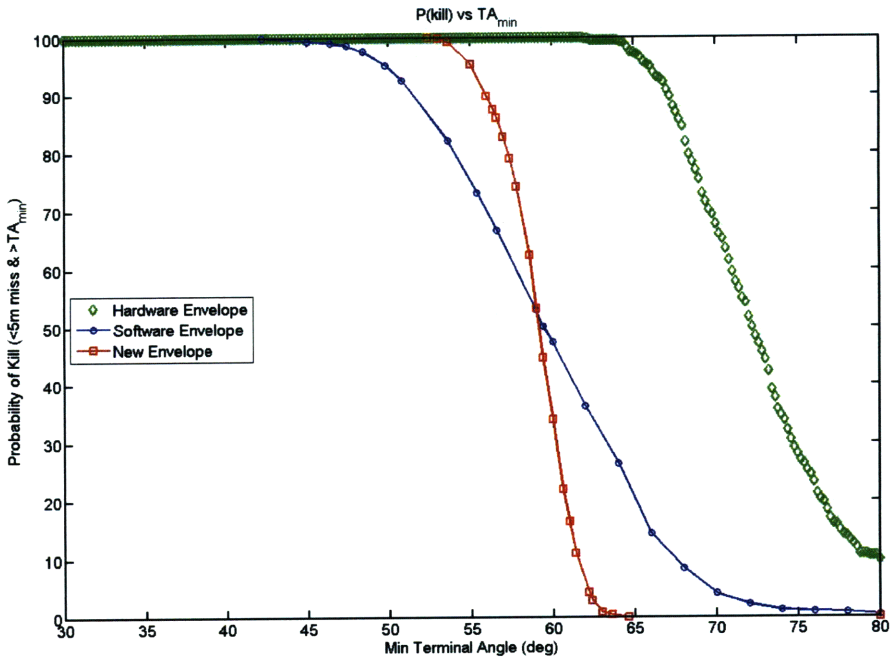


Figure 7.12: Predictive Rocket Ignition Time Results

allowing an earlier ignition time and those needing the entire safety factor. The major obstacle is the timeline. For the short range scenario, the earliest rocket ignition time is about 28.25 sec, meaning any decision to adjust the time of ignition must be made a second or two before that. For a real system, the GPS receiver on the munition would not start with an acquired signal. It is designed around the original need, to acquire prior to the guidance system turning on. Normally, this would not happen until the munition is near apogee, around 48 seconds for the short range scenario. Prior to calibration with the GPS, the INS is usable but not as accurate. This timeline becomes worse for the other scenarios, medium and long range, as the average rocket ignition time is earlier and apogee later. As a result, if a solution is found to improve performance it would only work for the short range scenario and if the assumed post-launch information is actually available at the necessary accuracy and time.

With these assumptions in mind, Table 7.2 shows the correlation factors between the ideal rocket ignition time found and actual state information at the time shown. Energy, ξ, is a specific energy with mass divided out. It is a summation of the munition's potential and kinetic energy at a particular time.

$$\xi = gh + \frac{v_x^2 + v_z^2}{2} \tag{7.1}$$

The table illustrates that the ideal rocket ignition time is difficult to determine from available state information in the short time between launch and rocket ignition. It does not show a strong propensity towards any one of the determined states, and the correlation coefficients do not significantly grow even as time progresses. Additionally, this is generated with the actual state information. Attempting to intelligently determine a more aggressive rocket ignition time with the actually available, imperfect state information would be even more difficult and dangerous.

Table 7.2: Predictive Rocket Ignition Time Correlation Coefficients

Time (sec)	Downrange	Altitude	$Velocity_x$	$Velocity_z$	Energy
5	-0.469	0.466	0.538	-0.541	0.536
6	-0.489	0.485	0.548	-0.551	0.546
7	-0.505	0.501	0.554	-0.557	0.552
8	-0.518	0.513	0.559	-0.561	0.557
9	-0.528	0.523	0.563	-0.564	0.561
10	-0.536	0.531	0.565	-0.566	0.563
11	-0.542	0.537	0.567	-0.567	0.565
12	-0.547	0.543	0.568	-0.569	0.567
13	-0.552	0.547	0.570	-0.570	0.569
14	-0.555	0.551	0.571	-0.571	0.570
15	-0.559	0.554	0.573	-0.573	0.573
16	-0.561	0.557	0.575	-0.575	0.574
17	-0.564	0.559	0.577	-0.577	0.576
18	-0.566	0.561	0.578	-0.578	0.577
19	-0.568	0.563	0.578	-0.578	0.578
20	-0.569	0.565	0.579	-0.578	0.578
21	-0.571	0.566	0.579	-0.578	0.578
22	-0.572	0.567	0.579	-0.578	0.579
23	-0.573	0.568	0.579	-0.577	0.579
24	-0.574	0.569	0.579	-0.577	0.579
25	-0.575	0.570	0.579	-0.576	0.579
26	-0.576	0.571	0.579	-0.576	0.579
27	-0.576	0.572	0.579	-0.575	0.579
28	-0.577	0.573	0.579	-0.575	0.579
29	-0.576	0.573	0.510	-0.298	0.508

7.3 Incorporating Post-Launch Information for Strongly Correlated Errors

This modification attempts to utilize post-launch information to reduce the errors most detrimental to performance. To show the value of such a modification, those errors are eliminated, revealing the potential performance improvement. Table 6.2 on page 67 indicates the two errors most strongly correlated with the performance factors are $X_{cp_{bt}}$ and RIT error. The data indicates that more accurately quantifying the former error should significantly improve both the downrange miss and terminal angle, while increased knowledge of the latter should improve only the terminal angle.

To assess the value of reducing these errors, first they are simply eliminated. Simulation performance without these errors compared to runs with these errors can show the value of such a change. Running the configurations that generated the software envelope, Figure 7.13 shows the performance change when the $X_{cp_{bt}}$ error is eliminated. Figure 7.14 illustrates the same when the RIT error is set to zero.

