Benchmark Characterization for Reusable Launch Vehicle Onboard Trajectory Generation using a Legendre Pseudospectral Optimization Method

by

Theodore R. Dyckman

B.S. Aerospace Engineering
United States Naval Academy, 2000

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 2002

Signature of Author_____
Department of Aeronautics and Astronautics
May 24, 2002

Certified by _____
Gregg H. Barton
Next Generation Guidance & Control Principle Investigator
The Charles Stark Draper Laboratory, Inc.
Technical Supervisor

Certified by _____
John J. Deyst
Professor of Aeronautics and Astronautics
Thesis Advisor

Accepted by _____
Wallace E. Vander Velde
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

[This page intentionally left blank]

Benchmark Characterization for Reusable Launch Vehicle Onboard Trajectory Generation using a Legendre Pseudospectral Optimization Method

by

Theodore R. Dyckman

Submitted to the Department of Aeronautics and Astronautics on
May 24, 2002, in partial fulfillment of the requirements for the
Degree of Master of Science in Aeronautics and Astronautics

Abstract

Draper Laboratory has developed a core technology that provides the foundation for onboard trajectory generators. The main goal is for these generators to create robust trajectories for the Terminal Area Energy Management phase of flight. The current onboard generator is the first working prototype that utilizes this core technology and has demonstrated the design of multiple trajectories for varying conditions. However, the optimality or robustness of these trajectories is not known. Therefore, in order to further progress the work done in real-time trajectory generation, it is necessary to establish the metrics for determining the robustness of a trajectory. This will also help in the creation of a suitable benchmark for which to judge and formulate future trajectory generators.

This thesis determines benchmark robust trajectories for the current onboard trajectory generator. This is accomplished by utilizing an optimization routine to generate a wide variety of trajectories around different cost functions. A physical measure and definition of robustness was then determined through a thorough analysis of all the parameters of the optimized trajectories. This led to the selection of a benchmark cost function that portrays the definitive characteristics of robust trajectories. In addition, the grading system used in the determination of the benchmark can be used to calculate a quantitative value of robustness for existing trajectories.

The robustness grading system, along with analysis of different trajectories, led to the characterization of the trajectories created from the current onboard generator. While these trajectories were determined to be solutions to tightly constrained problems, they also proved to be sub-optimal representations of the trajectory benchmark. In addition, they are shown to be fairly robust as well. Information and insight gained in the thesis is used to present recommendations for ways to continue the development and testing of new technology for onboard trajectory generation.

Technical Supervisor: Gregg H. Barton
Title: Principle Member of the Technical Staff, C.S. Draper Laboratory, Inc.

Thesis Advisor: John J. Deyst
Title: Professor of Aeronautics and Astronautics

[This page intentionally left blank]

ACKNOWLEDGMENT

May 24, 2002

Foremost, I would like to thank the Charles Stark Draper Laboratory for providing the opportunity, funding, and support necessary to pursue my graduate education at MIT. I am extremely grateful for all the help and guidance that was given to me by members of the staff, including Anil Rao, Ron Proulx, and Chris D'Souza for helping with optimization and any DIDO questions. Particular thanks goes especially to my technical advisor, Gregg Barton, for guiding, teaching, and helping me through this entire process. It is hard to believe this thesis has come so far, especially since we were both very nervous at how it was going to go, not to mention how we were going to tackle the problem we did. I can't say I truly enjoyed laying plot upon plot on your desk, but it was fun at times. Luckily all the chips fell in to place, and not a moment too soon, or too late for that matter. Thanks again.

Second, I would like to express my appreciation to the staff and professors in the MIT Aeronautics and Astronautics Department. Particular thanks go to my thesis advisor, Professor John Deyst, for his time, effort and support. I also enjoyed your Real-Time systems class. It was tough, but I learned a lot. Thanks.

Next, I thank my friends and fellow Draper fellows who have kept me both distracted and focused during these past two years. To the first group of fellows, Steve, Rich, and Raja, for letting me join you for lunch everyday. Raja, thanks for introducing me to the MIT and Draper ways, in addition to the heads up about Vance AFB. I sure you would like to know that the plant is doing well. To Andrew, the only other Navy guy at the time, thanks for your help on how to write the thesis, not to mention the actual template to follow. Also, thanks to Paul for help with MATLAB basics. To the second group of fellows, Geoff, Kim, Christine, Dave, and Jen, its been interesting. It was such a change to have so many new students, and to have to take up a few tables for lunch. To Stuart, for helping with the repotting of the plant, and for being my office mate until you were booted, although you still came in and chatted all the time. Thanks for reminding me to start writing, and see....I did finish it. To Steve, man those workouts are a pain in the butt. Thanks for helping me give a strong showing for the Navy, and for all the good times. Sorry about introducing you to the drive you have to make next year, although Marblehead is awesome. Thanks to Rob and Andy; you got to love those bi-weekly presentations.

Finally, I want to thank my family for their love, support, encouragement and guidance. I am sure you got sick of all my phone calls and complaints, but don't worry, I am sure you will hear it from flight school. I would not be where I am right now if it wasn't for all of you. Thank you so much.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Next Generation Guidance and Control, Internal Research and Development project #18544.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Theodore R. Dyckman

Table of Contents

1 Introduction 21

1.1 Background 21

1.2 Problem Definition 22

1.3 Thesis Objective 24

1.4 Thesis Overview 24

2 Vehicle Description 25

2.1 Overview 25

2.2 Mission Design 25

2.3 Aerodynamic Properties 26

2.4 Physical Description 28

3 Equations of Motion 31

3.1 Overview 31

3.2 Reference Coordinate Frames 31

3.3 Transformation of Coordinate Frames 32

3.4 Nonlinear Equations of Motion 35

4 Guidance and Control 43

4.1 Overview 43

4.2 Traditional Shuttle Guidance and Control 43

4.2.1 Guidance and Control Concept 43

4.2.2 Terminal Area Energy Management and Approach and Landing Phases 45

4.2.3 Limitations 49

4.3 Next Generation Guidance and Control (NGGC) 51

4.3.1 Guidance and Control Concept 51

4.3.2 Components 51

5 Onboard Trajectory Generation 55

5.1 Overview 55

5.2 Kernel Extraction Protocol (KEP) 55

5.2.1 Kernel Equations of Motion 55

5.2.2 KEP Methodology 59

5.3 Geometrically Constrained Methods 62
5.3.1 Definition 62
5.3.2 Auto-Landing I-load Program (ALIP) 63
5.3.3 ALIP3D 65
5.3.4 Limitations 68
5.4 Sub-optimal Nodal Application of the Kernel Extraction (SNAKE) Method 70
5.4.1 Overview 70
5.4.2 Dynamic Pressure Schedule 71
5.4.3 Ground Track Formulation 75
5.4.4 Rapid Trajectory Propagation 79
5.5 Proto-Snake 81
5.5.1 Methodology 81
5.5.2 Limitations 84
5.6 Summary 85
6 Optimization Tools 87
6.1 Overview 87
6.2 Optimization Problem 87
6.3 Optimization Methods 89
6.3.1 Pseudospectral Methods 89
6.3.1.1 Finite Difference Matrix 89
6.3.1.2 Pseudospectral Differentiation Matrix 91
6.3.2 Legendre Pseudospectral Optimization Method 93
6.4 Direct Indirect Dynamic Optimization (DIDO) 94
6.5 Software and Hardware Specifications 95
6.5.1 Software 95
6.5.2 Hardware 96
7 Benchmark Determination 97
7.1 Overview 97
7.2 Problem Setup 97
7.2.1 Mathematical Formulation 97
7.2.1.1 States and Controls 97

7.2.1.2 Dynamic Constraints 98
7.2.1.3 Event Constraints 99
7.2.1.4 Path Constraints 100
7.2.2 Non-Dimensionalization 101
7.2.3 Tables 103
7.2.4 Nodal Determination 104
7.2.5 Soft Knot 104
7.3 Cost Function Formulation 105
7.3.1 Proto-Snake Characterization and Verification 105
7.3.2 Robust Trajectories 108
7.3.2.1 Goals 108
7.3.2.2 Cost Layout 109
7.4 Trajectory Selection / Benchmark Selection 111
7.4.1 Desirability Criterion 111
7.4.2 Test Layout 112
7.4.3 Scoring / Grading 120
8 Results 129
8.1 Overview 129
8.2 Proto-Snake Characterization 129
8.3 Benchmark Trajectory / Cost Function 133
8.4 Summary 140
9 Conclusions 143
9.1 Summary and Conclusions 143
9.2 Recommendations for Future Work 145
Appendix A DIDO Problem Setup 151
A.1 User Interface Code 151
Appendix B DIDO Results 157
B.1 Graphical Presentation of DIDO Results 157
Appendix C Scoring Results 187
C.1 Tables of Parameter Scores 187
References 189

[This page intentionally left blank]

List of Figures

Figure 2.1: Trimmed Lift and Drag Coefficient (X-34) 26
Figure 2.2: Trimmed L/D versus Angle of Attack (X-34) 27
Figure 2.3: Schematic of Orbital Sciences' X-34 [1] 29
Figure 3.1: Inertial, Local Horizontal and Body Reference Frames 32
Figure 3.2: Heading Angle / Flight Path Angle Rotation 33
Figure 3.3: Bank Angle Rotation: 34
Figure 3.4: Stability and Body Reference Frames 35
Figure 3.5: Forces Projected onto the Vertical Plane of the Velocity Reference Frame 38
Figure 3.6: Forces Projected onto the Horizontal Plane of the Velocity Reference Frame . 39
Figure 4.1: Traditional Space Shuttle G&C Layout 44
Figure 4.2: Traditional Approach to Trajectory Generation [7] 44
Figure 4.3: TAEM Subphases [9] 46
Figure 4.4: Acquisition Subphase Segments 47
Figure 4.5: Flight Path along the HAC [9] 48
Figure 4.6: Approach and Landing Subphases [9] 49
Figure 4.7: Next Generation Guidance and Control System Concept 51
Figure 4.8: NGGC Approach to Onboard Trajectory Generation [7] 52
Figure 5.1: "Balancing the Dynamics" through Dynamic Pressure 61
Figure 5.2: Auto-Landing Subphases and Geometry [11] 63
Figure 5.3: Effect of Adjusting X_{ZERO} [11] 65
Figure 5.4: Elements of Lateral Geometry [13] 66
Figure 5.5: Elements of Longitudinal Geometry [13] 67
Figure 5.6: Possible High Mach Number Acquisition Turn [1] 69
Figure 5.7: ALI-relative Coordinate System [1] 70
Figure 5.8: Dynamic Pressure Schedules 72
Figure 5.9: Flight Path for Max Dive Trajectory 73
Figure 5.10: Typical Energy Corridor 74
Figure 5.11: Nodes and Ground Track Segments [1] 75

Figure 5.12: Ground Track Segment Orientation [1] 76
Figure 5.13: Family of Constant Nz Turns [1] 77
Figure 5.14: Analogy of Toy Snake to Ground Track [1] 78
Figure 5.15: Manipulation of Projected Ground Track to the ALI [1] 79
Figure 5.16: General Propagation Algorithm [1] 80
Figure 5.17: γ versus $^{d\gamma}/_{dh}$ Table for Segment Length Prediction [1] 82
Figure 5.18: Phase One of the Ground Track Solver [1] 83
Figure 5.19: Phase Two of the Ground Track Solver [1] 83
Figure 6.1: Position Data Taken at LGL Time Points [16] 93
Figure 7.1: Comparison of DIDO Verification Trajectory to a Proto-snake Trajectory 107
Figure 7.2: Proto-snake Reference Trajectories 114
Figure 7.3: Effect of $\dot{\alpha}$ and $\dot{\mu}$ in the Cost Functions 116
Figure 7.4: Effect of $\dot{\alpha}$ and $\dot{\mu}$ in "Smoothing" α and μ 117
Figure 7.5: Comparison of Different Dynamic Pressure Profiles 119
Figure 7.6: Dynamic Pressure Scoring Scheme 122
Figure 7.7: Nz_b Scoring Scheme 123
Figure 7.8: Heading Angle Scoring Scheme 124
Figure 7.9: Crossrange Parameter Scoring Area 125
Figure 7.10: Crossrange Parameter Scoring Scheme 126
Figure 7.11: $\dot{\alpha}$ and $\dot{\mu}$ Scoring Scheme 126
Figure 8.1: Effect of y Parameter in Cost Function 130
Figure 8.2: Nz_b Parameter Matching Issue 131
Figure 8.3: Comparison of (qam) Trajectories with Proto-snake 132
Figure 8.4: Most Robust Trajectory of Point 1 134
Figure 8.5: Most Robust Trajectory of Point 2 136
Figure 8.6: Most Robust Trajectory of Point 3 137
Figure 8.7: Most Robust Trajectory of Point 4 139
Figure 9.1: Dynamic Pressure Schedule Plateaus 146
Figure 9.2: "Steer-to-Zero" Logic 148
Figure 9.3: "Steer-to-Transition" Logic 149
Figure B.1: DIDO Run (qam1) versus Point 1 Reference 158

Figure B.2: DIDO Run (qamy1) versus Point 1 Reference 159
Figure B.3: DIDO Run (qamc1) versus Point 1 Reference 160
Figure B.4: DIDO Run (minEamnz1) versus Point 1 Reference 161
Figure B.5: DIDO Run (minEamy1) versus Point 1 Reference 162
Figure B.6: DIDO Run (maxEamc1) versus Point 1 Reference 163
Figure B.7: DIDO Run (maxEamy1) versus Point 1 Reference 164
Figure B.8: DIDO Run (qam2) versus Point 2 Reference 165
Figure B.9: DIDO Run (qamy2) versus Point 2 Reference 166
Figure B.10: DIDO Run (qamc2) versus Point 21 Reference 167
Figure B.11: DIDO Run (minEamnz2) versus Point 2 Reference 168
Figure B.12: DIDO Run (minEamy2) versus Point 2 Reference 169
Figure B.13: DIDO Run (maxEamc2) versus Point 2 Reference 170
Figure B.14: DIDO Run (maxEamy2) versus Point 2 Reference 171
Figure B.15: DIDO Run (qam3) versus Point 3 Reference 172
Figure B.16: DIDO Run (qamy3) versus Point 3 Reference 173
Figure B.17: DIDO Run (qamc3) versus Point 3 Reference 174
Figure B.18: DIDO Run (minEamnz3) versus Point 3 Reference 175
Figure B.19: DIDO Run (minEamy3) versus Point 3 Reference 176
Figure B.20: DIDO Run (maxEamc3) versus Point 3 Reference 177
Figure B.21: DIDO Run (maxEamy3) versus Point 3 Reference 178
Figure B.22: DIDO Run (qam4) versus Point 4 Reference 179
Figure B.23: DIDO Run (qamy4) versus Point 4 Reference 180
Figure B.24: DIDO Run (qamc4) versus Point 4 Reference 181
Figure B.25: DIDO Run (minEamnz4) versus Point 4 Reference 182
Figure B.26: DIDO Run (minEamy4) versus Point 4 Reference 183
Figure B.27: DIDO Run (maxEamc4) versus Point 4 Reference 184
Figure B.28: DIDO Run (maxEamy4) versus Point 4 Reference 185

[This page intentionally left blank]

List of Tables

Table 2.1: X-34 Physical Characteristics [3, 4] 29
Table 6.1: Computer Hardware Specifications 96
Table 7.1: Description of State Variables 98
Table 7.2: Path Constraint Bounds 101
Table 7.3: Longitudinal Parameters 110
Table 7.4: Lateral Parameters 110
Table 7.5: Desirability Criterion 111
Table 7.6: Center of Corridor Cost Functions 113
Table 7.7: Max Glide Cost Functions 113
Table 7.8: Max Dive Cost Functions 113
Table 7.9: Initial Conditions of Proto-snake Reference Trajectories 115
Table 7.10: Cost Functions Remaining After Second Down-Selection 118
Table 7.11: Cost Functions Remaining After Third Down-Selection 120
Table 7.12: Benchmark Selection / Robustness Parameters 121
Table 7.13: Scoring Parameter Weightings 127
Table 8.1: Point 1 Grading Results 134
Table 8.2: Point 2 Grading Results 136
Table 8.3: Point 3 Grading Results 137
Table 8.4: Point 4 Grading Results 138
Table 8.5: Overall Grading Results 139
Table C.1: Point 1 Trajectory Results 187
Table C.2: Point 2 Trajectory Results 187
Table C.3: Point 3 Trajectory Results 188
Table C.4: Point 4 Trajectory Results 188

[This page intentionally left blank]

List of Symbols

a ..... Acceleration

$\vec{a}$ ..... Acceleration vector

b ..... Wing span

$\overline{c}$ ..... Mean aerodynamic chord

C_D ..... Aerodynamic drag coefficient

CG ..... Center of gravity

C_L ..... Aerodynamic lift coefficient

D ..... Aerodynamic drag force

D_N ..... Differentiation matrix

e ..... Event constraint function

E/W ..... Energy-over-weight

$\vec{F}$ ..... Total force vector

$\vec{F}_{aero}$ ..... Aerodynamic force vector

$\vec{F}_{gravity}$ ..... Gravity force vector

g ..... Gravity force

h ..... Altitude

h_{norm} ..... Normalizing altitude value

h^* ..... Normalized altitude

Δh ..... Small altitude increment (step)

$k_{\Delta\alpha}$ ..... Adaptive gain for small change in α

L ..... Aerodynamic lift force

$\boldsymbol{L}$ ..... Integrand or Lagrange cost function

L/D ..... Lift-over-drag

m ..... Mass

M ..... Mach number

maxEamc ..... Cost function maximizing E/W and minimizing $\dot{\alpha}$, $\dot{\mu}$, and χ

maxEamy ..... Cost function maximizing E/W and minimizing $\dot{\alpha}$, $\dot{\mu}$, and y

minEamnz ..... Cost function minimizing E/W, $\dot{\alpha}$, $\dot{\mu}$, and Nz_b

minEamy ..... Cost function minimizing E/W, $\dot{\alpha}$, $\dot{\mu}$, and y

Nx_b ..... Body frame x-acceleration

Ny_b ..... Body frame y-acceleration

Nz_b ..... Body frame z-acceleration

Nx_v ..... Velocity frame x-acceleration

Ny_v ..... Velocity frame y-acceleration

Nz_v ..... Velocity frame z-accleration

p ..... Path constraint function

$\overline{q}$ ..... Dynamic pressure

qam ..... Cost function minimizing $(\overline{q} - 335), \dot{\alpha}, \dot{\mu}$

$qamc$ ..... Cost function minimizing $(\overline{q} - 335), \dot{\alpha}, \dot{\mu}$, and χ

$qamy$ ..... Cost function minimizing $(\overline{q} - 335), \dot{\alpha}, \dot{\mu}$, and y

$\vec{r}$ ..... Position vector

S ..... Planform area

t ..... Time

$\mathbf{T}_{a2b}$ ..... Transformation matrix from a to b

$\mathbf{u}$ ..... Vector of control variables

V ..... Inertial (ground-relative) velocity

V_{norm} ..... Normalizing velocity value

V^* ..... Normalized velocity

$\vec{V}$ ..... Velocity vector

W ..... Weight

x ..... Downrange distance (position component)

x_{norm} ..... Normalizing downrange value

x^* ..... Normalized downrange distance

$\mathbf{x}$ ..... Vector of state variables

y ..... Crossrange distance (position component)

y_{norm} ..... Normalizing crossrange value

y^* ..... Normalized crossrange distance

Y ..... Side force

z ..... Vertical position component

α ..... Angle of attack

β ..... Sideslip angle

δ_{sb} ..... Speedbrake position

χ ..... Runway-relative heading angle

ϕ ..... End point or Mayer cost function

Φ ..... Roll angle

γ ..... Flight path angle

μ ..... Bank angle

ρ ..... Density

τ ..... Clock time

τ_0 ..... Initial clock time

τ_f ..... Final clock time

List of Acronyms

A/L ..... Approach and Landing

ALI ..... Auto-Landing Interface

ALIP ..... Auto-Landing I-load Program

DIDO ..... Direct Indirect Dynamic Optimization

G&C ..... Guidance and Control

GTS ..... Ground Track Solver

HAC ..... Heading Alignment Cone

IG&C ..... Integrated Guidance and Control

KEP ..... Kernel Extraction Protocol

MD ..... Max Dive

MG ..... Max Glide

NGGC ..... Next Generation Guidance and Control

RLV ..... Reusable Launch Vehicle

SNAKE ..... Sub-optimal Nodal Application of Kernel Extraction

TAEM ..... Terminal Area Energy Management

TPBVP ..... Two Point Boundary Value Problem

[This page intentionally left blank]

Chapter 1

Introduction

1.1 Background

The flight success of the Space Shuttle is based in part on its re-entry guidance and control algorithms. So much so, that even after 25 years little has been done to improve upon the baseline shuttle Guidance and Control (G&C) framework. Even vehicles being designed today, like the X-34, X-37 and X-40, utilize G&C techniques designed to operate on circa 1970s flight computers. These algorithms were originally restricted by the limited memory and raw computing power available at the time. They rely on loaded trajectories defined well before launch, which involves labor-intensive pre-flight design, restricts the vehicle to tight, nominal flight corridors and reduces the vehicle's robustness to changing flight conditions. Any abort contingencies must also be pre-defined, and pre-loaded, which effectively prevents the full exploitation of a vehicle's recovery capacity. In addition, any flight condition encountered by the vehicle for which no pre-planned trajectory has been defined may result in the catastrophic loss of the vehicle.

The Charles Stark Draper Laboratory (Draper) has begun an initiative to develop the technologies necessary for the formulation of a next generation guidance and control framework for Reusable Launch Vehicles (RLV). This framework seeks to address and improve upon the limitations of the shuttle era G&C systems. The plan calls for three key components: an autonomous abort planner, an onboard trajectory generator and an Integrated G&C (IG&C) framework. Understanding the relationship between these components provides a clearer context for each individual technology.

Current shuttle G&C techniques rely on mission planners designing trajectories for every abort scenario they can envision and loading these "canned" responses into the vehicle's flight computer. While this may prove effective for a small range of conditions, the current system is not capable of robust abort, or the ability to direct the vehicle to a safe landing for a wide range of off-nominal failure conditions. In contrast, an autonomous abort planner would instantaneously assess the vehicle's current states and scrutinize the available "energy versus downrange" for a variety of landing options. It would select the

best runway within the vehicle's capability, but must rely on the guidance system to generate the necessary reference trajectory.

The onboard trajectory generator would autonomously create the reference trajectories from the vehicle's current position to the desired terminal conditions in real time. This could greatly enhance the G&C's capacity to handle off-nominal, anomalous conditions while taking advantage of the vehicle's full flight capability. While its use in this capacity may provide the greatest enhancement for next generation systems, it should also function equally well under nominal flight conditions. The use of an onboard generator will eliminate the need for mission-specific, pre-defined trajectories and can lead to greater robustness, improved overall performance and lower operational costs.

In order to fully capture the improvements offered by an onboard trajectory generator, it is necessary to couple it with an IG&C framework. This would enable the control system to utilize the new guidance inputs and recalculate the necessary control gains so that the vehicle accurately follows the reference trajectory. The current shuttle-based techniques separate the G&C functions, which has the undesired result that the guidance and control systems sometimes react to each other instead of cooperating to achieve the desired trajectory. An IG&C framework has the potential to overcome these difficulties by integrating and coupling the guidance and control systems, allowing real-time implementation of onboard generated trajectories and improved performance.

1.2 Problem Definition

Draper's work in pursuit of the onboard trajectory generation component has resulted in the creation of three different programs, all utilizing the X-34 as a representative RLV model. The Auto-Landing I-Load Program (ALIP) was developed by G. H. Barton as a rapid, pre-mission design tool for the generation of autolanding trajectories. This program laid the foundation for rapid, real-time trajectory propagation and served as starting point for further research conducted by A. R. Girerd. In his 2001 MIT Master's thesis, Girerd presented methodologies for onboard generation of trajectories throughout the subsonic portion of the Terminal Area Energy Management (TAEM) flight regime, consisting of altitudes less than 40,000 feet. His results provided the insight for the necessity of a more general design approach, which encompasses the full range of TAEM, including supersonic and high altitude flight. This led to the development of Proto-snake, by A. C. Grubler, which demonstrates a unique trajectory generation

methodology that uses dynamic pressure schedules and a real-time ground track solver. The methodology consists of predicting a straight-line ground track shape, derived from the full capabilities of the vehicle, and then "snaking" around the ground track to satisfy the given design constraints. This solution method enables the program to design trajectories throughout the TAEM regime and for a variety of different vehicle conditions.

Proto-snake does have its limitations however. Currently the ground track solver employed by Proto-snake provides an ad hoc solution for limiting cases, through the use of "brute force" techniques. While this does provide solutions for varying conditions, it does not create optimal trajectories or guarantee robustness*. In addition, the dynamic pressure schedules used in the formulation of the trajectories are user specified. This means that Proto-snake does not choose the best schedule for the given design conditions or present situation.

These limitations are due in part to the lack of a suitable benchmark that represents the defining characteristics of a robust trajectory. These characteristics include the actual lateral ground track shape, in addition to the dynamic pressure schedules flown by the vehicle, which define the longitudinal flight profile of the trajectory. It is hoped that a suitable benchmark will provide characteristics that can be mimicked by a real-time ground track solver to produce sub-optimal robust trajectories for a wide-variety of conditions.

Generating a robust trajectory through offline methods is difficult in itself. While methodologies exist for the propagation of optimal or feasible trajectories, no such program currently generates "robust" trajectories. Optimization routines can be used in the creation of trajectories, however they rely on the formulation of a cost functions that utilize vehicle states and controls. The robustness of a vehicle is not a vehicle state or even a well-defined vehicle parameter. This limits the ability to create benchmark robust trajectories, while at the same time provides freedom for a designer to establish his or her own set of characteristics for defining robust trajectories.

* A robust trajectory is defined as a trajectory that can handle future dispersions while still providing the means for the vehicle to attain the final end condition.

1.3 Thesis Objective

This thesis seeks to determine benchmark robust trajectories for the current onboard trajectory generators. This is done by utilizing a Legendre Pseudospectral Optimization method to generate a wide variety of trajectories around different cost functions. An analysis of the resulting trajectories, in addition to the use of desirability criterion, will lead to the determination of the characteristics of robust trajectories. This research also intends to characterize the optimality and robustness of the current trajectories created by A. Grubler's Proto-snake trajectory generator. Furthermore, the resulting trajectories are intended to provide a starting point for the formulation of future ground track solvers.

1.4 Thesis Overview

The present chapter provides an overall view of the main subject of research for this thesis. Subsequent chapters are narrower in their focus as they describe in detail the necessary background as well as procedures for the attainment of the thesis objective. Chapter 2 gives an overview of RLV type vehicles, including the X-34, which was the demonstration vehicle of choice for this research, as well as for previous onboard trajectory generators, while Chapter 3 presents the derivations of the equations of motion, which govern the vehicle in flight. Chapter 4 describes the shuttle-era G&C framework as well as Draper's vision for the Next Generation Guidance and Control (NGGC). The main technological component of Draper's NGGC, onboard trajectory generation, is covered in Chapter 5, including a more thorough description of the three developmental programs and their methodologies. Chapter 6 provides an overview of the Legendre Pseudospectral Method and it use while Chapter 7 describes the actual problem setup used for the determination of a trajectory benchmark. Chapter 8 describes the results of the research program and Chapter 9 concludes the thesis with an overall summation of the results and recommendations for future research.

Chapter 2

Vehicle Description

2.1 Overview

Draper Laboratory has been in pursuit of technologies that are necessary for the formulation of a Next Generation Guidance and Control (NGGC) system for Reusable Launch Vehicles (RLV). In order to understand the development of these technologies, particularly the onboard trajectory generation component, it is necessary to understand the vehicle for which they are designed to guide through the atmosphere. This chapter intends to provide an overview of RLVs in general, including their basic mission design and aerodynamic properties. The last section describes the X-34, which was chosen as the representative RLV model on which the trajectory generation technology was applied during previous research. This thesis continues to use this vehicle for convenience and continuity sake.

2.2 Mission Design

Reusable Launch Vehicles are spacecraft designed to perform specified missions, multiple times. The missions may include ferrying humans and supplies to orbit, launching satellites, or even performing experiments during flight. They were originally developed to dramatically reduce the cost of access to low Earth orbit, strictly due to their reusability. For instance, the Saturn V rocket was expended while sending humans to the Moon. In order to perform another Moon flight, it was necessary to construct another rocket. On the other hand, the Space Shuttle, the first reusable launch vehicle or first generation RLV, has performed over one hundred missions between four flight vehicles.

Draper Laboratory's NGGC concept is primarily focused on guiding shuttle-like RLVs. This includes vehicles designed with the same general characteristics of the Space Shuttle, in addition to the same relative flight dynamics. The shuttle is designed to make unpowered gliding approaches to horizontal landings on a conventional runway. Its general characteristics include low aspect ratio wings, a lifting body shape, and four primary flight control surfaces. The shuttle is also a low Lift-over-Drag (L/D) flight vehicle, and its corresponding trajectories are significantly different from gliding aircraft

or sailplanes. The X-33, X-34, X-37, and X-40 all fall into the category of "shuttle-like" and the NGGC system being developed can easily be applied to any of these vehicles.

2.3 Aerodynamic Properties

Shuttle-class gliding reentry vehicles fly differently from conventional aircraft. With no available thrust to provide a net positive energy source, low L/D vehicles must budget their total energy throughout their entire descent to ensure they will reach the target runway at the specified dynamic constraints. The low L/D characteristics impede efficient gliding performance and generally result in trajectories with steep equilibrium glide slopes and relatively high velocities. Therefore, shuttle-like RLVs usually have a more limited landing footprint and a smaller margin for trajectory errors than higher L/D vehicles. This fact helps to justify the need to design robust trajectories in order to guarantee vehicle safety and recoverability.

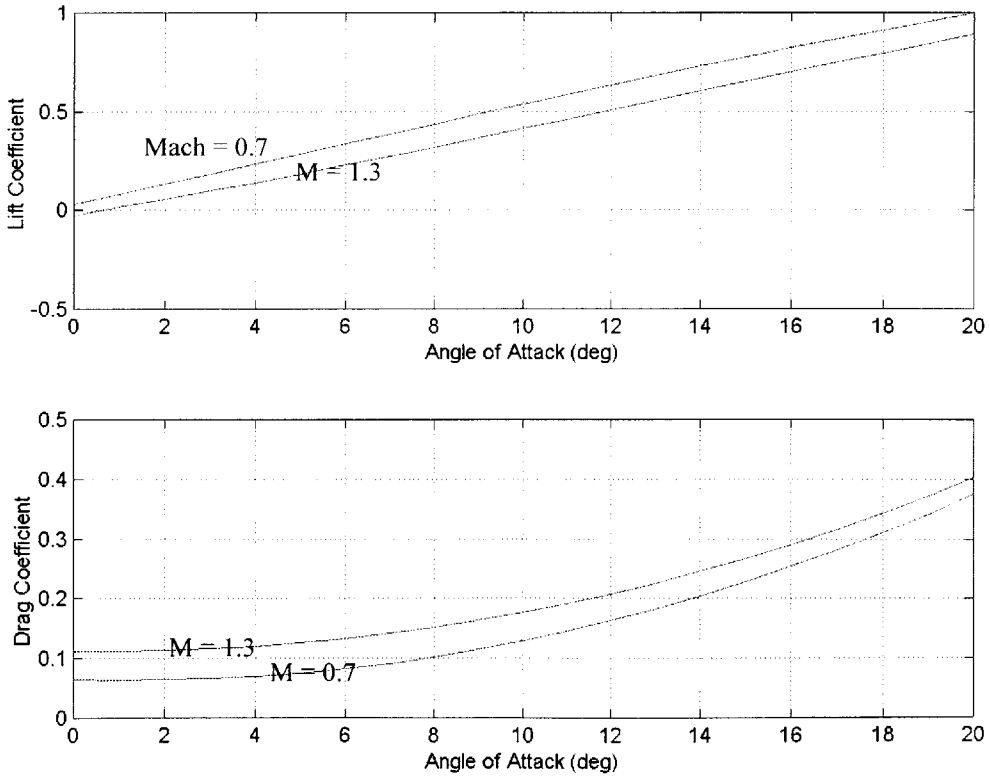


Figure 2.1: Trimmed Lift and Drag Coefficients (X-34)

A RLV can control its L/D ratio to some extent by varying its angle of attack, α. This is due to α's effect on the lift and drag coefficients of the vehicle, as shown in Figure 2.1. The aerodynamic data presented is for the X-34 with a speedbrake setting of 55°, but is indicative of RLVs in general. For values of α less than five degrees, the lift coefficient varies almost linearly with angle of attack, while the drag coefficient remains relatively flat. Beyond five degrees however, the drag grows exponentially, while the lift retains the linear trend. The resulting L/D curve, shown in Figure 2.2 for various Mach numbers, soon peaks and then starts decreasing. This causes the contour to be divided into a *front* side and *back* side. The maximum L/D of the vehicle occurs at the peak in the contour for the given angle of attack. The front side of the curve is usually quite short for low lift-to-drag vehicles such as the shuttle.

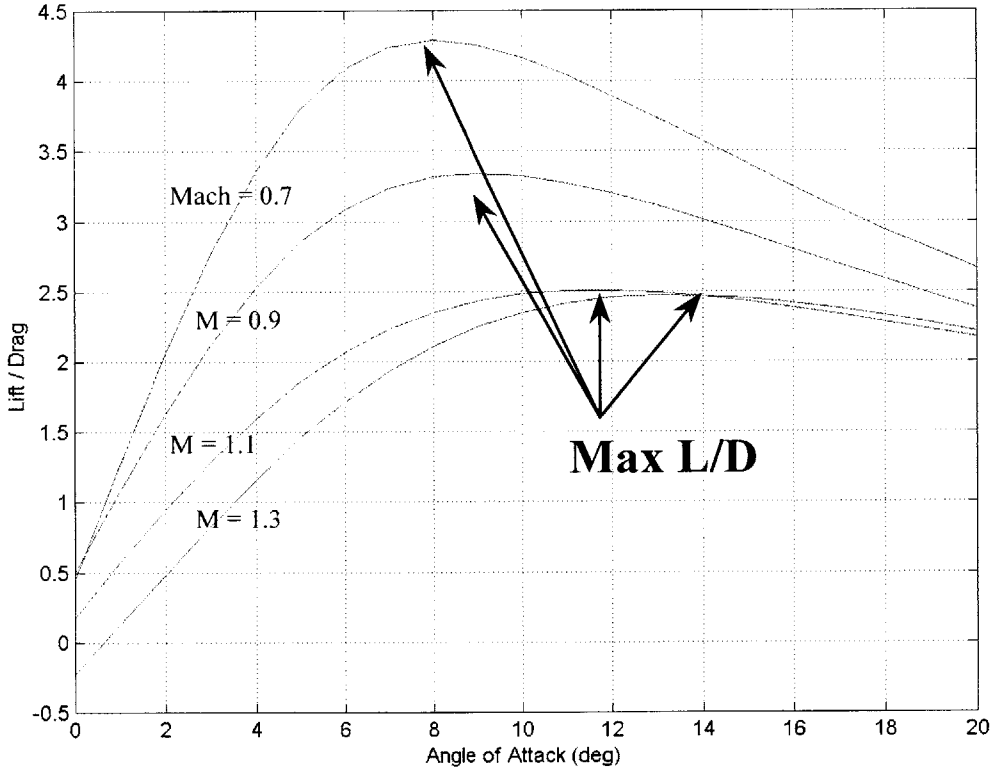


Figure 2.2: Trimmed L/D versus Angle of Attack (X-34)

It is important to keep the L/D curve in mind when designing trajectories. Controllability issues usually mandate keeping the RLV on the front side of the curve, where significant

changes in lift are accompanied by only modest changes in drag. These characteristics are required to sustain flight at a given flight path angle. If the vehicle flies on the back side of the curve, small changes in lift will result in large drag penalties. As speed is reduced due to the increased drag, the vehicle will try to pitch upward to increase the lift lost by the decrease in speed. This cascades into a non-returnable energy decay wherein a specific trajectory cannot be sustained and may lead to the loss of the vehicle [1].

2.4 Physical Description

The X-34 technology demonstrator, designed by Orbital Sciences, was chosen as the technical model for RLVs in this research. This vehicle was chosen in order to maintain continuity with previous research efforts, which also used the X-34, as well as to take advantage of the large amount of technical information and experience available for the vehicle at Draper Laboratory. Under contract with Orbital, Draper developed the entry and autolanding guidance, as well as the flight software, in support of previously planned flight tests. Also, Draper was awarded a Future-X flight demonstration of autonomous Robust Abort Technologies on the X-34 (RADX34) [2]. However, NASA cancelled the flight tests of the X-34 in April 2001, and the two vehicle specimens are currently in flyable storage. A schematic of the X-34 is presented in Figure 2.3.

The X-34 was developed as a single stage RLV, capable of flights up to 250,000 feet and speeds in excess of Mach 8, after being air-launched from the belly of an L-1011 carrier aircraft. It was designed to behave similarly to the shuttle in order to minimize risk, cost, and time of development, and even utilizes the same basic flight controls. These include a rudder, speedbrake, body flap and elevons. The body flap is used exclusively as a trim device for successive stages of entry and is not actively employed by the flight control system. All the pertinent physical characteristics of the vehicle are summarized in Table 2.1.

The X-34 is primarily a bank-to-turn vehicle. In its design, the rudder lacks the ability to cause large changes in the vehicle's heading angle and is only used to keep the vehicle in coordinated flight. The vehicle must bank to achieve any desired yaw rate, which results in a slow yaw response due to the need to roll before a change in heading can occur.

The X-34 is also powered by the reusable Fastrac engine, which runs off a mixture of liquid oxygen and kerosene. This was designed and developed by NASA's Marshall Space Flight Center, and can yield approximately 60,000 pounds of thrust. Even though the X-34 is powered, this research is only focused on the unpowered, gliding flight portion of the vehicle's flight envelope.

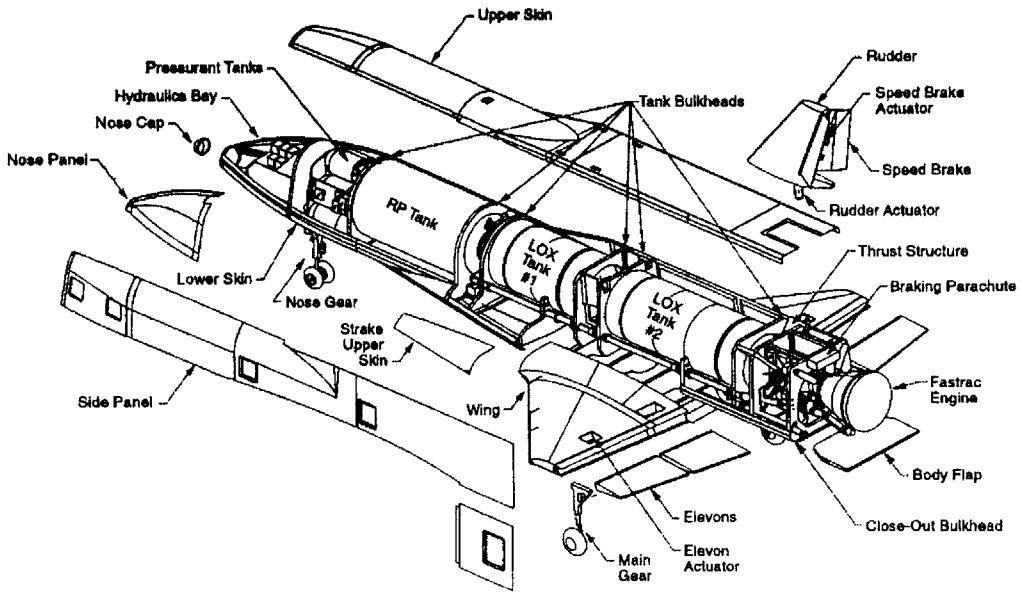


Figure 2.3: Schematic of Orbital Sciences' X-34 [1]

Table 2.1: X-34 Physical Characteristics [3, 4]

Length	58.3 feet
Wing Span, b	27.67 feet
Mean Aerodynamic Chord, $\bar{c}$	14.54 feet
Planform Area, S	357.5 feet^2
Gross Launch Weight	46,500 lb_f
Dry Weight	19,000 lb_f
Elevon Deflection Range	-34.2° to +15.8°
Speedbrake Deflection Range	0° to 103°

[This page intentionally left blank]

Chapter 3
Equations of Motion

3.1 Overview

The equations of motion for an atmospheric vehicle are vital to the formulation and the understanding of the developments in this thesis. They are presented in this chapter, along with their derivations and the appropriate assumptions. The first two sections of this chapter describe the necessary coordinate reference frames and the transformation matrices between the frames. The third section covers the derivations of the equations. For more detailed derivations, see References 5 and 6.

3.2 Reference Coordinate Frames

It is necessary to determine a set of coordinate reference frames when formulating the equations of motion for a vehicle. The following five reference coordinate frames are relevant to the movement of a vehicle in atmospheric flight. They all employ right-handed rectangular Cartesian axes.

Inertial Reference Frame $(\hat{i}_I, \hat{j}_I, \hat{k}_I)$: an Earth-fixed coordinate system, with its origin at the runway threshold as depicted in Figure 3.1. The Earth is assumed to be flat and stationary in inertial space, therefore the Earth is an inertial system, one in which Newton's laws are valid. Additionally, gravity is assumed uniform and constant, and hence the aircraft's center of mass and center of gravity (CG) are the same point.

Local Horizontal Reference Frame $(\hat{i}_h, \hat{j}_h, \hat{k}_h)$: a coordinate system with the origin at the vehicle CG, with axes parallel to the inertial reference frame, as shown in Figure 3.1. The rotation matrix between the inertial and local horizontal reference frames is the identity matrix and is time-invariant.

Velocity Reference Frame $(\hat{i}_v, \hat{j}_v, \hat{k}_v)$: a coordinate system with the origin at the vehicle CG and the $\hat{i}_v$ axis pointing along the velocity vector. The $\hat{j}_v$ axis remains in the local horizontal $\hat{i}_h - \hat{j}_h$ plane and the $\hat{k}_v$ axis completes the right-handed coordinate system.

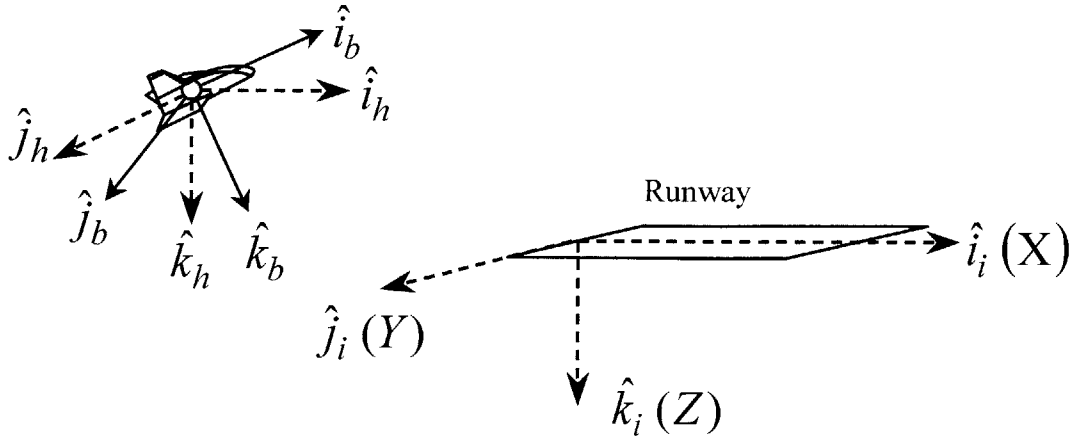


Figure 3.1: Inertial, Local Horizontal and Body Reference Frames

Body Reference Frame $(\hat{i}_b, \hat{j}_b, \hat{k}_b)$: a coordinate system in which the origin lies at the vehicle center of gravity. The $\hat{i}_b$ axis points through the nose of the aircraft and is coincident with the longitudinal axis of the aircraft. The $\hat{j}_b$ axis is positive out the right wing and the $\hat{k}_b$ axis points in the vehicle's ventral direction (positive downward) as depicted in Figure 3.1. If the vehicle is flying wings level and is pointing parallel to the runway, the transformation matrix between the body and local horizontal reference frames would be the identity matrix.

Stability Reference Frame $(\hat{i}_s, \hat{j}_s, \hat{k}_s)$: a coordinate system with the origin at the vehicle CG. The $\hat{i}_s$ axis lies along the projection of the velocity vector ($\vec{V}$) onto the body $\hat{i}_b - \hat{k}_b$ plane. With the assumption of zero sideslip, the velocity vector lies along the $\hat{i}_s$ axis. The $\hat{j}_s$ axis is coincident with the body $\hat{j}_b$ axis along the wing and the $\hat{k}_s$ axis completes the right-handed coordinate system. By definition, the airplane lift and drag vectors are perpendicular and parallel, respectively, to $\vec{V}$ and are aligned with the $-\hat{i}_s$ and $-\hat{k}_s$ axes of this frame, as shown in Figure 3.4.