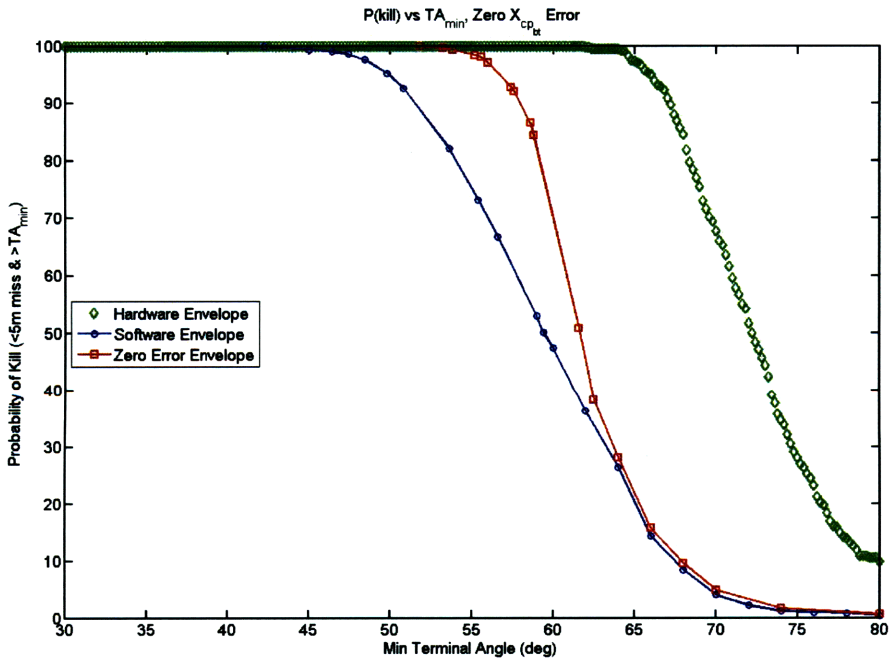


Figure 7.13: Performance with No $X_{cp_{bt}}$ Error

Although both figures show improvement above the baseline. Figure 7.13 shows that

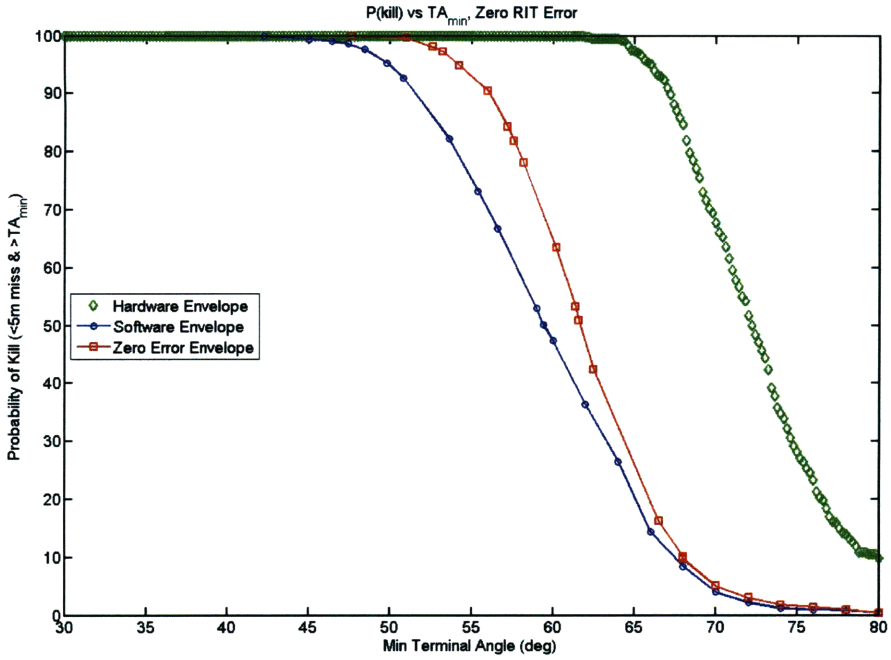


Figure 7.14: Performance with No Rocket Ignition Time Error

eliminating the error with the higher correlation factor did have a greater positive effect on performance.

7.4 Reducing $t_{go_{est}}$ Error

The final modification attempts to utilize post-launch information to reduce the time until impact estimate error. Before determining an implementable method, simulation runs can determine the greatest performance gain possible from reducing the time until impact estimation error. Using the method described in Chapter 6, Figures 7.15, 7.16, and 7.17 illustrate the change in the aiming point height versus time until impact. The guidance gain and rocket ignition time combinations are the same as in Figures 6.5, 6.6, and 6.7, but the particular cases displayed are not. The simple methodology for this test failed to completely eliminate the error for all the cases. For example, Case 446 in Figure 7.16 shows a noticeable difference between the truth and estimated aiming point height at a given $t_{go_{est}}$. In addition, there is a jump in the data as the guidance system switches from the truth data to the original guidance approximation, every case in Figures 7.15, 7.16, and 7.17 illustrate this gap as $t_{go_{est}}$ passes through -10 s. However, in the end, the

reduction in error is significant enough to demonstrate if this modification can create a substantial gain in performance.

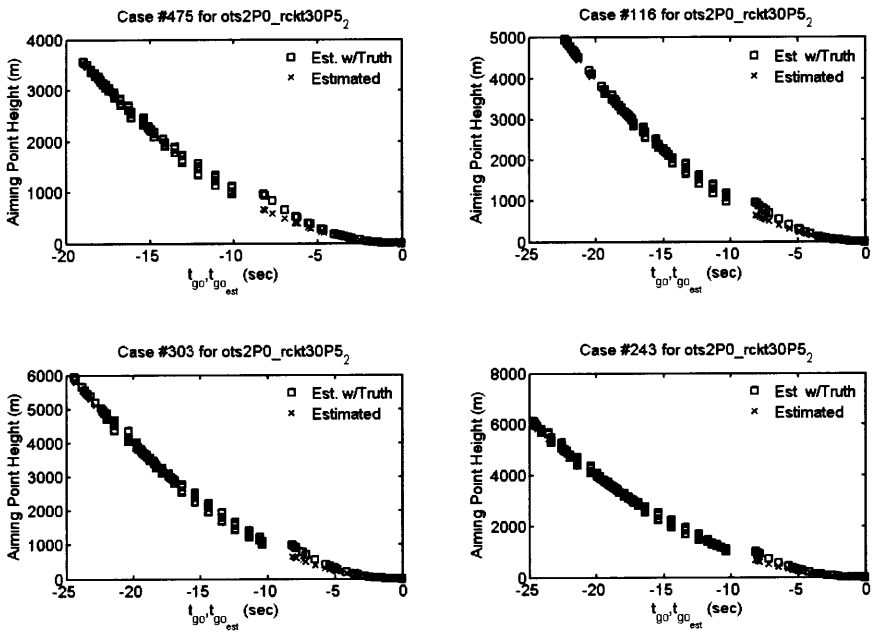


Figure 7.15: Corrected Aiming Height vs $t_{go}, t_{go_{est}}$ for OTS gain 2.0 and Rocket Ign of 30.5 sec

Running the same configurations in the software envelope, Figure 7.18 shows the performance for those runs where the time until impact error is almost entirely eliminated. The diamond marked line on the far right is the same hardware performance limit, displayed merely for a common relative measure to other figures. The six data sets create two envelopes. The three with the original time until impact estimate are displayed as "With Old t_{go} Estimate", while the other three with the near perfect time until impact estimate create the other envelope. Note, the "With Old t_{go} Estimate" is not the same as the software envelope. It is just the performance envelope formed by the three configurations tested. As the graph shows, nearly eliminating the $t_{go_{est}}$ error has a significant effect on performance. However, this method of reducing the impact error is obviously not implementable in a real guidance system.

For this to be a viable improvement, a change in the guidance system must incorporate post-launch information to maximize reductions in the impact error for the greatest number of scenarios and to detrimentally effect performance as infrequently as possible.

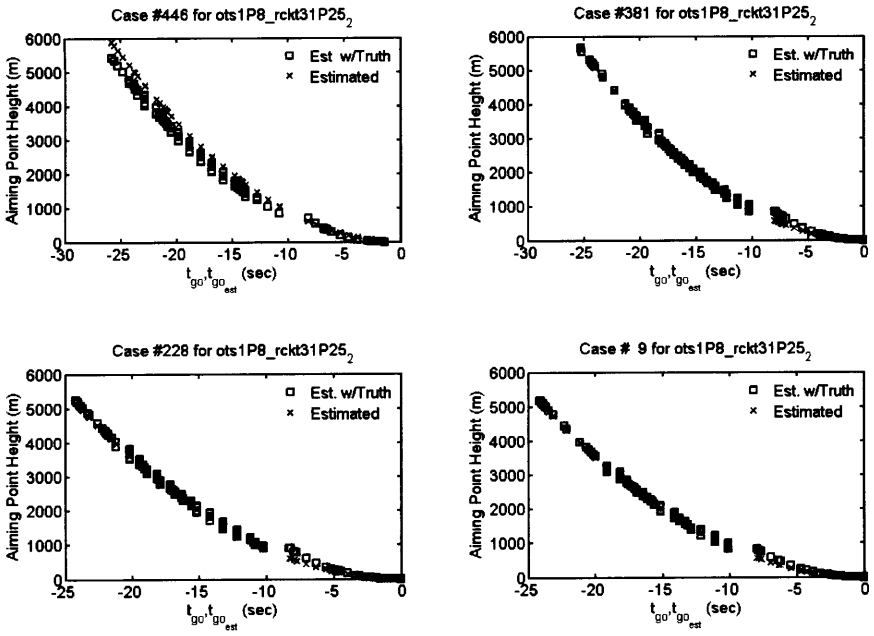


Figure 7.16: Corrected Aiming Height vs $t_{go}, t_{go_{est}}$ for OTS gain 1.8 and Rocket Ign of 31.25 sec

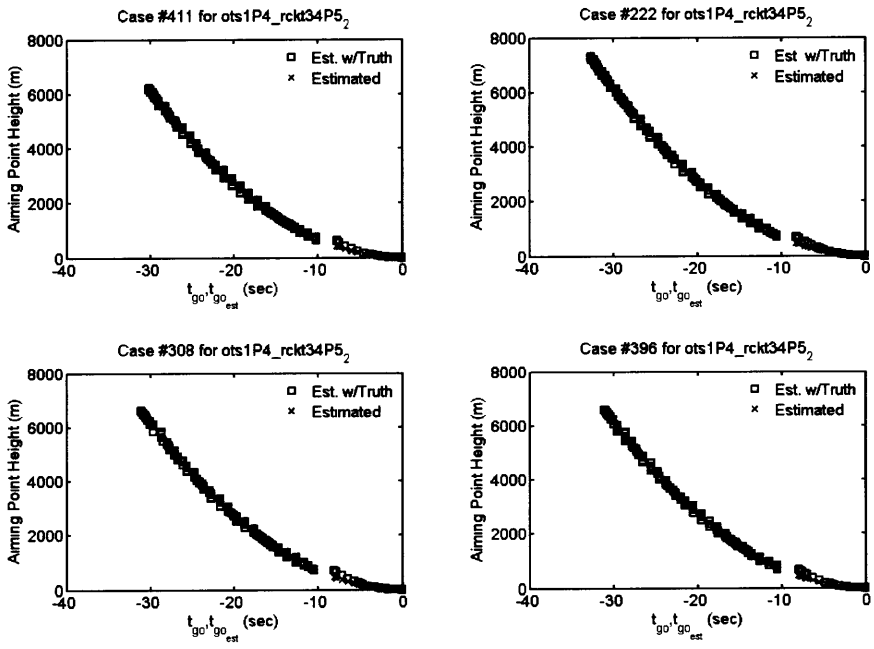


Figure 7.17: Corrected Aiming Height vs $t_{go}, t_{go_{est}}$ for OTS gain 1.4 and Rocket Ign of 34.5 sec

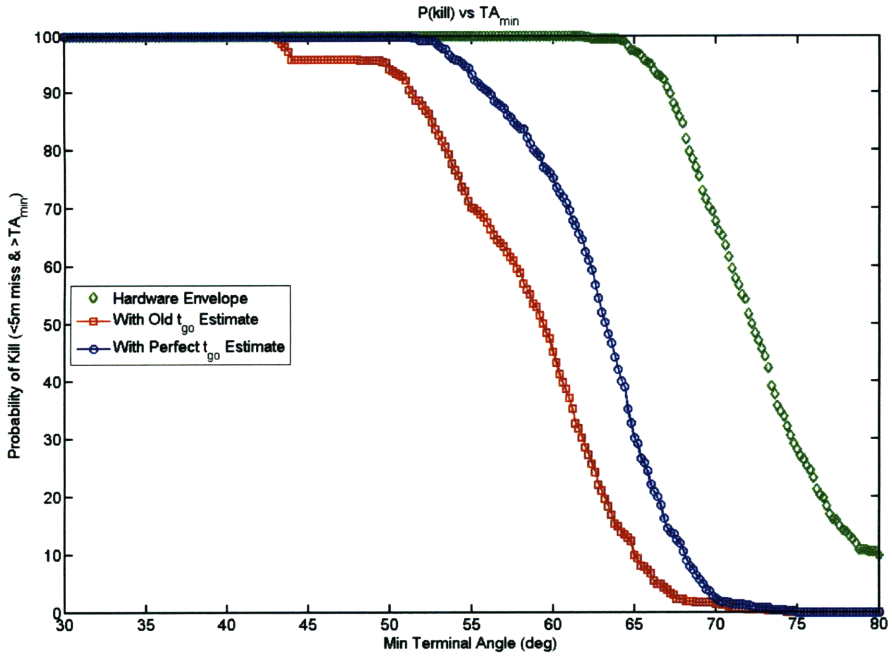


Figure 7.18: Change in Performance with Negligible $t_{go_{est}}$ Error

As stated back in Chapter 6, the data indicates a correlation between the amount and direction of impact error and the rocket ignition time. The data also demonstrates that the estimate time until impact converges to the truth so any added term to the estimate would have to decay as the munition approaches the target. The first attempt calculates a time to impact correction term until the estimate is less than 10 s. As defined, the error is positive when the estimated time until impact is greater than the actual. So once calculated, the correction term is subtracted from the current time until impact estimate. Figure 7.19 shows the mean error between the actual and estimated impact time for the 500 Monte Carlo scenarios, where:

$$t_{go_\epsilon} = t_{go_{est}} - t_{go_{act}} \tag{7.2}$$

The attempt fits a third-order polynomial regression to each of the error curves in Figure 7.19. The rocket ignition time for each case is pulled out and the resulting coefficients averaged for all three runs. This is merely one of several possible adjustments to the estimate time until impact calculation, chosen for ease of implementation not expected performance improvement [45]. As the simulation runs, the guidance system

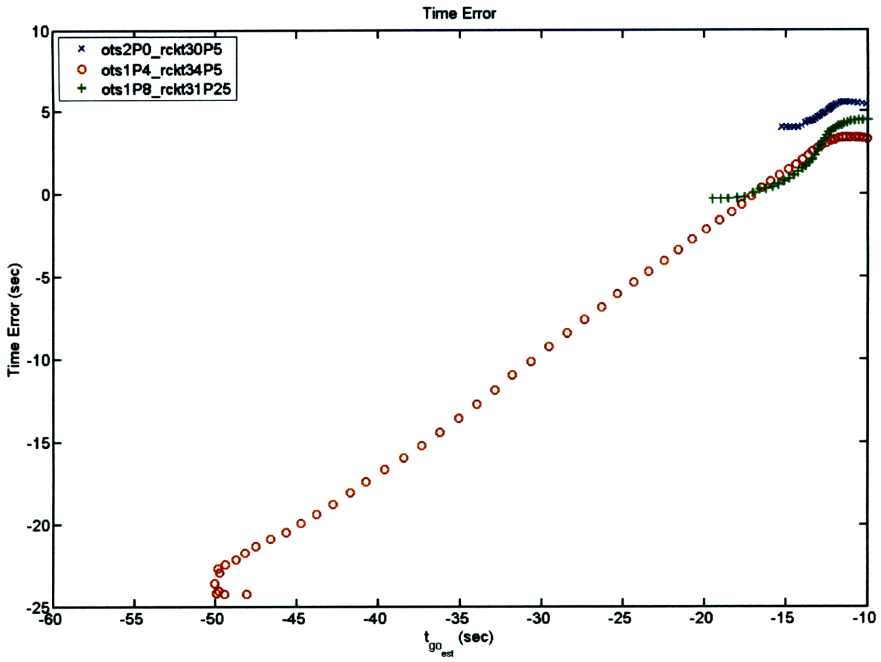


Figure 7.19: Time Until Impact Error vs Time Until Impact Estimate Error

takes in the INS detected RIT. This is the actual time of rocket ignition that varies from the specified time by a seed determined error for every scenario. The guidance then uses the RIT to determine the coefficients for the polynomial. Combined with the current time until impact estimate, these coefficients determine the time until impact correction. Figure 7.20 shows the correction factor for several rocket ignition times. With this adjustment in place, Figure 7.21 shows the same three configurations alongside their previous performance.

The results are mixed. For the run with an OTS gain of 1.4 and a RIT of 34.5 s, the correction term improves the performance along the entire metric except for the top 5% probability of kill. The new performance even outperforms the run with nearly zero time to impact error, possibly a coincidental benefit of too much correction in this guidance, rocket setup. However, this run is not defining any point of the performance envelope. Improving its performance is of no value.

The middle run, an OTS gain of 1.8 and RIT of 31.25 s, demonstrates a reduction in performance along most of the metric. The only area of increased performance occurs in the low minimum terminal angle, high probability of kill area. These results extend