3.3 Transformation of Coordinate Frames

For this thesis, it is assumed that the aircraft always flies coordinated turns. This means that the sideslip angle (β) is always assumed to be zero. The following transformations take this into account.

Transformations between coordinate frames involve Euler angle rotations about the $\hat{k}$, $\hat{j}$, and $\hat{i}$ axes, in order. A transformation matrix (denoted as $\mathbf{T}_{a2b}$) is a square array containing the Euler rotations between the vector components of the individual coordinate systems. The rotations are represented as sines and cosines of the separation angles between the frames. A vector is rotated from one reference frame into another reference frame by a multiplication of the transformation matrix. Rotating in the opposite direction simply involves multiplying by the transpose of the transformation matrix. Equation 3.1 demonstrates this principle.

$$\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = \mathbf{T}_{a2b} \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \qquad \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} = \mathbf{T}_{a2b}^{T} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} \tag{3. 1}$$

Heading / Flight Path Rotations: The orientation of the velocity reference frame with respect to the inertial and local horizontal reference frames is given by an Euler rotation sequence about the $\hat{k}_h$ and $\hat{j}_h$ axes as shown in Figure 3.2. In the inertial reference frame, the angles χ and γ are defined as the runway-relative heading angle and flight path angle respectively. The transformation matrix from the local horizontal reference frame to the velocity reference frame is

$$\mathbf{T}_{h2v} = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ -\sin\chi & \cos\chi & 0 \\ \sin\gamma\cos\chi & \sin\gamma\sin\chi & \cos\gamma \end{bmatrix} \tag{3. 2}$$

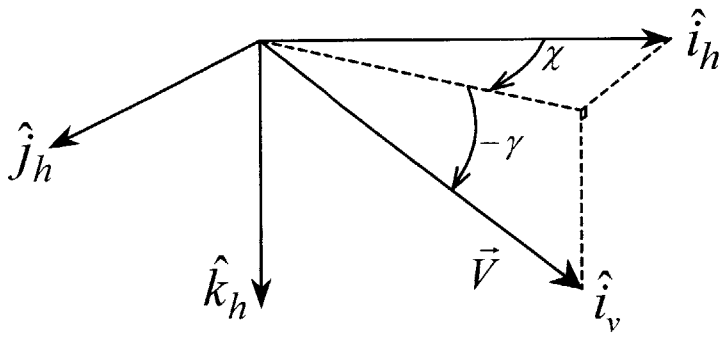


Figure 3.2: **Heading Angle / Flight Path Angle Rotation**

Bank Angle Rotation: The stability reference frame differs from the velocity reference frame by a bank angle μ rotation about their $\hat{i}$ axis, which is the velocity vector. The transformation matrix from the velocity reference frame to the stability reference frame is shown in Eq 3.3. The bank angle μ is shown below.

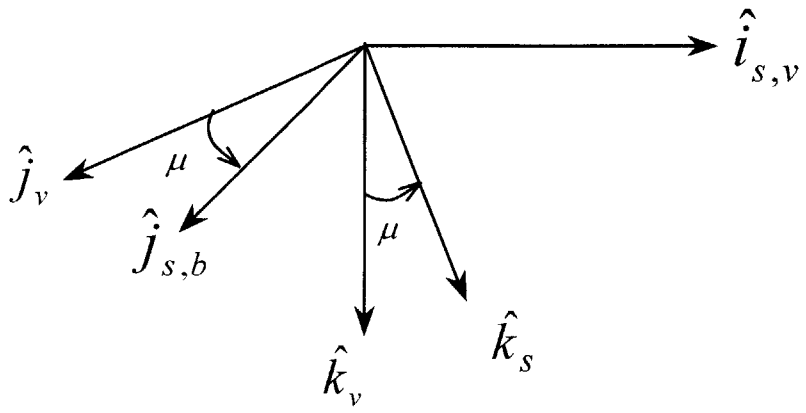


Figure 3.3: **Bank Angle Rotation**

$$\mathbf{T}_{v2s} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \tag{3.3}$$

Angle of Attack Rotation: The transformation from the stability reference frame to the body reference frame is given by a rotation of the angle of attack, α. The angle α is shown in Figure 3.4 and the transformation matrix is

$$\mathbf{T}_{s2b} = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{3.4}$$

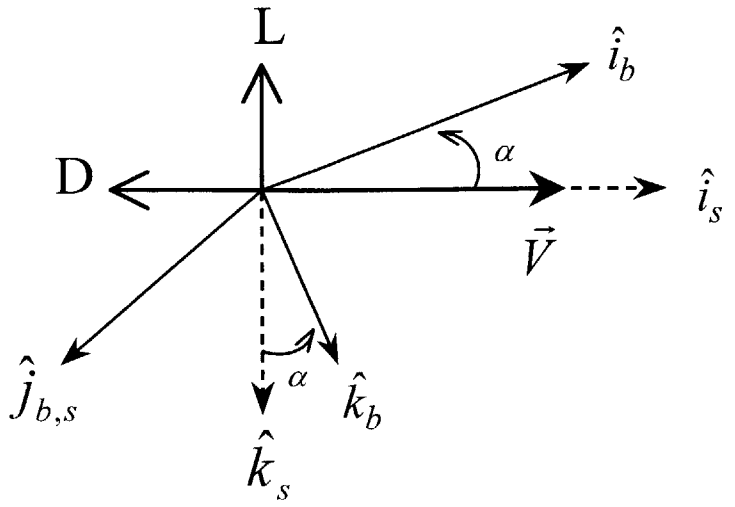


Figure 3.4: Stability and Body Reference Frames

Any other required transformation matrices can be expressed as products of those presented above. For example, the transformation matrix from the velocity reference frame to the body reference frame is given by

$$\mathbf{T}_{v2b} = \mathbf{T}_{s2b}\mathbf{T}_{v2s} \tag{3. 5}$$

simplifying yields

$$\mathbf{T}_{v2b} = \begin{bmatrix} \cos\alpha & \sin\alpha\sin\mu & -\sin\alpha\cos\mu \\ 0 & \cos\mu & \sin\mu \\ \sin\alpha & -\cos\alpha\sin\mu & \cos\alpha\cos\mu \end{bmatrix} \tag{3. 6}$$

3.4 Nonlinear Equations of Motion

The full nonlinear equations of motion for an atmospheric vehicle are derived using Newton's Laws. The derivations are subject to assumptions chosen to reduce the complexity of the formulation. These assumptions are listed below.

- the vehicle has a plane of symmetry
- the vehicle mass properties are constant

- the vehicle produces no thrust
- there are no aerodynamic moments (the vehicle is always in a state of static trim)
- there are no side forces (Y) present (no sideslip)
- the vehicle is a rigid airframe (no bending)
- the wind velocity is always zero

Refer to Reference 1 for derivations that include the complete set of nonlinear wind equations.

The derivation begins by expressing the position vector of the vehicle center of mass with respect to the inertial reference frame, as shown in Eq 3.7. In this frame (as well as the local horizontal reference frame), the positive $\hat{k}_{I,h}$ axis direction is down. However, this direction refers to the height of the vehicle above ground, and the convention is for a positive increase with vertical distance from the ground. Therefore, the $\hat{k}_{I,h}$ position component will be expressed as $z = -h$ (where h represents altitude), shown below,

$$\vec{r} = x\hat{i}_I + y\hat{j}_I + z\hat{k}_I = x\hat{i}_h + y\hat{j}_h - h\hat{k}_h \tag{3.7}$$

The velocity vector $\vec{V}$ is the time derivative of the position vector and is given by

$$\vec{V} = \frac{d\vec{r}}{dt} = \dot{x}\hat{i}_h + \dot{y}\hat{j}_h - \dot{h}\hat{k}_h \tag{3.8}$$

To track the flight path relative to the local horizontal reference frame, it is necessary to transform the velocity vector with respect to the velocity reference frame to the local horizontal reference frame. This is accomplished by using the inverse of the $\mathbf{T}_{h2v}$ matrix.

$$\vec{V} = \mathbf{T}_{h2v}^{T}\begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}_v = V\cos\gamma\cos\chi\,\hat{i}_h + V\cos\gamma\sin\chi\,\hat{j}_h - V\sin\gamma\,\hat{k}_h \tag{3.9}$$

The differential equations for the coordinates of the flight path are then

$$\begin{aligned} \dot{x} &= V\cos\gamma\cos\chi \\ \dot{y} &= V\cos\gamma\sin\chi \\ \dot{h} &= -\dot{z} = V\sin\gamma \end{aligned} \tag{3. 10}$$

The remaining equations are simply statements of Newton's second law of motion, namely

$$\sum \vec{F} = m\vec{a} = \vec{F}_{aero} + \vec{F}_{gravity} \tag{3. 11}$$

The forces affecting the vehicle consist of aerodynamic forces and gravity. The aerodynamic forces are lift (L) and drag (D). The lift and drag vectors represented in these equations are for the complete aircraft, including the wing, tail, fuselage, etc., and act in relation to the stability reference frame of the aircraft. The gravity force, materialized in the weight (W) vector, always acts downward, towards the Earth along the $\hat{k}_{I,h}$ axis.

Consider an aircraft CG to be the intersection of the axes of Figure 3.5. The figure is drawn so that the plane of the page is coincident with the $\hat{i}_v - \hat{k}_v$ axes of the velocity reference frame. The curvilinear motion of the aircraft along a curved flight path can be expressed by first taking a summation of the forces parallel to the flight path, and then taking a summation of the forces perpendicular to the flight path.

The sum of the forces parallel to the flight path is

$$\sum \bar{F}_1 = -D - W\sin\gamma \tag{3. 12}$$

The acceleration parallel to the flight path is

$$a_1 = \frac{d}{dt}V = \dot{V} \tag{3. 13}$$

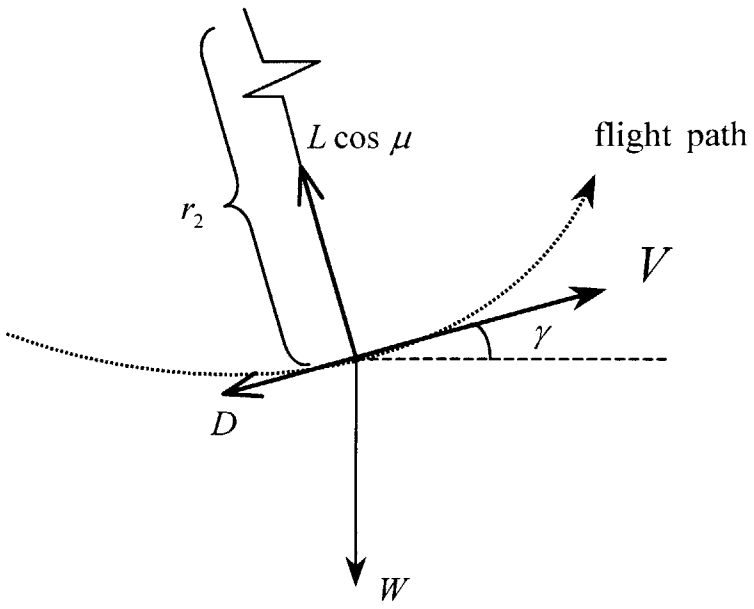


Figure 3.5: Forces Projected onto the Vertical Plane of the Velocity Reference Frame

Applying Newton's Law yields

$$\dot{V} = \frac{-D}{m} - g\sin\gamma \tag{3. 14}$$

The components of the forces perpendicular to the flight path are

$$\sum \vec{F}_2 = W\cos\gamma - L\cos\mu \tag{3. 15}$$

The radial acceleration, perpendicular to the flight path, is written in Eq 3.16. It follows the convention of the velocity reference frame, where $\hat{k}_v$ is positive down.

$$a_2 = -\frac{V^2}{r_2} = -V\dot{\gamma} \tag{3. 16}$$

where $\dot{\gamma}$ is the angular velocity equal to the rate of change of the flight path angle.

Applying Newton's Law and solving for $\dot{\gamma}$ yields

$$\dot{\gamma} = \frac{1}{mV}\left[L\cos\mu - W\cos\gamma\right] \tag{3. 17}$$

Consider now an aircraft CG to be the intersection of the axes of Figure 3.6. The figure is drawn so that the plane of the page is coincident with the $\hat{i}_v - \hat{j}_v$ axes of the velocity reference frame (top-down view).

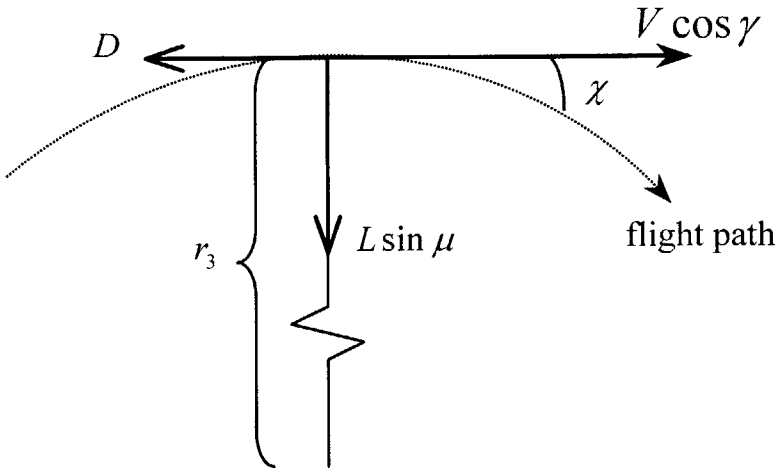


Figure 3.6: Forces Projected onto the Horizontal Plane of the Velocity Reference Frame

The sum of the forces perpendicular to the flight path is

$$\sum \vec{F}_3 = L\sin\mu \tag{3. 18}$$

The instantaneous radial acceleration along the flight path is

$$a_3 = \frac{(V\cos\gamma)^2}{r_3} = (V\cos\gamma)\dot{\chi} \tag{3. 19}$$

where $\dot{\chi}$ is the angular velocity equal to the rate of change of the runway-relative heading angle.

Applying Newton's Law and solving for $\dot{\chi}$ yields

$$\dot{\chi} = \frac{L \sin \mu}{mV \cos \gamma} \tag{3. 20}$$

The total acceleration vector of the vehicle in the velocity reference frame can be written as a summation of Eqs 3.13, 3.16 and 3.19, as shown below

$$\vec{a} = \dot{V}\,\hat{i}_v + (V \cos\gamma)\dot{\chi}\,\hat{j}_v - V\dot{\gamma}\,\hat{k}_v \tag{3. 21}$$

The acceleration loading on the vehicle expressed in the body frame is very important. It is a design parameter in certain instances and can be considered a measure of human 'ride-ability' for the vehicle. Specifically, the quantity of interest is the body Nz acceleration. It is defined as the sensed normal specific force or maneuver acceleration of the vehicle along the $\hat{k}_b$ axis. To determine the acceleration loading on the vehicle, each of the acceleration components are first normalized to gravity.

$$Nx_v = \frac{\dot{V}}{g} \tag{3. 22}$$

$$Ny_v = \frac{(V \cos\gamma)\,\dot{\chi}}{g} \tag{3. 23}$$

$$Nz_v = -\frac{V\,\dot{\gamma}}{g} \tag{3. 24}$$

Then, the accelerations are rotated from the velocity reference frame into the body reference frame using the transformation matrix in Eq 3.6 to give

$$Nx_b = Nx_v \cos\alpha + Ny_v \sin\alpha \sin\mu - Nz_v \sin\alpha \cos\mu \tag{3. 25}$$

$$Ny_b = Ny_v \cos\mu + Nz_v \sin\mu \tag{3. 26}$$

$$Nz_h = -Nx_v \sin\alpha + Ny_v \cos\alpha \sin\mu - Nz_v \cos\alpha \cos\mu \quad \textbf{(3. 27)}$$

Nz_h undergoes a sign change in order to follow the convention whereby positive Nz_h is up.

The drag and lift forces contained in the equations can be expressed as functions of the dynamic pressure $\overline{q}$, the vehicle planform area S, and the dimensionless coefficients of lift C_L and drag C_D.

$$L = \overline{q} S C_L \quad \textbf{(3. 28)}$$

$$D = \overline{q} S C_D \quad \textbf{(3. 29)}$$

Dynamic pressure can also be written as a function of atmospheric density (which is a function the altitude of the vehicle), and the velocity of the vehicle.

$$\overline{q} = \tfrac{1}{2} \rho V^2 \quad \textbf{(3. 30)}$$

[This page intentionally left blank]

Chapter 4
Guidance and Control

4.1 Overview

The purpose of an RLV's Guidance and Control (G&C) system is to guide the vehicle to a safe runway landing without violating certain constraints. These constraints may consist of thermal, dynamic pressure or acceleration loading limits imposed on each trajectory. In the classical arrangement, the guidance system regulates the vehicle's trajectory and energy, while the flight control system determines the actuator control deflections, based on steering commands from the guidance system. In the case of manned vehicles, including the shuttle, a human pilot typically acts as an intermediary between the guidance and control systems, and interprets cues provided by guidance into commands that are sent to the flight control system. However, future RLV's may be unmanned and will have no pilot available to make up for the limitations in the traditional G&C system layout. These limitations, described in the next section of this chapter, preclude autonomous systems from taking advantage of the full capabilities of the vehicle. Draper Laboratory feels that these limitations can be reduced by the introduction of a new G&C system layout. In order to explain the layout, this chapter first describes the "traditional" shuttle G&C concept and its limitations, and then presents Draper's Next Generation Guidance and Control (NGGC) system.

4.2 Traditional Shuttle Guidance and Control

4.2.1 Guidance and Control Concept

The traditional G&C system concept, including the relationships between various components, is illustrated in Figure 4.1. This concept was originally devised for the Space Shuttle, and due to its proven success, has become the standard for RLVs. Flight planning begins on the ground, by mission planners, who design a series of "reference profiles". These profiles are a set of vehicle states or control histories arranged with respect to some monotonically changing variable, such as time, downrange distance or velocity. The reference profiles are the result of the efforts of engineers to discover, through iterative design and optimization techniques, the nominal and abort trajectories

for a given vehicle mission. A typical profile (or trajectory) design approach used by the engineers is shown in Figure 4.2. Due to any revisions made to a vehicle's aerodynamic characteristics or configuration, and any alterations in expected initial flight conditions, both a nominal trajectory and a series of contingency abort trajectories must be redesigned, and then translated into reference profiles, for each and every flight made by a vehicle. Once the profiles are completed, they are uploaded onto the onboard guidance computers through a large sequence of I-loads, sometimes numbering in the thousands.

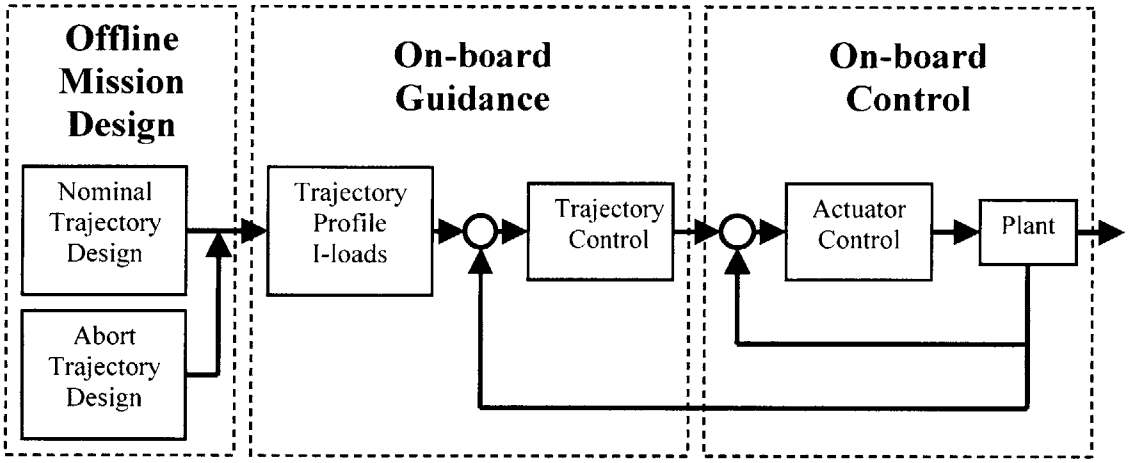


Figure 4.1: Traditional Space Shuttle G&C Layout

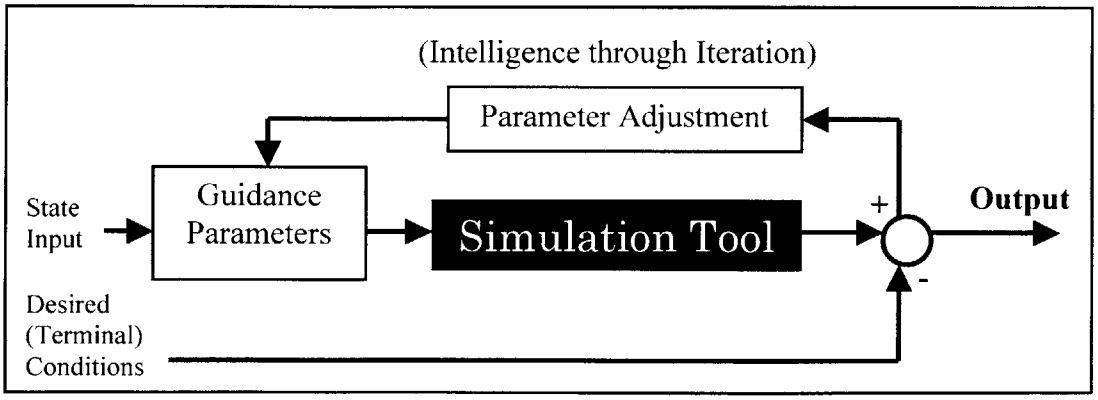


Figure 4.2: Traditional Approach to Trajectory Generation [7]

During a typical flight, the onboard guidance system generates commands that attempt to match the actual vehicle states with the preloaded reference states. These guidance commands are usually for the speedbrake position (δ_{sb}), roll (Φ), and acceleration along the vehicle's negative body $\hat{k}_b$ axis (Nz). Once generated, the commands are fed to an onboard control system that produces a set of deflection commands for the control surface actuators, in the case of an autonomous unmanned vehicle, or a set of pilot cues, in the case of a manned vehicle. It should be noted that in the shuttle-class G&C system, shown in Figure 4.1, the guidance and control systems are artificially partitioned into separate efforts, even though it is a coupled task. Additionally, separate guidance software is required for the nominal flight plan and each abort contingency.

4.2.2 Terminal Area Energy Management and Approach and Landing Phases

Due to the wide variations in conditions that the shuttle, or an RLV, may experience during a typical flight, its entry descent is divided into three sequential phases. This allows the guidance scheme to be broken up into three phases as well, making it more robust than a single scheme developed to handle the entire flight. The three phases are the Entry Phase, the Terminal Area Energy Management (TAEM) Phase, and the Approach and Landing (A/L) Phase. This thesis is only concerned with the flight of RLVs through the TAEM and A/L phases, so only these two areas will be discussed in this sub-section.

The TAEM phase is characterized by glider-type flight dynamics, and is initiated at a specified Mach number and/or altitude where the vehicle attains full control through the use of aerosurfaces only. The purpose of this phase is to control the energy state of the vehicle and direct it towards the landing site. This is accomplished conceptually by flying a predetermined Energy over Weight (E/W) profile as a function of range to the runway, or range-to-go. The E/W term, which captures both the potential and kinetic energies of the vehicle, and thus the total energy, is expressed as:

$$E/W = h + \frac{\bar{q}}{\rho g} \tag{4.1}$$

During the development of the TAEM phase, engineers decided to split the longitudinal and lateral channels, considering a full integration too complex [8]. Thus, the vehicle's angle of attack controls the longitudinal channel, while the bank angle controls the lateral channel.

The independent variable used during TAEM is range-to-go, and consists of two components: downrange distance and crossrange distance. This variable, more properly defined as the distance left along a predicted ground track, is easily calculated using well-defined geometric segments [9]. For this reason, TAEM is further divided into four distinct subphases, shown in Figure 4.3. These subphases, in order of occurrence, are S-turn, Acquisition, Heading Alignment, and Prefinal Approach. The TAEM phase terminates at a point known as the Auto-Landing Interface (ALI), where the A/L phase begins.

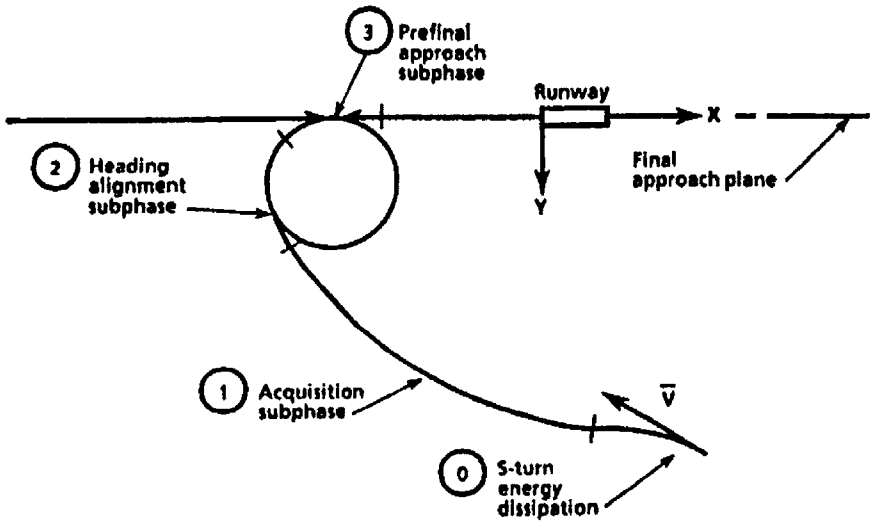


Figure 4.3: TAEM Subphases [9]

The S-turn subphase is used to provide large adjustments to the energy state of the vehicle. This subphase is only executed when the vehicle's energy state is too high to reach the ALI at the specified constraints. In other words, the predicted ground track length is not long enough to allow the excess energy to dissipate before reaching the ALI. In such cases, the guidance system commands the vehicle to turn at a maximum rate

away from the desired heading, while the resulting S-turn lengthens the ground track distance, thus dissipating excess energy. This turn is continued until the current energy state matches the E/W reference profile.

The Acquisition subphase is initiated to steer the vehicle towards a smooth interception with the Heading Alignment Cone (HAC), described in the subsequent paragraph. This phase, shown in Figure 4.4, consists of two segments. The first segment is a constant bank angle turn, approximated by an arc segment of constant radius. This arc is intended to change the vehicle's heading to that of a direction tangent to the HAC. The subphase is completed by connecting the endpoint of this segment to the tangent point on the HAC.

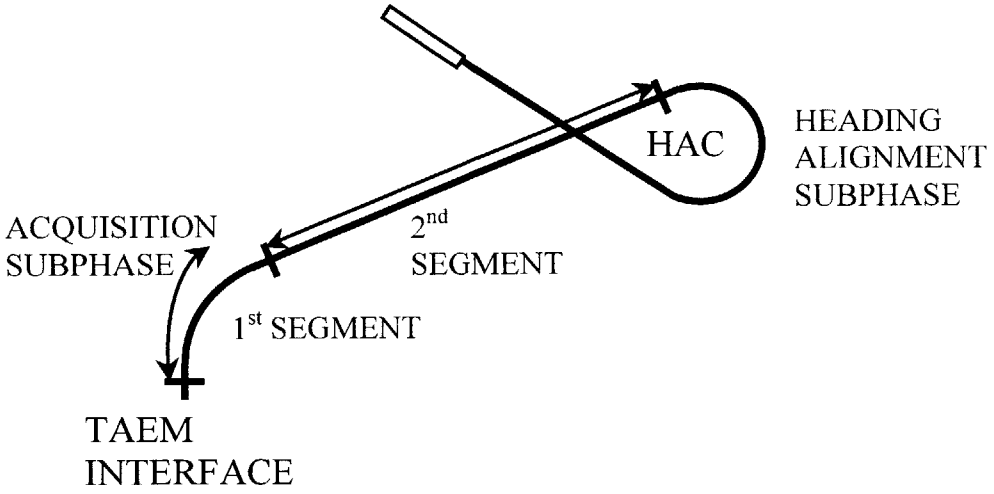


Figure 4.4: Acquisition Subphase Segments

The Heading Alignment subphase is initiated when the vehicle reaches the HAC tangent point. The HAC is an imaginary inverted cone placed uprange of the runway with its surface tangent to an extended projection of the runway centerline. Figure 4.5 provides an illustration of the HAC along with a vehicle's projected flight path. During this subphase, the lateral guidance keeps the vehicle on the HAC surface using radial position and rate errors to generate the necessary bank angle commands. By forcing the vehicle to fly around the outer surface of the HAC, this subphase removes crossrange position and vehicle heading errors prior to entering the Prefinal Approach subphase.

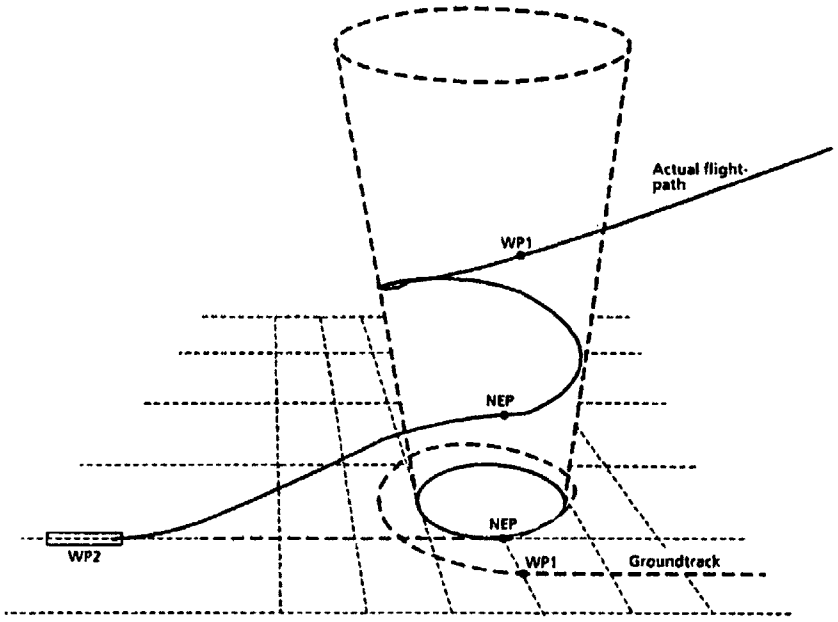


Figure 4.5: Flight Path along the HAC [9]

The last subphase in TAEM is the Prefinal Approach subphase. This is merely another straight-line segment intended to allow any existing crossrange errors to settle before the vehicle transitions from TAEM to the A/L phase. Guidance roll commands in this phase are calculated from any crossrange or crossrange rate errors from the runway centerline.

Upon completion of the TAEM phase at the ALI, the Approach and Landing (A/L) phase is initiated, which guides the vehicle along a predefined velocity and altitude profile to touchdown on the runway [10]. The ALI is usually placed at 10,000 feet and at a predetermined uprange distance, consistent with the particular design of the five autolanding subphases. These subphases consist of an initial steep glide slope to a circular flare, which exponentially decays into a shallow terminal glide slope. A final flare arrests the touchdown sink rate at a desired value and ensures the final vehicle pitch attitude. The A/L phase, including all defining constraints, is shown in Figure 4.6. The vehicle's flight, along with the A/L phase, ends after the termination of vehicle rollout.

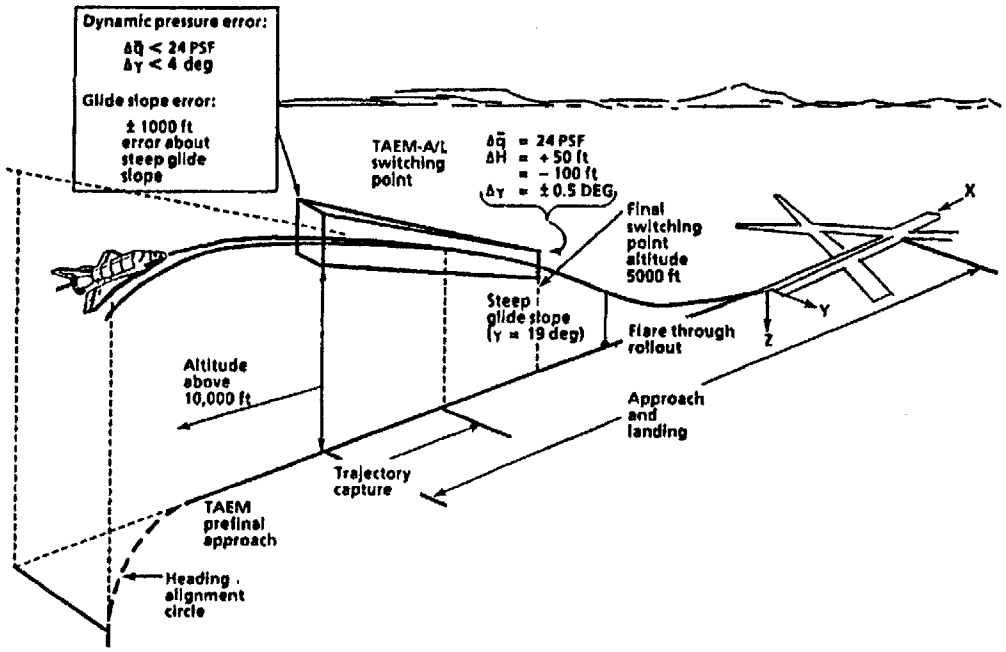


Figure 4.6: Approach and Landing Subphases [9]

4.2.3 Limitations

The traditional G&C concept has several important inefficiencies and limitations. These are the direct result of the fundamental design choices made during the formulation of the original G&C layout. This layout is based upon the use of fixed trajectory profiles that are determined pre-flight and in the uncoupling of the lateral and longitudinal channels throughout the TAEM and A/L guidance phases. In order to realize the potential that exists for improving vehicle performance by addressing the shortcomings of the existing TAEM methodology, it is necessary to understand the limitations of this system.

The use of pre-defined loaded trajectories is a big limitation. This process is very labor intensive and each flight requires a standing army of ground support, which contributes to high operational expenses. The use of preloaded trajectories confines a vehicle to tight flight corridors, which reduce the vehicle's robustness to changing flight conditions, while the use of preplanned abort contingencies effectively prevents the full exploitation of a vehicle's recovery capacity. In addition, no mission planner can devise abort

trajectories for every possible situation. This fact leads to the formation of "gaps" in the guidance coverage over the vehicle's flight path, for which there is no provision to intelligently guide the vehicle to a safe landing. These gaps will always exist for every pre-determined flight profile, and are potentially dangerous. The flight of a vehicle into a gap can result in the loss of the vehicle and possibly the loss of human life.

The geometry used in the formulation of the pre-defined trajectories has two important limitations. First, the Acquisition subphase is calculated by an arc segment of constant radius. This may hold true for small angles of rotation, but larger gliding turns will follow a spiral ground track instead, due to the effect of decreasing velocity and increasing density. In an abort situation, where the initial conditions are radically different from nominal initialization conditions, it may be necessary to make a large acquisition turn in order to fly towards a suitable landing location. With the present formulation, the resulting acquisition turn will lead to an error in the ground track prediction, causing a situation where the guidance system is tracking to an inadequate energy profile.

Second, the Heading Alignment Cone is always placed at a fixed location, uprange from the runway. With the current formulation, there exists the possibility of moving the HAC once during flight, from a distance 7nm from the runway to one 4nm from the runway, in an attempt to compensate for situations when the vehicle is low on energy. However, the use of only two locations removes the ability of fine-tuning the lateral ground track for the present energy situation, while also potentially forcing the vehicle from a state of low energy to state of extremely high energy versus range. This may lead to unwanted excess energy as the vehicle reaches the runway.

Finally, the separation of lateral and longitudinal channels in the guidance algorithms is a limitation on the overall system. The longitudinal guidance uses a wings level energy profile to predict a vehicle's projected ground track length, and thus its range capability. However, this approach completely ignores the large effect that banking has on a vehicle's lift vector, and thus the effect it has on a vehicle's energy and range. For trajectories with large acquisition turns (possible abort scenarios), this may lead to an incorrect determination of the overall ground track length, and the possibility of uncorrectable trajectory errors.

4.3 Next Generation Guidance and Control (NGGC)

4.3.1 Guidance and Control Concept

The goal of Draper's NGGC system is to significantly increase flight vehicle safety and reduce life cycle costs while gaining increased operation and performance capabilities for RLVs. This is the result of formulating a new G&C framework, which seeks to improve upon the limitations of the traditional G&C system concept. The NGGC overall system concept is shown in Figure 4.7. This concept is made up of three key components: an onboard autonomous abort planner, an onboard trajectory generator, and an Integrated Guidance and Control (IG&C) framework.

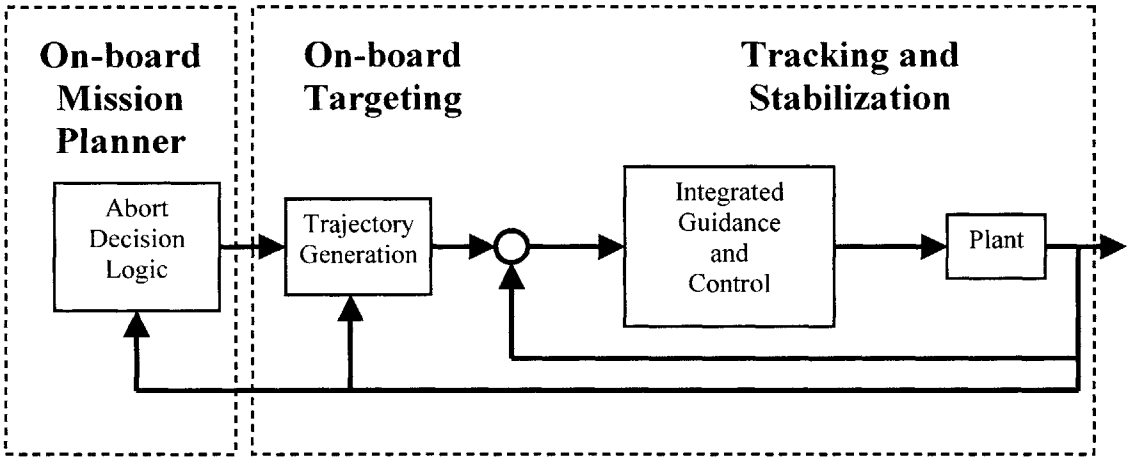


Figure 4.7: Next Generation Guidance and Control System Concept

4.3.2 Components

The abort decision planner would enhance the capability of current RLVs by providing a continuous, real-time, autonomous means of recovering the vehicle intact in the event of any major failure or off-nominal conditions. This NGGC component works by analyzing the current vehicle states to determine whether or not the vehicle can reach the nominal target landing site. If not, it would be able to select an alternative landing site based upon energy and vehicle constraints. In the event of those rare occasions when no emergency landing areas are available, the abort planner would direct the vehicle away from populated centers, improving overall system safety. The abort system presents whatever

flight plan is determined to the onboard trajectory generator, which actually targets the intended landing site and calculates the necessary reference trajectory.

The use of an onboard trajectory generator eliminates the dependency on pre-defined, preloaded trajectories. This, in turn, lowers recurring operational costs by removing the labor-intensive trajectory design process of each new flight for a one-time algorithm certification effort. The use of actual rather than predicted flight states grants flexibility to each trajectory and expands the vehicle's flight corridors. In addition, an onboard generator creates up-to-date profiles, which maximize robustness. In the case of an abort situation, an instantaneous assessment of current vehicle flight conditions allows for successful recovery trajectories to be produced for unanticipated cases, while still taking advantage of the full capabilities of the vehicle.

Draper is pursuing an onboard trajectory generator that follows a different approach from the conventional "brute-force" trajectory generators currently used to calculate the shuttle trajectories. The approach chosen is the use of a real-time design tool, which utilizes embedded intelligence and knowledge of key system dynamic tendencies as decision aids in the formulation of a trajectory. This design approach is shown in Figure 4.8. The next chapter covers in detail the fundamental layout as well as the operational and design background of Draper's current onboard trajectory generator.

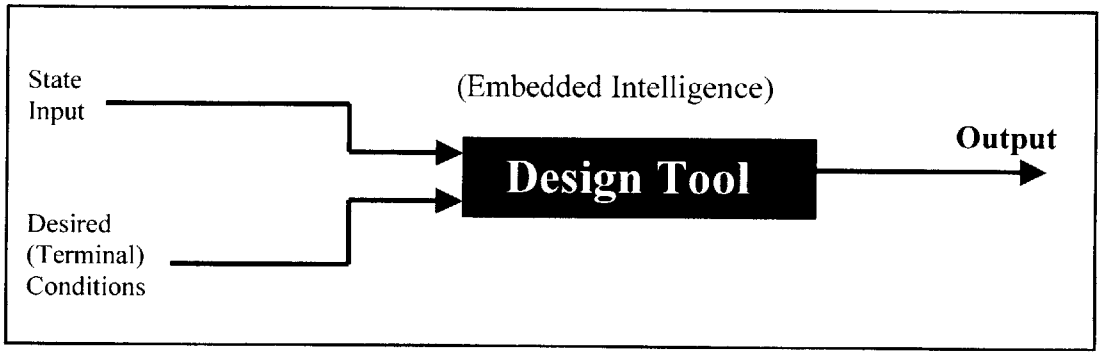


Figure 4.8: NGGC Approach to Onboard Trajectory Generation [7]

Future advances in onboard trajectory generation will not be fully realized without the creation of an IG&C system. This system is designed to perform over a wider range of conditions throughout the flight envelope, as compared to traditional G&C systems. Central to the design is the ability of the system to reconfigure to vehicle failures (such as a stuck elevon, or locked rudder) and to adapt to changing flight conditions as mandated

by the onboard trajectory generator. The adaptive nature of the control system will lead to automatic tuning of an optimal controller for various flight conditions. In addition, the tracking and stability control loops are integrated into a single process that optimizes the trajectory performance while accommodating the actuator control stability constraints.

With the development of each of these components, it is hoped that the resulting NGGC system will enable the performance of the vehicle to be only constrained by its physical design limitations and not the guidance and control algorithms.

[This page intentionally left blank]

Chapter 5
Onboard Trajectory Generation

5.1 Overview

Draper Laboratory's vision for a Next Generation Guidance and Control system is based on the ability to design trajectories with an onboard trajectory generator. Traditional trajectory design approaches are very time intensive and would not lend themselves to real time implementation. Draper has recently developed a new approach that could potentially form the core technology for an onboard system. This method uses a new dynamic procedure, known as the Kernel Extraction Protocol (KEP), where the trajectory dynamics are extracted from the non-linear kernel dynamic equations of motion. The use of KEP enables the near real time generation of trajectories, including all the corresponding flight dynamics and control histories for online optimization within the flight control system. This chapter presents the basic methodology behind the Kernel Extraction Protocol and its formulation. It begins with the derivations of the kernel equations of motion and the fundamentals of the protocol. The following sections present application and expansion of the KEP technology with regards to trajectory generation, while the last sections describe the status of Draper's onboard trajectory generator.

5.2 Kernel Extraction Protocol (KEP)

5.2.1 Kernel Equations of Motion

The conventional method of generating a trajectory involves integrating the equations of motion in the time domain in order to propagate the trajectory forward in space. This traditionally requires an iterative approach on adjusting the time-history control reference so that the resulting trajectory satisfies the design goals. However, a more efficient method, using the Kernel Extraction Protocol, designs the trajectory geometry based on desired characteristics, and then extracts the required dynamics and control history to fly that trajectory. The kernel equations of motion are essentially rearrangements of the equations of motion derived in Chapter 3, but differ by replacing the traditional guidance state (velocity), and the independent variable (time), with dynamic pressure and altitude.

The use of dynamic pressure ($\bar{q}$), as the dynamic variable, or guidance state, has two important benefits. First, $\bar{q}$ is a more slowly varying parameter than velocity, due to the fact that on an entry trajectory, the increasing density partially offsets the decreasing velocity [11]. Also, dynamic pressure is near constant at equilibrium conditions, and it experiences practically linear variations over small changes in altitude. These $\bar{q}$ characteristics enhance the stability and robustness when propagating the nonlinear kernel equations of motion. Second, many of the dynamic constraints imposed on trajectories are imposed directly on $\bar{q}$ and indirectly on vehicle velocity. These can be dynamic pressure limits at the ALI target conditions, or actual vehicle loading limits, which are functions of dynamic pressure. Additionally, controlling dynamic pressure still implies controlling velocity because $\bar{q}$ is directly related to velocity.

Changing the independent variable from time to altitude provides an easier and more convenient means of designing trajectories. Because KEP does not rely on the time integration of the equations of motion to generate a trajectory, the time history of the trajectory is not important from a design standpoint. Therefore, the independent variable can be any monotonically changing quantity. The choice of altitude allows the designing of trajectories based on predetermined altitude steps, which is a convenient method of handling gliding reentry trajectories. Also, because density is a function of altitude, it would be more difficult to characterize the change in density with respect to any other independent variable, such as downrange distance x, or time. It should be noted that the time history is not lost, since it too can be extracted from the KEP solution.

The formulations of the kernel equations begin with the equations of motion derived in Chapter 3 and reproduced here for ease of reference.

$$\dot{V} = \frac{-D}{m} - g\sin\gamma \tag{5.1}$$

$$\dot{\gamma} = \frac{1}{mV}\left[L\cos\mu - W\cos\gamma\right] \tag{5.2}$$

$$\dot{\chi} = \frac{L\sin\mu}{mV\cos\gamma} \tag{5.3}$$

Once again, the aerodynamic forces can be represented as functions of dynamic pressure, which is used to provide a more well-behaved iteration on the above equations. Dynamic pressure is defined as:

$$\bar{q} = \tfrac{1}{2}\rho V^2 \tag{5. 4}$$

Converting Eq 5.1 into a form governing the change in dynamic pressure begins with differentiating Eq 5.4 with respect to time.

$$\dot{\bar{q}} = \tfrac{1}{2}\dot{\rho}V^2 + \rho V\dot{V} \tag{5. 5}$$

The next step involves rewriting $\dot{\rho}$ as a function of altitude, using the chain rule:

$$\dot{\rho} = \left(\frac{d\rho}{dh}\right)\left(\frac{dh}{dt}\right) \tag{5. 6}$$

where $d\rho/dh$ is a known quantity of an atmospheric model, easily extracted from tabular data, and dh/dt is the vertical component of the derivative of the vehicle position vector given by:

$$\left(\frac{dh}{dt}\right) = \dot{h} = V\sin\gamma \tag{5. 7}$$

Recall the definition of the drag force from Chapter 3:

$$D = \bar{q}SC_D \tag{5. 8}$$

Now, substituting Eqs 5.1, 5.6 and 5.8 into Eq 5.5 and simplifying in respect to $\dot{h}$, yields an equation for the change in dynamic pressure with respect to time,

$$\dot{\bar{q}} = \dot{h}\left[\left(\frac{1}{\rho}\frac{d\rho}{dh} - \frac{\rho SC_D}{m\sin\gamma}\right)\bar{q} - \rho g\right] \tag{5. 9}$$

Next, the chain rule is used to replace time with altitude as the independent variable. This simplifies to just dividing by $\dot{h}$.

$$\frac{d(\ )}{dh} = \frac{d(\ )}{dt}\frac{dt}{dh} = \frac{d(\ )}{dt}\frac{1}{\dot{h}} \tag{5. 10}$$

After applying this change, and rearranging Eq 5.2 into an equation for dynamic pressure, Eqs 5.9, 5.2 and 5.3 can be written as:

$$\frac{d\bar{q}}{dh} = \bar{q}\left(\frac{1}{\rho}\frac{d\rho}{dh} - \frac{\rho S C_D}{m \sin\gamma}\right) - \rho g \tag{5. 11}$$

$$\bar{q} = \frac{W\cos\gamma}{SC_L\cos\mu - 2m\sin\gamma\left(\frac{1}{\rho}\frac{d\gamma}{dh}\right)} \tag{5. 12}$$

$$\frac{d\chi}{dh} = \frac{\rho S C_L \sin\mu}{m\sin 2\gamma} \tag{5. 13}$$

These are the kernel (or core) equations of motion used by the Kernel Extraction Protocol. Also, the advantage of using altitude as the independent variable is now apparent, because $\gamma(h)$ and $^{d\gamma}/_{dh}(h)$ can be expressed in closed-form based on the geometry of a given trajectory, where

$$\gamma(h) = \text{atan}\left(\frac{dh}{dx}\right) \tag{5. 14}$$

It should be noted that the $\bar{q}$ equation is a reformulation of the $\dot{\gamma}$ equation and establishes the lift for a given bank angle required to stay on the longitudinal profile of a trajectory at any given altitude. The $d\bar{q}/dh$ equation is a reformulation of the $\dot{V}$ equation and establishes the required drag to stay on the energy profile of a trajectory and the $d\chi/dh$ equation defines the bank angle necessary to maintain the desired heading [12].

5.2.2 KEP Methodology

The Kernel Extraction Protocol is a rapid method for solving the kernel equations of motion derived above. The equations already have the geometrical constraints built into the dynamics, so a trajectory is created by substituting the actual values for the geometry, such as γ, $d\gamma/dh$, and $d\chi/dh$, into the equations and then "balancing the dynamics" to satisfy the physics of flight mechanics to stay on the reference profiles. The Kernel Extraction Protocol is this process of "balancing the dynamics", which is explained more explicitly in the subsequent example. Propagating and balancing the three kernel equations as the vehicle descends along a trajectory yields an altitude-correlated solution.

The actual technical process of KEP is quite simple. The best way to describe it is with a simple trajectory design case: a wings level, straight trajectory that follows a constant glide slope. In this case, the following is true,

$$\gamma(h) = \gamma_1 = \text{constant} \tag{5. 15}$$

$$\mu = 0 \quad \text{and} \quad \frac{d\gamma}{dh} = 0 \tag{5. 16}$$

Therefore, the kernel equations of motion simplify to

$$\frac{d\overline{q}}{dh} = \left[\overline{q}\left(\frac{1}{\rho}\frac{d\rho}{dh} - \frac{\rho S C_D}{m \sin\gamma_1}\right) - \rho g\right] = \mathrm{f}_1(\overline{q}, \alpha, h) \tag{5. 17}$$

$$\overline{q} = \frac{W\cos\gamma_1}{SC_L} = \mathrm{f}_2(\alpha, h) \tag{5. 18}$$

$$\frac{d\chi}{dh} = 0 \tag{5. 19}$$

After simplifying the equations for the given geometry, it is apparent that the system complexity has been reduced to two equations of motion, $d\overline{q}/dh$ and $\overline{q}$, and one control, α. The angle of attack controls the lift and drag coefficients of the vehicle and hence Eqs 5.17 and 5.18. Also note, that the two equations are actually correlated so that the only state remaining is dynamic pressure. However, even though the problem is simplified, there is no closed-form solution to balancing the dynamics and completing the

trajectory design due to the fact that the equations are non-linear and the aerodynamic coefficients come from actual vehicle aerodynamic tables. Therefore, the Kernel Extraction Protocol is based upon an iterative technique where the angle of attack is adjusted such that the resulting aerodynamic properties balance the required lift to stay on the trajectory ($\bar{q}$ equation) with the proper drag to maintain the required energy state ($d\bar{q}/dh$ equation). This process is sometimes referred to as "balancing the dynamics", which is in fact just the balancing of the $\bar{q}$ equation, which gives the current state dynamic pressure based upon present conditions, with an estimated $\bar{q}$, which comes from an integration of the $d\bar{q}/dh$ equation.

The iteration of the trajectory begins at the top of the trajectory, at a point where the designer chooses. For example, this may be the airdrop location for a free-fall test vehicle, or the point in the space shuttle's trajectory where an abort is initiated and a new trajectory is needed. Either way, the initial altitude and all of the vehicle states are known. The derivative of $\bar{q}$ at the initial altitude is easily obtained from Eq 5.17. To solve for the angle of attack and the dynamic pressure at the next integration step, $h+\Delta h$, we first guess a value for α. This gives two estimates for $\bar{q}$ at altitude $h+\Delta h$, one calculated from the differential equation, $\bar{q}_{diff}$, and one calculated from Eq 5.18, $\bar{q}_{alg}$. The quantity $\bar{q}_{diff}$ is obtained by using an improved Euler numerical integration technique.

$$\bar{q}(h+\Delta h)_{alg} = f_2\left(\alpha(h+\Delta h), h+\Delta h\right) \tag{5.20}$$

$$\bar{q}(h+\Delta h)_{diff} = \bar{q}(h) + \frac{1}{2}\left(\frac{d\bar{q}}{dh}(h) + \frac{d\bar{q}}{dh}(h+\Delta h)\right) \tag{5.21}$$

where $$\frac{d\bar{q}}{dh}(h+\Delta h) = f_1\left(\bar{q}(h+\Delta h)_{alg}, \alpha(h+\Delta h), h+\Delta h\right).$$

The estimates $\bar{q}_{diff}$ and $\bar{q}_{alg}$ are compared to determine if the correct angle of attack was chosen to keep the vehicle on the specified geometry.

$$\bar{q}_{err} = \bar{q}_{alg} - \bar{q}_{diff} \tag{5.22}$$

If the estimates for dynamic pressure do not match, then $\alpha(h+\Delta h)$ is varied by a secant root finding method until they do match as shown below. Figure 5.1 also demonstrates this principle.

$$\alpha(h+\Delta h) = \alpha(h) + \overline{q}_{err} k_{\Delta\alpha} \tag{5. 23}$$

where $k_{\Delta\alpha}$ is an adaptive gain that provides an incremental change in angle of attack.

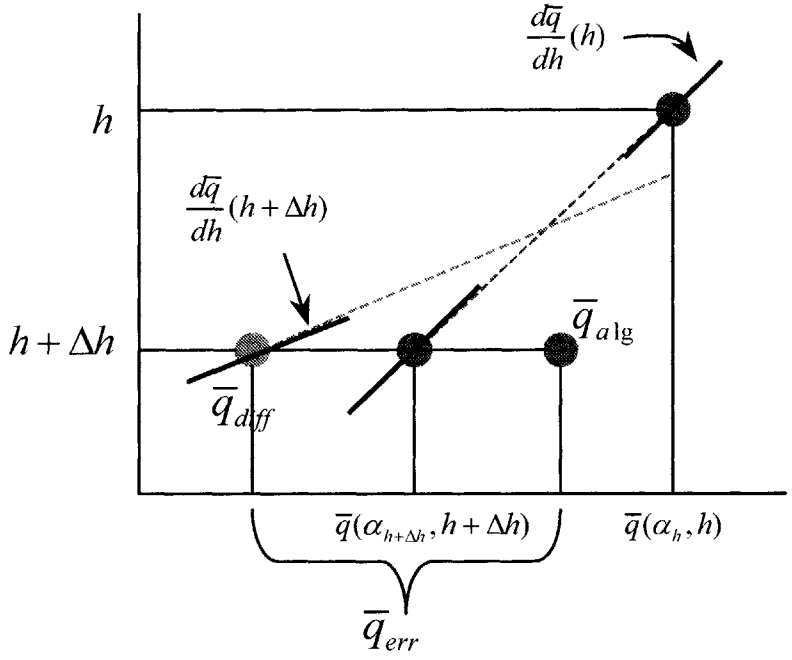


Figure 5.1: "Balancing the Dynamics" through Dynamic Pressure

Once the values for the dynamic pressures converge to the specified tolerance, the solution for $\alpha(h+\Delta h)$ and $\overline{q}(h+\Delta h)$ is complete and the process is repeated for every altitude step until ending at the specified target altitude. The vehicle control history, i.e. the elevon angle values, is determined by trimming the vehicle at every altitude step. This consists of adjusting the elevon to zero the pitching moments of the vehicle at each calculated angle of attack.

The process described above is the same for three-dimensional trajectories as well. In those cases however, there are two controllers, angle of attack and bank angle. The

solution technique typically involves using multiple nested loops to balance the three kernel equations of motion, unless some other clever rearrangement is used to simplify the equations. While the three-dimensional case may be a little more complicated, it follows the same process described above to balance the dynamics and obtain a solution for the states and controls.

The Kernel Extraction Protocol described here provides a simple, fast method for designing trajectories. It does not require, nor is it sensitive to, small changes in the control history vital to standard time-integrated shooting trajectory generators. This enables KEP to design trajectories potentially in real time, since the trajectory control history is correlated to the geometry at each altitude increment. Also, the protocol appears to be robust in this domain. If the profile under consideration is not within the capabilities of the vehicle, then the equations will not balance and no solution will be found. However, the potential for having no solution is mitigated by constraining the geometry or resulting $\bar{q}$ profiles to well-behaved, simple shapes. Therefore, every trajectory generated by KEP is intrinsically flyable.

Draper has devised two methods for using KEP in a trajectory generator. The first method, which is restricted to the subsonic region, adjusts a fixed number of geometric shapes and segments linked together to solve a two-point boundary value problem (TPBVP). The second method, which is used for the entire flight space (supersonic and subsonic regions), continuously adjusts the geometry to follow a specified energy profile. These two versions of the trajectory generator are covered in the following sections.

5.3 Geometrically Constrained Methods

5.3.1 Definition

Geometrically Constrained Methods refer to those trajectory designs that are constructed around pre-defined geometric shapes and segments. The geometric shapes may consist of straight-line constant glide slopes, circular or exponential flares, or even circular turns and vertical conical spirals. These shapes help to simplify the kernel equations of motion and are useful in reducing the order of the system to be solved, as discussed in the previous section.

5.3.2 Auto-Landing I-load Program (ALIP)

The Auto-Landing I-load Program (ALIP) was developed by G. Barton of Draper Laboratory as a pre-mission design tool for the X-34 project [11]. This program rapidly generates unpowered autolanding reference trajectories from the Auto-Landing Interface (ALI) at 10,000 feet to touchdown. It also generates the Mission Data Loads (I-loads), consisting of the state and control effector references, for input within the shuttle-class algorithms used by the X-34, and predicts the flight behavior while flying the reference trajectory. The primary motivation for the development of ALIP was to simplify the time-intensive conventional approach of designing autolanding trajectories by automating the process. This was accomplished by the use of KEP.

The autolanding (A/L) trajectory profile used by the X-34 is identical to the reference used by the Space Shuttle. This flight portion occurs on final approach, when the shuttle is flying wings level and its heading is aligned with the runway. The A/L profile is made up of well-defined geometric segments or subphases, consisting of a steep glide slope, circular flare, exponential decay, shallow glide slope, and final flare. These subphases are shown in Figure 5.2. It should be noted that a vehicle flying through the A/L flight phase is traveling subsonically and it is essentially restricted to longitudinal motion, i.e. pitching motion only.

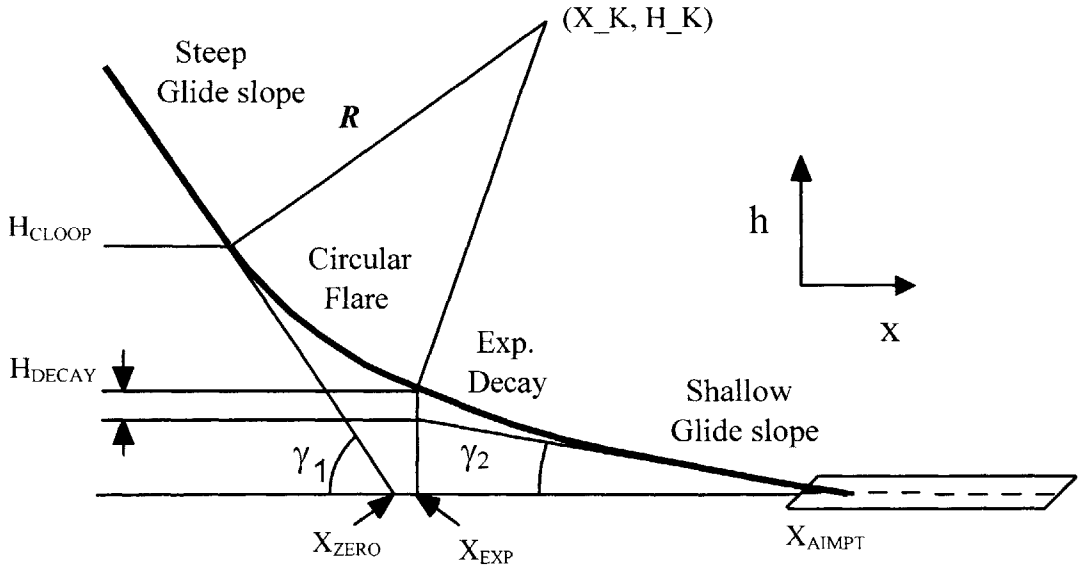


Figure 5.2: Auto-Landing Subphases and Geometry [11]

With the geometry well-defined, and enforcing physical constraints such as loads, vertical descent rate, continuity and smoothness, the design problem reduces to a two-point boundary value problem, with conditions on the initial and final dynamic pressure. The kernel equations simplify to Eqs 5.17, 5.18, 5.19, just like the simple example used to describe KEP. The substitutions for the geometry into the kernel equations are as follows:

-Steep / shallow glide slope

$$\frac{d\gamma}{dh} = 0 \tag{5. 24}$$

-Circular flare

$$\frac{d\gamma}{dh} = \frac{1}{(\mathrm{H\_K} - h)\tan\gamma} \tag{5. 25}$$

-Exponential decay

$$\frac{d\gamma}{dh} = \frac{\mathrm{H_{DECAY}}\cos^2\gamma}{\sigma^2\tan\gamma}\exp\left[\frac{\mathrm{X_{EXP}} - x}{\sigma}\right] \tag{5. 26}$$

where σ is the decay rate of the exponential flare.

In order to solve the TPBVP between the initial and final dynamic pressures, ALIP assumes an initial value for X_{ZERO} and uses KEP to design a trajectory from the ALI to the runway using each of the geometric segments. The variable X_{ZERO} represents the outer glide slope intercept with the landing altitude. If the touchdown $\bar{q}$ does not fall within tolerances, the program adjusts X_{ZERO}, sliding it either towards or away from the runway, until the trajectory satisfies the constraints (Figure 5.3). In this regard, ALIP uses the initial downrange state to remove landing energy errors at the final state, but in the process designs several complete trajectories before finding a solution. This method is sometimes referred to as a "shooting method", since it "shoots" several trajectories until it satisfies all the constraints. Also note that because ALIP adjusts X_{ZERO} to solve the TPBVP, it is solving a problem of unconstrained range, because the initial vehicle position or starting point is not fixed.

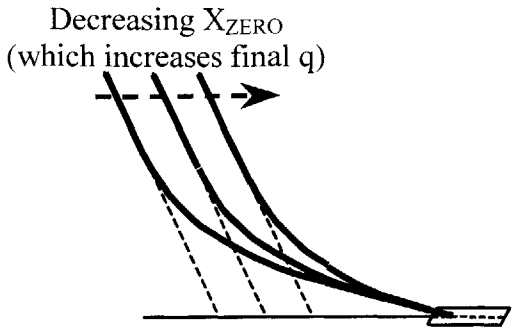


Figure 5.3: Effect of Adjusting X_{ZERO} [11]

5.3.3 ALIP3D

Draper Laboratory chose to expand the capabilities of ALIP and KEP by extending it into the subsonic portion of the TAEM flight regime, consistent with the expected X-34 unpowered drop tests. This led to the development of ALIP3D, designed by A. Girerd, which is covered in an MIT thesis entitled *Onboard Trajectory Generation for the Unpowered Landing of Autonomous Reusable Launch Vehicles* [13]. This trajectory generator has many similarities with ALIP, including designing subsonic flight trajectories around predefined geometric shapes derived from the Space Shuttle's guidance scheme. However, ALIP3D uses the full three-dimensional equations of motion, and successfully couples the lateral and longitudinal flight dynamics into the resulting trajectories.

The entire flight geometry is designed from the subsonic portion of the shuttle's TAEM region. Four geometric segments comprise the elements of the lateral flight portion and are shown in Figure 5.4. The first segment, the circular acquisition turn, changes the vehicle's heading towards tangency to the Heading Alignment Cone (HAC). The second segment is a straight-line tangent to both the acquisition turn and the HAC. The third segment is defined as the distance traveled along the HAC spiral and the fourth segment is a straight line from the HAC to the ALI. The longitudinal flight portion generally consists of three segments, which includes an initial straight-line glide slope to a flare/anti-flare, and ends with a final glide slope to the ALI. An anti-flare is defined as a dive or constant increase in the magnitude of γ with respect to altitude. The longitudinal geometry is shown in Figure 5.5.

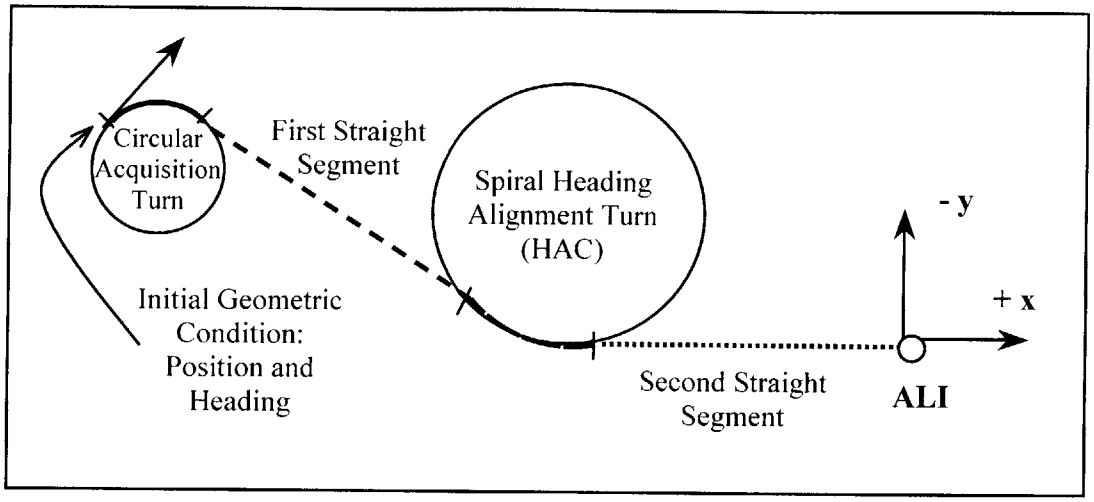


Figure 5.4: Elements of Lateral Geometry [13]

With the addition of the lateral geometry, the KEP methodology must make use of all three kernel equations of motion. However, a clever rearrangement of the $d\chi/dh$ can be used to reduce the system back down to two equations. After substituting the longitudinal geometry (γ and $d\gamma/dh$) into the kernel equations and rewriting, the system to be solved consists of:

$$\frac{d\overline{q}}{dh} = \mathrm{f}_1(\overline{q}, \alpha, h) \tag{5. 27}$$

$$\overline{q} = \mathrm{f}_2(\alpha, h, \mu) \tag{5. 28}$$

$$\frac{d\chi}{dh} = \mathrm{f}_3(\alpha, h, \mu) \tag{5. 29}$$

The $d\chi/dh$ equation is then rewritten by solving for bank angle μ:

$$\mu = \sin^{-1}\left[\frac{d\chi}{dh}\frac{m\sin 2\gamma}{\rho S C_L}\right] = f_4(\alpha, h) \tag{5. 30}$$

Therefore, because the bank angle is a function of the other variables in Eq 5.28, Eq 5.28 can be condensed into:

$$\overline{q} = f_2(\alpha, h) \tag{5.31}$$

With all of the geometry substituted into the kernel equations, it can be seen that the problem has been simplified back down to two equations (Eqs 5.27 and 5.31). Once again, KEP is used to balance the dynamic pressure equations through the use of angle of attack.

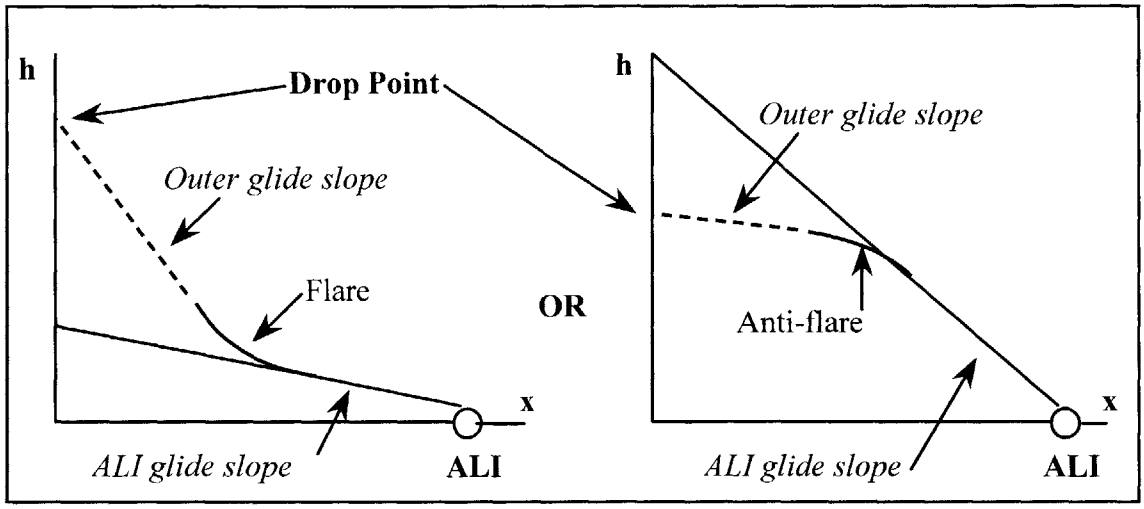


Figure 5.5: Elements of Longitudinal Geometry [13]

In general, ALIP3D begins the trajectory design by using the initial vehicle dynamic state and position to determine the location and radius of the acquisition turn. The minimum turn radius is defined by the maximum allowable Nz_b. The maximum turn radius is determined by calculating the largest acquisition circle that still allows tangency to the HAC. ALIP3D can be programmed to choose a radius based on user specifications, so long as it lies in between the two requirements described above. Once this segment has been established, the program lays out the remaining ground track segments, including the pre-positioned HAC, and calculates the bank angles necessary to follow this lateral geometry. It is important to reiterate that the radius of the acquisition circle, and thus the maximum turning capability of the vehicle at the trajectory initiation, is determined

solely on the initial vehicle energy state. Designing the lateral portion first enables ALIP3D to predict the effect of the lateral maneuvers on the longitudinal profile.

The program continues with the design of the longitudinal flight path following the geometry of Figure 5.5. The altitude flight space for these trajectories is usually between 40,000 ft and the ALI (10,000 ft). Once the lateral and longitudinal profiles have been finalized, KEP is used to propagate the states and controls and complete the trajectory design.

5.3.4 Limitations

Although ALIP and ALIP3D rapidly calculate flyable trajectories, neither trajectory generator is well suited for onboard implementation. First, the use of pre-defined fixed geometry in the calculation is undesirable because it takes away from the main design feature of an onboard trajectory generator, that is, the ability to design for abort scenarios. These scenarios are anticipated to occur during off-nominal conditions, so the use of nominal pre-defined geometric profiles is not feasible. While using fixed geometry simplifies the kernel equations of motion, it also reduces the possible number of trajectories that can be designed by forcing them to contort to those geometrical constraints. For example, one can clearly understand the difficulty of generating an abort trajectory around off-nominal conditions while still trying to make it to a pre-defined HAC, which is designed for certain, *nominal* conditions. Also, ALIP and ALIP3D only design trajectories feasible in the subsonic regime. This is a limiting feature considering that this regime makes up the smallest portion of an RLV's gliding flight. Additional limitations are program specific.

ALIP's use of a shooting method that uses X_{ZERO} as the design parameter forces ALIP to solve a TPBVP of unconstrained range. This precludes ALIP's use as an onboard trajectory generator. In reality, an onboard system must design a trajectory using the current vehicle states, including its exact position. Therefore, the TPBVP to be solved by an onboard trajectory generator is necessarily constrained in range. However, ALIP was designed as a pre-flight design tool and not as an onboard system. It calculates the optimum approach to touchdown for a given set of design conditions, and does it very well. Additionally, it is unlikely that A/L trajectories will ever need to be generated in real time. As a vehicle passes through 10,000 ft during an approach to a runway, it is improbable that it will be able to choose a different landing spot and design a new

trajectory at that moment. Therefore, it is safer, and more convenient to design trajectories to a predefined ALI point and blend it in to an A/L trajectory created by ALIP. ALIP3D follows this approach and tries to adapt ALIP methodologies into a working onboard system.

ALIP3D correctly designs trajectories correlated to the initial states and exact position of the vehicle, but it presets the radius of the circular acquisition turn by an evaluation of the initial vehicle velocity. This limits the capabilities of the program by reducing the solution space. For instance, a vehicle at subsonic speeds possesses a much tighter turning capability than at supersonic velocities. If the radius of the acquisition turn is formulated around an initial velocity of Mach 3.5, the resulting turning maneuver would have a girth greater than the total distance to the HAC and the program would fail to find a solution (Figure 5.6). However, a solution might be possible if a smaller acquisition turn is used later, after the vehicle loses velocity. In this respect, ALIP3D does not account for the increase in maneuverability with decrease in flight speed. This severely limits its application in the supersonic flight regime.

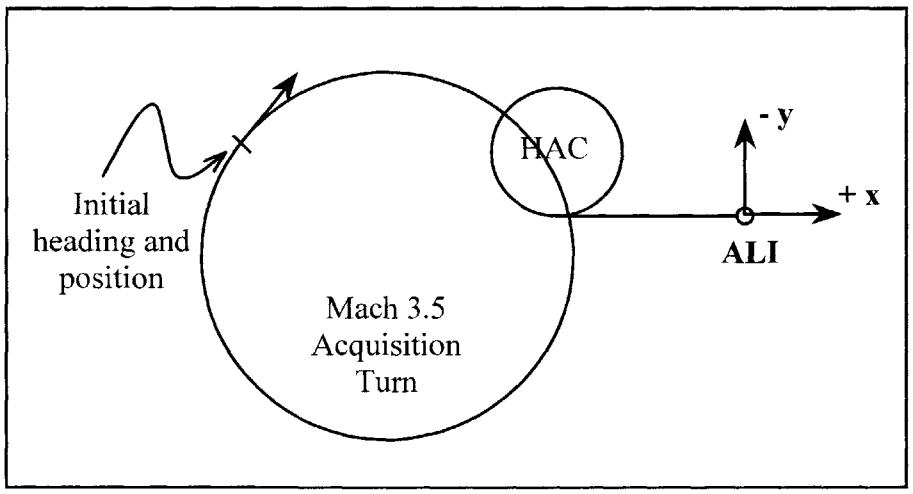


Figure 5.6: Possible High Mach Number Acquisition Turn [1]

5.4 Sub-optimal Nodal Application of the Kernel Extraction (SNAKE) Method

5.4.1 Overview

To specifically address the supersonic flight regime design challenges, A. Grubler developed the Sub-optimal Nodal Application of the Kernel Extraction (SNAKE) method. This method does not rely on pre-defined fixed geometry and is valid over the entire TAEM region, including high altitudes and supersonic flight speeds. The SNAKE method generates trajectories in three distinct steps, covered in the following sections. Step 1): is the design of a dynamic pressure schedule, from which the longitudinal flight profile of the vehicle is derived. Step 2): is a formulation of the vehicle's ground track, or lateral flight profile, by means of a real-time ground track solver. And Step 3): is the use of KEP to balance the dynamics around the geometry calculated in the first two steps and to complete the trajectory design.

Note: All SNAKE trajectories are designed to fly to the ALI, and utilize an ALI-relative coordinate system, as shown in Figure 5.7. The origin lies at the ALI transition point of X_{ALI} downrange, zero crossrange, and 10,000 ft altitude. The positive x-axis points in the direction of the runway, while the positive y-axis points perpendicularly to the right of the runway. Additionally, the heading angle χ is equal to zero when it is aligned with the positive x-axis. This reference system is used in the remainder of this thesis.

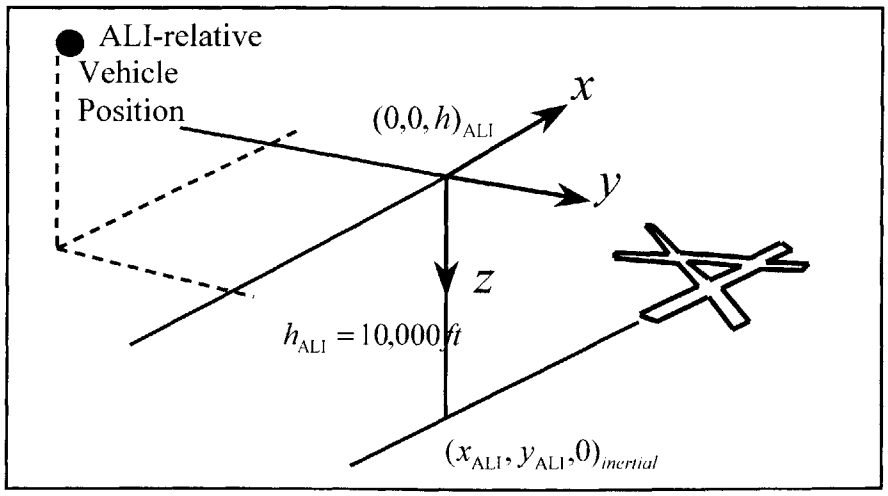


Figure 5.7: ALI-relative Coordinate System [1]

A. Grubler's work is covered in an MIT thesis entitled *New Methodologies for Onboard Generation of Terminal Area Energy Management Trajectories for Autonomous Reusable Launch Vehicles,* and can be referred to for a more detailed explanation of the technical process of the SNAKE method [1].

5.4.2 Dynamic Pressure Schedule

All trajectories that are designed to end at the ALI are required to meet the dynamic and geometric constraints mentioned in Chapter 4. This includes the dynamic pressure constraint of 335-psf. ALIP designs the autolanding outer glide slope so that the vehicle experiences a state of near-static equilibrium, or nearly constant $\overline{q}$, as it descends. In reality, a vehicle flying this constant glide slope will experience slight variations in dynamic pressure due to density and velocity changes. For this reason, the unique glide slope has been termed the quasi-static equilibrium glide slope. However, this glide slope is only valid at lower altitudes and velocities. In order to maintain a constant $\overline{q}$ or follow a $\overline{q}$ schedule in the design space utilized by SNAKE, the vehicle must hold to a continuously varying flight path as it descends through TAEM. This results in the alteration of the trajectory from pre-defined shapes, such as glide slopes. Therefore, the SNAKE method utilizes a $\overline{q}$ schedule or profile to determine the longitudinal shape of the trajectory, which consists of continuously varying geometry.

A $\overline{q}$ schedule is a pre-defined change in dynamic pressure with a given altitude. A typical schedule may consist of an unvarying or constant $\overline{q}$ profile, or a schedule that varies through different $\overline{q}$ values before settling in on the ALI target $\overline{q}$ condition (Figure 5.8). In general, the nature of these schedules is arbitrary and left up to the choice of the trajectory designer, although the focus of this thesis may guide in the selection of these schedules. The only major concerns are that the changes in dynamic pressure between altitude steps remain physically realistic and that the schedule presents a path to the constraint $\overline{q}$ value at ALI. Reference 1 presents a method for using a quadratic $d\overline{q}/dh$ form to ensure that the schedule moves from the initial to final value smoothly, with no discontinuities or dynamic constraint violations. Additionally, the $\overline{q}$ schedule chosen must also fall within the vehicle energy corridor. This corridor contains the range of possible dynamic pressures with which the vehicle can maintain flight, and represents the physical range limits for the vehicle. The edges of the energy corridor are dubbed the Max Dive (MD) and the Max Glide (MG) limits.

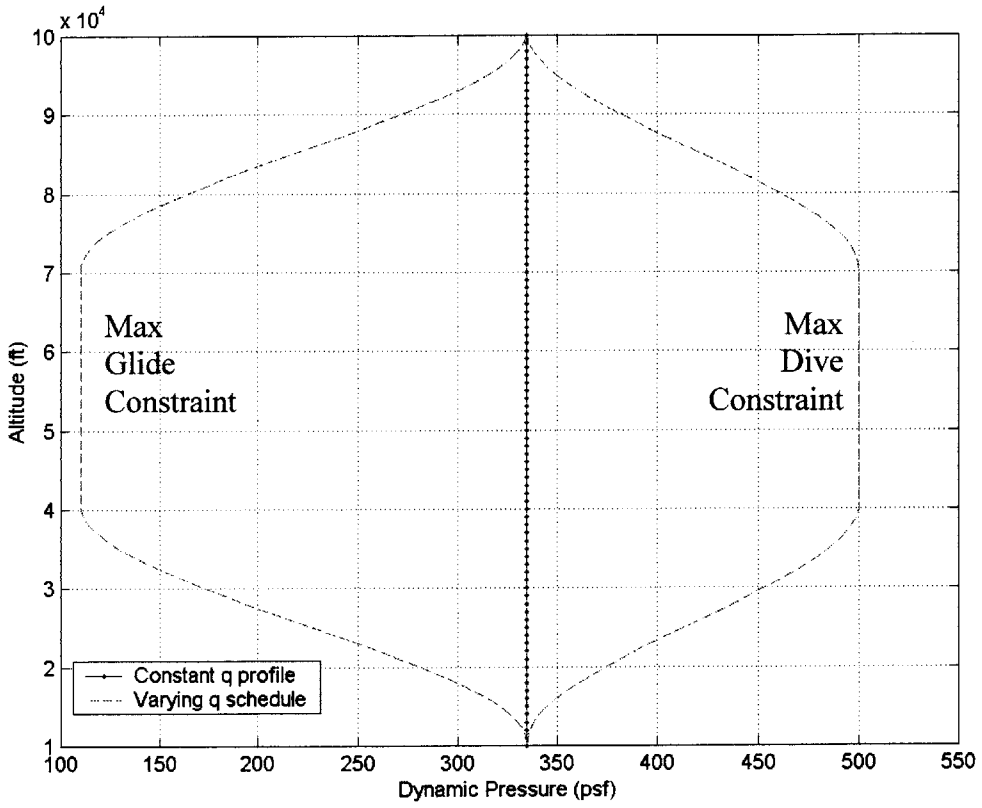


Figure 5.8: Dynamic Pressure Schedules

The Max Dive boundary represents the steepest path a vehicle may follow without violating its maximum dynamic pressure limit. It also defines the shortest possible ground track length the vehicle may achieve. A trajectory designed to follow the MD limit utilizes a constant $\overline{q}$ profile and represents the uttermost limit of the vehicle dive capability. The X-34 is limited to a maximum dynamic pressure of 500-psf, beyond which structural damage may occur. A typical Max Dive trajectory through the TAEM region is shown in Figure 5.9. It can be seen that following a constant $\overline{q}$ profile results in a continuously evolving flight path, and not the quasi-static equilibrium glide slope utilized by ALIP and ALIP3D. The vehicle must continuously pitch downward to maintain the high dynamic pressure as the vehicle slows and descends into the ever-thickening atmosphere. As it approaches the ALI, it must then pitch upward to decrease the dynamic pressure in order to satisfy the 335-psf constraint.

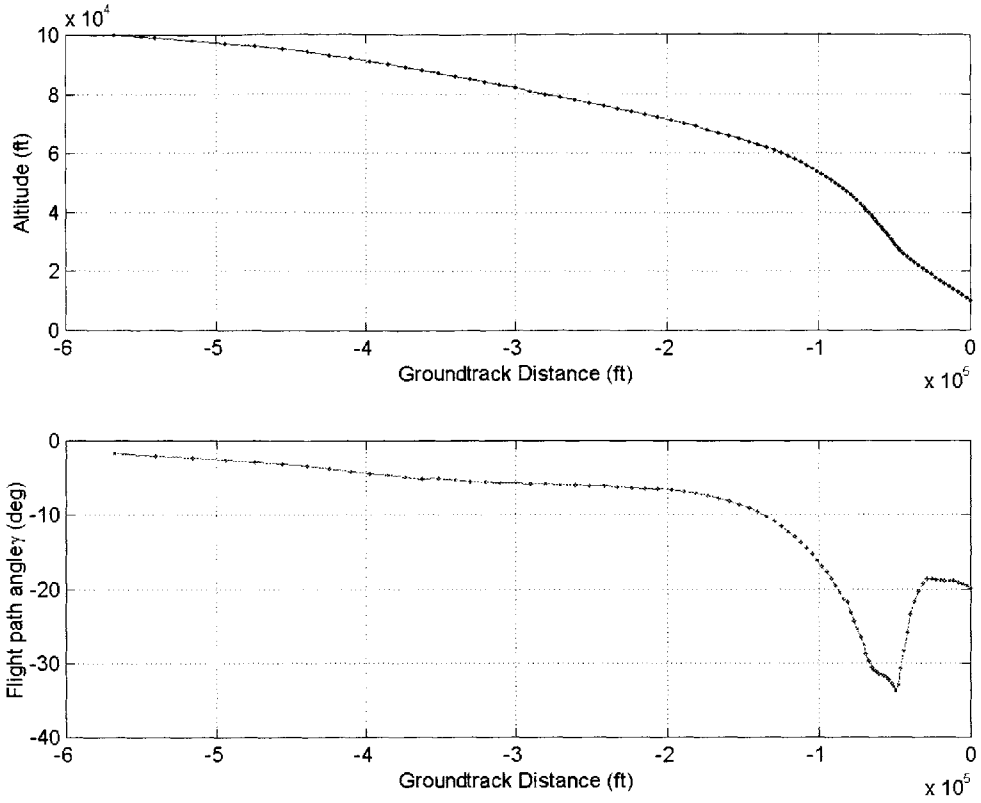


Figure 5.9: Flight Path for Max Dive Trajectory

The Max Glide limit represents the furthest horizontal distance a vehicle may achieve, and is determined by following the vehicle's maximum lift-over-drag contour. In order to fly at the max L/D, a vehicle must maintain an angle of attack corresponding to the peak in the L/D curve. This actually varies due to velocity and speed brake changes over the course of a typical trajectory. A suitable approach for determining the MG limit is to substitute it with a constant $\overline{q}$ profile that closely approximates the Max Glide line. Not only does this provide for a smoother trajectory, but it also standardizes the energy corridor boundaries with a constant $\overline{q}$ profile much like that of the MD limit. The MG line for the X-34 is set at a dynamic pressure value of 110-psf, in accordance with Reference 1. A graph of the nominal energy corridor, including the Max Dive and Max Glide boundary lines, is shown in Figure 5.10.

A center of corridor profile is a trajectory that follows a constant dynamic pressure of 335-psf. This type of profile allows a vehicle to fly in between its MD and MG limits

and theoretically may represent the point of maximum robustness for the vehicle because it has the maximum ability to change between energy states. Maximum robustness is defined as the condition of greatest tolerance to future dispersions. For example, if a vehicle is following a center of corridor profile and suddenly finds itself short of the runway, it can extend its range by transitioning towards the max glide boundary. However, if it were initially following an MG trajectory, it would not be able to extend its range and would fail to reach the target location. Once again, it should be noted that the choice of the dynamic pressure schedule is left up to the trajectory designer, be this a human operator or an automated program.

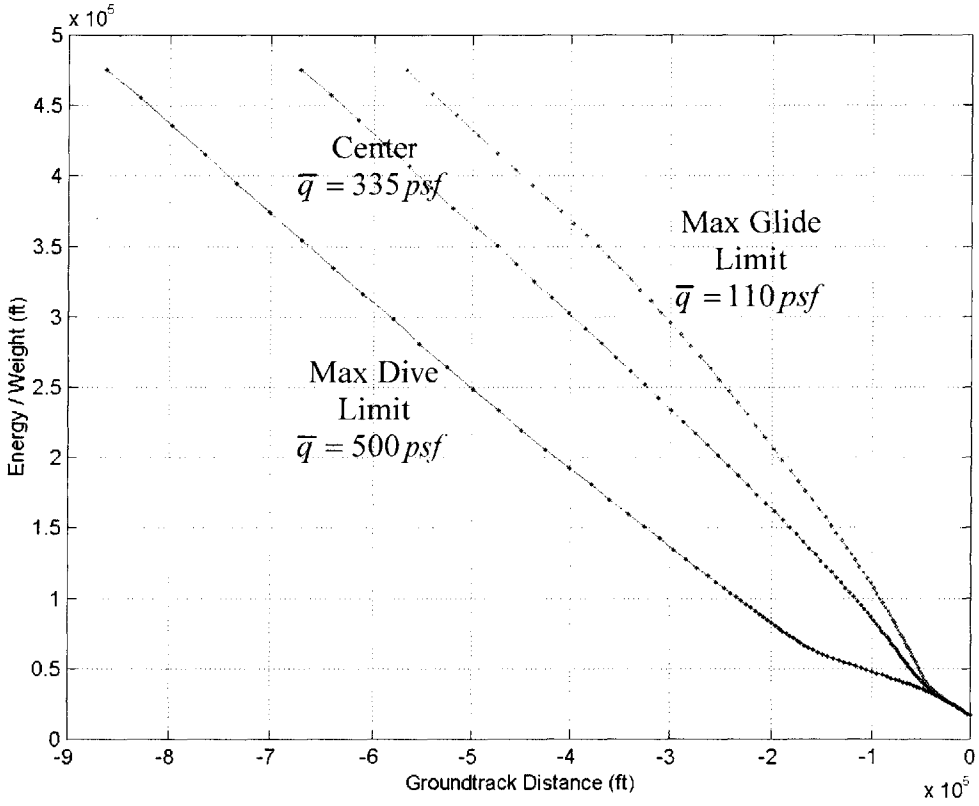


Figure 5.10: Typical Energy Corridor

5.4.3 Ground Track Formulation

The next step in the application of the SNAKE methodology is the formulation of the ground track or the design of the lateral trajectory profile. This is done by the use of a real-time ground track solver that takes advantage of the varying maneuverability resulting from the large variations in speed and altitude over the trajectory, and manipulates a projected ground track to reach the final target location. The formulation of the projected ground track is derived from the SNAKE's method of handling the design space through the use of nodes and straight-line segments.