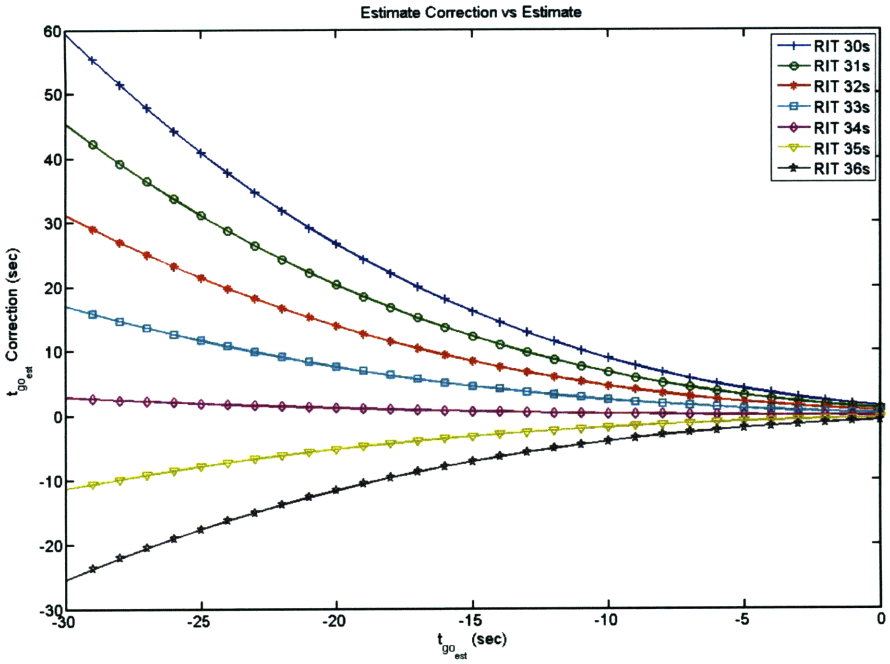


Figure 7.20: $t_{go_{est}}$ Correction Factor for Various Rocket Ignition Times

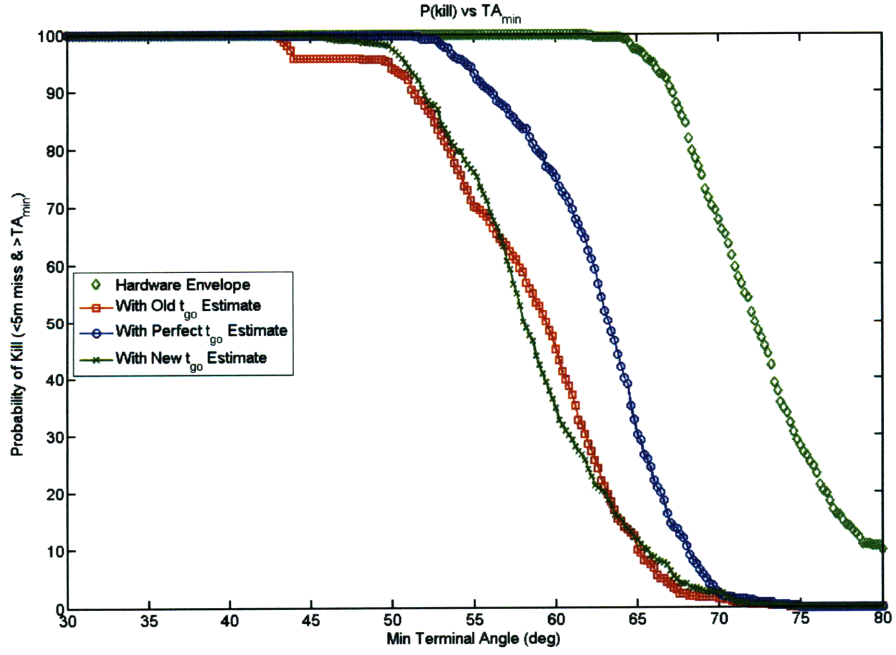


Figure 7.21: Probability of Kill vs. Min Terminal Angle for $t_{go_{est}}$ Correction Error

the top 1% probability of kill performance envelope a few more degrees. This is a small improvement but a positive change to the envelope.

The final run, 2.0 OTS gain and 30.5 s RIT, shows the most significant benefits from the time to impact correction term. Performance is significantly improved in the high probability of kill, lower minimum terminal angle region. However, below around 65% probability of kill, the time until impact correction negatively affects the run's performance.

Chapter 8

Conclusions

Overall, Dynamic programming demonstrated the most realizable, performance improvement for the entire metric. Even with only two runs, its envelope outperformed the optimal performance of the industry standard guidance algorithms except at the lowest probability of kill configurations.

Upon investigation, the simulation found the rocket ignition time (RIT) to be an extremely sensitive factor. While properly adjusting it greatly increased performance, the more likely improper adjustment doomed a firing to failure from the beginning. In the current architecture, as established in this thesis, intelligent changes to the RIT are very difficult for even the most ideal short range scenarios.

Simulation runs demonstrated that reducing the time to impact error significantly improved performance, but the method for reducing that error investigated in this thesis was cumbersome and ineffective. Results indicated that the utilized methodology improved with more simulation runs.

The simulation demonstrated significant performance improvement by eliminating certain key errors with strong correlation factors to the performance parameters.

The control policy generated with a limited number of Monte Carlo scenarios performed exceptionally well when tested on the entire seed. Varying the number of Monte Carlo scenarios significantly affected the percentage of unique minimum cost, state dependent gains but not the performance. While demanding a large amount of time to generate the control table, dynamic programming only does this once. After determining the control policy, dynamic programming becomes a very low computationally demanding

guidance addition, requiring only a table lookup. To save more CPU time, the frequency of the table lookup can be set much lower than that of the rest of the guidance system because of the speed of this munition's dynamics relative to the software. Additionally, the current version of dynamic programming is merely at a utility level, and, with refinement, reducing the computational time by several orders of magnitude is easily possible. These savings could reduce the current run time of between 20 hours and 4 days, depending on the number of Monte Carlo scenarios, or increase the resolution of the state gridding or guidance gain options. Finally, dynamic programming offers a solution for the entire munition's range of operation. While this work focused on the short range scenario, dynamic programming could generate a control policy for the entire state space defined from the short to the long range scenario.

Based on the gathered data, the best method of improving performance for this system is a conglomeration of the tested methods. Ideally, dynamic programming is run, coded either independently of RIT or run with several RITs. If coded as another state variable, the post-launch processing produces a control policy where guidance looks up the INS detected RIT along with the state information to determine the guidance gain. If several runs with different hard coded RITs are stored, the system can linearly interpolate between the control policies of nearest state combination and rocket ignition time. Additionally, the other post-launch methods could use the state and RIT information dynamic programming requires to further increase performance.

In the end, this work achieved, to different extents, both the theoretical and practical objectives. Different attempts demonstrated varying amounts of potential performance improvement. However, the unquestionable result is that better utilizing pre- and/or post-launch data improved the munition's performance. By pushing the software envelope closer to the hardware envelope, software modifications help the munition perform closer to its potential. For the user, these changes result in practical benefits. After implementation, the munition is able to simultaneously hit the target with greater accuracy and a higher terminal angle, reducing cost and increasing the munition's mission envelope. To compare, for a required minimum terminal angle of 55 deg, the user's probability of kill improves by 14.31%, a 19.03% increase, when using dynamic programming. Conversely, for a required probability of kill of 90%, dynamic programming improves the achievable minimum terminal angle by 3.34 deg, a 6.48% increase. While the particular performance improvement for a single run differs according to that run's constraints, in the end, the user has a better, more capable munition for all missions.

Chapter 9

Future Work

Restructuring the munition's architecture to allow earlier GPS acquisition would significantly help any possibility of properly adjusting the RIT. With enough reliable state information available prior to ignition, the munition would have a reasonable chance to intelligently change its RIT and greatly improve performance.

Despite concerns expressed for the current method of correcting the $t_{go_{est}}$ error, the possible performance improvement, see Figure 7.18, and availability of post-launch information strongly suggest the benefits and possibility of generating a better way to reduce the error. More work is required to realize this great potential.

This thesis leaves the task of finding a practical method to reduce the errors strongly correlated to performance and realize the potential demonstrated to another, see Figures 7.13 and 7.14. Even if practical solutions do not eliminate errors entirely, it is completely within the current architecture and work load of the system to process post-launch information to reduce the most damaging errors and significantly improve performance.

With more time, post-launch information could be used for multiple means not only to reduce keys errors but estimate important, unaccounted forces-such as drag or lift correction.

The work in dynamic programming has created a large number of questions. The difference between 5 and 25 scenarios was examined, however the conclusions drawn should be extended to the limits on either side. A careful examination of why the performance of the control policy does not change, yet parts of the control policy do change would answer many questions. Also dealing with the number of Monte Carlo scenarios, the

cost for the terminal states is determined from the 50% median of all the Monte Carlo scenarios run. The consequences of using the median with such a small sample size versus other statistics, such as mean, and the ability of the median to portray the most critical attributes of the data remains inadequately examined. The method of discretization needs reworked to create a state matrix with fewer doomed state combinations.

A great deal of work is necessary for the described amalgamation to work. Implementable solutions must be designed and tested to intelligently change the RIT, reduce the time to impact error, and decrease the most significant errors. Before using it, dynamic programming needs to be computationally streamlined and translated to a faster language. It needs to be modified to either add an addition state variable or allow guidance to interpolate between control policies generated with different RITs. Then, these attempts need to be tested simultaneously. It may turn out that some of the improvements are not concurrently implementable. However, based on the currently collected data, the work seems worthwhile as it would drastically improve performance.

Appendix A: Abbreviations

Table 9.1: Abbreviation List

A	Aerodynamic Frame
ATK	Alliant Techsystems
BTERM	Ballistic Trajectory Extended Range Munition
CEP	Circular Error Probable
CPU	Central Processing Unit
COTS	Commercial-Off-the-Shelf
deg	Degrees
DOF	Degree of Freedom
DP	Dynamic Programming
E	Earth-Centered, Earth-Fixed
ERGM	Extended Range Guided Munition
ft	Feet
g	Factors of Average Earth Gravity
in	Inches
INS	Inertial Navigation System
K	Kelvin
kg	Kilograms
lb	Pounds
LCEF	Launch-Centered, Earth-Fixed
LOS	Line-Of-Sight
m	Meters
NED	North-East-Down
OTS	Offset-Target-Scale
PGG	Predictive Geometric Guidance
PRIT	Predictive Rocket Ignition Time
ProNav	Proportional Navigation
ProNav GB	Proportional Navigation with Gravity Bias
rad	Radians
RIT	Rocket Ignition Time
s	Seconds
SE	Software Envelope
$t_{go_{est}}$	TimetoGo Estimate and Time Until Impact
TLAM	Tomahawk Land Attack Missile
TOF	time-of-flight
yr	Years

[This Page Intentionally Left Blank]

Appendix B: Coordinate Frames

Table 9.2: Earth-Centered, Earth-Fixed (E) Coordinate Frame

Origin	Center of Earth
Fundamental Plane	Earth's mean equator
1^{st} **Axis & Definition**	I—Direction of Equator/Prime Meridian intersection
2^{nd} **Axis & Definition**	K—Direction of mean rotational axis (North is +)
3^{rd} **Axis & Definition**	J—Completes right-hand system

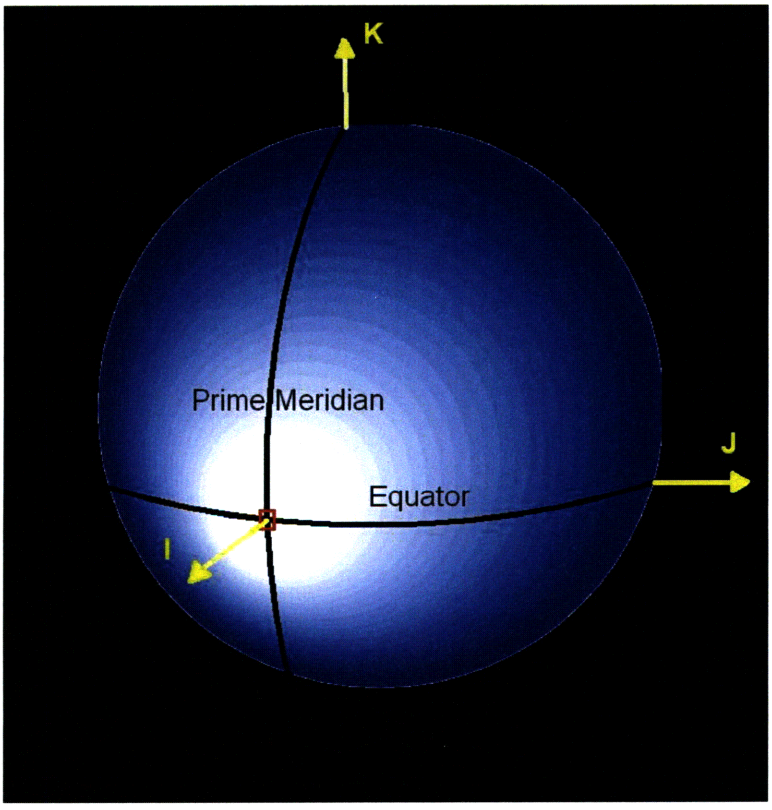


Figure 9.1: Earth-Centered, Earth-Fixed (E) Coordinate Frame

This section details the coordinate systems used in thesis ad nauseam for the sake

of avoiding confusion. Table 9.2 describes and Figure 9.1 illustrates the Earth-Centered, Earth-Fixed frame. Occasionally, this work references LCEF, Launch-Centered, Earth-Fixed, or TCEF, Target-Centered, Earth-Fixed. These frames are exactly the same as Earth-Centered, Earth-Fixed except they are centered at the launch point or target, respectively. In this thesis, the Earth is approximated as a sphere and so the conversion magnitude is always one earth radius, see **Nomenclature** for details, distributed among the axes as a function of launch point latitude and longitude.

Looking at the intersection of the Equator and the Prime Meridian, there is a red box that shows the area Figure 9.2 has zoomed into. Also note the change in the cardinal directions.

Table 9.3: North-East-Down (N) Coordinate Frame

Origin	Munition center of mass
Fundamental Plane	Plane created by 1^{st} and 3^{rd} axis
1^{st} **Axis & Definition**	N—Points North
2^{nd} **Axis & Definition**	E—Points East
3^{rd} **Axis & Definition**	D—Completes right-hand system

Table 9.4: Aerodynamic Coordinate Frame (A)

Origin	Munition center of mass
Fundamental Plane	Plane created by 1^{st} and 3^{rd} axis
1^{st} **Axis & Definition**	x—Along munition's longitudinal axis
2^{nd} **Axis & Definition**	y—Out right "wing", 90° to x-axis
3^{rd} **Axis & Definition**	z—Completes right-hand system

Tables 9.3 and 9.4 describe the other coordinate frames. These frames move much more both relative to each other and the other systems. Figure 9.2 shows their relative orientation at one point in the trajectory. At other points in the flight, the relationship would be drastically different. As a side note, the size and shape of the trajectory relative to the Earth is approximately correct for a short-range firing with a gun elevation of 60° and rocket ignition at 33 seconds. The total downrange travel is around 28 kilometers, while the maximum altitude is just under nine kilometers.