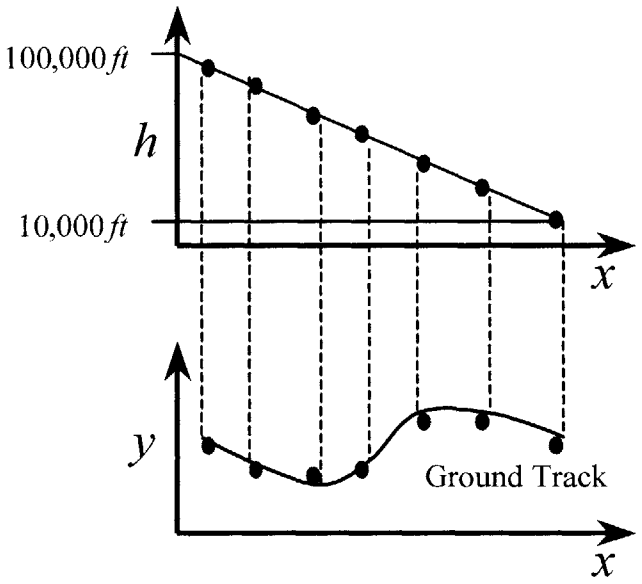


Figure 5.11: Nodes and Ground Track Segments [1]

SNAKE divides the TAEM region into altitude steps with a node corresponding to each altitude value. The nodes are then connected to each other by straight-line trajectory segments shown in Figure 5.11. A projection of these trajectory segments onto the $x_{ALI} - y_{ALI}$ plane makse up the ground track shape of the trajectory. As can be seen in Figure 5.12, the 2-D ground track segment lengths and orientations are geometrically determined from node to node by the vehicle's flight path and heading angles respectively. The vehicle's flight path angle is dynamically constrained to maintain a $\overline{q}$ schedule as discussed in the previous section, and yields an estimation of all the segment

lengths. Thus, the overall predicted ground track is specified by a summation of all the ground track segments.

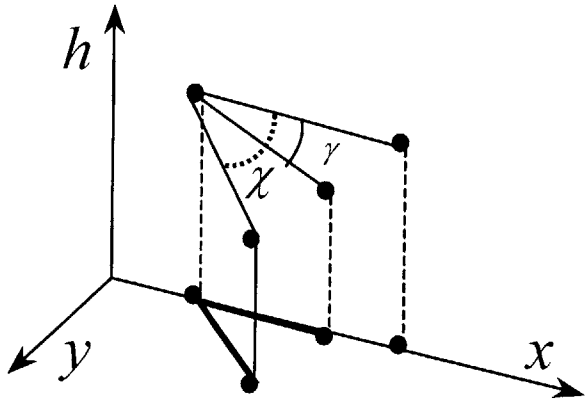


Figure 5.12: Ground Track Segment Orientation [1]

In the previous subsection, the ground track range was determined assuming wings-level flight for a specific $\bar{q}$ schedule. Now, it is important to add in the turning capability of the vehicle. Unlike ALIP3D, where the turning radius is determined by a given Nz load and the initial vehicle velocity, the SNAKE method calculates the turning capability for a given Nz load at every altitude step. This allows designing trajectories within the full capability of the vehicle as well as correctly handling the dynamics of flight. To do this, the SNAKE method calculates the ground track for a constant Nz turn from the initialization of a TAEM trajectory to the end. The resulting ground tracks are spiral in nature due to the fact that any constant Nz turn in a gliding vehicle will lead to a turn of ever decreasing radius as the vehicle loses velocity. Figure 5.13 depicts the ground tracks for a family of constant Nz turns. These spiraling turns provide a measure of the flexibility of the entire ground track for a particular $\bar{q}$ schedule at a specified Nz limit and lead directly to the derivation of a $d\chi$ array, which delineates the maximum number of degrees of heading angle change allowable between any two trajectory segments. The heading angle changes define how far the segmented ground track is allowed to bend at each node.

Correlating the ground track to a segmented toy snake is the best way of visualizing the calculation of the final ground track shape. For this analogy, the joints of the toy snake represent the trajectory nodes, while the body segments between the joints correspond to the ground track segments of the trajectory. The orientation of the snake's tail

characterizes the current vehicle position and heading, and the head symbolizes the final position and heading of the vehicle that must eventually be brought to lie at the ALI. Figure 5.14 clearly shows the relationship between the toy snake and the trajectory ground track.

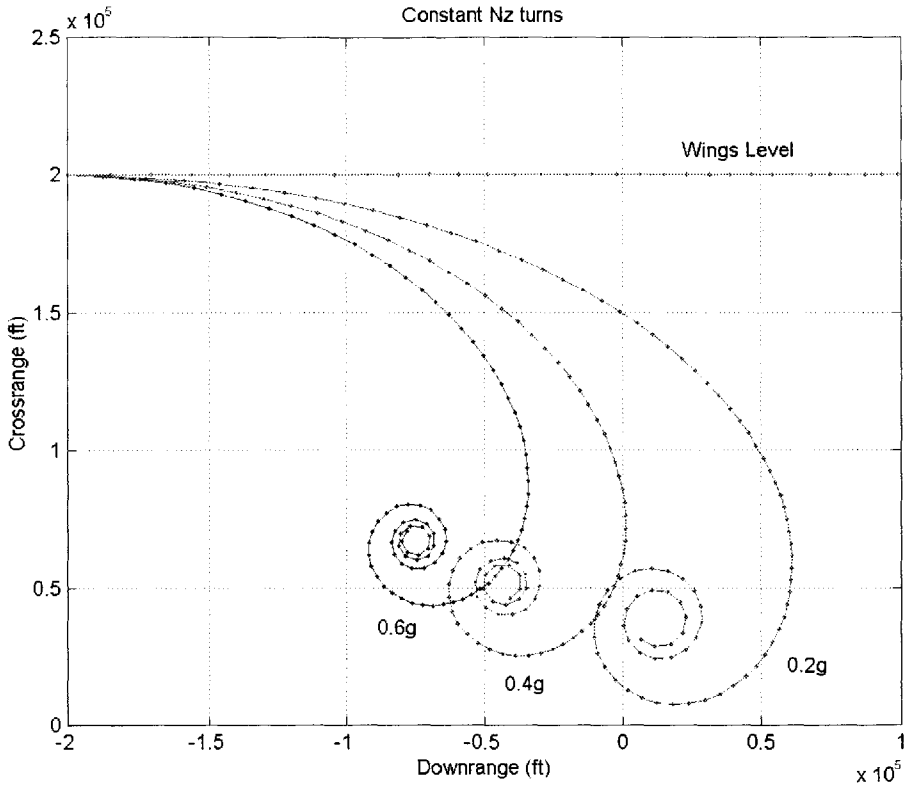


Figure 5.13: Family of Constant Nz Turns [1]

The lateral movements of the snake segments are constrained by the range of the motion of the joints, and correspond directly to the flexibility of the ground track with respect to the Nz banking limits, as determined from the propagation of the spiral turns. In the same fashion, the toy snake is more flexible near the head due to the increase in turning capability as a vehicle loses velocity. Also, the snake segments decrease in length as they are bent, caused by the increase in flight path angle as a vehicle banks. This effect is sometimes referred to as "dumping the lift" [1].

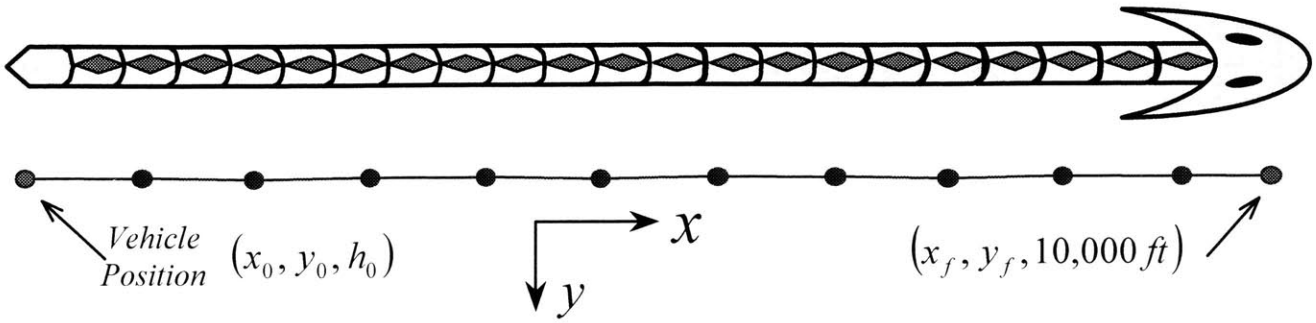


Figure 5.14: Analogy of Toy Snake to Ground Track [1]
(Figure not drawn to scale. In actuality, the number of ground track segments equals the number of snake body segments)

With the analogy of the toy snake established, it may be employed to further explain the formulation of the trajectory ground track. For example, assume a trajectory is being designed for a vehicle with an initial crossrange and downrange offset from the Auto-landing Interface. The dynamic pressure schedule has already been selected and the maximum straight-line ground track profile is propagated through an estimation of the flight path angles. This ground track distance overshoots the ALI by several thousands of feet. The only way for the vehicle to arrive at the ALI while flying the selected $\overline{q}$ schedule is to "snake" the ground track around so that it terminates at the ALI target. This must be done without violating any of the turning constraints placed upon each segment and must also take into account the loss of segment length caused by each turn. The toy snake demonstrates this example in Figure 5.15.

Defining the lateral profile in this fashion completely discards any dependency on fixed geometric shapes. The resulting ground track is designed entirely from the vehicle's capabilities and the coupled lateral and longitudinal dynamics. The actual manipulation of the snake-like ground track is handled by a real-time ground track solver, which must bend it to the final conditions, just like the example above. Once the final ground track shape has been finalized, the γ, $d\gamma/dh$, and $d\chi/dh$ values are passed on to be used by KEP in the propagation of the trajectory. It should be noted that the intelligence of the SNAKE methodology lies in the formulation of the lateral ground track. If the ground track solver can generate a solution for all types of initial conditions, then the SNAKE method can generate trajectories for all conditions.

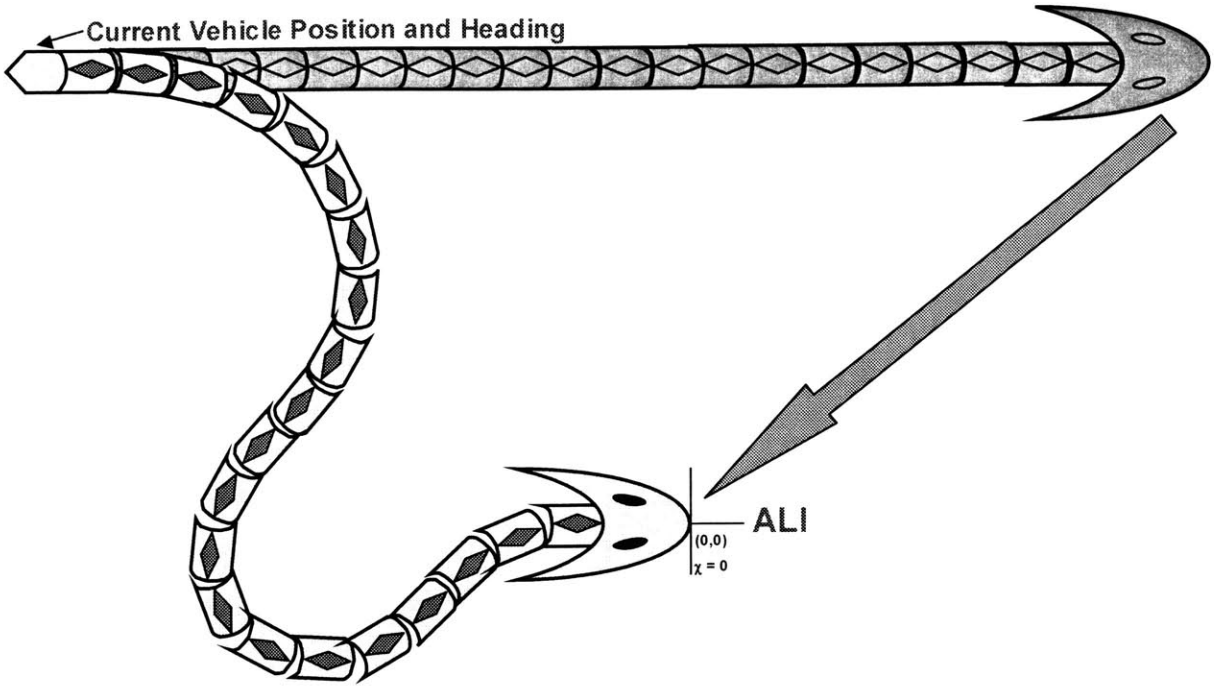


Figure 5.15: Manipulation of Projected Ground Track to the ALI [1]

5.4.4 Rapid Trajectory Propagation

Once the dynamic pressure schedule is determined, and the ground track is formulated, a trajectory is rapidly propagated through the use of the Kernel Extraction Protocol. This involves the balancing of all three kernel equations of motion, because the order of the system has not been reduced by introducing constraining geometry. The system of equations to be solved consists of:

$$\frac{d\overline{q}}{dh} = \mathrm{f}_1(\overline{q}, \gamma, \alpha, h) \tag{5. 32}$$

$$\overline{q} = \mathrm{f}_2(\gamma, \alpha, h, \mu) \tag{5. 33}$$

$$\frac{d\chi}{dh} = \mathrm{f}_3(\gamma, \alpha, h, \mu) \tag{5. 34}$$

The general solution technique involves using three nested loops to achieve the required balance between the three equations. This program layout is shown in Figure 5.16. The innermost loop adjusts the flight path angle γ to balance the two estimates of the dynamic pressure derived from Eqs 5.32 and 5.33. The next outward loop adjusts the angle of attack to ensure that the dynamic pressure determined in the inner loop matches the $\bar{q}$ of the chosen dynamic pressure schedule. Finally, the outermost loop adjusts the bank angle μ to satisfy the required $d\chi/dh$ or Nz value determined during the ground track formulation. Once all three equations are in agreement, the process continues for each altitude step until the completion of the trajectory design.

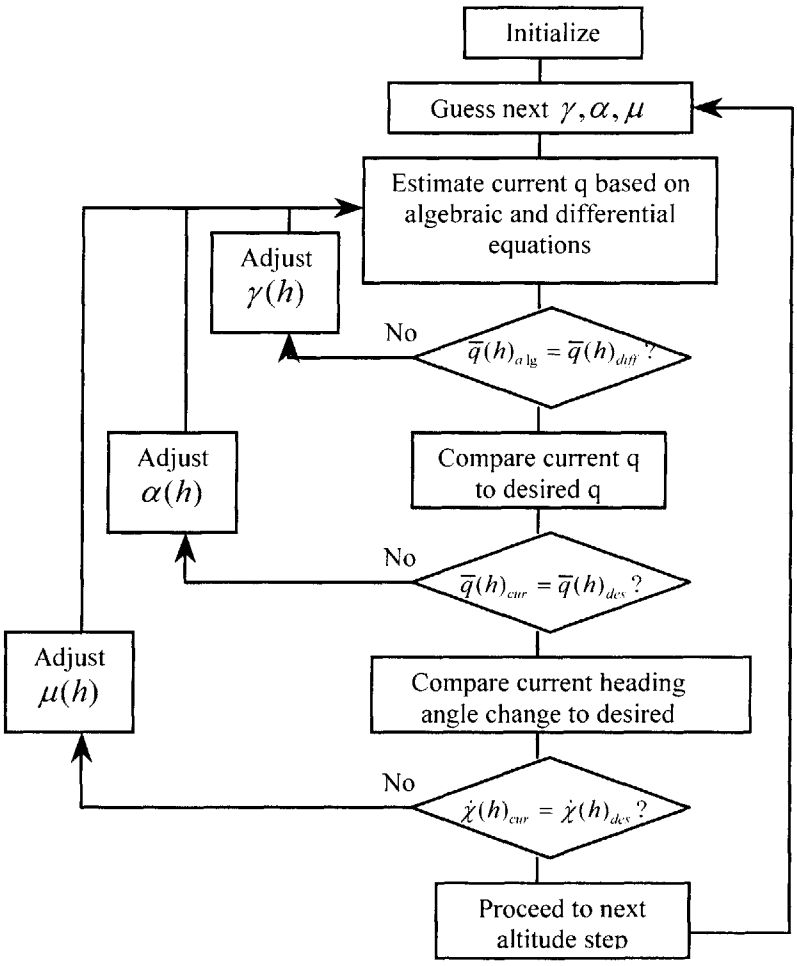


Figure 5.16: General Propagation Algorithm [1]

5.5 Proto-Snake

5.5.1 Methodology

The Proto-snake is a prototype trajectory generator, designed by A. Grubler, which makes use of the SNAKE methodology. It is the first generator to utilize the three SNAKE technological components and the first to demonstrate the potential application of an onboard ground track solver. The Proto-snake can generate trajectories throughout the TAEM region and has proven to be fairly robust in this domain. Reference 1 contains a collection of results for this generator.

Proto-snake begins its trajectory design, as per the SNAKE methodology, with the determination of a dynamic pressure schedule. At the present time, the generator does not have the intelligence or ability to design or select the most appropriate dynamic pressure schedule for the given conditions. However, Proto-snake does have the ability to step the trajectory up or down from the initial $\overline{q}$ state using the previously described quadratic function, which can blend the step with the initial and terminal boundary constraints.

With the $\overline{q}$ profile decided, the program uses KEP to propagate both a wings level trajectory at the specified $\overline{q}$, and a spiral constant Nz trajectory. The Nz used for the spiral turn is set by the user, and again is not chosen automatically by the trajectory generator. These two trajectories are used by the ground track solver in determining the lateral profile shape, or in regards to the previously used analogy, the shape of the toy snake. The wings level trajectory gives the ground track solver the projected flight path angles, and the spiral trajectory gives it the maximum allowable heading angle changes at each node and the impact on the wings level flight path angle. If the ground track solver needs to turn at less than the maximum allowable, the corresponding flight path angle is an interpolation between the wings level and spiral values of γ. This is shown in Figure 5.17.

The real-time Ground Track Solver (GTS) used by Proto-snake is the only real automated process within the trajectory generator. It takes the current vehicle location, and the output from the wings level and spiral trajectories and designs the necessary ground track shape to reach the ALI. While maintaining the dynamic pressure schedule, it solely designs trajectories following a pre-designed method. In this manner, it is an ad hoc

solution for limiting cases, but it is does demonstrate the unique capabilities and advantages of the SNAKE method.

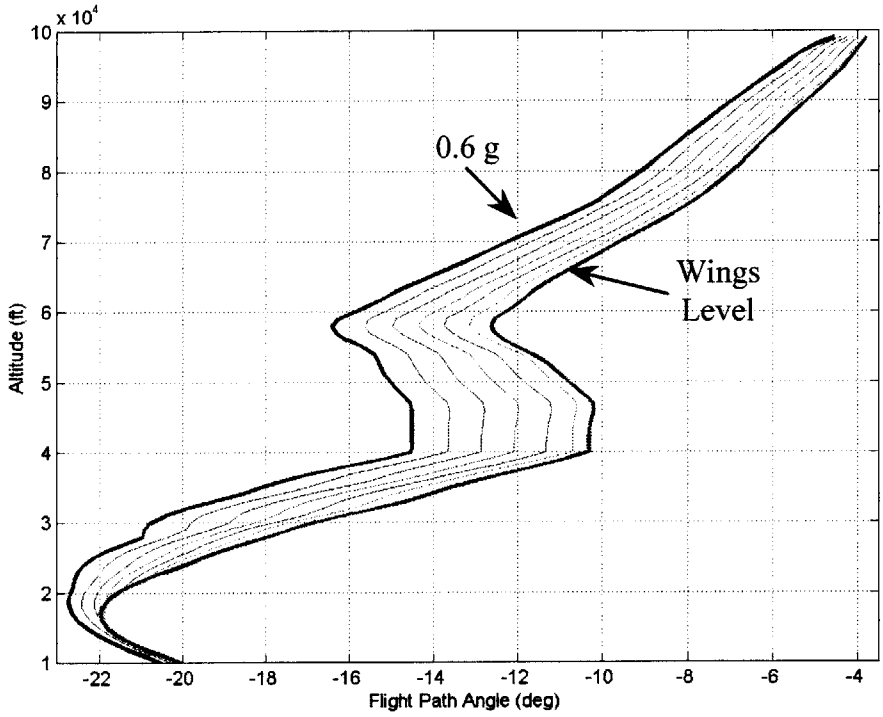


Figure 5.17: γ versus $^{d\gamma}/_{dh}$ Table for Segment Length Prediction [1]

The ad hoc approach taken in the design of the GTS was to force it to operate in much the same way as an objective function. By using "brute force" methods, the GTS drives the trajectory crossrange errors to zero as quickly as possible. This takes place in two phases. In phase one, shown in Figure 5.18, the GTS begins by marching down the nodes of the projected ground track while bending each node towards the x-axis at the maximum allowable bend (determined from the Nz limit). Each adjacent segment to the node currently being bent is flexed in the opposite direction until the remaining ground track is parallel with the x-axis. This continues until the remaining projected ground track is coincident with the x-axis, which is when the crossrange error is zero.

The second phase is used to pull out the excess ground track that extends beyond the ALI threshold. To do this, the GTS continues to bend the segments at the maximum Nz away from the ALI while bending the remaining segments back towards the x-axis and holding them coincident with the axis (Figure 5.19). This is continued until the downrange error

has been removed. Once the final node is coincident with the ALI location, the GTS ends and passes on the heading angle profile for the resulting ground track to the general KEP propagator for the completion of the trajectory design.

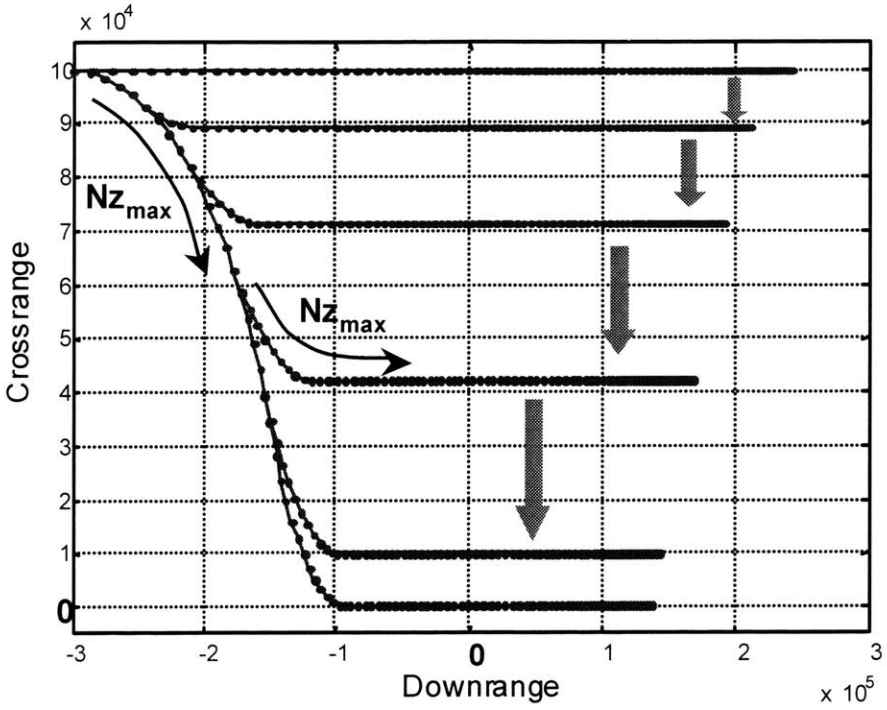


Figure 5.18: Phase One of the Ground Track Solver [1]

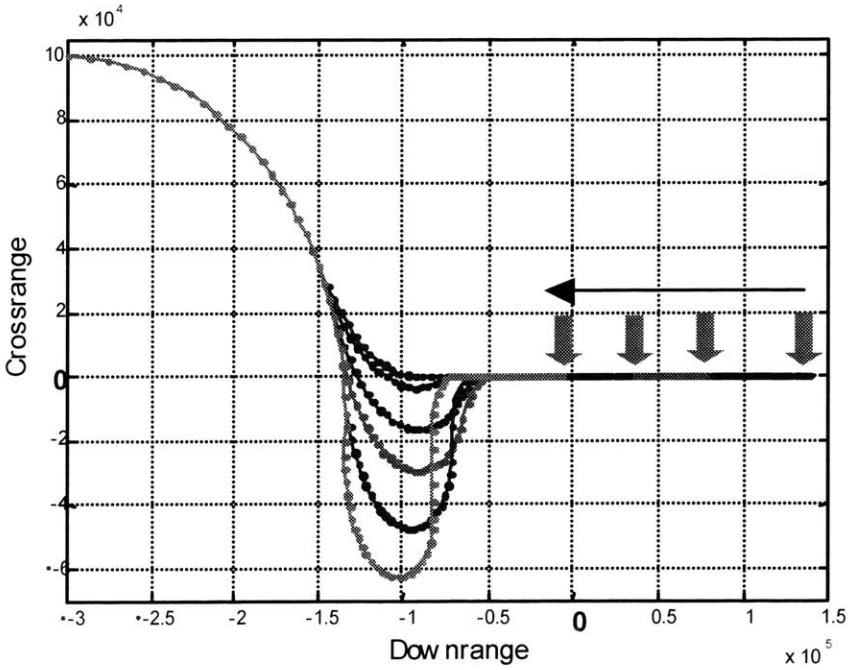


Figure 5.19: Phase Two of the Ground Track Solver [1]

5.5.2 Limitations

The SNAKE method has successfully addressed the shortcomings that the geometric constrained methods have in the supersonic flight regime. However Proto-snake, as the first prototype in the application of the SNAKE methodology, has its limitations. These include the following:

Dynamic Pressure Schedules: These are currently user defined and are not designed differently for varying trajectory scenarios. This means that the $\overline{q}$ schedule selected may not provide the most suitable path for flying from the initialization point to the ALI. The greatest benefit of an onboard trajectory generator is its ability to design trajectories during an abort scenario. Hopefully the trajectories designed would be robust enough to handle future dispersions during the flight. If the $\overline{q}$ schedule defined is non-optimal for the given conditions, the resulting trajectory will not be the best solution to the problem.

Currently there is no fine measure of an optimal dynamic pressure schedule or a protocol that determines how that schedule is formulated. The research presented in this thesis hopes to address this problem and lay out possible criteria for the calculation and formulation of more robust $\overline{q}$ schedules and thus more robust trajectories. This research focuses on benchmarking trajectories at a variety of different conditions using an offline optimizer, and determining the desirable physical characteristics of robust trajectories for an ad-hoc onboard trajectory generator. By analyzing the results, dynamic pressure schedules that define robust trajectories can be identified.

Ground Track Solver: The current solver uses brute force methods to force a trajectory towards the ALI, but only by one method. While this may work and prove feasible in designing a score of trajectories for a variety of different initial conditions, it actually limits the design space and reduces the number of possible solutions. For example, if the Proto-snake is initialized for an altitude and $\overline{q}$ reference in which the resulting ground track is just long enough to make it to the ALI, the GTS would try and bend it through its two design phases and would discover that the trajectory is short on range and fails to supply a solution to the problem. However, a possible solution exists if the trajectory is bent only slightly towards the runway so that the resulting ground track shape is basically a straight-line flight to the ALI. This is an obvious shortcoming when the program cannot find a trajectory for a given set of conditions even though a solution does exist.

Additionally, the GTS does not guarantee the optimality or robustness of the trajectory shapes designed. This is due in part to the lack of a suitable benchmark that represents the ground track characteristics of a robust trajectory. The research presented in the later chapters of the thesis is focused primarily on this issue. Once an optimal trajectory benchmark is established, a ground track solver may be programmed to produce ground tracks that emulate the features of those trajectories.

Proto-snake's design time for the creation of trajectories is also slowed down by the process of having to design two trajectories before designing the final one. This is reminiscent of the time penalty incurred by ALIP's use of a shooting method. Hopefully the wings level and spiral trajectories can be pre-designed for different $\bar{q}$ profiles and stored to reduce this computation time.

Predictor / Corrector: In the present form, Proto-snake passes on the γ values predicted during the wings level and spiraling trajectories along with the $d\chi$ values determined from the GTS to the KEP propagation algorithm for the completion of the trajectory design. However, the bank angle of a vehicle has a direct effect on the flight path angle necessary to hold on to a given dynamic pressure schedule. This means that any adjustment to the ground track will result in a modification of the γ profile. Proto-snake takes this into account, but incurs a small error if the bank angle required by the GTS is less than the one used during propagation of the spiral turns. This is shown in Figure 5.17, when the GTS interpolates the γ value for a given bank angle. While this is a good approximation, it is not equal to the actual γ calculated during the KEP procedure. Therefore, a corrector should be added that compensates for the change in the γ values due to the resulting ground track shape. Adding this component would greatly reduce any errors present in the final trajectories.

5.6 Summary

Draper's progress in the development of a Next Generation Guidance and Control system has been highlighted in this chapter by the presentation of the work done on the formulation of an onboard trajectory generator. The designs of three different prototypes, covering two different fundamental design methods, were explained along with their limitations. Each system improved upon the shortcomings of the previous trajectory generators and expanded upon the use of the Kernel Extraction Protocol, or the core technology, key to the rapid propagation of trajectories within the chosen design space.

Out of these programs, the Sub-optimal Nodal Application of the Kernel Extraction method has shown the greatest promise for future development. This involves addressing the limiting issues of the method's first functioning prototype, Proto-snake, and improving upon its design.

The following chapters discuss the work completed in this research for the advancement of the SNAKE method. This involves characterizing benchmark trajectories that future ground track solvers should strive to emulate and creating guidelines for the determination of appropriate dynamic pressure schedules. Chapter 6 explains the type of optimization method used to attain these goals while Chapter 7 describes the actual problem setup and execution method.

Chapter 6

Optimization Tools

6.1 Overview

Proto-snake demonstrated the feasibility of using the SNAKE methodology to design reentry trajectories suitable for reusable launch vehicles. Its use of a real-time ground track solver enabled the program to automatically generate realistic flight paths for a variety of different flight conditions and dynamic pressure schedules. However, the optimality and robustness of the trajectories created was never proven or verified. The goal of this thesis is to characterize the ground tracks created by the GTS and to establish the benchmark trajectory for an onboard trajectory generator. This is accomplished by creating trajectories through the use of an optimization routine. This chapter covers the optimization tools used to realize this goal. The first section provides an overview of a dynamic optimization problem, while the second section describes the pseudospectral optimization method. The final sections describe the software and hardware tools used to complete this research, including the MATLAB routine DIDO.

6.2 Optimization Problem

The basic idea in an optimization problem is the minimization of an objective (or cost) function, subject to certain constraints. The cost function can be made up of any combination of state and/or control variables, and possibly state variable values at certain clock times. Once the problem is set up, a technique is chosen to solve for the minimization of the cost function by manipulating the state and control variables, while satisfying the dynamic constraints, such as the equations of motion.

It is necessary to describe the mathematical formulation of the basic optimization problem before describing the solution techniques or direct application to trajectory generation. The setup for the trajectory optimization contained in this thesis can be described in its simplest form as follows, as stated in Reference 14.

Let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$. The variable $\mathbf{x}$ is a vector made up of the state variables of the problem, while $\mathbf{u}$ is a vector made up of the control variables. The goal is to determine

the trajectory-control pair, $\{\mathbf{x}(\cdot), \mathbf{u}(\cdot)\}$, and possibly the clock times or "event" times that minimize the objective function. The clock times, τ_0 and τ_f, are the initial and final times respectively. The objective function, also known as the Bolza cost functional, can be written as:

$$J[\mathbf{x}(\cdot),\mathbf{u}(\cdot),\tau_0,\tau_f] = \phi\left(\mathbf{x}(\tau_0),\mathbf{x}(\tau_f),\tau_0,\tau_f\right) + \int_{\tau_0}^{\tau_f} L(\mathbf{x}(\tau),\mathbf{u}(\tau),\tau)\,d\tau \tag{6.1}$$

where J represents the objective function, ϕ represents the end point cost or Mayer cost and L is known as the integrand or Lagrange cost.

The objective function is subject to the dynamic constraints,

$$\dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau),\mathbf{u}(\tau),\tau) \tag{6.2}$$

and the end point constraints, also known as event constraints, that occur at certain "event" times,

$$e_l \le e\left(\mathbf{x}(\tau_0),\mathbf{x}(\tau_f),\tau_0,\tau_f\right) \le e_u \tag{6.3}$$

Note: the subscripts $_l$ and $_u$ stand for lower and upper respectively. An equality constraint is simply expressed by setting the lower bound equal to the upper bound.

The objective function is also subject to constraints on the state and control variables, as well as mixed trajectory-control path constraints. These can be written as:

$$\mathbf{x}_l \le \mathbf{x}(\tau) \le \mathbf{x}_u \quad \text{and} \quad \mathbf{u}_l \le \mathbf{u}(\tau) \le \mathbf{u}_u \tag{6.4}$$

$$p_l \le p(\mathbf{x}(\tau),\mathbf{u}(\tau),\tau) \le p_u \tag{6.5}$$

Also note, that all functions are piecewise differentiable and if there are any discontinuities in the state and control variables they are of finite number.

6.3 Optimization Methods

6.3.1 Pseudospectral Methods

There are many methods available for the solving of optimization problems, applicable to trajectory generation. Essentially these methods are broken down into two categories, indirect and direct methods. The indirect methods basically use calculus of variations to explicitly solve for the optimality conditions of the optimality problem. This is done by reducing the problem to a nonlinear multi-point BVP. Direct methods, on the other hand, convert the optimization problem into a parameter optimization problem, by discretizing the time domain into a set of subintervals. The end points of these intervals are called nodes, and the parameters are the values of the states and controls at these nodes [15]. The resulting problem can be solved using any one of many nonlinear programming codes. However, using pseudospectral methods in solving optimization problems is faster and more accurate than the two methods mentioned above even though it makes use of essentially the same direct technique. The difference comes from using a pseudospectral differentiation matrix as opposed to finite difference matrices or finite element methods [16].

Pseudospectral methods have an advantage in solving optimization problems because their approximations of differential equations are more accurate than the traditional finite difference or finite element methods. This accuracy comes from the pseudospectral differentiation matrix, which is used to discretize the nonlinear differential equations into nonlinear algebraic equations. The resulting equations are then rearranged into a parameter optimization problem, which is solved through the use of a numerical optimizer. This differentiation matrix is best described by starting with an explanation of a finite difference matrix.

6.3.1.1 Finite Difference Matrix

Assume that there is positional data for an object moving in a straight line, taken at 6 times separated by equal time steps. The velocity of the object is the time derivative of the position.

$$V = \frac{dx}{dt} \tag{6. 6}$$

A finite difference formula can be used to estimate the velocity at each of the six points. This is done by using a forward difference approximation for the first point, a backward difference approximation for the last point, and a central difference approximation for all points in between. The formulas for these approximation are derived from the first two terms of a Taylor series expansion and are shown below. The error, denoted by e, is due to the truncation of the Taylor series expansion [16].

Forward Difference: $$V_i = \frac{x_{i+1} - x_i}{\Delta t} + e \quad \text{where } e \approx O(\Delta t) \tag{6. 7}$$

Central Difference: $$V_i = \frac{x_{i+1} - x_{i-1}}{2\Delta t} + e \quad \text{where } e \approx O(\Delta t^2) \tag{6. 8}$$

Backward Difference: $$V_i = \frac{x_i - x_{i-1}}{\Delta t} + e \quad \text{where } e \approx O(\Delta t) \tag{6. 9}$$

Using these expressions, the velocity at each time point can be written as:

$$\begin{aligned} V_1 &\approx \frac{x_2 - x_1}{\Delta t} \\ V_2 &\approx \frac{x_3 - x_1}{2\Delta t} \\ V_3 &\approx \frac{x_4 - x_2}{2\Delta t} \\ V_4 &\approx \frac{x_5 - x_3}{2\Delta t} \\ V_5 &\approx \frac{x_6 - x_4}{2\Delta t} \\ V_6 &\approx \frac{x_6 - x_5}{\Delta t} \end{aligned} \tag{6. 10}$$

Expressing the above equations in matrix form yields the finite difference matrix of Eq 6.11. This matrix is referred to as the "*D*-matrix" and is denoted as D_6, where the subscript represents the number of points used in the calculation of the matrix.

$$\frac{1}{\Delta t}\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \approx \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \end{bmatrix} \tag{6. 11}$$

The error in the approximation of the velocity can be reduced if more terms from the Taylor series expansion are added into the difference approximations.

The most important concept in the formulation of the finite difference matrix is the idea that a continuous differential equation can be approximated by a discrete set of algebraic equations [16]. If a vector of position data taken at different time points is given for an object, then the velocity at those time points can by approximated using the *D*-matrix.

$$V = \frac{d(x)}{dt} \tag{6. 12}$$

$$\vec{V} \approx D_6 \vec{x} \tag{6. 13}$$

This equation can be extended to systems of equations with finite set of N points, where the *D*-matrix in those cases would be N x N matrices denoted by D_N.

6.3.1.2 Pseudospectral Differentiation Matrix

The pseudospectral *D*-matrix can be used to approximate derivatives in the same manner as the finite difference *D*-matrix. However, the theory behind the pseudospectral matrix differs from that of the finite difference matrix by the choice of spacing between the points. Whereas the finite difference approach uses equally spaced points, the pseudospectral technique does not. This is mainly due to the choice of interpolation used to estimate a curve fit for the points, made up of a polynomial or sum of polynomials.

The pseudospectral method uses a Lagrange interpolation to fit polynomials between the data points. For a set of N data points, the Lagrange interpolation formula is:

$$y(t) = \sum_{j=1}^{N} y_j \phi_j(t) \tag{6. 14}$$

where
$y(t)$ = the approximating polynomial
y_j = the value of y at t_j
$\phi_j(t)$ = the set of interpolating functions

The approximation of Eq 6.14 only applies to values of t between the t_i node points. This is defined to be exactly equal to the data points at the values of t_i. It should be noted that there is no reason that the data nodes be evenly spaced. Therefore, this is a method for finding a polynomial to approximate a set of arbitrarily spaced points [16].

The formulation of the pseudospectral D-matrix comes about by using the polynomial approximation above to estimate derivatives at the node points, with respect to t. This begins by taking the derivative of Eq 6.14.

$$\dot{y}(t) = \sum_{j=1}^{N} y_j \dot{\phi}_j(t) \tag{6. 15}$$

Only the derivatives at the node points are desired. This yields:

$$\dot{y}(t_i) = \sum_{j=1}^{N} y_j \dot{\phi}_j(t_i) = \sum_{j=1}^{N} D_{ij} y_j \tag{6. 16}$$

where
$\vec{y}$ = vector of N data points at the nodes
D_N = the N × N pseudospectral differentiation matrix

Therefore, the elements of the D_N matrix are given by:

$$D_{ij} = \dot{\phi}_j(t_i) \tag{6. 17}$$

For a more detailed derivation of the pseudospectral differentiation matrix, refer to Reference 16.

6.3.2 Legendre Pseudospectral Optimization Method

As mentioned previously, the node spacing of the data points is arbitrary. However, it is possible to choose the spacing that will yield the best polynomial approximation. Approximation theory has shown that the optimal node spacing occurs when the nodes are the roots of orthogonal polynomial such as Tschebyscheff or Legendre polynomials [16].

The Legendre pseudospectral method is a pseudospectral method that places its nodes coincident with the set of Legendre-Gauss-Lobatto (LGL) points. The LGL points are defined as:

$$\begin{aligned} t_1 &= -1 \\ t_j &= \text{roots of } L'_{N-1}(t) \ (j = 2...N-1) \\ t_N &= 1 \end{aligned}$$

where $L_{N-1}(t)$ is the Legendre polynomial of order $N-1$.

These points must be computed numerically because there are no closed form equations. Figure 6.1 shows the characteristic spacing of the nodes. The nodes actually "bunch up" near the ends and "spread out" in the middle of the domain.

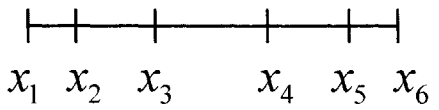


Figure 6.1: Position Data Taken at LGL Time Points [16]

In summary, the Legendre Pseudospectral Method is a way of writing any optimization problem modeled with differential equations as a problem with only algebraic constraints by use of the pseudospectral differentiation matrix. It is then rearranged as a parameter optimization problem and solved by a numerical optimizer. This method was developed by F. Fahroo and I. M. Ross of Naval Postgraduate School, and has been demonstrated to work well with both continuous and discontinuous states and controls. The results of this thesis are the result of an application of this method. For more detailed information concerning the legendre pseudospectral method, refer to References 17 and 18.

6.4 Direct Indirect Dynamic Optimization (DIDO)

DIDO is a MATLAB routine for solving possibly discontinuous dynamic optimization problems by using the legendre pseudospectral method. This program was developed by I. M. Ross and F. Fahroo, and has several key characteristics that make it useful for trajectory optimization problems. It is easy to use, demonstrates robust convergence and is fairly quick in obtaining a solution. It provides an intuitive front end for setting up an optimization problem to those not familiar with optimization procedures. One only needs to understand how to set up the basic problem, then input that problem into a specific DIDO format and start the program. DIDO does the rest.

DIDO works by taking the problem input by the user, and transforming it from the domain of the dynamic equations to the LGL domain. Then it calculates the pseudospectral *D*-matrix and uses it to transform the derivatives into algebraic expressions. This information is transferred to a numerical optimizer that actually solves the problem. The solution data is then returned to the MATLAB command window for analysis by the user. The numerical optimizers currently used by DIDO are either NPSOL or SNOPT, which are explained in the next section.

DIDO is the optimization routine used during this thesis, and all data presented is the solution results supplied by the program. The basic procedure for creating trajectories through DIDO is as follows:

1. Choose the states and control variables of the trajectory problem.
2. Write out the dynamic equations. These usually consist of any differential equations that describe the system and that must be met at all points in the

trajectory problem. For example, this may be the governing equations of motion of the problem, such as those derived in Chapter 3.

3. Non-dimensionalize the problem. This is done by choosing values that will make all the values in the optimization vector roughly the same order of magnitude. A canonical scaling system or ad hoc method can be used. Non-dimensionalizing the problem usually improves the numerical properties of the problem as well as working better within the numerical optimizer.
4. Write out all the trajectory constraints, appropriate path constraints and bounding values for the states and controls.
5. Formulate the cost or objective function. This function is the designing factor in the creation of the trajectories.
6. Code steps 1-5 into the DIDO format and run the code.

A sample of the DIDO interface code is contained in Appendix A. This format is what is required to run DIDO, and is the direct result of step 6 above.

It should be noted that this version of DIDO can only solve an optimization problem using one cost function. Future versions may have the ability for solving problems that utilize multiple cost functions.

6.5 Software and Hardware Specifications

6.5.1 Software

MATLAB—DIDO is run using MATLAB, version 5.3.1.29215a (R11.1). Its producer, The Math Works, Inc, released this version on September 28, 1999.

NPSOL—this is one of the numerical optimizers used by DIDO. It was developed by P. Gill at the University of California, W. Murray and M. Saunders at Stanford University, and M. Wright at Bell Laboratories. This program was originally written as a set of FORTRAN subroutines for nonlinear programming problems. DIDO is able to run the code through the use of an NPSOL mexfile. For more information, see Reference [19].

SNOPT—this numerical optimizer is setup in the same fashion as NPSOL. It is written as a set of FORTRAN subroutines used for solving nonlinear programming problems. However, SNOPT applies a sparse sequential quadratic programming (SQP) method that

requires less matrix computation than NPSOL. It was developed by P. Gill at the University of California, and W. Murray and M. Saunders at Stanford University. For more information, see Reference [20].

6.5.2 Hardware

The computer used for this research was supplied by Draper Laboratory, Inc. and its specifications are listed in Table 6.1 below.

Table 6.1: Computer Hardware Specifications

Manufacturer	Gateway Inc.
Main Memory	392 MB
Operating System	Microsoft Windows 2000 5.00.2195
CPU Type	Intel Pentium 4 1500MHZ CPU

Chapter 7

Benchmark Determination

7.1 Overview

The limitations of Proto-snake, presented in Chapter 5, are its inability to determine the best dynamic pressure schedule for the given conditions, in addition to its creation of trajectories for which optimality and/or robustness is unknown. These limitations are due in part to the lack of a suitable trajectory benchmark that provides the characteristics inherent to robust trajectories. Chapter 6 provided an overview of the Legendre Pseudospectral Optimization Method that is used by DIDO in the creation of optimal trajectories, in addition to an explanation of DIDO itself. The sections of this chapter present the actual optimization problem formulation used to set up DIDO. The following sections also detail the procedure used to determine the different cost functions as well as the criteria and process used to determine the robust trajectories, or trajectory benchmarks.

7.2 Problem Setup

DIDO is used to create the optimal trajectories analyzed in this thesis. Before executing the program, it is necessary to set up the trajectory problem in the same format as the basic optimization problem described at the beginning of Chapter 6. This involves specifying the state and control variables, and all the constraints that apply to the problem. The following subsection presents this formulation. Once the problem is set up, it can be coded into the required DIDO format as specified in Appendix A.