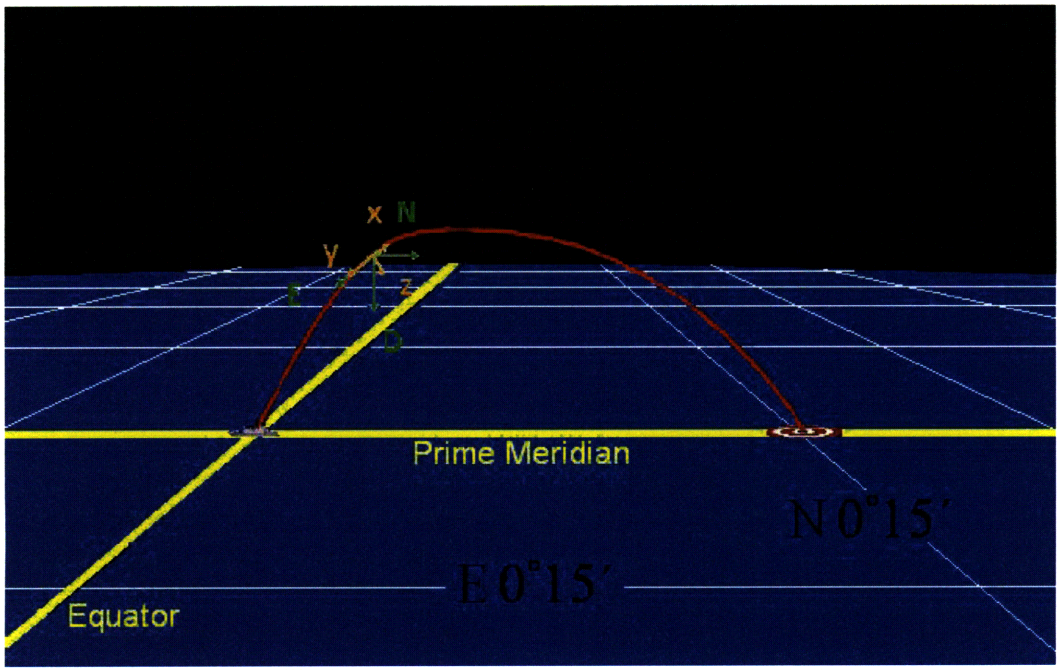


Figure 9.2: North-East-Down (N) and Aerodynamic (A) Coordinate Frames

[This Page Intentionally Left Blank]

Bibliography

[1] Ergm: Extended range guidance munition, 2006-2007. <http://www.raytheon.com/products/ergm/>.

[2] F. P. Adler. Missile guidance by three-dimensional proportional navigation. In *Journal of Applied Physics*, volume 27, pages 500–7, 1956.

[3] ATK. *Ballistic Trajectory Extended Range Munition (BTERM).* <http://www.atk.com/customer_solutions_missionsystems/cs_ms_w_gp_bterm.asp>.

[4] S. N. Balakrishnan and Jie Shen. Hamiltonian based adaptive critics for missile guidance. In *Guidance, Navigation and Control Conference, San Diego, CA*, Jul 29-31, 1996. AIAA-1996-3836.

[5] K. Becker. Closed-form solution of pure proportional navigation. In *IEEE Transactions on Aerospace and Electronic System*, volume 26, pages 526–32, 1990.

[6] Richard Bellman. *Dynamic Programming.* Dover Publications, dover edition of 1957 original edition, 2003.

[7] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 1995.

[8] John T. Betts. Survey of numerical methods for trajectory optimization. In *Journal of Guidance, Control, and Dynamics*, volume 21, pages 193–207, 1998. 0731-5090.

[9] B. Bowring. Transformation from spatial to geographical coordinates. In *Survey Review*, volume XXIII, pages 323–327, 1976.

[10] Ying-Chwan Chiou and Chen-Yuan Kuo. Qualitative analysis of missile guidance problem. In *AIAA Guidance, Navigation, and Control Conference, New Orleans, LA*, Aug 11-13, 1997,. AIAA-1997-3689. Collection of Technical Papers. Pt. 2 A97-37001, pp 10-63.

[11] Ying-Chwan Chiou and Chen-Yuan Kuo. Geometric approach to three-dimensional missile guidance problem. In *Journal of Guidance, Control, and Dynamics*, volume 21, pages 335–341, 1998. 0731-5090.

[12] R. G. Cottrel. Optimal intercept guidance for short-range tactical missiles. In *AIAA Journal*, volume 9, pages 1414–5, 1971.

[13] Jeffrey S. Dalton and S. N. Balakrishnan. Neural on-line learning in missile guidance. AIAA-1993-3872.

[14] Department of the Navy: Research, Development and Acquisition. *ERGM Extended Range Guided Munition*, May 2007. <http://acquisition.navy.mil/programs/weapons/ergm>.

[15] Department of the Navy: Research, Development and Acquisition. *Tactical Tomahawk*, May 2007. <http://acquisition.navy.mil/content/view/full/4709>.

[16] Clark R. Dohrmann, G. R. Eisler, and Rush D. Robinett. Dynamic programming approach for burnout-to-apogee guidance of precision munitions. In *Journal of Guidance, Control, and Dynamics*, volume 19, pages 340–346, 1996. 0731-5090.

[17] Global Security Online. *Ballistic Trajectory Extended Range Munition (BTERM) Autonomous Naval Support Rounds (ANSR)*, Jan 2006. <http://www.globalsecurity.org/military/systems/munitions/bterm.htm>.

[18] Global Security Online. *Extended Range Munition (ERM)*, Feb 2006. <http://www.globalsecurity.org/military/systems/munitions/erm.htm>.

[19] C. Gracey, E. M. Cliff, F. H. Lutze, and H. J. Kelley. Fixed-trim re-entry guidance analysis. In *AIAA Guidance and Control Conference, Alburquerque, New Mexico*, volume 5, Aug 1981.

[20] M. Guelman. A qualitative study of proportional navigation. In *IEEE Transactions on Aerospace and Electronic Systems*, volume 7, pages 637–43, 1971.

[21] P. Gurfil. Robust guidance for electro-optical missiles. In *IEEE Transactıons on Aerospace and Electronis Systems*, volume 39, pages 450–461, 2003.

[22] P. Gurfil. Zero-miss distance guidance law based on line-of-sight rate measurement only. In *Control Engineerıng Practice*, volume 11, pages 819–832, 2003.

[23] P. Gurfil and J. Kasdin. Improving missile guidance performance by in-flight two-step nonlinear estimation of radome aberration. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, Austin, Texas*, Aug 11-14, 2003. AIAA-2003-5723.

[24] S. Gutman. On optimal guidance for homing missiles. In *Journal of Guidance, Control, and Dynamıcs*, volume 2, pages 296–300, 1979.

[25] Jonathan How. Principles of optimal control. Lecture Notes for 16.323, Spring 2006. MIT OpenCourseWare, <http://ocw.mit.edu>.

[26] S. H. Jalali-Naini. Analytical study of a modified los guidance. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, Montreal, Canada*, Aug 6-9, 2001. AIAA-2001-4045.

[27] Donald E. Kirk. *Optimal Control Theory: An Introduction.* Dover Publications, 1998.

[28] Carlo Kopp. Cruise missiles. *Australian Aviation*, Sep 1985. <http://www.ausairpower.net/TE-Cruise-Missiles-1985.html>.

[29] R. R. Kumar, H. Seywald, and E. M. Cliff. Near-optimal three-dimensional air-to-air missile guidance against maneuvering target. In *Journal of Guidance, Control, and Dynamics*, volume 18, pages 457–464, 1995.

[30] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[31] R. V. Lawrence. Advanced missile guidance. AIAA-1991-2726.

[32] N. Lechevin and C.A. Rabbath. Lyapunov-based nonlinear missile guidance. In *Journal of Guidance, Control, and Dynamics*, volume 27, pages 1096–1102, 2004. 0731-5090.

[33] S. Levinson, H. Weiss, and J. Ben-Asher. Trajectory shaping and terminal guidance using linear quadratic differential games. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, Monterey, California*, Aug 5-8, 2002. AIAA-2002-4839.

[34] C. Li and W. Jing. New results on three-dimensional differential geometric guidance and control problem. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado*, Aug 21-24, 2006. AIAA-2006-6086.

[35] C. F. Lin. Classical vs. modern control system design for terminal guidance of bank-to-turn intercept missiles. In *Proceedings of the AIAA Guidance and Control Conference, AIAA, New York*, pages 283–301, 1983. AIAA Paper 83-2203.

[36] Ching-Fang Lin, John E. Bibel, Ernest Ohlmeyer, and Steve Malyevac. Optimal design of integrated missile guidance and control. In *AIAA and SAE, 1998 World Aviation Conference, Anaheim, CA*, Sep 28-30, 1998. AIAA-1998-5519.

[37] Fred Lisy and Troy Prince. End game enhancement using reflexive decision making. In *3rd Annual Missiles and Rockets Symposium*, Apr 19, 2002.

[38] Jongki Moon, Kiseok Kim, and Youdan Kim. Design of missile guidance law via variable structure control. In *Journal of Guidance, Control, and Dynamics*, volume 24, pages 659–664, 2001. 0731-5090.

[39] Y. Ochi. Missile guidance law design based on two-degree-of-freedom bearing control. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, Austin, Texas*, Aug 11-14 2003. AIAA-2003-5578.

[40] Craig A. Phillips and James M. Chisholm. Missile guidance and control challenges for short range anti-air warfare. In *AIAA Guidance, Navigation and Control Conference, Baltimore, MD*, Aug 7-10, 1995,. AIAA-1995-3282. Technical Papers. Pt. 2 A95-39609 10-63, Washington, DC, American Institute of Aeronautics and Astronautics, 1995, pp 1002-1019.

[41] William C. Pittman and Rex B. Powell. Trends in tactical missile guidance and control strategy and program formulation. In *AIAA Defense and Space Programs Conference and Exhibit - Critical Defense and Space Programs for the Future, Huntsville, AL*, Sep 23-25, 1997,. AIAA-1997-3952. A Bound Collection of Papers A97-41701, pp 11-12.

[42] S. Rogers. Missile guidance comparison. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, Rhode Island*, Aug 16-19, 2004. AIAA-2004-4882.

[43] Ilan Rusnak. Multiple model-based terminal guidance law. In *Journal of Guidance, Control, and Dynamics*, volume 23, pages 742–746, 2000. 0731-5090.

[44] I[lan] Rusnak and L. Meir. Optimal guidance for high-order and acceleration constrained missile. In *Journal of Guidance, Control, and Dynamics*, volume 14, pages 589–596, 1991.

[45] Jason L. Speyer, Kevin D. Kim, and Minjea Tahk. A passive homing missile guidance law based on new target maneuver models. In *Guidance, Navigation and Control Conference, Portland, OR*, Aug 20-22, 1990. AIAA-1990-3378.

[46] Robert F. Stengel. *Optimal Control and Estimation.* Dover Publications, 1994.

[47] T. L. Vincent and R. Morgan. Guidance against maneuvering targets using lyapunov optimizing feedback control. In *Proceddings of the American Control Conference*, pages 215–220, New York, 2002. IEEE Press.

[48] P. Weinacht, G. Cooper, and J. Newill. Prediction of direct fire munition trajectories using an analytical approach. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit, San Francisco, California*, Aug 15-18 2005. AIAA-2005-5816.

[49] B. White, R. Zbikowski, and A. Tsourdos. Direct intercept guidance using differential geometry concepts. In *AIAA Guidance, Navigation, and Control Conference and Exhibit, San Franciso California*, Aug 15-18, 2005. AIAA-2005-5969.

[50] C. D. Yang and F. B. Yeh. Optial proportional navigation. In *Journal of Guidance, Control, and Dynamics*, volume 11, pages 375–7, 1988.

[51] R. T. Yanushevsky and W. J. Boord. A new approach to guidance law design. In *AIAA Paper*, Aug 2003. 2003-5577.

[52] S. H. Yoo and T. E. Bullock. Missile guidance based on weight adjusted singular perturbation. In *AIAA Guidance, Navigation and Control Conference, Boston, MA*, Aug 14-16, 1989,. AIAA-1989-3482. Technical Papers. Part 1 A89-52526 23-08. Washington, DC, American Institute of Aeronautics and Astronautics, 1989, pp 496-503.

[53] P. J. Yuan and J. S. Chern. Ideal proportional navigation. In *Journal of Guidance, Control, and Dynamics*, volume 15, pages 268–271, 1992.

[54] P. Zarchan. Tactical and strategic missile guidance. In *AIAA Progress in Astronautics and Aeronautics, Reston, VA*, volume 199, pages 11–29, 2002,.