7.2.1 Mathematical Formulation

7.2.1.1 States and Controls

Recall from Chapter 6 that $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$, where the variable $\mathbf{x}$ is a vector made up of the state variables, while $\mathbf{u}$ is a vector made up of the control variables. A TAEM trajectory normally involves both lateral and longitudinal dynamics, so a complete

representation of the vehicle's flight path requires that **x** be made up of 6 quantities: three position states and three velocity states. Table 7.1 identifies these state variables.

Table 7.1: Description of State Variables

State Description	Symbol
x Position Coordinate (downrange distance)	x
y Position Coordinate (crossrange distance)	y
Altitude	h
Inertial (Ground-Relative) Velocity	V
Flight Path Angle	γ
Heading Angle	χ

Note that **x** is a vector of the state variables, while x is a state variable.

The control variables used for trajectory propagation by the Kernel Extraction Protocol are the vehicle's angle of attack, α, and bank angle, μ. However, in order to facilitate the use of a soft-knot, described in subsection 7.2.4 below, the controls chosen for DIDO are the angle of attack rate, $\dot{\alpha}$, and bank angle rate, $\dot{\mu}$. Therefore α and μ join the six variables above to complete the state vector **x**. The vectors **x** and **u** can be written as:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ h \\ V \\ \gamma \\ \chi \\ \alpha \\ \mu \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} \dot{\alpha} \\ \dot{\mu} \end{bmatrix} \tag{7.1}$$

7.2.1.2 Dynamic Constraints

The objective function of an optimization problem is subject to dynamic constraints, defined in Chapter 6 as:

$$\dot{\mathbf{x}}(\tau) = f\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) \tag{7.2}$$

The dynamic constraints for the trajectory problem are the equations of motion that govern the flight of the vehicle, derived in Chapter 3. Rewriting the equations in vector/constraint form yields:

$$\dot{\mathbf{x}}(\tau) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{h} \\ \dot{V} \\ \dot{\gamma} \\ \dot{\chi} \\ \dot{\alpha} \\ \dot{\mu} \end{bmatrix} = \begin{bmatrix} V\cos\gamma\cos\chi \\ V\cos\gamma\sin\chi \\ V\sin\gamma \\ \dfrac{-D}{m} - g\sin\gamma \\ \dfrac{1}{mV}\left[L\cos\mu - W\cos\gamma\right] \\ \dfrac{L\sin\mu}{mV\cos\gamma} \\ u_1 \\ u_2 \end{bmatrix} \tag{7.3}$$

where u_1 and u_2 are the two components of the control vector **u**.

7.2.1.3 Event Constraints

The event constraints are also known as the end point constraints, and are written as:

$$e_l \le e\left(\mathbf{x}(\tau_0), \mathbf{x}(\tau_f), \tau_0, \tau_f\right) \le e_u \tag{7.4}$$

Applying these event constraints can be described as fixing the initial and/or final values of particular state variables. Therefore, these constraints can actually be split and written as:

$$e_{0_l} \le e_0\left(\mathbf{x}(\tau_0), \tau_0\right) \le e_{0_u} \tag{7.5}$$

$$e_{f_l} \le e_f\left(\mathbf{x}(\tau_f), \tau_f\right) \le e_{f_u} \tag{7.6}$$

where the subscripts $_0$ and $_f$ refer to the initial and final values respectively.

For this particular problem, all of the initial state values are fixed, and are set equal to the initial values of a Proto-snake trajectory. In this case, in order to fix the state at a certain value, the constraint is made an equality constraint by simply setting the lower bound of Eq 7.5 equal to the upper bound. The final values are handled the same way, and are set to the final values of a Proto-snake trajectory. However, the final values for γ and α are left free, in order to give some leeway to the optimizer and to be consistent with the constraints imposed on the ALI target, since these values are not specified. In addition, the final time, τ_f, is left free and is not specified by the event constraints.

The choice to fix the event constraints to the end point values of a Proto-snake trajectory was made in order to make suitable comparisons between DIDO and Proto-snake trajectories. This allows DIDO to design an optimal trajectory around the same set of conditions imposed upon Proto-snake. For instance, if Proto-snake happens to produce an optimal trajectory, and its end points are fed into DIDO, the resulting trajectories would theoretically match. Therefore, the event constraints force DIDO to design trajectories between the same two points given to Proto-snake. This is important to understand, particularly when the trajectory comparisons are presented.

In all the trajectories produced, the end conditions are specified by the ALI constraints described in Chapter 4, while the initial conditions may vary. For comparison sake, a Proto-snake trajectory is referred to as the reference, and is given a number that corresponds to a given set of initial conditions or starting point, while a DIDO trajectory is referred to by the cost function used to generate it along with the number of the Proto-snake reference. For this thesis, four reference trajectories were produced, corresponding to trajectories starting at four different locations. This means that four different sets of initial conditions were used to formulate trajectories in DIDO as well. The four reference trajectories are explained further in subsection 7.4.

7.2.1.4 Path Constraints

The path constraints confine the optimizer to stay within a set "path" when determining the trajectory. These constraints apply to parameters that are functions of the states and/or controls, but are not part of the dynamic equations.

The mathematical representation of the constraints can be written as

$$p_l \le p(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau) \le p_u \tag{7.7}$$

For the trajectory problem, the path constraints refer to limits on certain vehicle parameters that define the vehicle's capability. These include dynamic pressure and vehicle loading limits. Therefore p can be written as:

$$p = \begin{bmatrix} Nx_b \\ Ny_b \\ Nz_b \\ \bar{q} \end{bmatrix} = \begin{bmatrix} Nx_v \cos\alpha + Ny_v \sin\alpha \sin\mu - Nz_v \sin\alpha \cos\mu \\ Ny_v \cos\mu + Nz_v \sin\mu \\ -Nx_v \sin\alpha + Ny_v \cos\alpha \sin\mu - Nz_v \cos\alpha \cos\mu \\ \tfrac{1}{2}\rho V^2 \end{bmatrix} \tag{7.8}$$

The actual values for the upper and lower constraint limits were set to correspond to the predefined vehicle dynamic limits. These limits are shown in Table 7.2

Table 7.2: Path Constraint Bounds

Path Constraint	Bounds
Body x-acceleration, Nx_b	-1 to 0.6
Body y-acceleration, Ny_b	-1 to 1
Body z-acceleration, Nz_b	-1 to 3
Dynamic Pressure, $\bar{q}$	110 psf to 500 psf

7.2.2 Non-Dimensionalization

Optimization codes, including most numerical codes, behave in a fundamentally superior fashion (accuracy, speed, etc.) when the variables are scaled properly [14]. It is necessary to scale, or non-dimensionalize, this problem before coding it into DIDO. The goal of scaling is to make every state and control variable about the same order of magnitude. This is accomplished by simply dividing the appropriate quantities by their non-dimensionalizing counterparts. There are different ways of determining the non-

dimensionalizing values. These include canonical scaling schemes or ad hoc methods. Both techniques yield the same results: enhancing the numerical computation.

For this particular problem formulation, an ad hoc scaling method was used for non-dimensionalization. The process involves choosing three different length scales that correspond to the three state variables with units of length. The length scales are chosen to make the states the same order of magnitude, even though the three states may be of different order when dimensionalized, such as y = 10 ft and h = 100,000 ft. A velocity scale is also chosen to non-dimensionalize the velocity state. Once these non-dimensionalized values are determined, the equations are rewritten to reflect the change. This is shown in Eqs 7.9 through 7.13.

$$x^* = x / x_{norm} \tag{7.9}$$

$$y^* = y / y_{norm} \tag{7.10}$$

$$h^* = h / h_{norm} \tag{7.11}$$

$$V^* = V / V_{norm} \tag{7.12}$$

$$\dot{\mathbf{x}}(\tau) = \begin{bmatrix} \dot{x}^* \\ \dot{y}^* \\ \dot{h}^* \\ \dot{V}^* \\ \dot{\gamma} \\ \dot{\chi} \\ \dot{\alpha} \\ \dot{\mu} \end{bmatrix} = \begin{bmatrix} \dfrac{V^* V_{norm} \cos\gamma \cos\chi}{x_{norm}} \\ \dfrac{V^* V_{norm} \cos\gamma \sin\chi}{y_{norm}} \\ \dfrac{V^* V_{norm} \sin\gamma_r}{h_{norm}} \\ \dfrac{1}{V^*}\left[-D/m - g\sin\gamma\right] \\ \dfrac{1}{mV^* V_{norm}}\left[L\cos\mu - W\cos\gamma\right] \\ \dfrac{L\sin\mu}{mV^* V_{norm}\cos\gamma} \\ u_1 \\ u_2 \end{bmatrix} \tag{7.13}$$

The subscript "$_{norm}$" in the above equations stands for the normalizing value, and the superscript " * "marks the scaled parameters.

The values of γ, χ, α and μ are converted to radians, which is already non-dimensional. It should also be noted that because three length scales are used, a normalizing value for time cannot be derived from a length scale and a velocity scale, as is done in the canonical scheme. Instead, a normalizing value for time can be chosen. For this problem however, time was relatively small, so it was left in its dimensionalized units, along with the path constraint of dynamic pressure. This means that dynamic pressure must be scaled before using it in a cost function. For all cost functions containing $\overline{q}$ in the thesis, the value of $\overline{q}$ is normalized by the max dive limit of 500 psf.

7.2.3 Tables

The aerodynamic properties for flight vehicles are normally stored in aerodynamic tables. These tables are the result of wind tunnel testing, computational fluid dynamics and/or actual flight-testing. The data contained in the longitudinal tables consists of the lift, drag, and pitching moment coefficients of the vehicle for different flight speeds, and control position deflections. For this problem, it was decided to use the actual aerodynamic tables in the optimization routine in order to capture trajectories designed around the aerodynamics of a true vehicle, in this case the X-34. However, it is computationally intensive for DIDO to numerically calculate the Jacobian matrix from table data, especially if the multi-dimensional tables are not smooth, as will be the case for the X-34 as it flies through Mach one.

Several options are available in order to reduce the numerical complexity of the problem to a manageable size. One solution involves fitting a spline to the data. However, for every iteration made by the optimizer, a spline would have to be re-computed, increasing the computation time. Another possible solution involves calculating analytic functions that are representative of the data. This method defeats the purpose of using the "actual" table data. The solution chosen for this thesis involves reducing the three-dimensional data to two dimensions by setting the X-34's speedbrake to 55 degrees and trimming the vehicle at every node in the trajectory. A new table is then produced that contains the reduced aerodynamic data spread over an evenly spaced grid. This allows the use of a linear interpolation function in MATLAB that is especially designed for evenly spaced tables, and reduces the computation time to a reasonable time frame.

7.2.4 Nodal Determination

The number of nodes used in the optimization routine is an important parameter. The accuracy of the solution is directly correlated to the number of nodes. Theoretically, an infinite number of nodes would provide the best possible solution, in terms of accuracy. However, the tradeoff for using a large number of nodes is the speed of the optimization process. This is caused by the increased complexity of a large problem. By increasing the number of nodes, the size of the Jacobian increases. This in turn slows the routine because of the large computational burden placed upon the numerical solver to calculate the derivatives for every entry in the Jacobian matrix. Therefore, the process for deciding the nodal size is a balancing of the desired solution accuracy with a reasonable solution time.

For this trajectory problem, the number of nodes was set between 50 and 60 nodes. This size was determined through a trial and error process that was primarily concerned with time. The number of nodes was increased until the time of the solution made it infeasible to obtain the number of results needed for analysis. A trajectory of 60 nodes can take anywhere between 12 and 30 hours to solve. Adding more nodes would only make it worse. In comparison, this nodal size is about half the number of nodes used by Proto-snake. A typical Proto-snake trajectory is created with nodes placed at every 1000 ft increment in altitude space. This, on average, yields trajectories consisting of about 100 nodes that take less than 5 minutes to design. DIDO may take up to a week to design a 100 node trajectory.

7.2.5 Soft-Knot

A soft-knot is a pseudo-event defined by state continuity. It is used to improve the accuracy during a change or switch in the dynamics or controls at a certain time, without increasing the number of points in the solution. For this problem, the vehicle flies through Mach one, where the spike in the aerodynamic data occurs. This is a perfect spot to add a soft knot. In this case, a *free* soft knot is added, where the location of the knot is determined as part of the optimization routine. The optimizer may or may not place the soft knot at the Mach one point, but it will choose a location that yields the best solution and improves overall accuracy [14].

The soft knot can be thought of as splitting the optimized trajectory into two parts, one before the knot and one afterwards. The optimizer changes each part of the trajectory, while maintaining state continuity over the knot, in order to calculate the best solution. In a trajectory without a soft knot, the trajectory nodes lie at the Legendre-Gauss-Lobatto (LGL) points spread out over the entire solution. However, with a soft knot added, the nodes of the first part of the trajectory lie at the LGL points spread over the first trajectory segment up to the knot, while the remaining nodes lie at the LGL points spread over the second trajectory segment after the knot. This in a way shows that there are two solutions, which are joined at the knot to make one overall solution. Recall that the LGL points have a characteristic where the nodes "bunch up" near the ends (see Figure 6.1). With a soft knot in place, this means that the nodes "bunch up" at the ends and also near the soft knot, which accounts for the increased accuracy near the knot. For this problem, 35 nodes were always delegated to the trajectory segment before the soft knot. The remaining nodes, either 15 or 25, were assigned to the segment after the knot.

While the soft-knot guarantees state continuity throughout the solution, it does not necessarily assure continuity of the controls over the knot. This means that the vehicle's trajectory controls, α and μ, may be discontinuous at the soft knot. A discontinuous control reference is not a desirable solution from a guidance and control standpoint in regards to aerodynamic flight controls. Therefore, the controls chosen for this trajectory problem were $\dot{\alpha}$ and $\dot{\mu}$, as described in the subsection 7.2.1.1, in order to overcome this problem. Angle of attack and bank angle were added to the states in order to make them continuous over the knot and ensure trajectory control continuity. For more information on soft knots, please see Reference 17.

7.3 Cost Function Formulation

7.3.1 Proto-Snake Characterization and Verification

The main purpose of an optimization routine is the minimization of the cost or objective function. The characteristics of the solution are directly dependent upon the parameters contained within the cost function. These include any combination of the states and/or controls, or functions of the states and controls. Therefore, in order to generate trajectories through optimization, it is necessary to formulate an appropriate cost function. For the Legendre Pseudospectral Method, this cost function is also known as the Bolza cost functional and can be written as:

$$J[\mathbf{x}(\cdot),\mathbf{u}(\cdot),\tau_0,\tau_f]=\phi\left(\mathbf{x}(\tau_0),\mathbf{x}(\tau_f),\tau_0,\tau_f\right)+\int_{\tau_0}^{\tau_f}L(\mathbf{x}(\tau),\mathbf{u}(\tau),\tau)\,d\tau \qquad \textbf{(7. 14)}$$

Note that the end point cost, denoted by ϕ, is a function of the state vector at the initial and final clock times. This means that this portion of the cost function imposes constraints on the initial and final states. However, in this particular problem formulation, the constraints on the initial and final conditions are determined by the event constraints mentioned previously. Therefore, the only portion of the cost function that needs to be established is the integrand or Lagrange cost denoted by L.

The first cost function formulated was used to verify the implementation of the Proto-snake and DIDO methodologies. The goal was to force the DIDO optimizer to follow the same lateral and longitudinal geometry established by Proto-snake in order to compare the resulting dynamics and controls. If DIDO produces optimal and feasible trajectories, then the dynamics and controls determined by Proto-snake should match. The converse is also true. If Proto-snake creates intrinsically flyable and accurate trajectories, and DIDO is working properly, than both programs will produce the same results. This verification was carried out first, in order to make sure both programs were in agreement and so that the future results from DIDO can be applied to Proto-snake. The verification cost function can be written as:

$$J=\int_{t_0}^{t_f}\left[0.75(h-h_{ptp})^2+0.25(y-y_{ptp})^2\right]dt \qquad \textbf{(7. 15)}$$

The subscript "$_{ptp}$" refers to Proto-snake trajectory profiles. These references are fed into DIDO by interpolating the appropriate trajectory parameter profiles at the nodes being used during the optimization. Therefore, the Proto-snake references and the state variables are the same size and are oriented to the same nodes. The first term in the cost function forces the altitude state to match Proto-snake's altitude profile at every node, while the second term forces the crossrange state to match. In this way, the optimized trajectory is forced to match both the lateral and longitudinal geometry of a Proto-snake trajectory. The weighting chosen for both terms is a design choice. It was decided that

the longitudinal portion is a more important parameter and should be weighted heavier than the lateral component. This decision is carried out in all remaining trajectories.

Figure 7.1 shows a comparison of a Proto-snake trajectory and a DIDO trajectory. It is apparent from the graphs that both the geometry and the control variable α are nearly identical. The remaining states and the bank angle control variable also compare favorably but are not displayed here. It should be noted that Proto-snake uses KEP to balance the equations through a manipulation of both angle of attack and bank angle, in order to create a trajectory around a given geometry. For this cost function, DIDO has done the same thing. It determined the controls, α and μ, necessary to minimize the cost function, which in this case results in a matching of the reference geometry fed into the cost. The fact that the resulting control histories for the constrained geometry are nearly identical verifies the implementation of both methodologies.

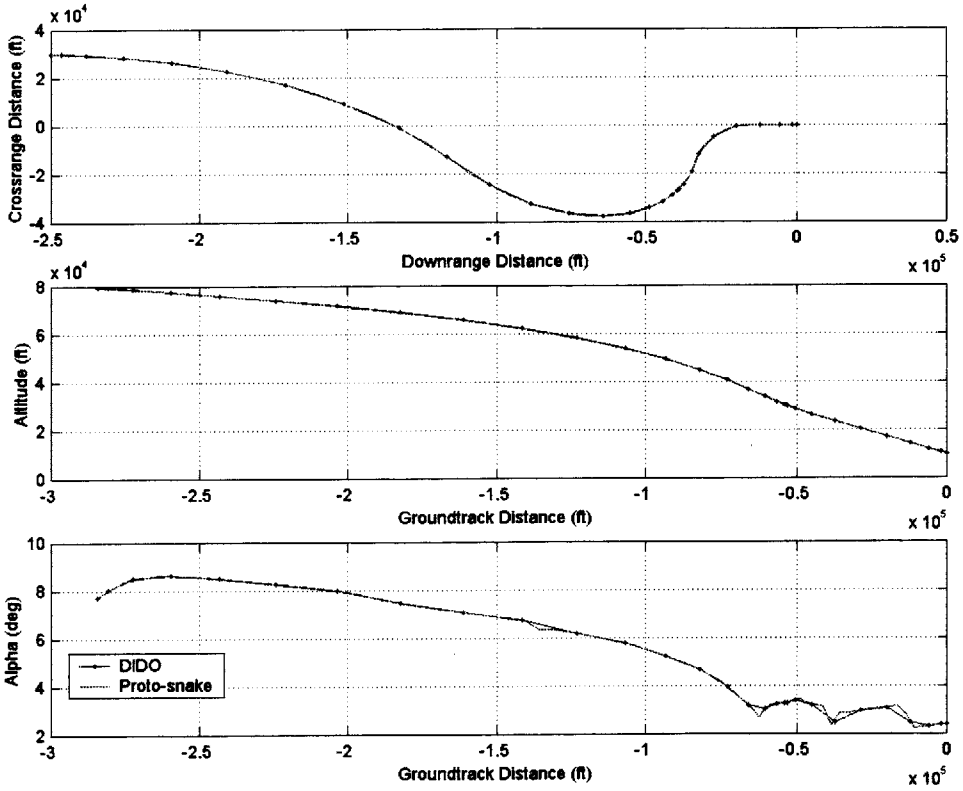


Figure 7.1: Comparison of DIDO Verification Trajectory to a Proto-snake Trajectory

The next cost function was established in an attempt to characterize the optimality of Proto-snake trajectories. Because Proto-snake's ground track solver (GTS) uses an ad hoc approach to solving the lateral ground track, it is not known if the resulting trajectories are optimal for a given set of parameters. The only way to determine this is to create trajectories for a given cost function and see how they compare. If the trajectories come out close or even match, then the GTS theoretically designs sub-optimal trajectories about the same parameters used in the cost function.

The cost function created for the characterization task is formulated to follow the same logic utilized by Proto-snake's GTS. As described in Chapter 5, the GTS manipulates a predicted ground track, produced from following a dynamic pressure schedule, to meet the terminal conditions. The resulting shape initially bends towards the runway in order to eliminate any crossrange error as quickly as possible. The characterization cost function *idealized* to capture the same effect is:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.4(y)^2 \right] dt \qquad \textbf{(7. 16)}$$

The subscript "$_{ref}$" refers to a reference value of $\bar{q}$ for a constant $\bar{q}$ profile. This $\bar{q}$ reference can also be replaced by a $\bar{q}$ schedule, which is implemented in much the same way as the Proto-snake trajectory profiles used in the previous cost function. The first term of this cost function forces the trajectories designed to follow a set $\bar{q}$ schedule while the second term tries to eliminate crossrange errors.

7.3.2 Robust Trajectories

7.3.2.1 Goals

The focus of this thesis is to design robust trajectories that can be used as benchmarks for onboard trajectory generators. Once again, a robust trajectory is defined as a trajectory that can handle future dispersions while still providing the means for the flight vehicle to attain the final end condition. Because the trajectories are being created through the use of an optimization routine, the easiest way to create robust trajectories is to input the negative of the "robustness parameter" in the cost function. This would have the effect of producing a trajectory with maximum robustness. Unfortunately, there is no such thing

as a "robustness parameter" or variable. Instead, cost functions must be produced through the clever arrangement of different parameters in order to obtain different trajectories. Then, a metric must be created in order to compare the different trajectories and to classify robustness. Through the design of scores of cost functions, it is hoped to establish certain parameters that govern both trajectory geometry and robustness.

7.3.2.2 Cost Layout

The parameters in the cost function control the outcome and shape of the optimized solution. For a trajectory optimization problem, these parameters can almost be thought of as a type of guidance for the resulting trajectory. Because of this it is necessary to include appropriate trajectory control parameters in every cost function. A flight through the TAEM region involves both lateral and longitudinal dynamics. Therefore, in order to control a trajectory through this region, the cost function should be made up of both lateral and longitudinal parameters.

The basic format for all trajectory cost functions used for benchmark determination in this thesis is written as:

$$J = \int_{t_0}^{t_f} \left[0.6\left(\text{Longitudinal Parameters}\right)^2 + 0.4\left(\text{Lateral Parameters}\right)^2 \right] dt \qquad (7.17)$$

The longitudinal parameters are defined as a variable or combination of variables that have a direct influence on longitudinal motion or position of the vehicle throughout the trajectory. They can be made up of any of the vehicle states, controls or functions of the states and controls. The longitudinal parameters chosen for use are listed in Table 7.2. These were chosen to create trajectories in the same fashion as Proto-snake. That is, to be able to create trajectories that follow the center of the energy corridor $\bar{q}$ schedule, as well as trajectories that fly on the max-dive and max-glide $\bar{q}$ limits. These three types of trajectories are referred to as center-of-corridor, max-glide and max-dive trajectories. By choosing these longitudinal parameters, the DIDO trajectories can be compared against Proto-snake trajectories of the same type, whose simple designs are easily implemented in an onboard guidance algorithm. In addition, the different trajectories will show the variances of different dynamic pressure schedules amongst optimal trajectories. This will

help in determining what type of $\overline{q}$ schedule should be used by Proto-snake during the initial phase of trajectory formulation.

Table 7.3: Longitudinal Parameters

Type of Trajectory	Parameter
Center-of-Corridor	$(\overline{q}-335)$
Max Glide (maximize E/W)	$(-E/W)$
Max Dive (minimize E/W)	(E/W)

The lateral parameters of the cost function are made up of those combinations of states and/or controls that directly influence the lateral motion or position of the vehicle throughout the trajectory. Unlike the longitudinal parameters, where there are three main terms, there are no clear overriding lateral terms that directly control the entire lateral space. Therefore, this portion of the cost function is made up of multiple combinations. Table 7.4 lists the possible lateral parameters. The actual combinations of variables used in the cost function formulations are described in the next section.

Table 7.4: Lateral Parameters

Description	Parameter
Heading Angle	χ
Heading Angle Rate	$\dot{\chi}$
Bank Angle	μ
Bank Angle Rate	$\dot{\mu}$
Crossrange Distance	y
Body z-acceleration	Nz_b

7.4 Trajectory Selection / Benchmark Selection

7.4.1 Desirability Criterion

There are certain trajectory attributes that are desirable in the trajectories created through DIDO. These attributes, or desirability criterion, were determined from an intuitive knowledge of the trajectory problem, including an overarching concept of what might define robust characteristics. Table 7.5 includes a list of the desirability criterion and explanations.

Table 7.5: Desirability Criterion

Criterion	Explanation
Smooth Trajectory	No major oscillations or radical state changes, and smooth control histories (real-time flight control virtues)
Low Dynamic Loads	Reduced body accelerations, particularly Nz_b
Follows $\overline{q}$ Schedule	Follows a type of schedule that can easily be mimicked by Proto-snake for future applications in a real-time guidance scheme
Time on Centerline	Reduced crossrange errors. Spends more time flying towards the runway centerline

This set of criterion was established to help in the down-selection of 'robust' trajectories from a broad test matrix. In this manner, a trajectory that is oscillatory and choppy can be eliminated over trajectories that are very smooth and continuous. In addition, these attributes were also used in the formulation of cost functions. For instance, the body z-acceleration term was added as a lateral parameter in the cost functions in an attempt to meet the low dynamic load criterion. Also, the establishment of this criterion led to the consideration of using $\dot{\alpha}$ in some of the cost functions. This term helps to smooth the control history of respective trajectories. When implemented, it is grouped within the lateral parameter term and equally shares the 0.4 lateral weighting shown in Eq. 7.17. However, experimentation has shown that $\dot{\alpha}$ should actually be increased by a factor of 50 to return the desired results. An example of $\dot{\alpha}$ in a cost function is:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(y)^2 \right] dt \qquad \textbf{(7. 18)}$$

where the lateral parameter can be written as $(50\dot{\alpha})^2 + (y)^2$. For simplicity sake, this 50 is not mentioned when describing the $\dot{\alpha}$ term. However, it is included in every cost function utilizing $\dot{\alpha}$.

7.4.2 Test Layout

The test layout chosen for the determination of benchmark trajectories involved testing a wide variety of cost functions and analyzing the resulting solutions. This process was initiated by creating cost functions made up of combinations of longitudinal and lateral parameters. These cost functions were used to create trajectories and gain insight into what type of influence the different parameters have over the trajectory characteristics. Initially, the cost functions were written by choosing a longitudinal parameter and joining it with a "family" of lateral parameters. For example, the center-of-corridor parameter joined with the "chi-family" of lateral parameters would yield some of the cost functions written in Eqs 7.19. Note: for simplicity sake, the cost functions are abbreviated by only writing the pertinent parameters, and not the full mathematical expression and appropriate weightings.

$$\begin{aligned} J &= \int (\bar{q} - 335)^2 + \chi^2 && \text{or} \\ J &= \int (\bar{q} - 335)^2 + \dot{\chi}^2 && \text{or} \\ J &= \int (\bar{q} - 335)^2 + \chi^2 + \dot{\chi}^2 && \text{or} \\ J &= \int (\bar{q} - 335)^2 + (\chi - \chi_{ref})^2 && \text{etc.} \end{aligned} \qquad \textbf{(7. 19)}$$

This type of family exploration led to the creation of 30 different cost functions. Tables 7.6, 7.7 and 7.8 present these cost functions, organized by longitudinal parameter. As a reminder, the cost is made up of a longitudinal parameter and the listed combinations of the lateral parameters. All but 3 of the cost functions contain the $\dot{\alpha}$ term.

Table 7.6: Center of Corridor Cost Functions

$J = \int 0.6(\bar{q} - 335)^2 + 0.4(....[below]....)$	
$0.5(\dot{\alpha})^2 + 0.5(\dot{\mu})^2$	$0.5(\dot{\alpha})^2 + 0.5(\mu - 20°)^2$
$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 - .25(\mu)^2$	$0.5(\dot{\alpha})^2 + 0.5(\mu - 40°)^2$
$0.5(\dot{\alpha})^2 + 0.5(\chi)^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(\mu - 20°)^2$
$0.5(\dot{\alpha})^2 + 0.5(\dot{\chi})^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(\mu - 40°)^2$
$0.5(\dot{\alpha})^2 + 0.5(\chi - \chi_{ref})^2$	$0.5(\dot{\alpha})^2 + 0.5(y)^2$
$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(\chi)^2$	$(y)^2$
$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(\dot{\chi})^2$	$\left((y)^2\right)^*$
$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(\chi - \chi_{ref})^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(y)^2$
$(Nz_b)^2$	$\left(0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(y)^2\right)^*$
$0.5(\dot{\alpha})^2 + 0.5(Nz_b)^2$	$\left(0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(Nz_b)^2\right)^*$
$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(Nz_b)^2$	

Note that the " * " refers to cases when the dynamic pressure scaling was removed. This had the effect of weighting the longitudinal parameter more than the lateral parameter.

Table 7.7: Max Glide Cost Functions

$J = \int -0.6(E/W)^2 + 0.4(....[below]....)$	
$0.5(\dot{\alpha})^2 + 0.5(\dot{\mu})^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(Nz_b)^2$
$0.5(\dot{\alpha})^2 + 0.5(\mu)^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(y)^2$
$0.5(\dot{\alpha})^2 + 0.5(y)^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(\dot{\chi})^2$

Table 7.8: Max Dive Cost Functions

$J = \int 0.6(E/W)^2 + 0.4(....[below]....)$	
$0.5(\dot{\alpha})^2 + 0.5(\dot{\mu})^2$	$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(Nz_b)^2$
$0.5(\dot{\alpha})^2 + 0.25(\dot{\mu})^2 + 0.25(y)^2$	

The cost functions for the max glide and max dive trajectories were established after analyzing most of the results from the center of corridor trajectories. This gave insight into which parameters were more desirable and effective, and helped reduce the number of combinations for those cases significantly.

All thirty cost functions were used to generate trajectories between the initial and terminal conditions given from a Proto-snake reference trajectory that is used to specify the DIDO event constraints. Four reference trajectories were established from Proto-snake. These trajectories are referred to by numbers corresponding to different starting points. Figure 7.2 is a crossrange versus downrange plot, or top down view of the four trajectories. The trajectory used for the first 30 cost functions is the point 1 reference.

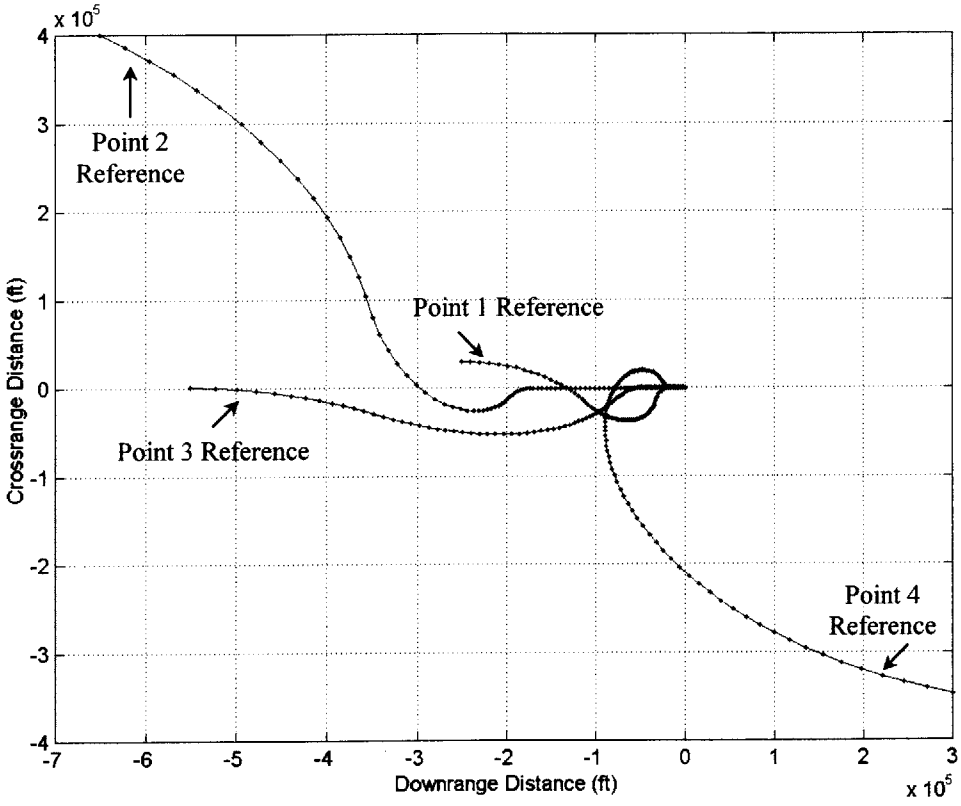


Figure 7.2: Proto-snake Reference Trajectories

Table 7.9 describes the initial conditions for the Proto-snake references. Each of the four starting points and initial headings were chosen to be distinctly different from each other.

This helps to test the capabilities of the cost functions and to ensure that the results for a cost function are not "initial condition specific". Note that all of the reference trajectories have the same "snaking" trend that is characteristic of the Proto-snake GTS. Both point 1 and point 4 references are designed to be center-of-corridor trajectories. The point 2 and point 3 references are max glide and max dive trajectories, respectively. Both points have an initial and final dynamic pressure of 335 psf and follow the dynamic pressure schedules depicted in Figure 5.8.

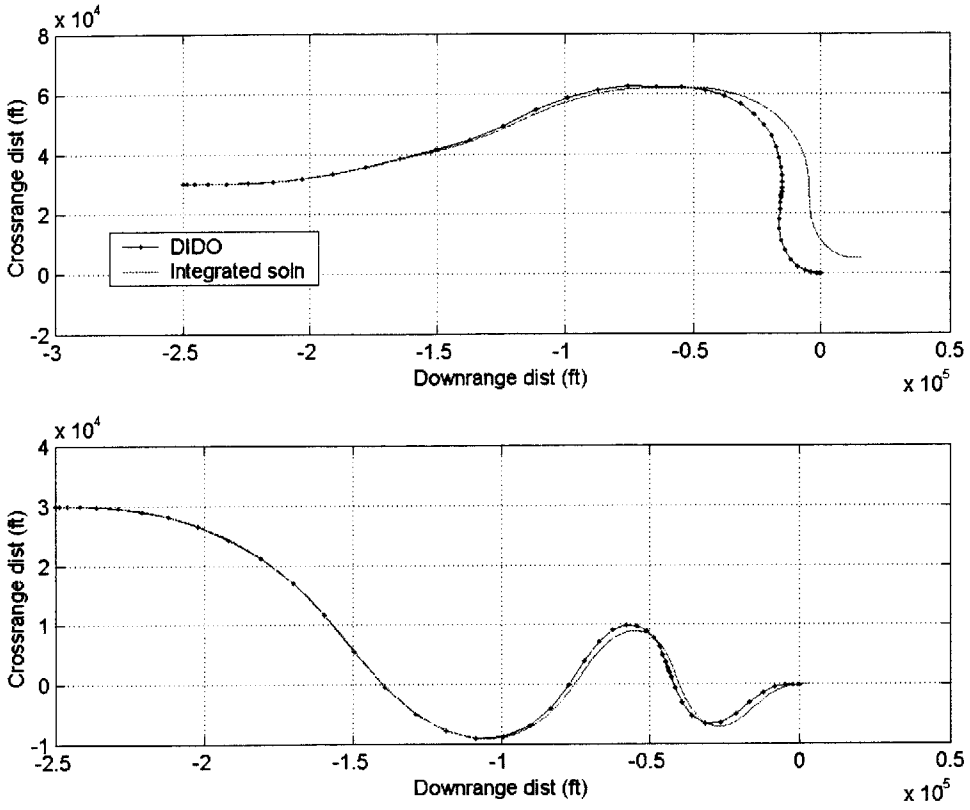
Table 7.9: Initial Conditions of Proto-snake Reference Trajectories

Parameter	Point 1	Point 2	Point 3	Point 4
Altitude, h	80,000 ft	100,000 ft	100,000 ft	100,000 ft
Downrange Distance, x	-250,000 ft	-650,000 ft	-550,000 ft	300,000 ft
Crossrange Distance, y	30,000 ft	400,000 ft	1,000 ft	-347,000 ft
Heading Angle, χ	0°	-25°	0°	170°
Mach Number, M	3.1	4.9	4.9	4.9
$\bar{q}$ Schedule (psf)	Constant 335	$\begin{bmatrix} 335 \\ 110 \\ 335 \end{bmatrix}$	$\begin{bmatrix} 335 \\ 500 \\ 335 \end{bmatrix}$	Constant 335

Unfortunately, running the thirty different cost functions resulted in the generation of thirty different trajectory characteristics. It became apparent that it was necessary to reduce the size of this test matrix to something more reasonable, before being able to choose the benchmark trajectory. Thus, an initial down-selection was done to reduce the option-space by applying the desirability criterion presented in subsection 7.4.1.

The first criterion was for smooth trajectories. The easiest way to determine which trajectories are smooth is to integrate the control variables of the trajectory in the time domain. If the controls are sufficiently smooth, an integration of the controls will yield the exact same states and dynamics as included in the solution. The trajectories can then be compared to determine what variables of the cost function dictated this "smoothness". The most obvious parameters that guarantee smooth controls are the control parameters themselves, which proved in fact to be the case. The use of $\dot{\alpha}$ and $\dot{\mu}$ in the cost function produced the smoothest trajectories. Figure 7.3 shows a comparison between trajectories

with and without $\dot{\alpha}$ and $\dot{\mu}$ in the cost function. The graph on the top is the trajectory solution created without $\dot{\alpha}$ and $\dot{\mu}$ in the cost function. The graph on the bottom is the trajectory solution created with $\dot{\alpha}$ and $\dot{\mu}$ in the cost function. Both graphs are plotted against a trajectory shape resulting from the integration of their respective control



histories.

Figure 7.3: Effect of $\dot{\alpha}$ and $\dot{\mu}$ in the Cost Functions

The top graph shows that without $\dot{\alpha}$ and $\dot{\mu}$ in the cost function, an integration of the trajectory controls will not yield the same solution. There is so much error or lost information over the time span that the integrated solution does not even end at the terminal condition. The bottom graph shows a dramatic improvement in the integrated solution and a change in the lateral characteristics when including the "smoothing" costs.

Figure 7.4 presents graphs comparing the control histories for the trajectories in figure 7.3. Note the much-improved behavior when including the "smoothing" costs, particularly in the bank angle profile. Based on these results, all trajectories without $\dot{\alpha}$

and $\dot{\mu}$ in the cost function were eliminated from the test matrix. This left eighteen cost functions after the first down-select.

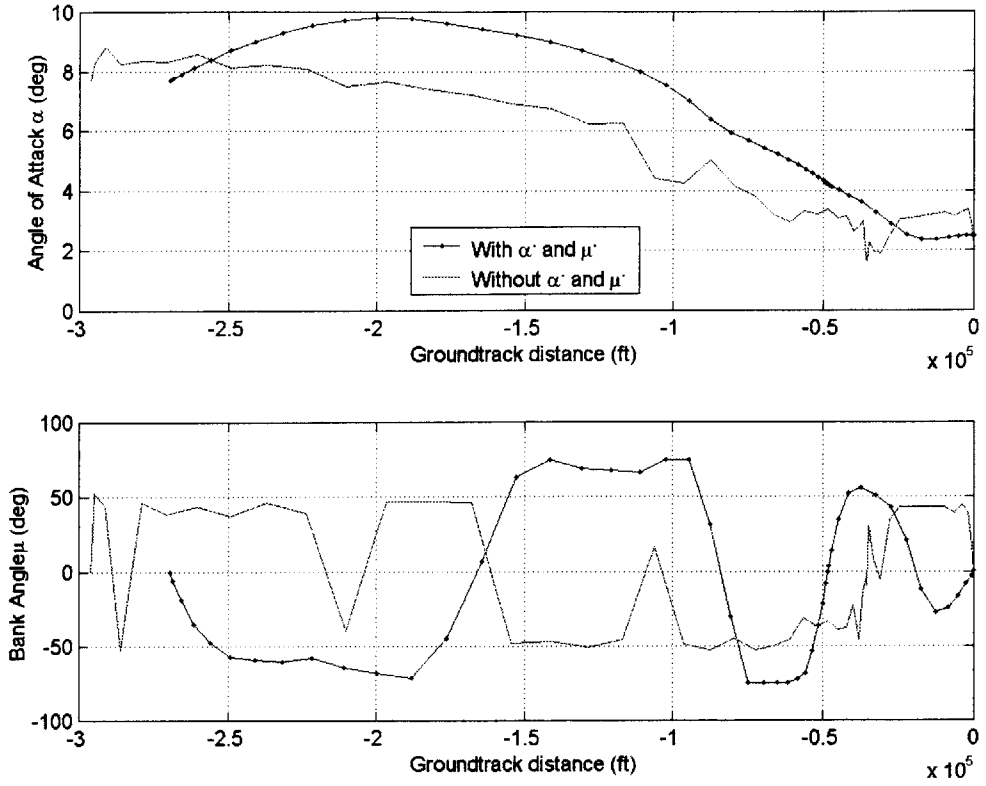


Figure 7.4: Effect of $\dot{\alpha}$ and $\dot{\mu}$ in "Smoothing" α and μ

The second criterion for down-selection consisted of eliminating cost functions that contained intuitively undesirable design characteristics. Included in that category are cost functions with constant bank angle and arbitrary heading angle references, in addition to the cost functions where the dynamic pressure was left un-scaled. It was also decided that cost functions that minimize $\dot{\chi}$ or $\dot{\mu}$ have the effect of also minimizing χ and μ. Therefore, the cost functions with just χ or μ were removed. At the end of this down-selection, eleven cost functions remained and are presented in Table 7.10.

Table 7.10: Cost Functions Remaining After Second Down-Selection

Longitudinal Parameter → **Lateral Parameter** ↓	$(\overline{q}-335)$ *Center of Corridor*	$(-E/W)$ *Max Glide*	(E/W) *Max Dive*
$\dot{\alpha}+\dot{\mu}$	X	X	X
$\dot{\alpha}+\dot{\mu}+Nz_b$	X	X	X
$\dot{\alpha}+\dot{\mu}+\dot{y}$	X	X	X
$\dot{\alpha}+\dot{\mu}+\dot{\chi}$	X	X	

The weightings for the parameters, including the factor of 50 on $\dot{\alpha}$, are not indicated in the table. This makes it easier to see the general layout of the cost functions. The X's in the table signify the existence of a single cost function and its corresponding trajectory. A cost function is a combination of a row and column of the table. Therefore the cost function corresponding to the first entry in the table can be written as:

$$J=\int_{t_0}^{t_f}\left[0.6(\overline{q}-335)^2+0.2(50\dot{\alpha})^2+0.2(\dot{\mu})^2\right]dt \qquad \textbf{(7. 20)}$$

The third criterion for the down-selection involved a critical look at the shapes of the eleven trajectories. The focus this time was on how the overall trajectory looked with regards to its dynamic loading, dynamic pressure profile and crossrange errors. This process was explicitly subjective with regards to how the down-selection was made. Even though all the parameters of a trajectory were analyzed, there was no quantifying measure used to distinguish one trajectory from another one. It was simply a designer's choice. Reasons selected for eliminating trajectories during this down-selection are described below.

The two primary reasons for eliminating trajectories included those that had, on average, higher g-loads or a fluctuating dynamic pressure profile over the course of the flight. An example of this is shown in Figure 7.5. It is desired that the resulting trajectories follow a fairly constant or well-behaved dynamic pressure profile. The goal of this research is to establish the best $\overline{q}$ schedule for easy implementation in an onboard guidance algorithm. Once established, the onboard generators could be programmed to mimic these

schedules. Although the "nearly constant dynamic pressure profile" in Figure 7.5 is oscillatory, for all practical purposes, it can be approximated by a constant $\overline{q}$ schedule. However, it would be undesirable to duplicate a random fluctuating $\overline{q}$ schedule. Therefore, the cost functions that resulted in these varying $\overline{q}$ schedules were eliminated.

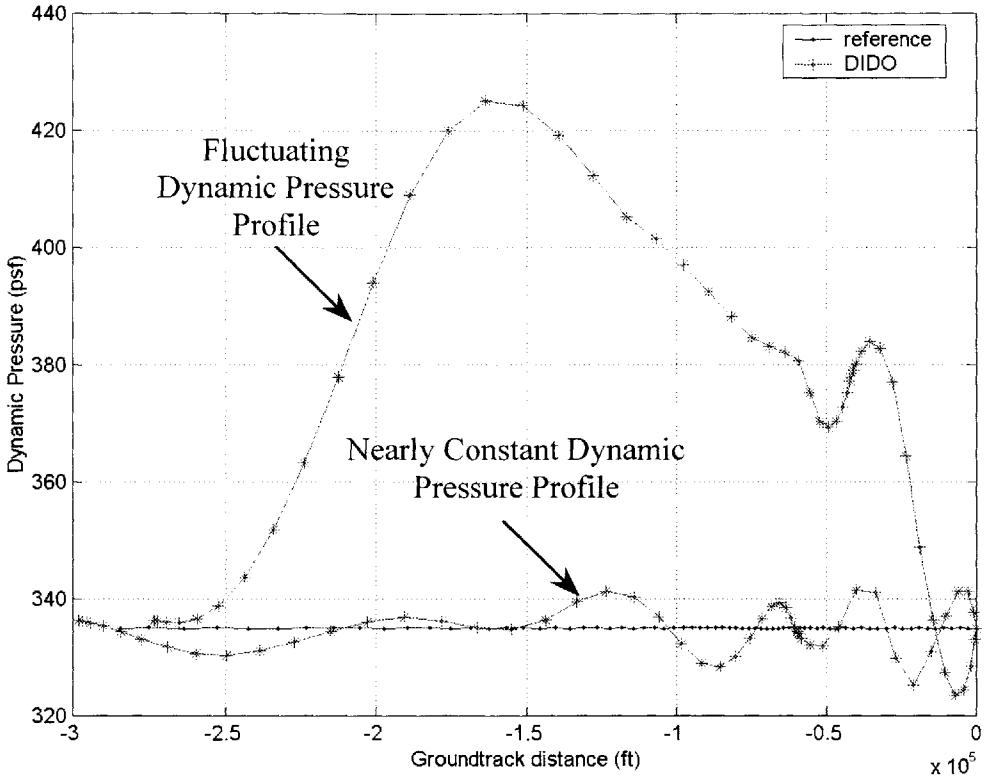


Figure 7.5: Comparison of Different Dynamic Pressure Profiles

After the third down-selection, there are only seven cost functions remaining. The cost functions are listed in Table 7.11 along with appropriate names. This naming scheme is meant to quickly describe the cost function and all resulting trajectories. The names result from the components of the cost function. A trajectory created by that cost function is referred to by the cost function name and the number of the Proto-snake reference trajectory it was designed around. For example, *qam* is used to name the cost function of Eq 7.20. The *q* stands for the dynamic pressure term, the *a* stands for $\dot{\alpha}$ and the *m* stands for $\dot{\mu}$. If this cost function is used to create a trajectory around the point 1 reference, the resulting trajectory is named *qam1*.

Table 7.11: Cost Functions Remaining After Third Down-Selection

Longitudinal Parameter → / **Lateral Parameter** ↓	$(\bar{q}-335)$ Center of Corridor	$(-E/W)$ Max Glide	(E/W) Max Dive
$\dot{\alpha}+\dot{\mu}$	*qam*		
$\dot{\alpha}+\dot{\mu}+Nz_b$			*minEamnz*
$\dot{\alpha}+\dot{\mu}+y$	*qamy*	*maxEamy*	*minEamy*
$\dot{\alpha}+\dot{\mu}+\dot{\chi}$	*qamc*	*maxEamc*	

These seven cost functions are the primary focus of the test plan. All give insight into the characterization of robust trajectories, in addition to stimulating ideas for improving Proto-snake. In order to complete the test plan, these cost functions were run at the other three reference points in order to yield a total of four trajectories per cost function. This allowed collection of more data for each cost function, which will hopefully lead to better results and conclusions.

7.4.3 Scoring / Grading

To complete the research, it is necessary to select which cost function produces the desired benchmark characteristics. This involves one final down-selection. However, the process should involve a more objective approach, rather than choosing which one "looks" better. This requires determining quantifiable and measurable characteristics for each trajectory that can be repeated from test to test. The method used is common in systems engineering when down-selecting amongst various system architectures, whereby a scoring system is applied to certain variables of a particular problem. This results in a raw score for each characteristic or attribute in the trajectories. Then a weighting scheme is applied to the attributes in order to assign an order of precedence when adding the scores. The weighted scores are added which results in the assignment of a grade to each trajectory. The lowest scoring trajectory has the best mix of desirable characteristics for that particular reference point, and is set as the winner for that point. The cost function with the most winning trajectories over the four reference points is selected as the benchmark cost function.

The process begins by first determining the attributes or parameters of the trajectories that are important. These parameters are actually dictated by the desirability criterion established in subsection 7.4.1. Table 7.12 lists the down-select parameters along with a description of the effect they have on the trajectories. These parameters were also chosen to classify robustness.

Table 7.12: Benchmark Selection / Robustness Parameters

Description of Effect in Cost Function	Parameter
Dictates $\overline{q}$ schedule, controls vehicle energy	$\overline{q}$
Regulates dynamic loads	Nz_h
Lateral controller, reduces dynamic loads	χ
Lateral controller, reduces crossrange errors	y
Smoothes trajectory solutions	$\dot{\alpha}$
Smoothes trajectory solutions	$\dot{\mu}$

The next step involves developing a scoring system that is applied to each parameter so that the trajectories can receive a score dependent upon those parameters. This also includes scaling the scores so that a $\overline{q}$ score is comparable in magnitude to a Nz_h score. The easiest way to accomplish this is to give each trajectory a score between 0 and 1 for each parameter. A score of 0 means that the trajectory has met the desirability criterion. A score of 1 means the trajectory has not satisfied the criterion and is undesirable. These scores are assigned to each node of the trajectory for a given parameter. The scores of the nodes are then averaged to give the total score for that parameter. The actual scoring assignment varies for each parameter, depending upon the desired results. The scoring assignment or scheme for each parameter is presented below.

For the dynamic pressure parameter, it is desired that the trajectory follow a center of corridor profile. This keeps the vehicle off the design limits in addition to guaranteeing the ability to change dynamic states dependent upon disturbances. For instance, if a vehicle follows a max dive trajectory profile and finds itself being pushed by a strong tailwind, it cannot dive any steeper or shorten the ground track any further due to the fact that it is riding the 500 psf limit. However, if the vehicle followed a center of corridor trajectory profile, than it would be able to dive to a different $\overline{q}$ value in order to handle the disturbance. Therefore a score of 1 will penalize trajectories designed on the 500 psf

limit or on the 110 psf limit. Figure 7.6 shows the scoring system applied to each node of the trajectory for the $\overline{q}$ parameter.

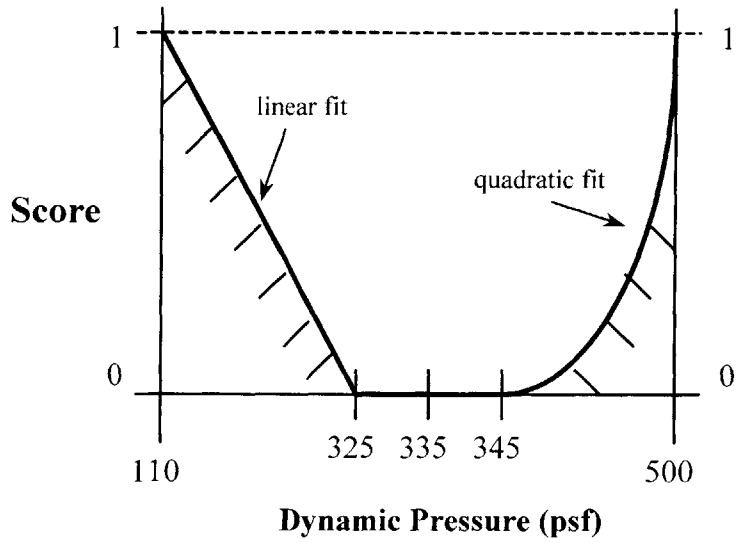


Figure 7.6: Dynamic Pressure Scoring Scheme

Each node of the dynamic pressure profile of the trajectory is analyzed and given a score. For this scheme, if the node lies between 325 and 345 psf, it receives a score of 0, which corresponds to a center of corridor profile. This small range is sort of a dead-band area to handle any noise or oscillations in the signal. If the node lies at 422 psf, it will receive a score of $\frac{1}{4}$, while a node at 218 psf will receive a score of $\frac{1}{2}$ and so on. All of the scores of the nodes are then averaged to give a total dynamic pressure score for that particular trajectory. For the dynamic pressure score, it was decided that having slightly higher values of $\overline{q}$ is better than being low on energy. That is why the upper values are governed by a quadratic curve and not by the linear curve applied to the lower $\overline{q}$ values. The reasoning behind this is that a vehicle is more likely to experience a loss in energy due to the effect of dispersions rather than a net gain in energy. Therefore, a vehicle that is at flying at a $\overline{q}$ of 400 psf is considered more robust than if it was flying at 300 psf. The scoring scheme takes this into account.

The scoring system for the Nz_b parameter is a little simpler. The goal here is to apply low scores to those trajectories that have low dynamic loading values. Proto-snake designs trajectories at an Nz_b value of 0.6, while the maximum value for the vehicle is 3.

However, if the vehicle carried humans returning from orbit, the lower the values the better. Therefore, the scoring scheme chosen is shown in Figure 7.7. This indicates a dead-band area of about 0 to 0.2. All values between 1 and 3 are given a full score of 1.

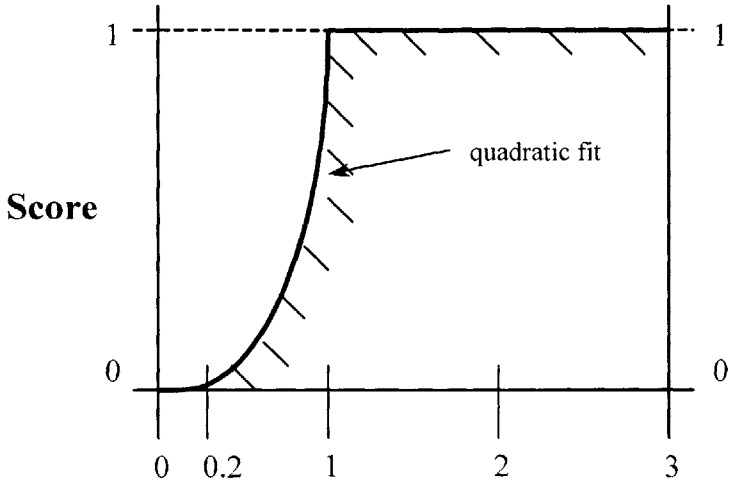


Figure 7.7: Nz_h Scoring Scheme

The scoring assignment for the χ and y parameters is slightly different. It is not clear what heading and crossrange errors should be penalized during the early flight regime, so no penalty is applied. Theoretically, a trajectory may take any shape to the ALI, so long as it meets the end constraints. However, those trajectories that remove most of the crossrange and heading errors early, and spend most of the time flying on the required end conditions should receive lower scores. These trajectories are considered more robust than trajectories that turn at the very last second to meet the boundary constraints. Therefore, only the nodes near the end of the flight should be given scores for the χ and y parameters.

There are different ways of selecting the nodes near the end of the trajectory. Assigning the weightings to the nodes once the vehicle reaches a certain ground track distance from the runway is one possibility. However, a vehicle flying faster will reach the ALI in a shorter amount of time and thus has less time to correct errors. Therefore, the "switch" for assigning weightings to the nodes should occur at a set time from the ALI. This gives each trajectory, independent of velocity, the same amount of time to reduce the crossrange and heading angle errors. The switch for this problem occurs when the trajectory has 80 seconds of flight time remaining. This corresponds to about 10 nautical miles of ground track distance to the ALI. Once this switch has occurred, the nodes for χ and y are given scores. The total score for the respective parameter is an average of those scores over the 80 seconds of flight. The scoring assignment for χ is shown in Figure 7.8.

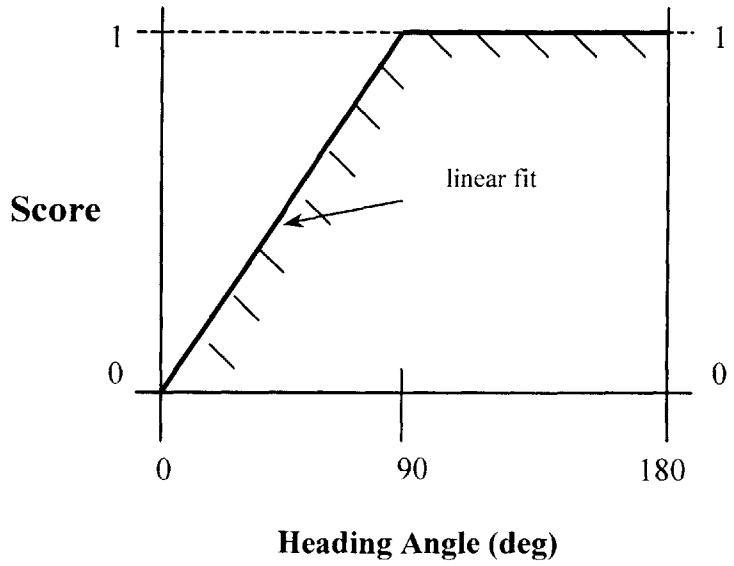


Figure 7.8: Heading Angle Scoring Scheme

A score of 0 is assigned to the node when it is aligned with the runway centerline. A score of 1 is applied to those nodes or trajectories that are aligned perpendicular to the runway centerline or are actually pointed away from the ALI. Turning away from the ALI is undesirable due to the effect it has on projecting vehicle energy away from the target area. If a sudden loss of energy occurs due to some type of dispersion, the ability to recover safely and effectively is drastically reduced. Therefore it is good practice to keep vehicle momentum and energy projected towards the ALI. The x-axis in the figure

corresponds to the absolute value of the χ parameter. A dead-band area is not included in this scoring scheme in order to penalize trajectories in which the heading angle is not absolutely aligned with the centerline.

The scoring scheme for the crossrange distance y is shown in Figure 7.9. This shows that a trajectory that has approximately zero crossrange error when it reaches the last 80 seconds of flight will receive a score of 0. Zero crossrange error corresponds to a trajectory lying within a wedge of five degrees about the runway centerline. This five degree wedge can be considered the dead-band area for this parameter. A score of 1 is assigned if the trajectory is outside a wedge of 20 degrees. These wedges actually extend in both directions from the ALI. In this manner, a trajectory that is riding the runway centerline, but approaching from the opposite direction (from the right) still receives a score of 0. However that trajectory will receive a score of 1 for the heading angle error it exhibits. The scoring between the two wedges is a linear curve, as shown in Figure 7.10. The value x is the trajectory's downrange distance from the ALI.

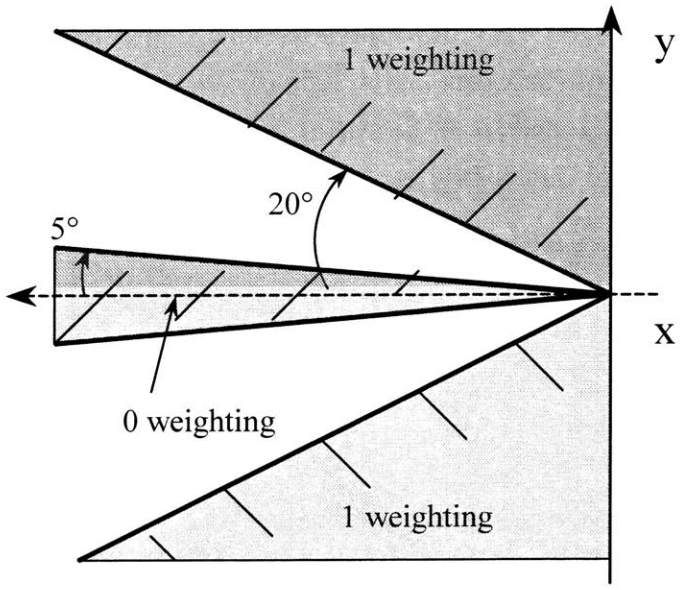


Figure 7.9: Crossrange Parameter Scoring Area

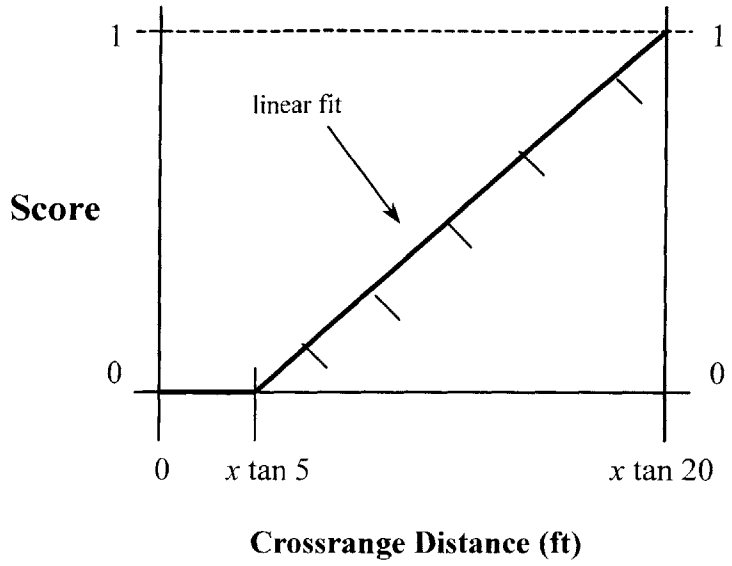


Figure 7.10: Crossrange Parameter Scoring Scheme

The scoring assignment for the last two parameters is nearly the same. The goal is to minimize both $\dot{\alpha}$ and $\dot{\mu}$ values for a trajectory. A score of 1 is applied to the nodes that lie above a chosen maximum value for each parameter. The dead-band area, where a score of 0 is applied, is set between 0 and 10% of the chosen maximum value. The scoring scheme for both $\dot{\alpha}$ and $\dot{\mu}$ is shown in Figure 7.11. The x-axis in the figure corresponds to the absolute value of the parameters.

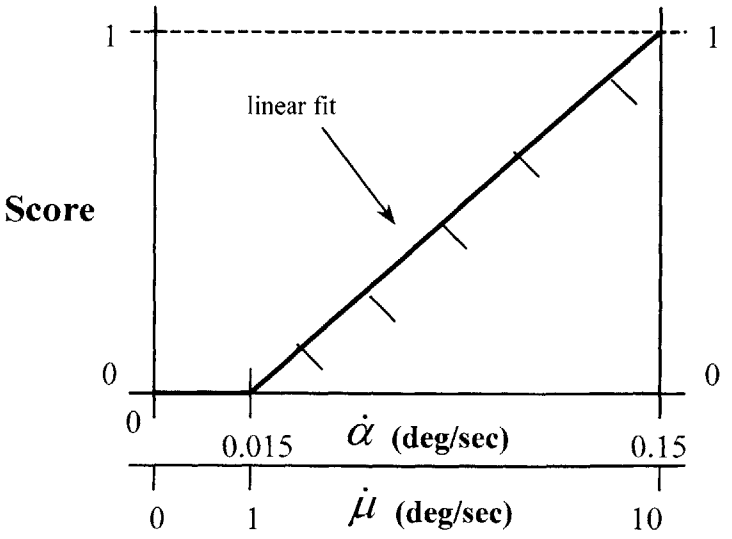


Figure 7.11: $\dot{\alpha}$ and $\dot{\mu}$ Scoring Scheme

With all the parameters of a given trajectory assigned a score, it is necessary to determine how the scores are added to result in a final grade. This is also a subjective process. The designer must apply a weighting to each parameter score that reflects how important the parameter is with regards to robustness. The weightings assigned to each parameter are listed in Table 7.13. This shows that the dynamic pressure profile is the most important attribute of a robust trajectory. The dynamic loading is the second most important parameter and so on. Therefore, a center of corridor trajectory will win over a max dive trajectory in most circumstances, even if all other parameter scores remain the same. However, it is still possible for a center of corridor trajectory to have an undesirable mix of benchmark attributes and score poorly. This scoring and grading system was used to determine the benchmark trajectories and the corresponding cost function.

Table 7.13: Scoring Parameter Weightings

Applied Weighting	Parameter
50	$\overline{q}$
25	Nz_h
10	χ
5	y
5	$\dot{\alpha}$
5	$\dot{\mu}$
100	

[This page intentionally left blank]

Chapter 8

Results

8.1 Overview

This chapter presents the results of the benchmark trajectory selection process. The first section describes a characterization of Proto-snake trajectories. The next section summarizes the scoring and grading results for each of the four reference points. Finally, a single cost function is presented for representing the formulation of robust trajectories, and serves as the benchmark for onboard trajectory generators. All of the results give insight into the characteristics of robust trajectories and provide an understanding of how to make improvements to Proto-snake.

8.2 Proto-Snake Characterization

The goal of the characterization cost function and resulting trajectories is to classify the optimality of the Proto-snake trajectories. As stated in Chapter 7, the initial cost function formulated for this task is:

$$J = \int_{t_0}^{t_f} \left[0.6(\overline{q} - \overline{q}_{ref})^2 + 0.4(y)^2 \right] dt \qquad \textbf{(8. 1)}$$

However, the trajectories that resulted from this cost function were choppy and did not follow the Proto-snake trajectories. In order to smooth the output from DIDO, $\dot{\alpha}$ and $\dot{\mu}$ were added to the cost function (as reported in section 7.4). This put the cost function in the form of:

$$J = \int_{t_0}^{t_f} \left[0.6(\overline{q} - \overline{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(0.5(\dot{\mu})^2 + 0.5(y)^2 \right] dt \qquad \textbf{(8. 2)}$$

This cost function is also known as *qamy* from the previous chapter, and is one of the seven that was tested for creating robust trajectories. However, the resulting trajectories from *qamy* did not match all the Proto-snake attributes. Different weightings were tried

and still the results did not prove fruitful. In the end, the Proto-snake characterization activity did provide valuable insight into the type of trajectories produced by Proto-snake, and was instrumental in developing the benchmark metrics. However, there were primary differences that precluded the characterization by a single cost function, as discussed below.

It was discovered that Proto-snake does not really minimize crossrange errors, in the exact sense. Figure 8.1 provides a comparison between two DIDO trajectories and the appropriate reference trajectories from Proto-snake. The cost function used in the creation of the DIDO trajectories is expressed in Eq 8.2. Note that the y parameter in the cost function forces the trajectory to stay as close to centerline as possible whereas Proto-snake only bends the trajectory towards the centerline initially. It will continue to overshoot and develop additional crossrange distance errors until the predicted ground track lies at the ALI. This "initial bend towards the runway" cannot be captured by a simple crossrange parameter in a cost function.

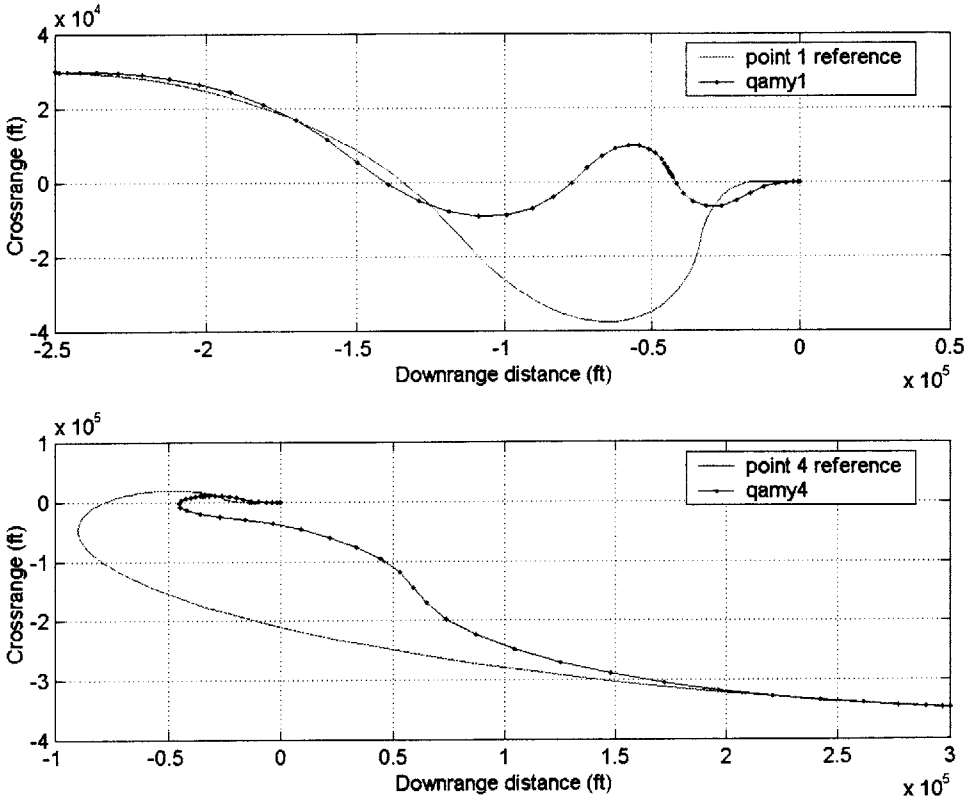


Figure 8.1: Effect of y Parameter in Cost Function

Another limiting factor in determining the characteristic cost function is the dynamic loading parameter, Nz_b. In Proto-snake's program formulation, a maximum Nz_b is set and used in the determination of the lateral ground track. In this case Nz_b is set at 0.6. When the trajectories are calculated, the entire turning profile occurs at the constrained value. The only time the trajectory is not bent at the constraint is near the end of the flight where only a slight adjustment may be need to bend the trajectory back to centerline. Figure 8.2 depicts a comparison between a Proto-snake trajectory and a optimized DIDO trajectory. It should be noted in the upper graph of the figure, that Proto-snake flew the entire trajectory at an Nz_b of 0.6, whereas the DIDO trajectory resulted in a varied Nz_b profile. This is significant, considering that both trajectories follow the same dynamic pressure schedule. Therefore, even though DIDO flew the same $\bar{q}$ schedule, it selected a different Nz_b profile in order to minimize the cost function.

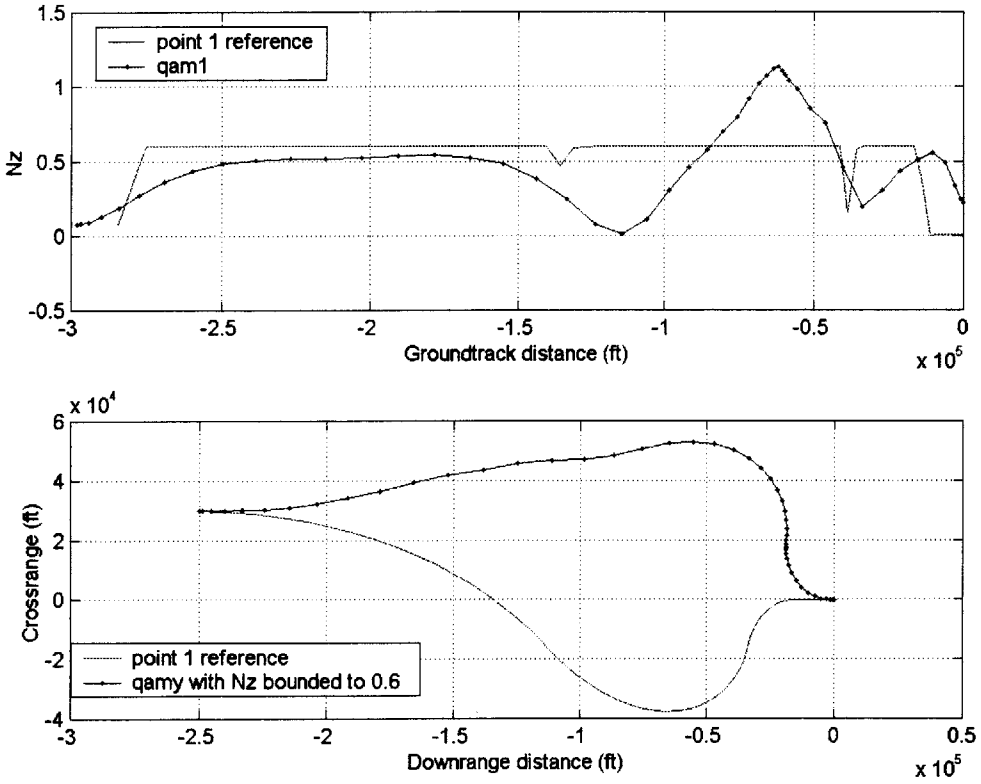


Figure 8.2: Nz_b Parameter Matching Issue

The bottom graph in Figure 8.2 shows a crossrange versus downrange plot of a DIDO trajectory and the point 1 reference. In this particular trajectory, the maximum Nz_b value in DIDO was constrained at 0.6. This shows that when DIDO was forced to fly on the

Nz_b constraint, it chose instead to release the $\overline{q}$ schedule in order to minimize the cost function. With a departure from the dynamic pressure schedule, the resulting trajectory actually ended up bending in the opposite direction from the reference. Therefore, constraining both $\overline{q}$ and Nz_b in Proto-snake precludes it from designing a more nearly optimal trajectory. Thus, Proto-snake trajectories are solutions of a constrained problem, which is not a bad ad hoc design approach. This is how the trajectory generator is able to generate the solutions so quickly. It reduces the design space to a set of constraints and generates trajectories along those constraints.

With these two factors limiting the initial characterization cost function, it was necessary to see if any other cost function embodied the trend of Proto-snake trajectories. The cost function *qam* actually exhibited similar characteristics. Figure 8.3 shows a comparison of *qam* and Proto-snake trajectories for reference points 1 and 4. These points were chosen because the Proto-snake references were designed to be center of corridor trajectories. This is consistent with *qam*, which optimizes about the center of the corridor.

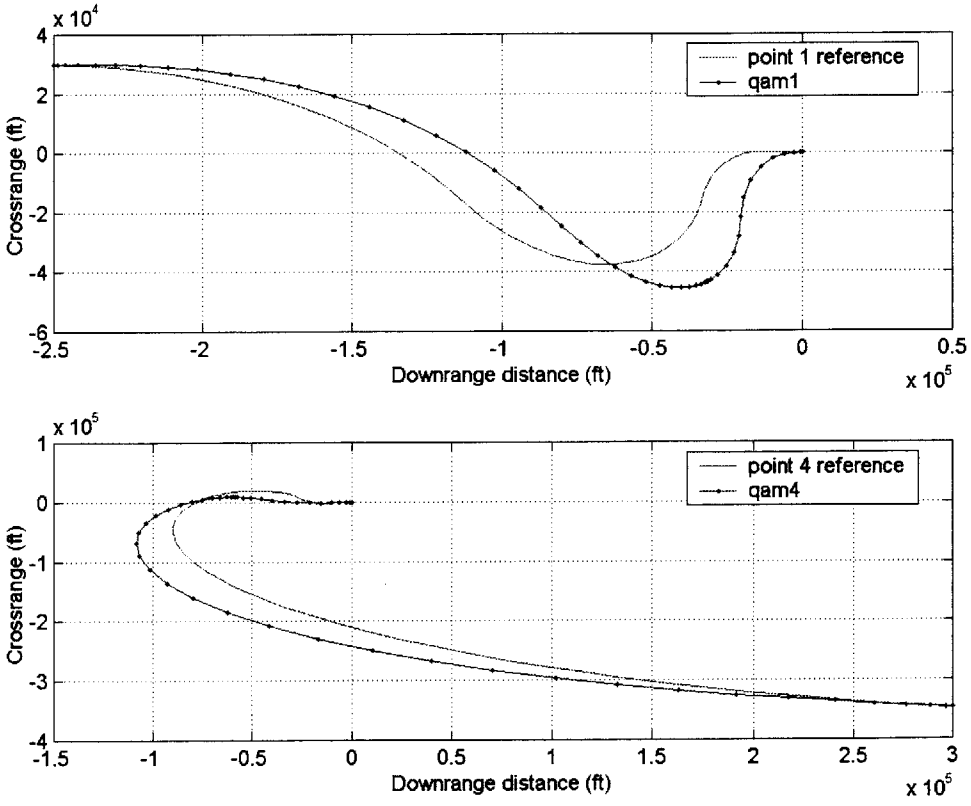


Figure 8.3: Comparison of (qam) Trajectories with Proto-snake

As can be seen in the Figure 8.3, both *qam* and Proto-snake generated similar trajectory attributes. The only real differences occurred in the slight variations of Nz_b and the dynamic pressure profiles. This verifies that Proto-snake is not simply minimizing crossrange errors, as the y parameter was not included in the *qam* cost function. Even though there are slight variances between the two, *qam* provides the closest approximation in the characterization of the cost function qualities observed in the Proto-snake algorithm.

8.3 Benchmark Trajectory / Cost Function

As presented in Chapter 7, a grading system was used to down-select to one or two cost functions that create robust trajectories from a field of seven. This involved analyzing 28 different trajectories, corresponding to four different sets of initial conditions. The process consisted of developing a scoring scheme that determined the values of various attributes within a trajectory. Then different scoring weights were applied to the parameters in proportion to their importance in governing what are considered the robust characteristics of trajectories. A sum of the weighted parameter scores yielded a grade for each trajectory and corresponding cost function. These values were compared to each other in order to determine the best cost function for each reference point. The results of this process are organized below. Appendix B contains a graphical presentation of all 28 trajectories that were compared through this method.

The results for the first reference point are contained in Table 8.1. This table presents the resulting grade for each trajectory and the rank of the trajectory when compared to the others. Appendix C contains tables that include all of the parameter scores, in addition to the raw total of the scores and final grade, for the 28 trajectories analyzed. Please refer to this appendix, as well as Appendix B, for a more detailed comparison between the actual trajectory characteristics and parameters. All of the information presented in the tables of this section is a summation of the tables of Appendix C.

Table 8.1: Point 1 Grading Results

Cost Function	Grade	*Rank*
qam	17.21	3
qamy	23.35	7
qamc	17.42	4
minEamnz	13.66	1
minEamy	20.38	5
maxEamc	21.72	6
maxEamy	54.51	8
Proto-snake	14.20	2

The highlighted row within the table indicates the most robust trajectory of this reference point. In this case it happens to be the result of the *minEamnz* cost function. Figure 8.4 shows the ground track shape of *minEamnz1* and its corresponding $\bar{q}$ profile. The Proto-snake reference trajectory is also presented on the graphs.

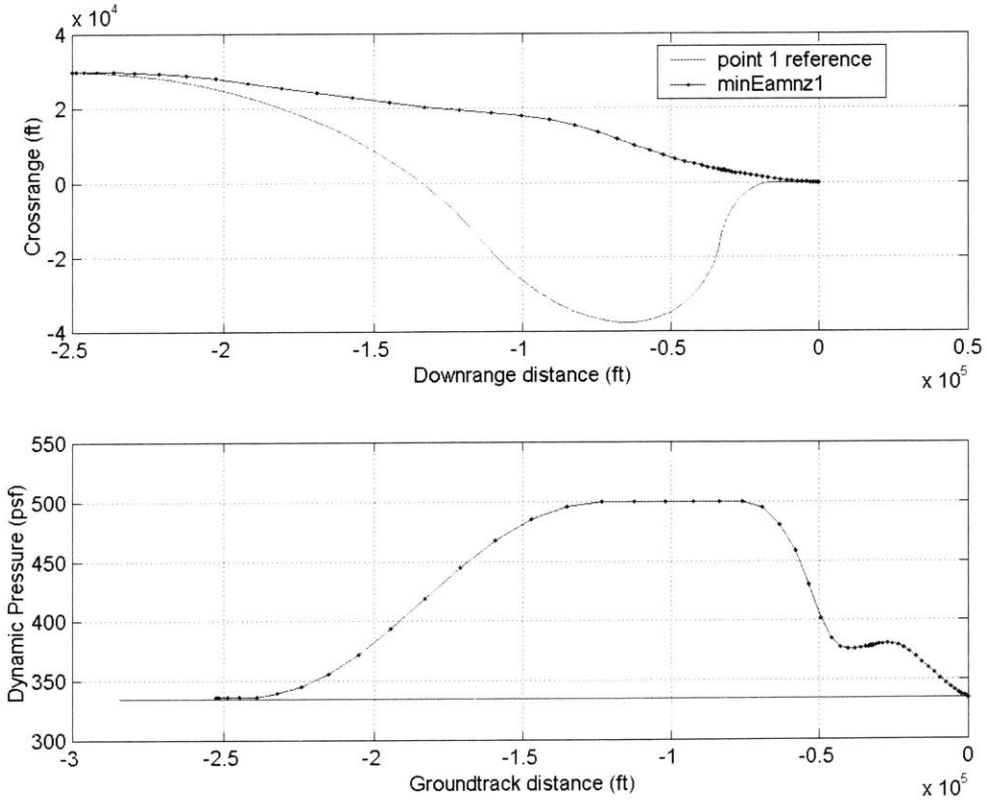


Figure 8.4: Most Robust Trajectory of Point 1

This particular trajectory distinguished itself from the others by its low crossrange errors, heading errors and dynamic loading. It accomplished this by following a higher dynamic pressure schedule, which reduced total ground track length and enabled it to fly practically straight to the ALI. This in turn reduced the number of turns necessary to accomplish the task and reduced most of the errors in the parameters. The parameter scores were so low for y, Nz_b, and χ, that they offset the penalty the trajectory gained by flying on the 500 psf limit. It should be noted that for this particular reference point, Proto-snake scored second in terms of robustness. This accomplishment was the result of smaller dynamic loading than the characteristic cost function *qam*, even though they both had similar shapes.

Table 8.2 presents the results for the second reference point. The most robust trajectory of this set is the result of the cost function *qamy*. This particular trajectory is presented in Figure 8.5, along with the corresponding reference trajectory. The *qamy* trajectory scored better than the other two center of corridor trajectories (*qam* and *qamc*), because of slightly reduced y and χ errors during the last 80 seconds of flight. This produced a slight bowing of the trajectory toward the runway centerline midway through flight.

This reference point also highlights a limitation of Proto-snake's ground track solver. In order to find a snaked ground track solution, the program bends the trajectory towards centerline at the maximum turn rate. For the given initial conditions, a center of corridor $\overline{q}$ schedule produces a ground track distance that is too short to reach the ALI by this method. In order for Proto-snake to generate a solution for this case, a max glide dynamic pressure schedule had to be input, even though the DIDO trajectories prove that a center of corridor (335 psf) trajectory is a viable solution. Therefore, Proto-snake's present GTS is not able to generate trajectories that fly straight to the terminal point.

The results for the third reference point are displayed in Table 8.3. The most robust trajectory of this set is *qamc3*, which defeated *qam3* by a slightly smaller Nz_b parameter score. This trajectory, along with the point 3 reference, is presented in Figure 8.6. Proto-snake finished in fourth place in this case. This is primarily due to the fact that it used a max dive dynamic pressure schedule instead of a center of corridor schedule as utilized by the top three finishers. The $\overline{q}$ parameter penalty far outweighed the other characteristics, even though the crossrange and heading errors, in addition to the dynamic loading proved to be the lowest of all of the trajectories.

Table 8.2: Point 2 Grading Results

Cost Function	Grade	*Rank*
qam	7.04	2
qamy	5.45	1
qamc	7.21	3
minEamnz	20.96	4
minEamy	24.56	5
maxEamc	51.01	7
maxEamy	55.28	8
Proto-snake	36.82	6

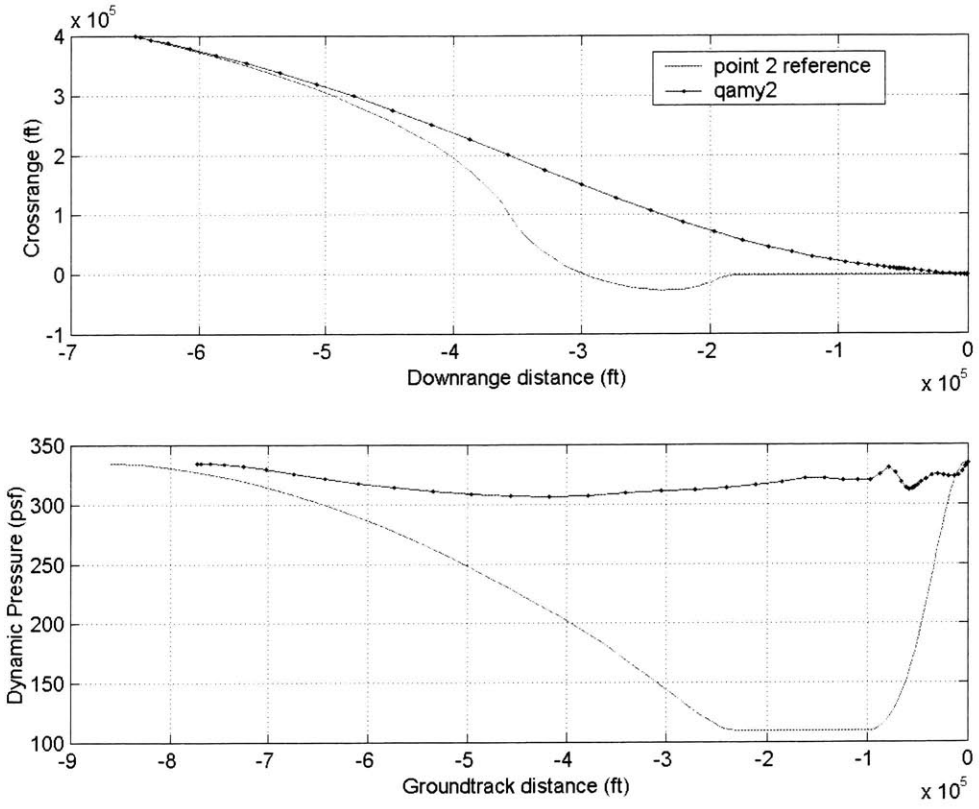


Figure 8.5: Most Robust Trajectory of Point 2

Table 8.3: Point 3 Grading Results

Cost Function	Grade	*Rank*
qam	23.87	2
qamy	28.81	3
qamc	21.98	1
minEamnz	37.93	6
minEamy	30.57	4
maxEamc	51.95	7
maxEamy	55.56	8
Proto-snake	36.17	5

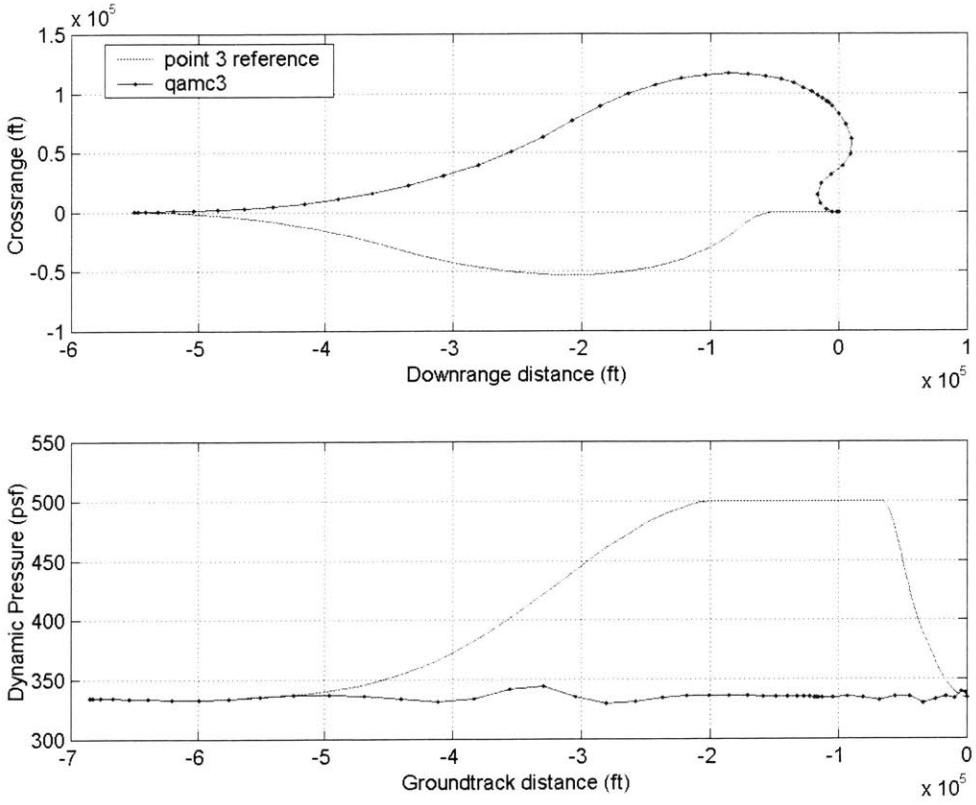


Figure 8.6: Most Robust Trajectory of Point 3

Table 8.4 presents the results of the fourth reference point. In this case, the trajectories created by *qam* and *qamc* proved to be the most robust. In addition, the Proto-snake trajectory came in a close third. The top three trajectories demonstrated lower Nz_b parameter scores than the cost functions containing a crossrange parameter. In those cases the dynamic loading increased significantly in order to reduce the overall crossrange errors. Those trajectories also incurred penalties in the χ parameter due to their nature of turning into the ALI at the last moment. The top three trajectories demonstrate a delayed turn towards the runway that results in increasing the time they spend aligned with the ALI near the end of the flight. The *qam4* trajectory is shown in Figure 8.7, along with its dynamic pressure schedule and the corresponding point 4 reference trajectory.

Table 8.4: Point 4 Grading Results

Cost Function	Grade	*Rank*
qam	8.10	1
qamy	23.05	4
qamc	8.22	2
minEamnz	30.82	5
minEamy	49.04	8
maxEamc	47.45	7
maxEamy	47.20	6
Proto-snake	12.89	3

A final selection of the appropriate benchmark cost function involves a summation of the results from each of the four reference points. This method is actually similar to what is done for sailing regattas. Visualize that the rank of the cost function for each reference point is the place it scored in that "race". A summation of its places yields an overall score for the regatta. The cost function with the lowest score wins the regatta, or in this case, is chosen as the benchmark. Table 8.5 summarizes the selection process by listing the results for all four reference points. The cost functions in this table are listed in order of how they placed overall.

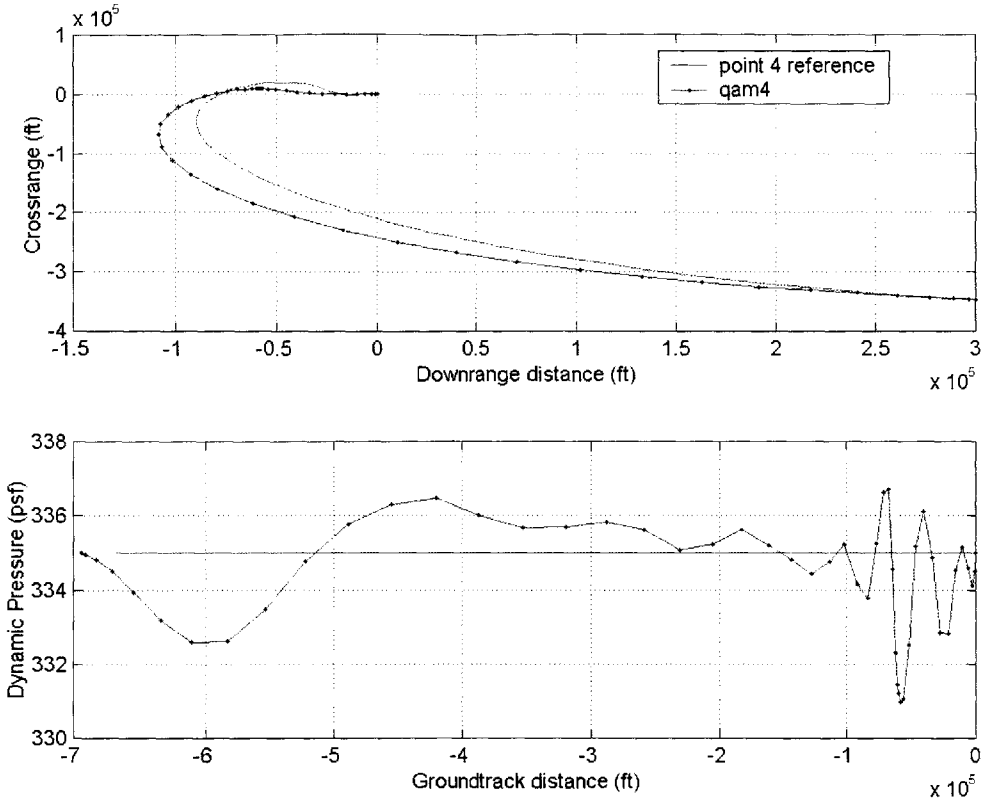


Figure 8.7: Most Robust Trajectory of Point 4

Table 8.5: Overall Grading Results

Cost Function	Point 1	Point 2	Point 3	Point 4	Total	***Overall Rank***
qam	3	2	2	1	8	***1***
qamc	4	3	1	2	10	***2***
qamy	7	1	3	4	15	***3***
Proto-snake	2	6	5	3	16	***4***
minEamnz	1	4	6	5	16	***4***
minEamy	5	5	4	8	22	***6***
maxEamc	6	7	7	7	27	***7***
maxEamy	8	8	8	6	30	***8***

This shows that the cost function that generates benchmark trajectories for onboard trajectory generators based upon the listed desirability criterion is *qam*, and is written as:

$$J = \int_{t_0}^{t_f} \left[0.6(\overline{q} - \overline{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(\dot{\mu})^2 \right] dt \qquad \textbf{(8. 3)}$$

Also observe that the top three finishers are cost functions that produce center of corridor trajectories. The next two are the costs that produce the max dive trajectories. Max glide trajectories finished last. It should also be pointed out that Proto-snake actually tied for fourth place in its current implementation. This result is actually surprising, considering that Proto-snake is the first working prototype that demonstrates the SNAKE methodology. Though it is encouraging that it placed so well, the results gained in the benchmark determination process will still be used for future improvements.

8.4 Summary

The results presented in this chapter can be summarized primarily by three main points. First of all, the *qam* cost function was selected as the benchmark for future trajectory generators, based on the following desirability features. It creates center of corridor trajectories that exhibit very smooth control histories. The smooth controls result in gradual turns and smooth transitions from the initial starting point to the autolanding interface. This in turn reduces dynamic loading, and ensures an entrance into the ALI that experiences relatively small crossrange distance and heading angle errors. These are all defining characteristics of robust trajectories, as determined and presented by this thesis.

Second, the Proto-snake trajectory generator produces sub-optimal trajectories that are characterized by the benchmark cost function. The trajectories are solutions to a constrained problem, whereby the dynamic loading is fixed throughout the turns. In addition, the initial turn must always be made towards the runway centerline. While this method is effective, it sometimes precludes the determination of a solution for a given set of conditions, even if one exists. This basically reveals itself in Proto-snake's inability to produce straight-line trajectories from an initial starting point if there is any crossrange

error present. All Proto-snake trajectories must snake the excess energy in the approach trajectory to satisfy the ALI boundary constraint.

Finally, the trajectories currently produced by Proto-snake do exhibit a fair amount of robustness. This was revealed by the benchmark scoring and grading system presented in Chapter 7. While the generator tied for 4th out of eight possible trajectory formulators, it still has its limitations. Recommendations for improvements, derived from the results of the benchmark selection, are presented in the next chapter.

[This page intentionally left blank]

Chapter 9

Conclusions

9.1 Summary and Conclusions

Draper Laboratory has been in pursuit of a Next Generation Guidance and Control (NGGC) system that may be applied to future Reusable Launch Vehicles (RLV). This system seeks to significantly increase flight vehicle safety and reduce life cycle costs while gaining increased operation and performance capabilities. A key technological element of the NGGC system is an onboard trajectory generator. This tool should rapidly generate flyable trajectories, in real time, around an RLV's current flight conditions. Several trajectory generators have been demonstrated including one version that utilizes the SNAKE methodology. This prototype generator is known as Proto-snake and has shown the greatest promise for fulfilling the requirements necessary to operate within the NGGC framework.

The Proto-snake trajectory generator creates trajectories from the supersonic TAEM region to the auto-landing interface. It utilizes a real time Ground Track Solver (GTS) that manipulates the predicted ground track by "snaking" it around in order to solve a two-point boundary value problem. The predicted ground track is determined from a user- input dynamic pressure schedule. While this program has proven successful in the generation of multiple trajectories for various initial conditions, it does have limitations. Because the dynamic pressure schedule used by Proto-snake is pre-defined, it can only provide robust trajectories for a limited subset of the total design space. Also, the current GTS utilizes an ad hoc scheme, developed from its designer's engineering intuition, for determining the lateral ground track. This means that the robustness of each trajectory for the given conditions has not been evaluated due to the lack of a suitable benchmark that defines those characteristics. In addition, no methodology currently exists for quantitatively grading the robustness of a trajectory.

This thesis presented a methodology for further advancing the work done on onboard trajectory generation through the determination of suitable benchmarks. A number of trajectories were created through the use of a Legendre Pseudospectral Optimization method, which involved formulating several different cost functions. A physical measure

and definition of robustness was then determined through a thorough analysis of all the optimized trajectories and their corresponding attributes. The use of a detailed scoring and grading system enabled a down-selection to one final cost function, which was set as the benchmark cost function for future work. Trajectories created through the use of this cost function are considered benchmark trajectories that portray the definitive characteristics of robust trajectories. These attributes should be modeled and mimicked by future onboard generators to create better reference trajectories. The grading system used in the determination of the benchmark can also be used to calculate a quantitative value of robustness for existing trajectories. In this manner, Proto-snake trajectories were shown to be sub-optimal representations of the benchmark trajectories. Additionally, this research has provided insight into possible ways to improve upon the baseline Proto-snake generator.

Some of the insight gained is with regards to the interaction between various components and variables that comprise a trajectory and how they may be manipulated by an onboard generator in order to create a solution. In particular, the interaction between overall trajectory shapes and dynamic pressure schedules has been explored. The use of a $\overline{q}$ schedule enables the formulation or prediction of an overall ground track length. This ground track must then be manipulated in order to create a trajectory between the initial starting point and the termination point at the ALI. There are only two ways of doing this; adjusting the dynamic pressure schedule, and "snaking" the trajectory, either horizontally or vertically. A vertical snake experiences positive and negative dynamic loading and resembles a ride on NASA's KC-135, also known as the vomit comet to occupants experiencing the undesirable trajectory features of such a motion. A horizontal snake occurs in the x-y plane and usually results in higher constant positive dynamic loading. If the ground track length is too short, the $\overline{q}$ schedule must be reduced (i.e. lower $\overline{q}$) in order to lengthen the overall trajectory length. If the ground track is too long, the trajectory can be snaked, or the dynamic pressure schedule increased (i.e. higher $\overline{q}$). Both adjusting the $\overline{q}$ schedule and snaking the trajectories affect overall robustness. Therefore, creating the most robust trajectory involves a balancing of the tradeoffs between them.

Proto-snake can readily handle the horizontal snaking of trajectories to arrive at viable solutions. However, the dynamic pressure schedules that should be followed were not explored during its development. The results presented in this thesis have shown that robust trajectories are the result of a center of corridor $\overline{q}$ schedules. Therefore, Proto-

snake should primarily design center of corridor trajectories. If the ground track length is too short to reach the targeted end condition, then the dynamic pressure schedule should be reduced slightly until the vehicle flies an essentially straight trajectory (in the lateral sense) to the end point. The GTS should not excessively snake the ground track when the $\overline{q}$ is below 335 psf, because low $\overline{q}$ values result in poor robustness to future dispersions. However, if the ground track is too long, it should just be snaked around to satisfy the boundary conditions. Proto-snake should only switch to a higher $\overline{q}$ schedule if the snaking is too great, which may lead to the growth of dynamic loading throughout the trajectory, in addition to crossrange and heading errors at the ALI. In the future, a $\overline{q}$ schedule that allows too much snaking may be identified by analyzing the Energy over Weight (E/W) value at a predetermined altitude.

In conclusion, the results presented here are derived from standardized systems engineering pratices, whereby a quantifiable metric is established for measuring a previously unknown quantity. This led to an objective approach in differentiating between and comparing various trajectories. In the end, a benchmark was established for onboard trajectory generators that should make it easier to formulate future ground track solvers. In addition, the amount of information derived in the process of creating this thesis has led to a greater understanding of trajectory generation that can be directly applied to improving Proto-snake.

9.2 Recommendations for Future Work

The initial focus of this research was to advance the work done on Draper's onboard trajectory generator. This was accomplished by determining trajectory benchmarks that can be used to further improve the current version of Proto-snake. In addition, it is hoped that the results of this thesis will be used in the formulation of a more advanced trajectory generator, one that is designed around the findings of this research. Therefore, the recommendations made here are for two improvements that can be made to Proto-snake to increase its capability of generating robust trajectories, in addition to suggestions that may be used in the formulation of future trajectory generators. The improvements for Proto-snake involve its dynamic pressure schedules and its ground track solver. These are presented below.

Dynamic Pressure Schedules: Currently, the dynamic pressure schedules used by Proto-snake are input by the user. This is due to the fact that Proto-snake is the first working

prototype that utilizes the SNAKE methodology and a dynamic pressure selector was not necessary for the demonstration of SNAKE. However, without a $\overline{q}$ selector, cases arise where a $\overline{q}$ schedule is entered into the program that yields a ground track length that is too short to reach the end condition, which causes Proto-snake to fail. Therefore, a dynamic pressure schedule selector should be added to Proto-snake. This selector should be able to choose a $\overline{q}$ schedule that always returns a trajectory solution, while at the same time maximizes the robustness of the trajectories.

One possible way to implement this $\overline{q}$ schedule selector is through the use of dynamic pressure plateaus. These plateaus represent a given dynamic pressure value, and their respective ground track distance, for a given starting altitude. By storing a number of these plateaus, a $\overline{q}$ schedule selector can choose a different schedule based on its need for a specific ground track distance. A visual representation of these plateaus is shown in Figure 9.1.

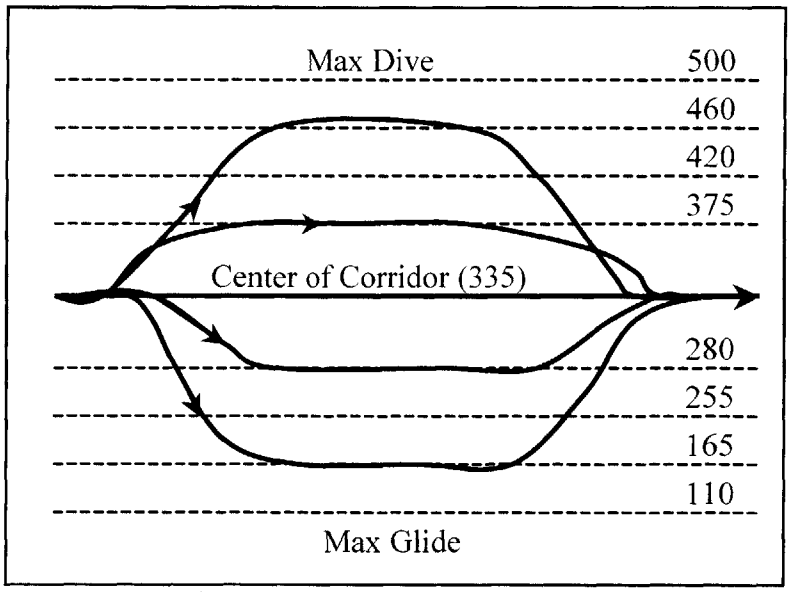


Figure 9.1: Dynamic Pressure Schedule Plateaus

A trajectory is formulated to maintain a center of corridor $\overline{q}$ schedule. This results in the generation of the most robust trajectories. It must also always end with a $\overline{q}$ value that matches the ALI target constraints. If the ground track distance is too long, the trajectory is snaked to meet the end conditions. However, if the snaking violates too many

robustness parameters, such as dynamic loading or crossrange errors, or even ends up bending the trajectory away from the ALI, a different $\overline{q}$ schedule is selected. This is depicted in Figure 9.1. The selector chooses a higher $\overline{q}$ schedule in order to reduce ground track distance. It is yet to be determined how the selector creates the variation from 335 psf to the chosen plateau value. One possibility may be to use the $d\overline{q}/dh$ quadratic presented in Reference 1. Additionally, the selector must choose whether the dynamic pressure schedule should use a higher plateau for a shorter period of time or a lower plateau for a longer period of time. In terms of maximizing robustness, the later should be implemented. Also, the selector only chooses a lower plateau value for cases where there is not enough ground track distance to make the ALI. It should never select a lower plateau for cases where a center of corridor schedule yields a solution.

Ground Track Solver: The ground track solver utilized by Proto-snake follows a logic coined "steer-to-zero". This logic results in the GTS starting a trajectory design by bending the ground track towards the runway at one node, while bending it back to a heading angle of zero at the next node. The process is continued down through all the nodes until the projected trajectory lies coincident with the ALI centerline. Figure 9.2 shows the shapes of the ground track during this process. This logic is the first phase of the ground track formulation procedure followed by the GTS, and is described more fully in Chapter 5. The second phase results in the addition of extra loops or snaking of the ground track away from centerline in order to allow the end of the trajectory to lie at the ALI.

While this GTS has yielded sub-optimal solutions that are fairly robust, it can be improved upon. The steer-to-zero logic causes Proto-snake to fail in certain instances where it determines the ground track distance is too short to allow a solution for the given dynamic pressure schedule. This problem has been clearly shown in Chapter 8, where Proto-snake had to use a max glide $\overline{q}$ schedule in order to generate a solution for the point 2 initial conditions, even though DIDO proved a center of corridor trajectory is a viable answer. While this problem can be somewhat alleviated by the introduction of a $\overline{q}$ schedule selector, the GTS can be modified to help in the solution as well. For the most part, the steer-to-zero logic causes the ground track shapes to snake too much and prohibits the GTS from designing essentially straight-line trajectories from the starting point to the ALI. Changing the logic from a "steer-to-zero" to a "steer-to-transition" can remove this problem.

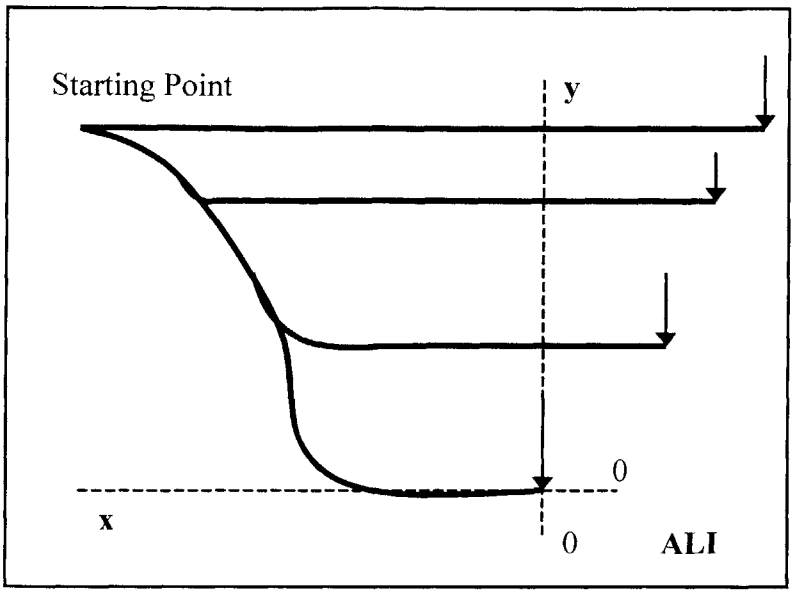


Figure 9.2: "Steer-to-Zero" Logic

A steer-to-transition logic is a slight modification of the steer-to-zero logic. The steer-to-transition refers to aligning the trajectory with a transition point that lies ahead of the ALI. It is yet to be determined where exactly this point should be, but perhaps it is around a downrange value of 10 nm, which corresponds approximately to the last 80 seconds of flight for a vehicle. During the trajectory formulation, as the GTS marches down the trajectory and bends the nodes towards the runway, it should not bend each subsequent node back toward a heading of zero as is done in the steer-to-zero logic. Instead, the GTS should keep the trajectory pointed in the direction of the last heading change until this heading is coincident with the heading to the transition point. While this is occurring, the last 10 nm of the ground track is kept aligned with a heading of zero. This logic idea is shown in Figure 9.3. The ground track is continually bent until the end point lies at the ALI. If the ground track is too long, the phase two snaking is carried out as before, except this time the excess looping is done from the transition heading and not the ALI centerline.

The result of this steer-to-transition logic will be the generation of fairly straight trajectories, for an appropriate dynamic pressure schedule. This should increase the solution space of Proto-snake. In addition, the use of the transition point for aiming allows the trajectory to turn towards the ALI at this point and not right at the ALI. This

mimics the Heading Alignment Cone (HAC) used by the shuttle, and enables the reduction of crossrange distance and heading angle errors at a predetermined distance from the runway. Because the scoring scheme presented in this thesis only measures crossrange and heading errors during the last moments of flight, this transition point logic will maximize the robustness of the trajectories created by Proto-snake.

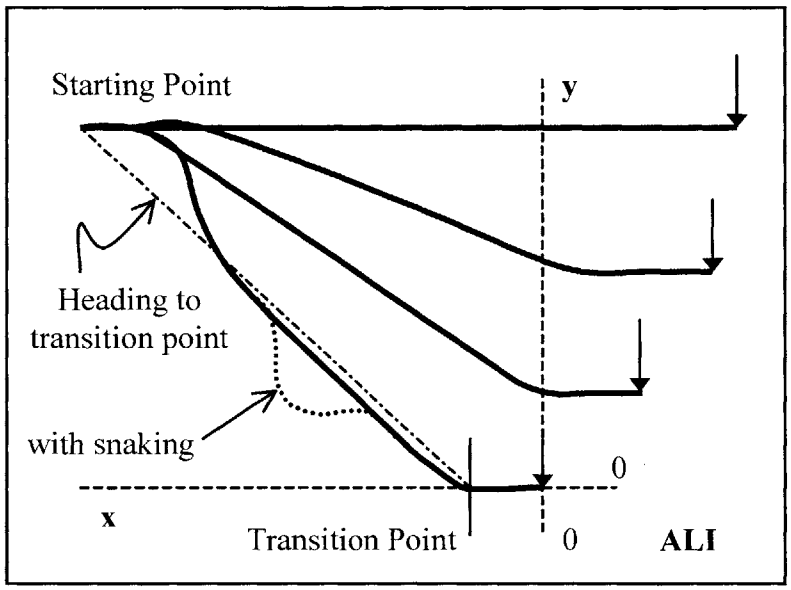


Figure 9.3: "Steer-to-Transition" Logic

Future Trajectory Generators: The future SNAKE trajectory generators should encompass the same recommendations presented here for Proto-snake, in addition to a warm start capability. A warm start is where the generator can begin the trajectory creation process at any node in the trajectory. It does not have to always follow the same beginning to end procedure if an initial trajectory already exists. The warm start capability will enable the trajectory generator to recalculate a new trajectory at any given time. This time may correspond to when the vehicle finds itself low on energy or near a constraint and it needs to create an updated trajectory that takes this into account.

A warm start capability will also enable the formulation of a more advanced guidance and control concept. This process may begin by the generation of an initial trajectory that is fed to the flight management system. As the vehicle begins to fly down the trajectory, the mission manager uses the grading system presented in this thesis to determine the overall robustness of the initial trajectory. It then identifies the attribute with the highest

value or error and issues a command to the trajectory generator to correct it with another trajectory design. Because the vehicle is already in flight, this new trajectory does not start from the initial position, but the current vehicle location. A warm start capability is the key piece that may make this possible. As the vehicle continues through the flight, the mission manager continuously assesses the trajectory robustness and modifies the remaining ground track to improve upon any deficiencies, similar to the way a highly trained pilot would react to changing circumstances. This has the effect of creating the most robust trajectory for that particular flight, no matter how the conditions change during throughout the flight. A trajectory generator of this type would help the overall NGGC system concept come to full fruition.

Appendix A

DIDO Problem Setup

A.1 User Interface Code

TaemMainmu—Main file needed to run DIDO. This sets up the bounds on the states, controls, path constraints and event conditions. It also calls the four separate files. These include TaemCostmu, TaemDynamics, TaemEvents and TaemPath.

```
%=========================================================
load aero_tables
load output335

%----------------------
%   Setup constants
%----------------------
global CONSTANTS MACHTAB ALPHATAB
global CDTAB CLTAB

CONSTANTS.W        = 18013.7;      % Weight(lb)
CONSTANTS.g        = 32.174;       % Gravity(ft/sec)
CONSTANTS.S        = 357.5;        % Wing Area(ft^2)
CONSTANTS.Hrun     = 3840.5;       % Runway height(ft)
CONSTANTS.Vnorm    = 6000;         % Velocity(ft/sec) normalizing value
CONSTANTS.hnorm    = 150000;       % Norm Altitude(ft)
CONSTANTS.xnorm    = 300000;       % Norm Downrange(ft)
CONSTANTS.ynorm    = 80000;        % Norm Crossrange(ft)
MACHTAB      = mach_sav;     % Mach table
ALPHATAB     = alpha_sav;    % Alpha table
CDTAB        = cd_sav;       % Drag coeff table
CLTAB        = cl_sav;       % Lift coeff table

%--------------------
%   Problem
%--------------------
taemProblem.cost         = 'taemCostmu';
taemProblem.dynamics     = 'taemDynamics';
taemProblem.events       = 'taemEvents';
taemProblem.path         = 'taemPath';


%--------------------------
%  Setup constants
%--------------------------
t0        = 0;
tfMax     = 1000;
tfguess   = t.nodes(end)
```

```
%------------------------------
%   Setup knots
%------------------------------
knots.locations      = [t0 125 tfguess];
knots.definitions    = {'hard', 'soft', 'hard'};
knots.bounds.lower   = [0 95 0];
knots.bounds.upper   = [0 155 tfMax];
knots.numNodes       = [35 15];

Ntotal = sum(knots.numNodes);


%---------------------------------------
%   Setups bounds and BC's
%---------------------------------------
V0 = t.states(1,1);  Vf = t.states(1,end);
c0 = t.states(2,1);  cf = t.states(2,end);
h0 = t.states(3,1);  hf = t.states(3,end);
x0 = t.states(4,1);  xf = t.states(4,end);
y0 = t.states(5,1);  yf = t.states(5,end);
g0 = t.states(6,1);
a0 = t.states(7,1);
m0 = t.states(8,1);  mf = t.states(8,end);

bounds.lower.states = [500/CONSTANTS.Vnorm; -pi; 0;...
      -30e4/CONSTANTS.xnorm; -100000/CONSTANTS.ynorm;
(-45*(pi/180)); 0; -75*(pi/180)];
bounds.upper.states = [6000/CONSTANTS.Vnorm; pi;
150000/CONSTANTS.hnorm; 30e4/CONSTANTS.xnorm;...
100000/CONSTANTS.ynorm; 0; 20*(pi/180); 75*(pi/180)];

bounds.lower.controls = [-15*(pi/180); -180*(pi/180)];
bounds.upper.controls = [15*(pi/180); 180*(pi/180)];

bounds.lower.events = [V0; c0; h0; x0; y0; g0; a0; m0;...
                       Vf; cf; hf; xf; yf; mf];
bounds.upper.events = bounds.lower.events;

bounds.lower.path = [-1; -1; -1; 110];
bounds.upper.path = [0.6; 1; 3; 500];


%----------------------------
%   Provide a guess
%----------------------------
guess.states(1,:) = [t.states(1,1:35) t.states(1,37:end)];
guess.states(2,:) = [t.states(2,1:35) t.states(2,37:end)];
guess.states(3,:) = [t.states(3,1:35) t.states(3,37:end)];
guess.states(4,:) = [t.states(4,1:35) t.states(4,37:end)];
guess.states(5,:) = [t.states(5,1:35) t.states(5,37:end)];
guess.states(6,:) = [t.states(6,1:35) t.states(6,37:end)];
guess.states(7,:) = [t.states(7,1:35) t.states(7,37:end)];
guess.states(8,:) = [t.states(8,1:35) t.states(8,37:end)];
guess.controls(1,:)  = [t.controls(1,1:35) t.controls(1,37:end)];
guess.controls(2,:)  = [t.controls(2,1:35) t.controls(2,37:end)];
guess.time        = [t.nodes(1,1:35) t.nodes(1,37:end)];
```

```
%=======================
%   Run Dido
%=======================
[cost, primal] = dido(taemProblem,knots,bounds,guess,useropt);

save tOutmuA
```

TaemCostmu—File containing the setup for the cost function.

```
function [zeroEndPointCost, integrandCost] = taemCostmu(primal)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global CONSTANTS MACHTAB ALPHATAB

V = primal.states(1,:);     % velocity
h = primal.states(3,:);     % altitude
mu = primal.states(8,:);    % bankangle

u1 = primal.controls(1,:);
adot = u1;

wf = 0.6;    % weighting function

%-----------------------
%   Setup variables
%-----------------------
for II = 1:(length(V(1,:)));
    [tmp(II),pres(II),rho(II),sos(II)] = atmos4(0,((h(II).*...
        CONSTANTS.hnorm)+CONSTANTS.Hrun));
end
qbar     = (0.5)*(rho).*((V.*CONSTANTS.Vnorm).^2);


%=========================================================================
zeroEndPointCost = 0;
integrandCost = 0.6*(((qbar-335)./500).^2)+0.2*((mu).^2)+...
0.2*((50*adot).^2);
%=========================================================================
```

TaemDynamics—File containing the setup of the dynamic constraints, or equations of motion.

```
function residuals = taemDynamics(primal)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global CONSTANTS MACHTAB ALPHATAB
global CDTAB CLTAB

V = primal.states(1,:);      Vdot = primal.statedots(1,:);    %velocity
c = primal.states(2,:);      cdot = primal.statedots(2,:);    %chi
h = primal.states(3,:);      hdot = primal.statedots(3,:);    %altitude
x = primal.states(4,:);      xdot = primal.statedots(4,:);    %downrange
y = primal.states(5,:);      ydot = primal.statedots(5,:);    %crossrange
```

```
g = primal.states(6,:);      gdot = primal.statedots(6,:);    %gamma
alpha    = primal.states(7,:);     adot = primal.statedots(7,:); %angle
of attack
mu       = primal.states(8,:);     mdot = primal.statedots(8,:); %bank
angle

u1 = primal.controls(1,:);      % (alpha)dot
u2 = primal.controls(2,:);      % (bank angle)dot


%-----------------------
%   Setup variables
%-----------------------
for II = 1:(length(V(1,:)));
    [tmp,pres,rho(II),sos(II)] = atmos4(0,((h(II).*...
        CONSTANTS.hnorm)+CONSTANTS.Hrun));
end
Mach     = ((V.*CONSTANTS.Vnorm)./sos);
qbar     = (0.5)*(rho).*((V.*CONSTANTS.Vnorm).^2);
CL  = interp2(MACHTAB,ALPHATAB,CLTAB,Mach,(alpha.*(180/pi)),'*linear');
CD  = interp2(MACHTAB,ALPHATAB,CDTAB,Mach,(alpha.*(180/pi)),'*linear');


%----------------------------------------
%   Setup state equations/residuals
%----------------------------------------
N = length(primal.states(1,:));
residuals = zeros(8,N);

%=========================================================================
residuals(1,:) = Vdot-((-CONSTANTS.g./(CONSTANTS.W.*
CONSTANTS.Vnorm)).*((qbar.*CONSTANTS.S.*CD)+(CONSTANTS.W. *sin(g))));
residuals(2,:) = cdot-((CONSTANTS.g./(CONSTANTS.W*V.
*CONSTANTS.Vnorm)).*((qbar.*CONSTANTS.S.*CL.* sin(mu))./(cos(g))));
residuals(3,:) = hdot-((V.*CONSTANTS.Vnorm.*sin(g))./
(CONSTANTS.hnorm));
residuals(4,:) = xdot-((V.*CONSTANTS.Vnorm.*(cos(c)).*(cos(g)))./
(CONSTANTS.xnorm));
residuals(5,:) = ydot-((V.*CONSTANTS.Vnorm.*(sin(c)).*(cos(g)))./
(CONSTANTS.ynorm));
residuals(6,:) = gdot-(CONSTANTS.g./(CONSTANTS.W*V.*CONSTANTS.Vnorm)).*
((qbar.*CONSTANTS.S.*CL.*cos(mu))-(CONSTANTS.W.*cos(g))));
residuals(7,:) = adot - u1;
residuals(8,:) = mdot - u2;
%=========================================================================
```

TaemEvents—File containing the specification of the event conditions. These are essentially specifications on the states at the initial and final times.

```
function boundaryConditions = taemEvents(primal)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global CONSTANTS
```

```
V0 = primal.states(1,1);        Vf = primal.states(1,end); % velocity
c0 = primal.states(2,1);        cf = primal.states(2,end); % chi
h0 = primal.states(3,1);        hf = primal.states(3,end); % altitude
x0 = primal.states(4,1);        xf = primal.states(4,end); % downrange
y0 = primal.states(5,1);        yf = primal.states(5,end);  % crossrange
g0 = primal.states(6,1);        gf = primal.states(6,end);  % gamma
a0 = primal.states(7,1);        af = primal.states(7,end);  % alpha
m0 = primal.states(8,1);        mf = primal.states(8,end);  % mu

boundaryConditions = zeros(12,1);

%=========================================
boundaryConditions(1)    = V0;
boundaryConditions(2)    = c0;
boundaryConditions(3)    = h0;
boundaryConditions(4)    = x0;
boundaryConditions(5)    = y0;
boundaryConditions(6)    = g0;
boundaryConditions(7)    = a0;
boundaryConditions(8)    = m0;
%-----------------------------------------
boundaryConditions(9)    = Vf;
boundaryConditions(10)   = cf;
boundaryConditions(11)   = hf;
boundaryConditions(12)   = xf;
boundaryConditions(13)   = yf;
boundaryConditions(14)   = mf;
%=========================================
```

TaemPath—File that specifies the path constraints for the problem. In this case, these include the dynamic pressure $\overline{q}$, Nx_b, Ny_b, and Nz_b.

```
function pathconstraints = taemPath(primal)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global CONSTANTS MACHTAB ALPHATAB
global CDTAB CLTAB

V = primal.states(1,:);      % velocity
c = primal.states(2,:);      % chi
h = primal.states(3,:);      % altitude
x = primal.states(4,:);      % downrange
y = primal.states(5,:);      % crossrange
g = primal.states(6,:);      % gamma
alpha    = primal.states(7,:); % angle of attack
mu       = primal.states(8,:); % bank angle

u1 = primal.controls(1,:);      % (alpha)dot
u2 = primal.controls(2,:);      % (bank angle)dot
```

```
%-----------------------
%   Setup variables
%-----------------------
for II = 1:(length(V(1,:)));
   [tmp,pres,rho(II),sos(II)] = atmos4(0,((h(II).*...
      CONSTANTS.hnorm)+CONSTANTS.Hrun));
end
Mach    = ((V.*CONSTANTS.Vnorm)./sos);
qbar    = (0.5)*(rho).*((V.*CONSTANTS.Vnorm).^2);
CL      =
interp2(MACHTAB,ALPHATAB,CLTAB,Mach,(alpha.*(180/pi)),'*linear');
CD      =
interp2(MACHTAB,ALPHATAB,CDTAB,Mach,(alpha.*(180/pi)),'*linear');

Vdot = ((-CONSTANTS.g./(CONSTANTS.W.*CONSTANTS.Vnorm)).*...
   ((qbar.*CONSTANTS.S.*CD)+(CONSTANTS.W.*sin(g))));
cdot = ((CONSTANTS.g./(CONSTANTS.W*V.*CONSTANTS.Vnorm)).*...
   ((qbar.*CONSTANTS.S.*CL.*sin(mu))./(cos(g))));
gdot = ((CONSTANTS.g./(CONSTANTS.W*V.*CONSTANTS.Vnorm)).*...
   ((qbar.*CONSTANTS.S.*CL.*cos(mu))-(CONSTANTS.W.*cos(g))));


%-------------------------
%  Setup functions
%-------------------------
Nxa = ((Vdot.*CONSTANTS.Vnorm)./CONSTANTS.g);
Nya = ((cdot.*V.*CONSTANTS.Vnorm.*(cos(g)))./(CONSTANTS.g));
Nza = ((-gdot.*V.*CONSTANTS.Vnorm)./(CONSTANTS.g));

Nxb = Nxa.*(cos(alpha))+Nya.*(sin(alpha)).*(sin(mu))-...
   Nza.*(sin(alpha)).*(cos(mu));
Nyb = Nya.*(cos(mu))+Nza.*(sin(mu));
Nzb = (-Nxa.*(sin(alpha)))+Nya.*(cos(alpha)).*(sin(mu))-...
   Nza.*(cos(alpha)).*(cos(mu));


N = length(primal.states(1,:));
pathconstraints = zeros(4,N);

%=============================
pathconstraints(1,:) = Nxb;
pathconstraints(2,:) = Nyb;
pathconstraints(3,:) = Nzb;
pathconstraints(4,:) = qbar;
%=============================
```

Appendix B

DIDO Results

B.1 Graphical Presentation of DIDO Results

The figures presented on the following pages are graphical presentations of the data output from DIDO. Each figure compares a DIDO trajectory to the reference produced by Proto-snake. The first seven figures are the results of DIDO running each of the seven cost functions at the first reference point. The second group of seven figures are the results of the cost functions at the second reference point and so on. The cost function used for the formulation of the DIDO trajectory presented in a particular figure is displayed at the bottom of the page.

For each figure, the top two plots present the longitudinal flight of the trajectory, represented by its altitude and flight path angle profiles. The next two plots are of the trajectory control states, angle of attack and bank angle. In DIDO, the actual control variables are the derivatives of these states. The next two plots are of the path constraints, dynamic pressure and dynamic loading. All six states / constraints shown at the top each figure are plotted against the overall ground track distance. The largest plot at the bottom of each figure is a top down view of the trajectory, which represents the lateral flight or lateral ground track shape of the trajectory. For this plot, the crossrange distance is plotted against the trajectory downrange distance.

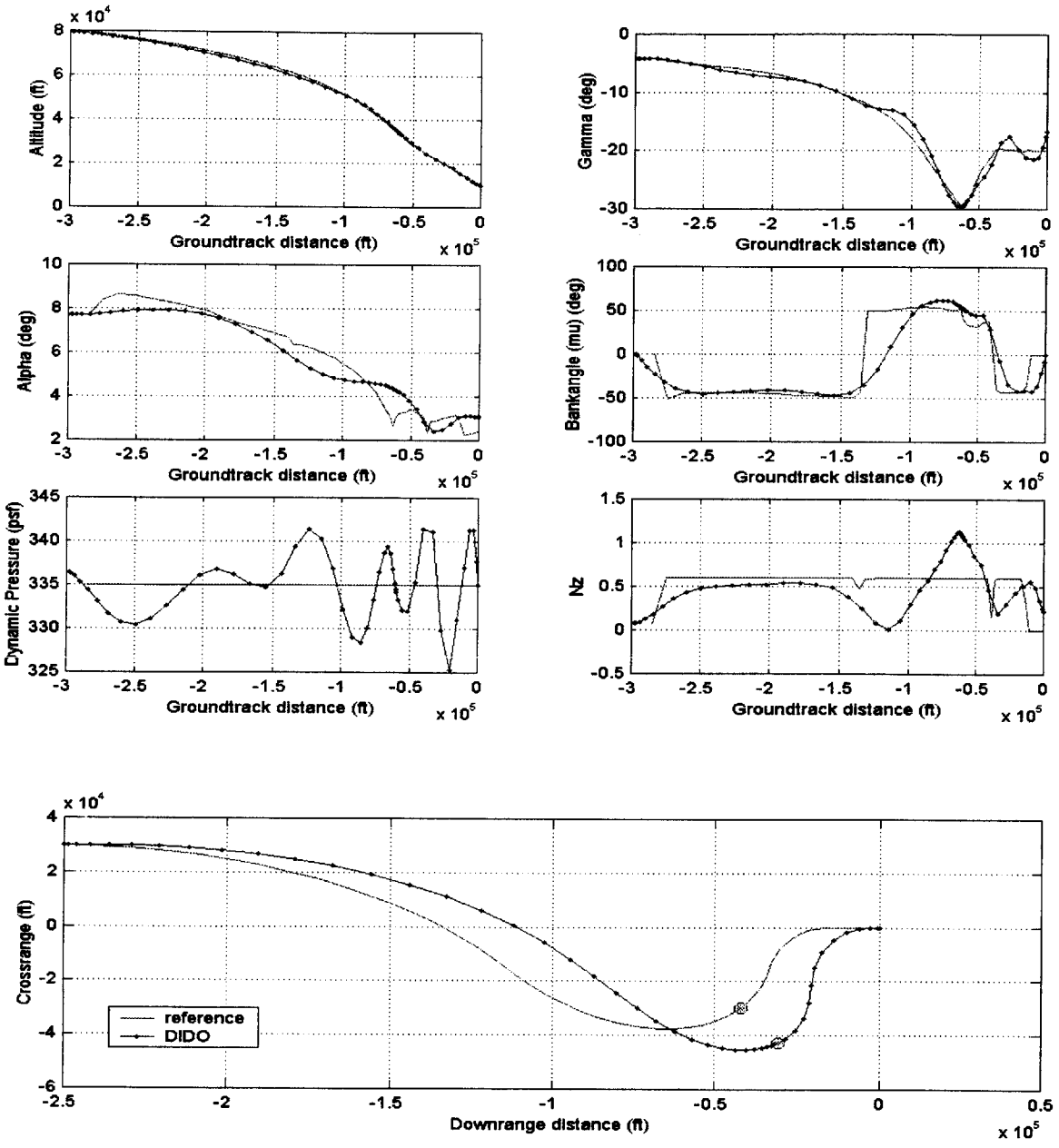


Figure B.1: DIDO Run (qam1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\overline{q} - \overline{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(\dot{\mu})^2 \right] dt$$

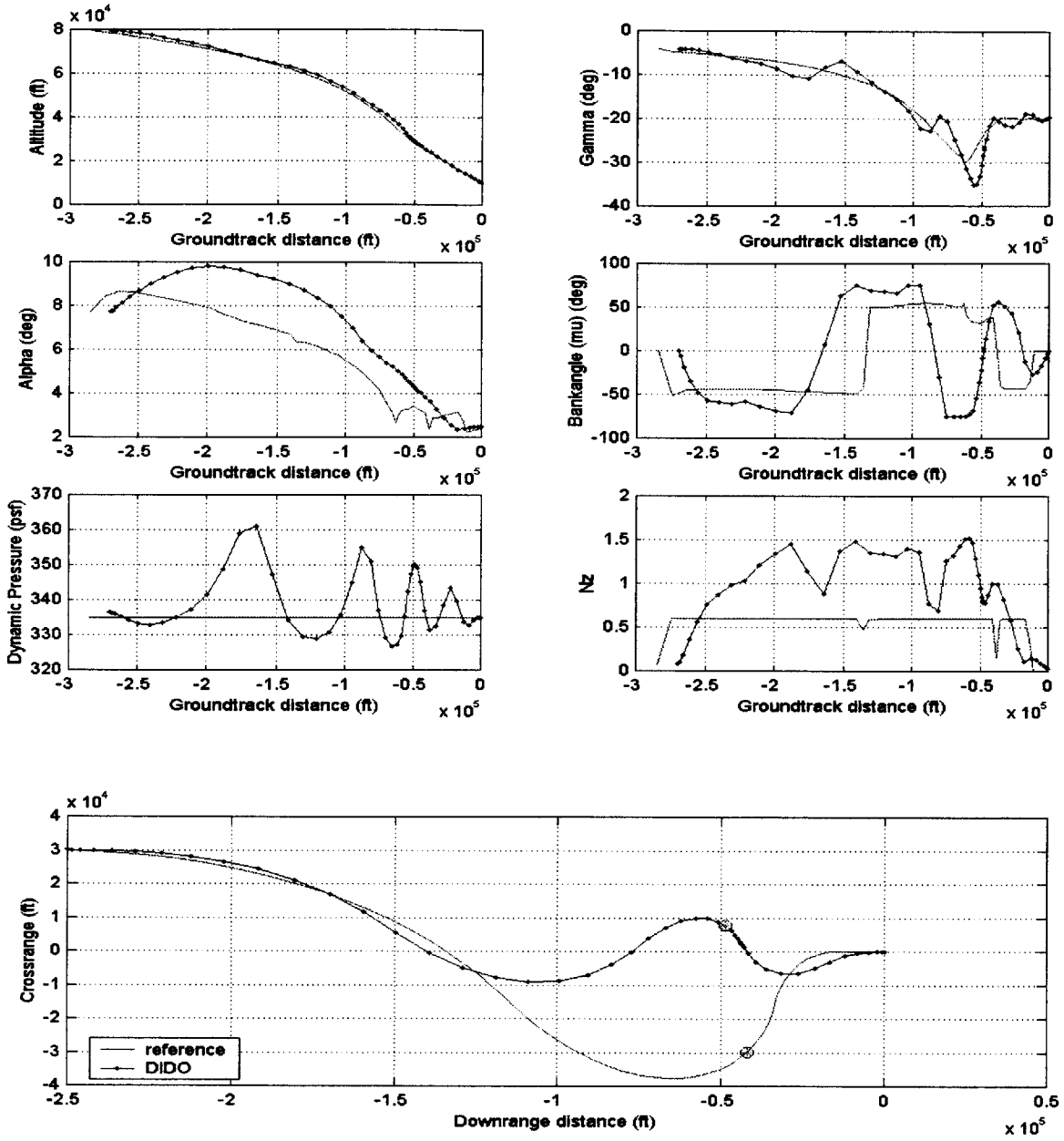


Figure B.2: DIDO Run (qamy1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

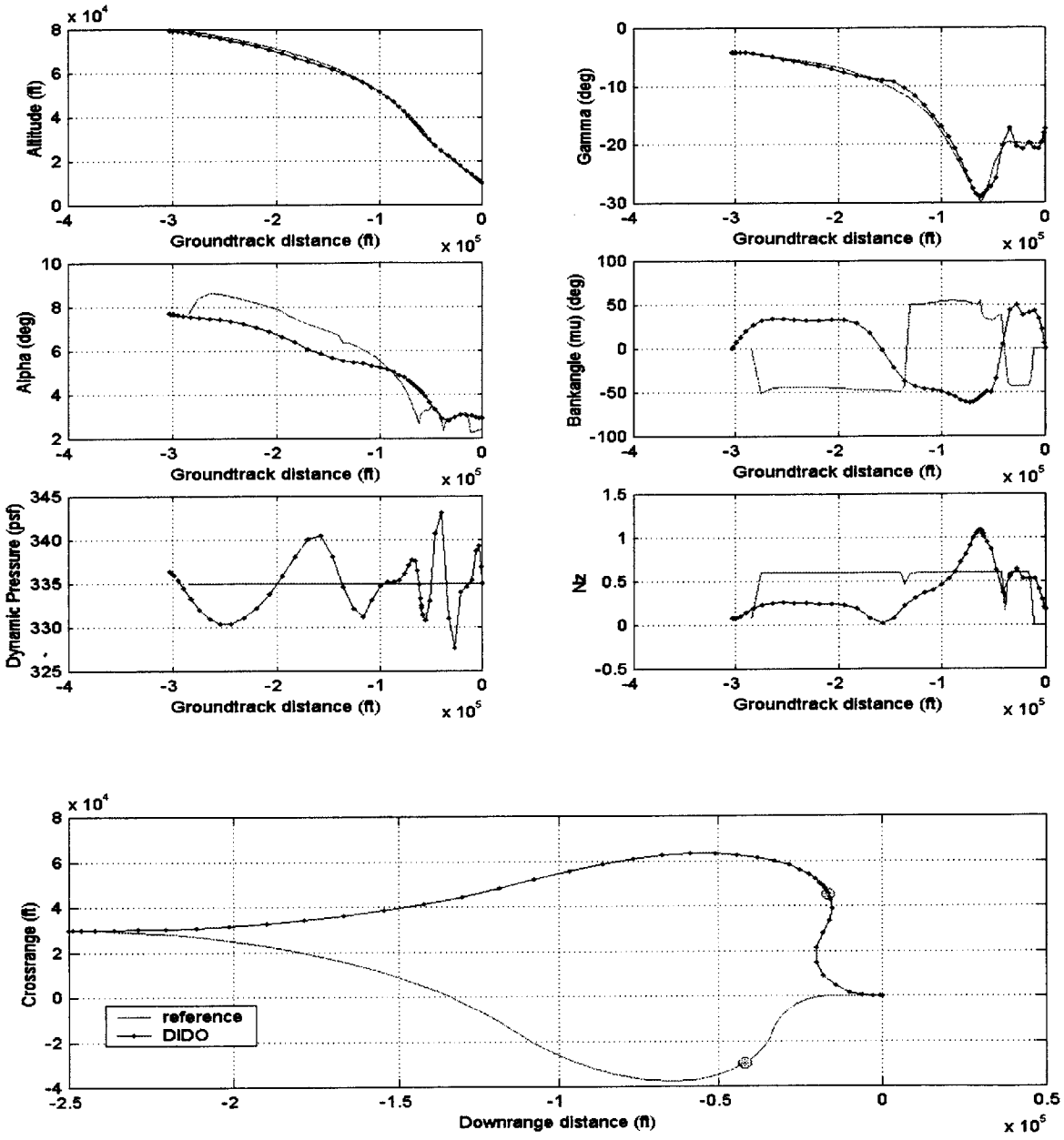


Figure B.3: DIDO Run (qamc1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\overline{q} - \overline{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

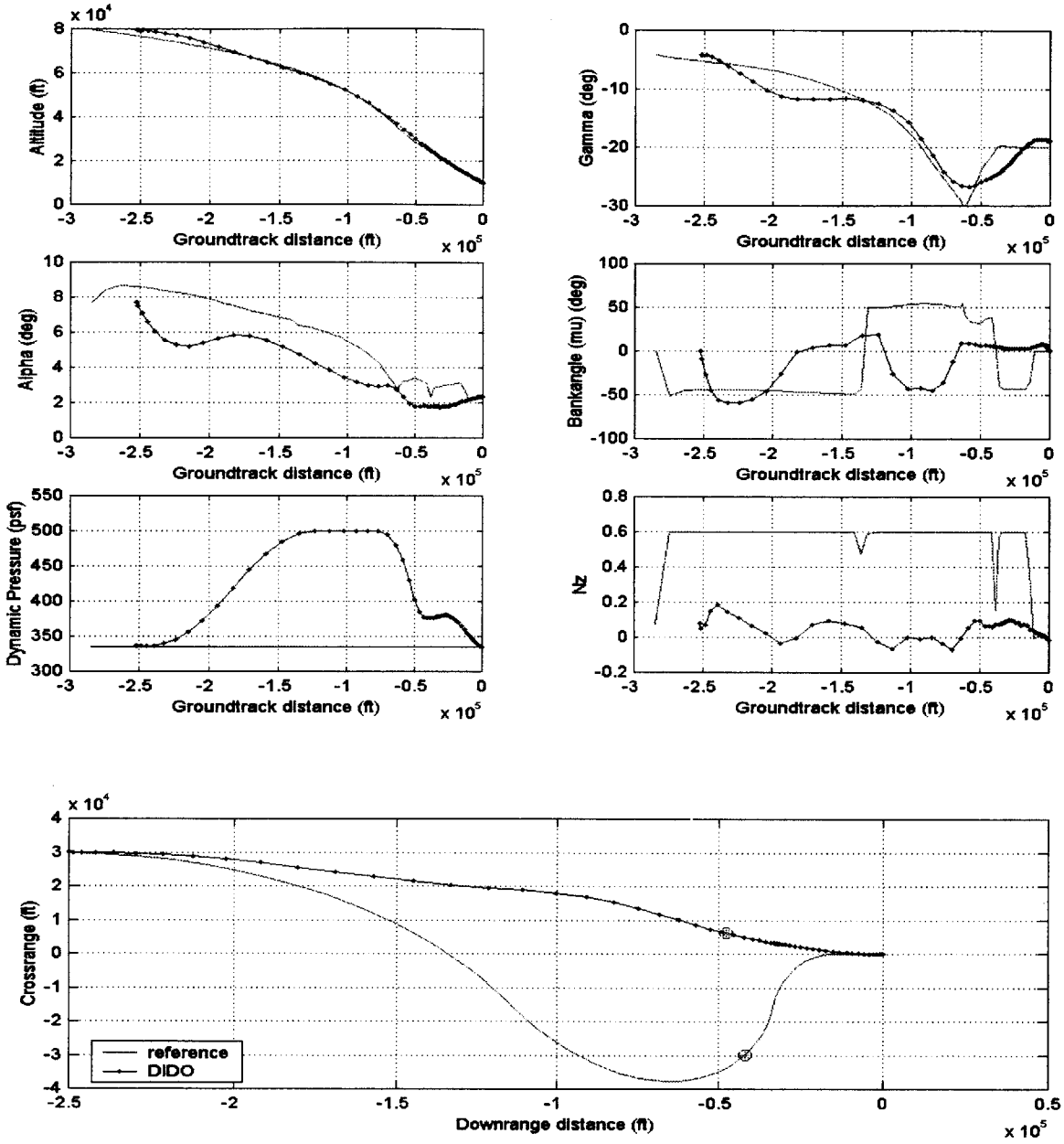


Figure B.4: DIDO Run (minEamnz1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(Nz_b)^2\right) \right] dt$$

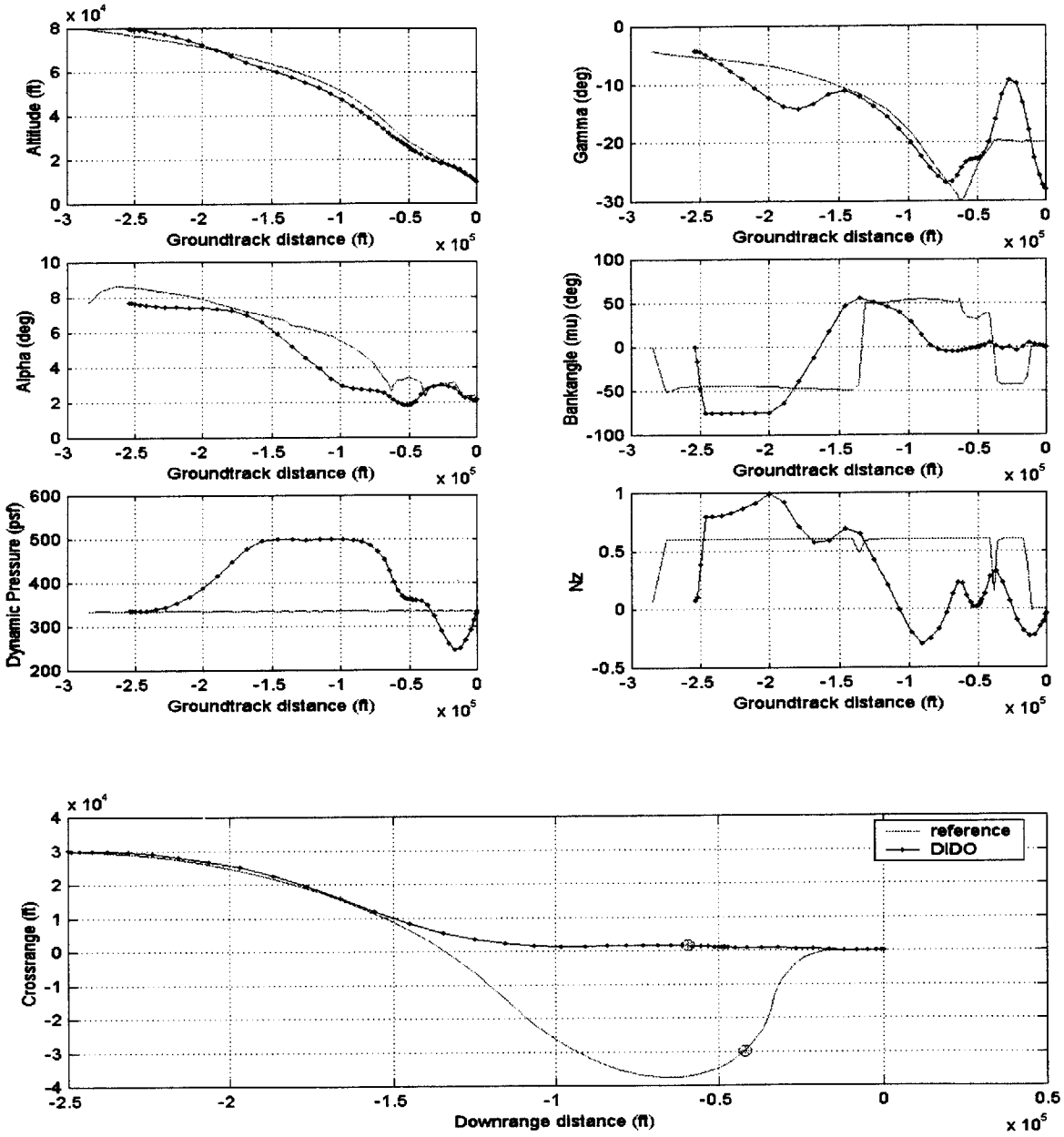


Figure B.5: DIDO Run (minEamy1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

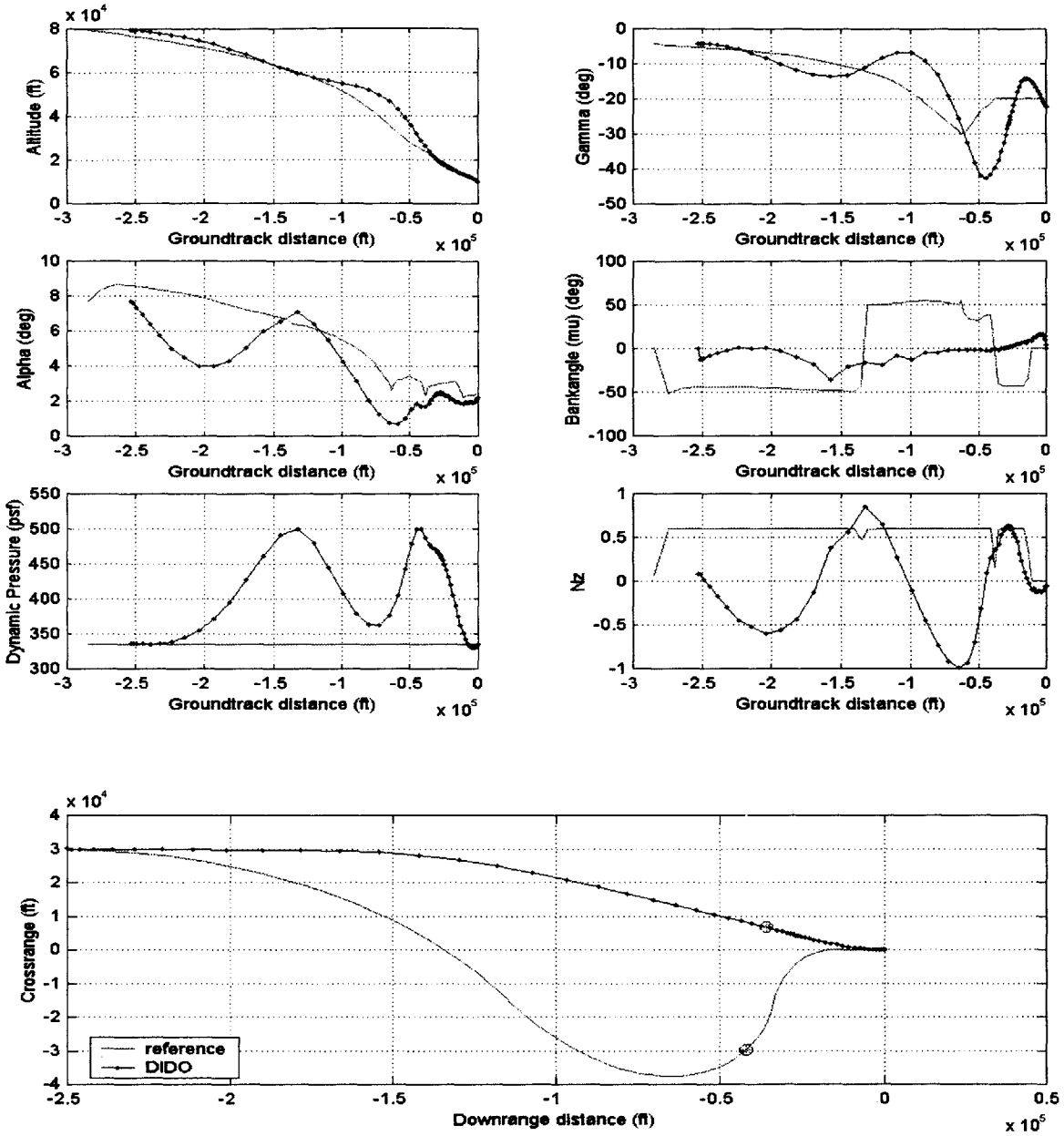


Figure B.6: DIDO Run (maxEamc1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

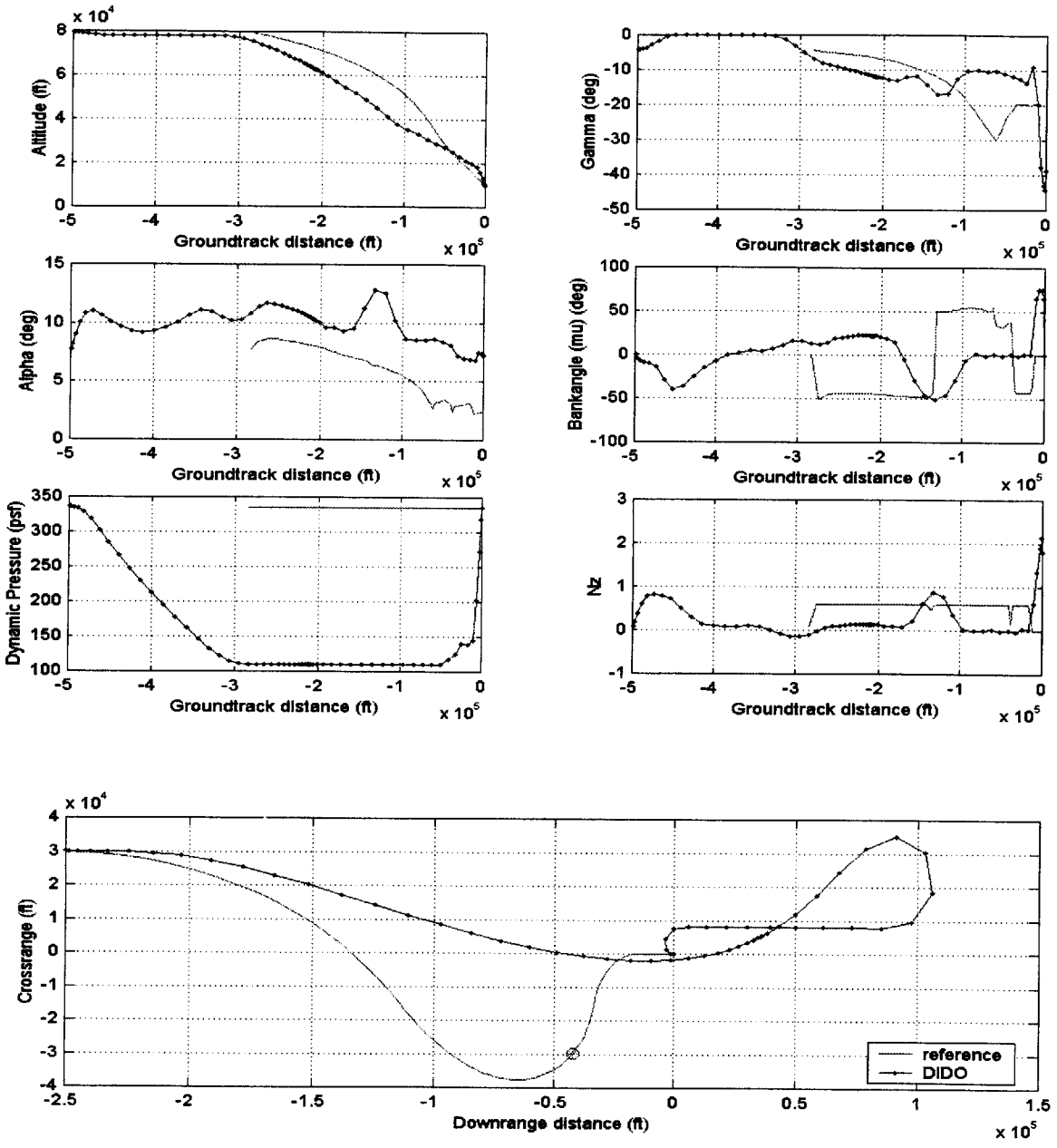


Figure B.7: DIDO Run (maxEamy1) versus Point 1 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

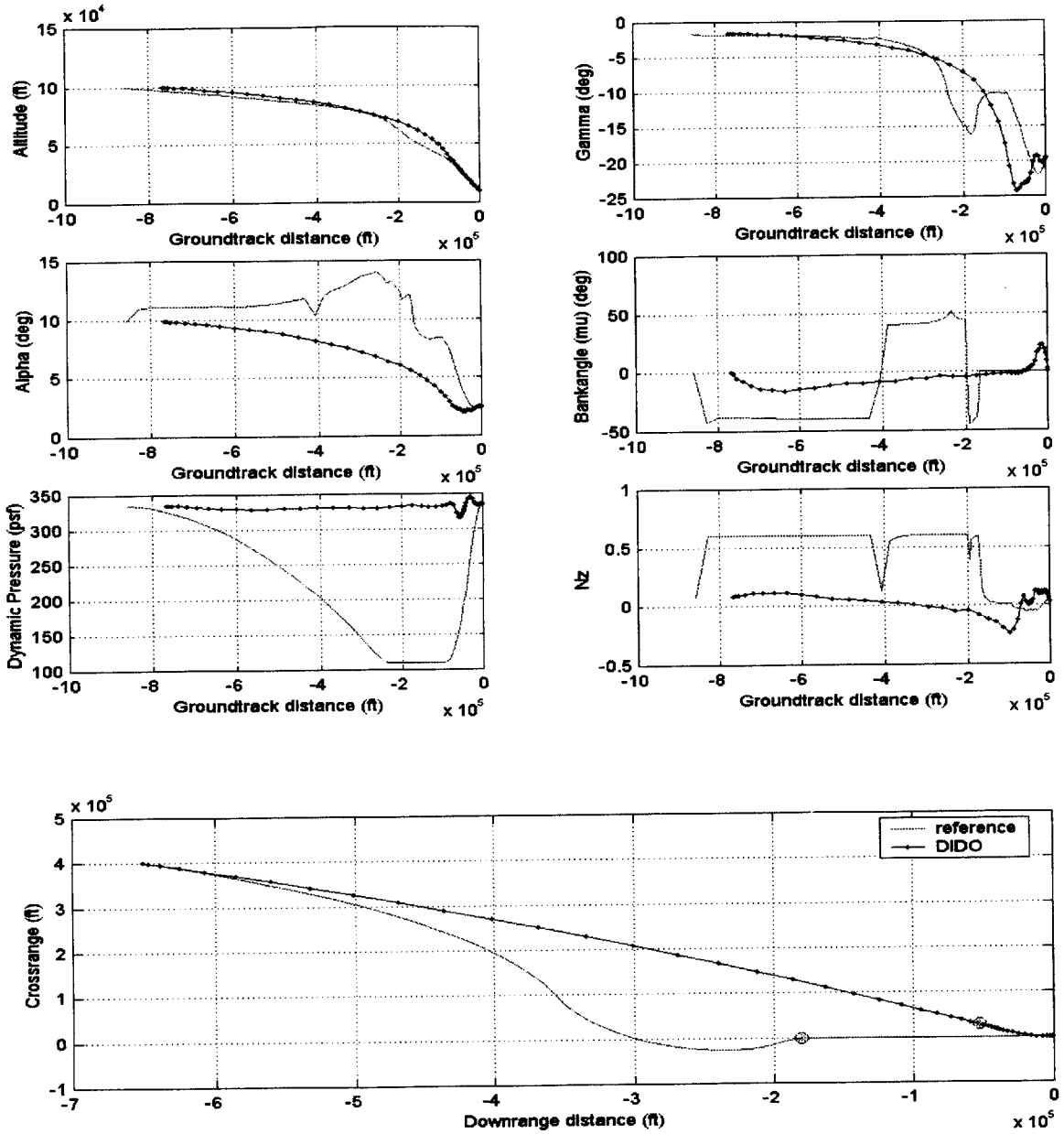


Figure B.8: DIDO Run (qam2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(\dot{\mu})^2 \right] dt$$

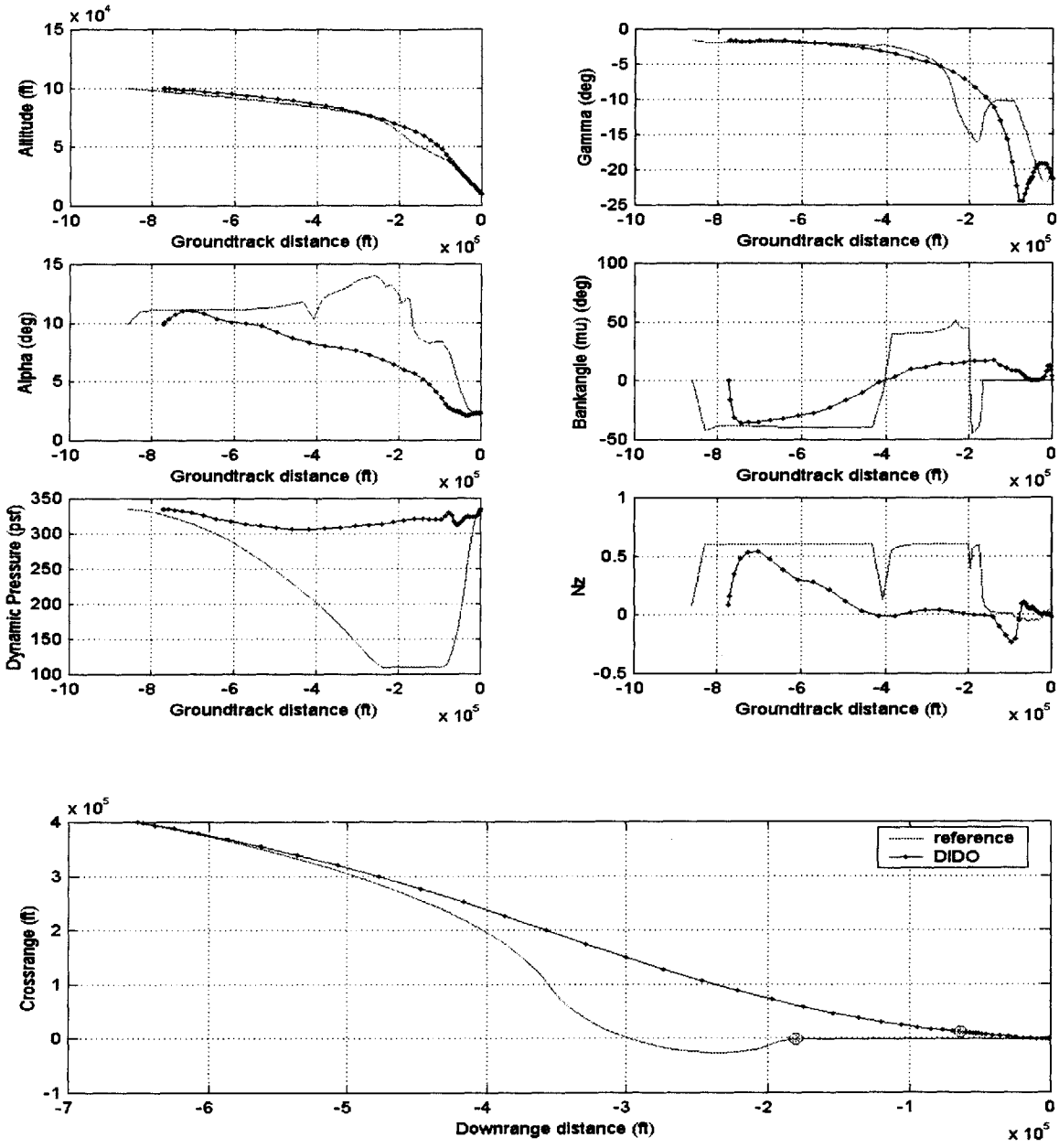


Figure B.9: DIDO Run (qamy2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

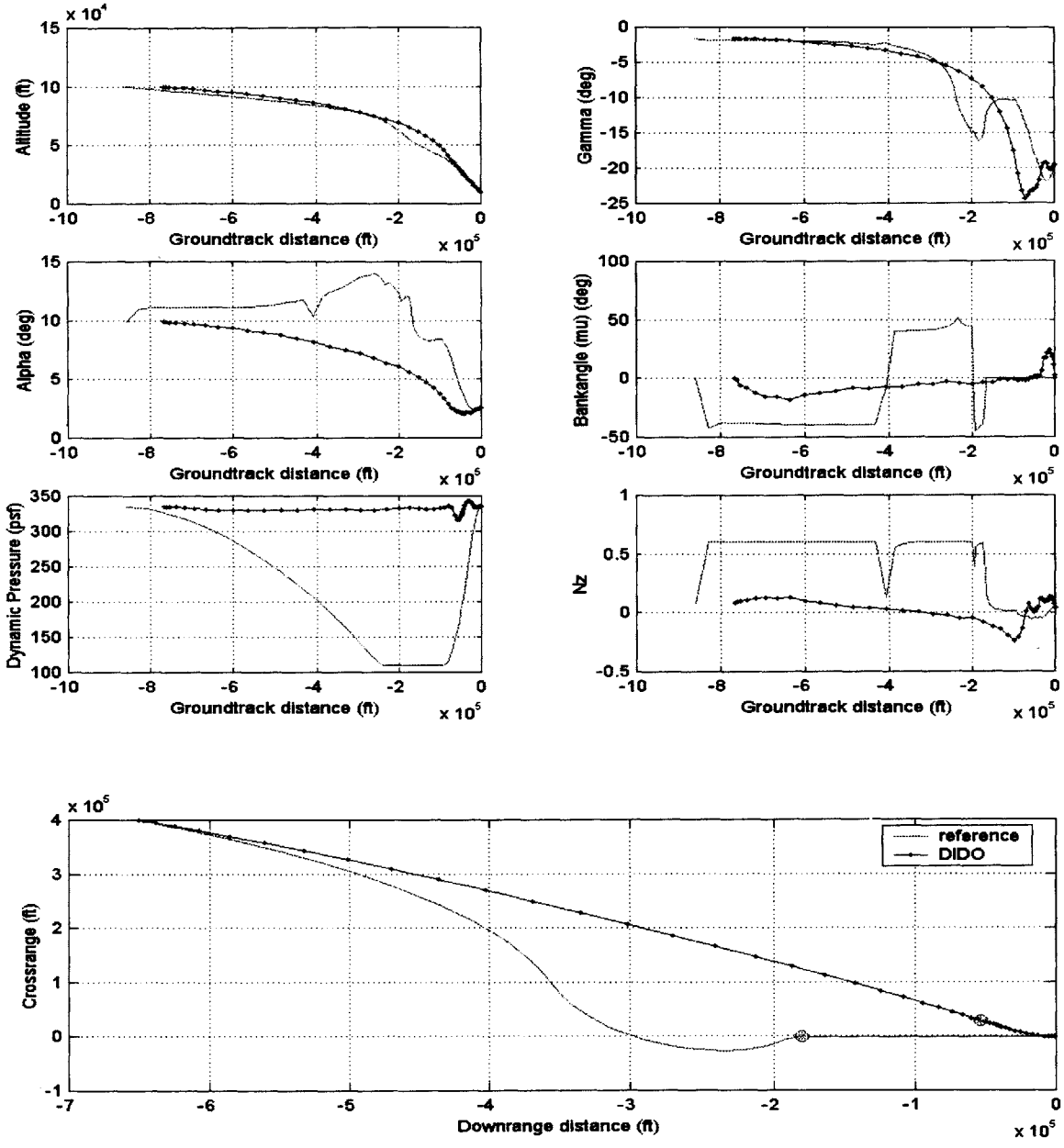


Figure B.10: DIDO Run (qamc2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

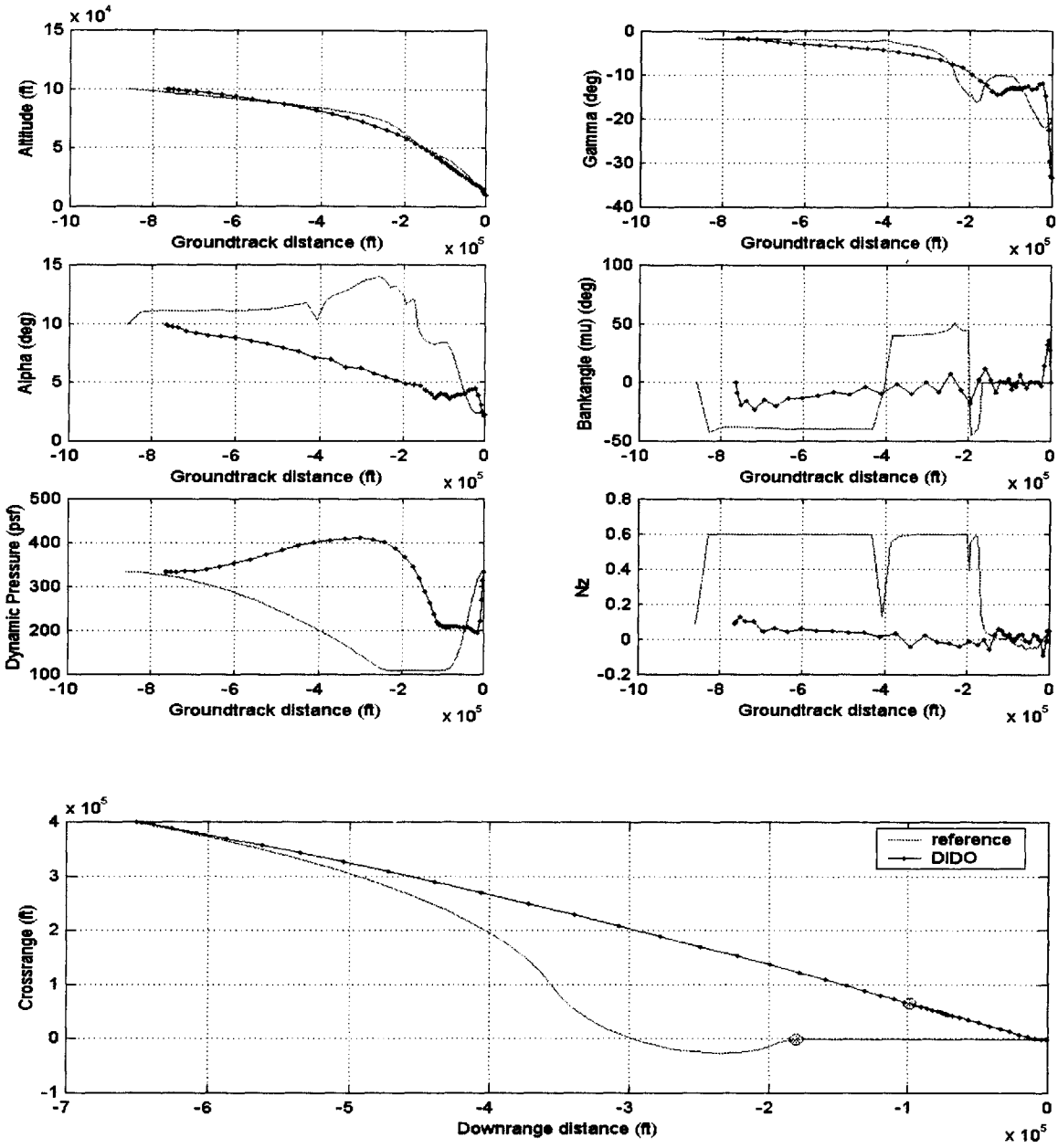


Figure B.11: DIDO Run (minEamnz2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(Nz_b)^2\right) \right] dt$$

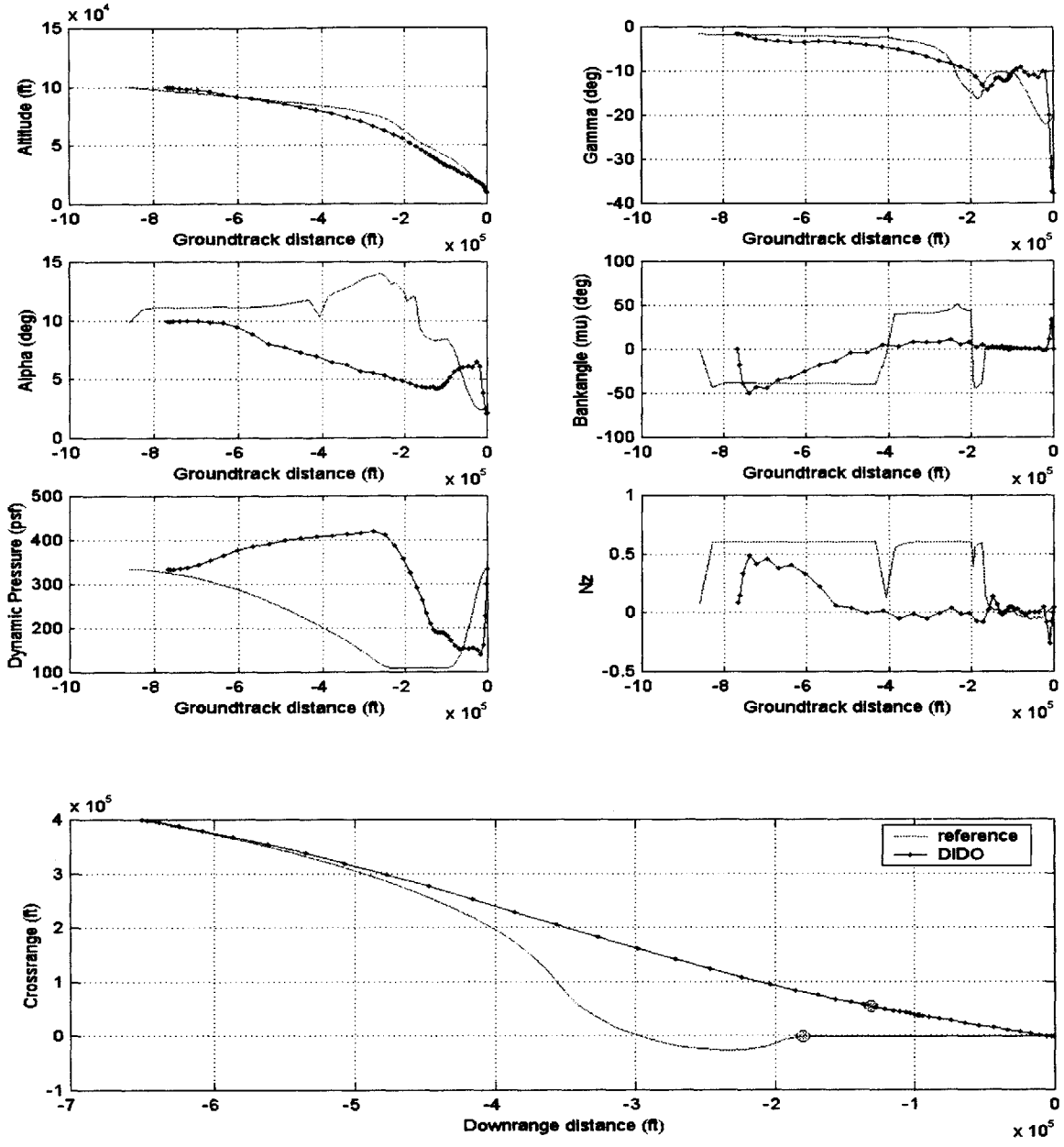


Figure B.12: DIDO Run (minEamy2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

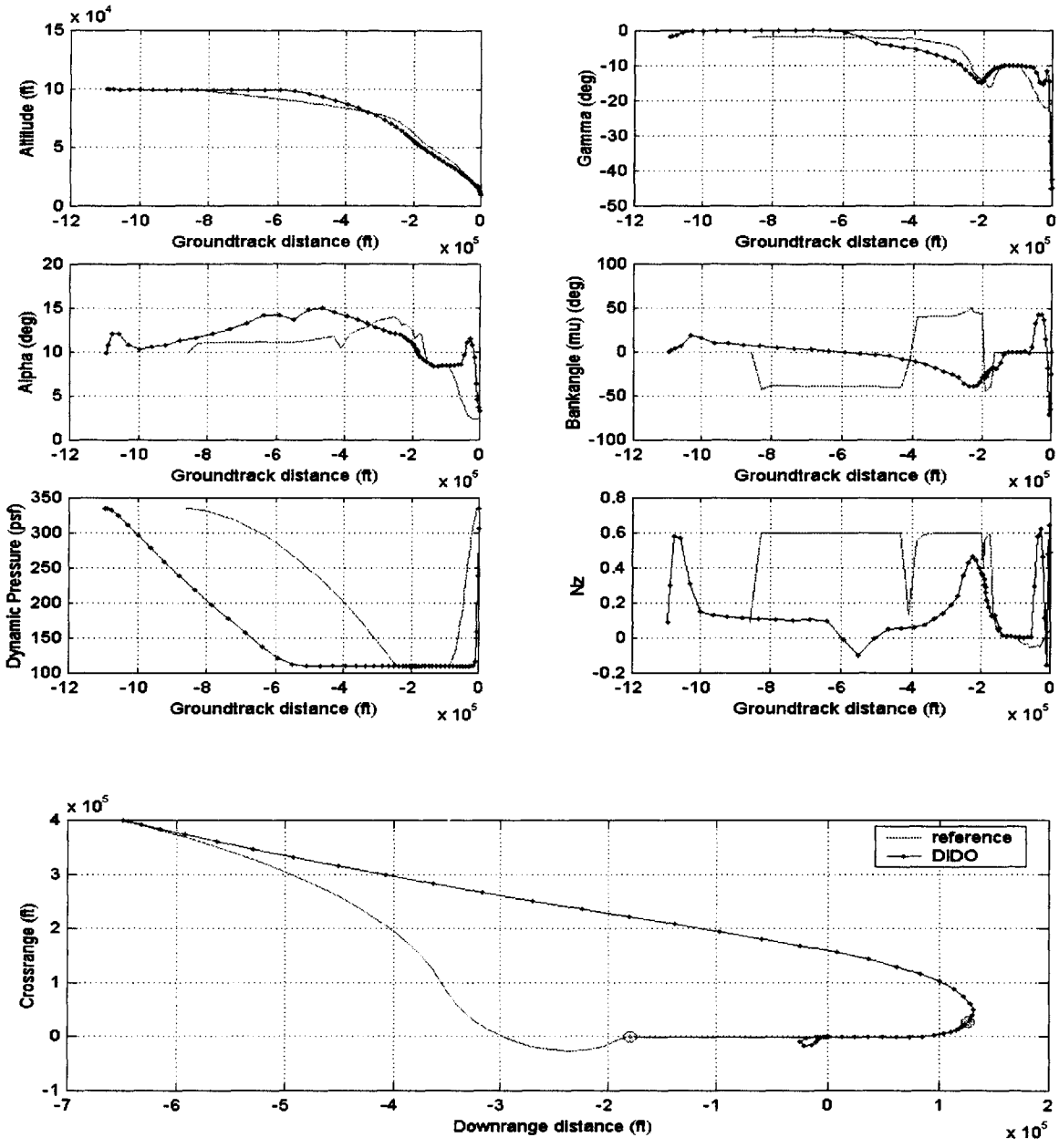


Figure B.13: DIDO Run (maxEamc2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

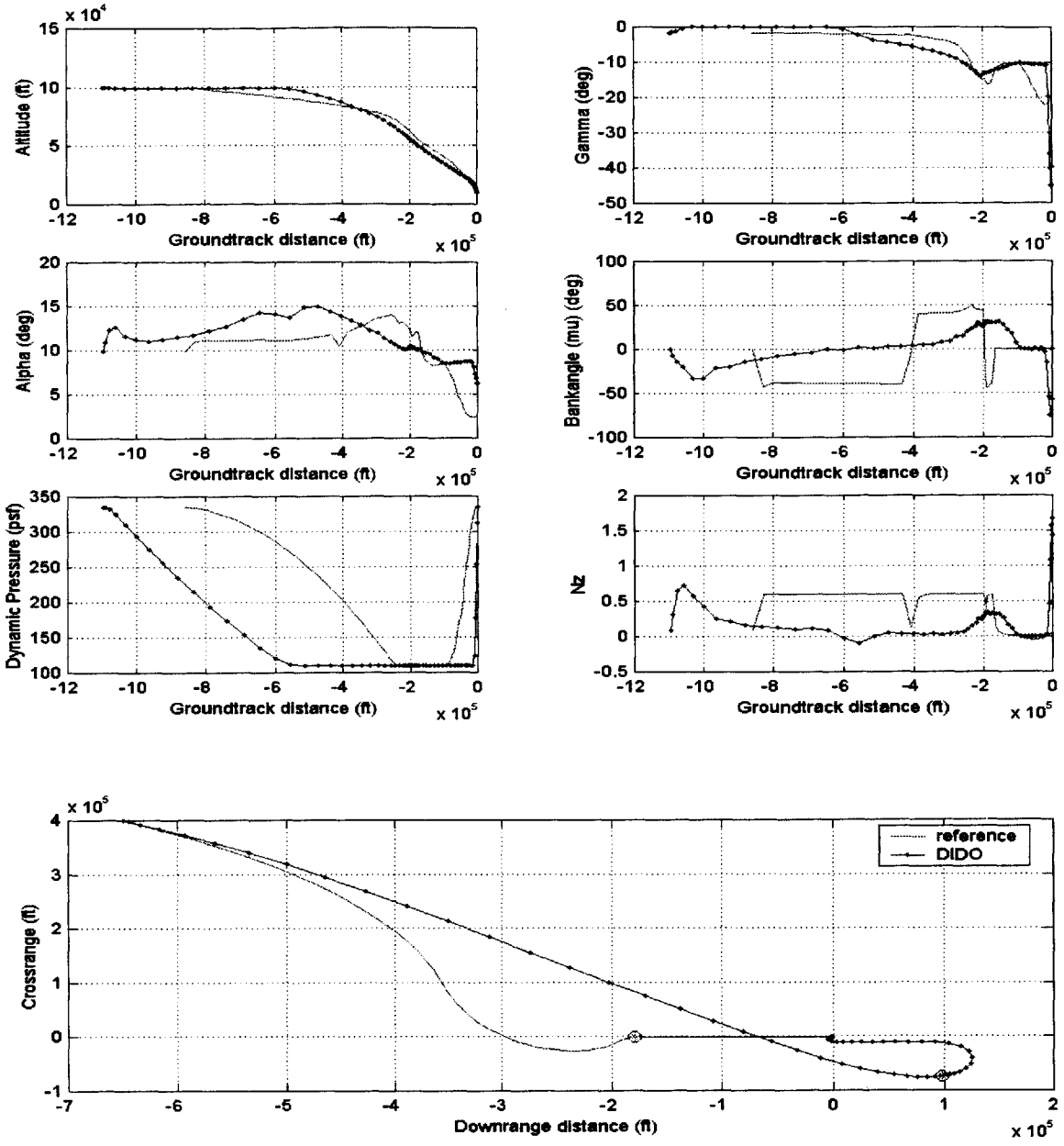


Figure B.14: DIDO Run (maxEamy2) versus Point 2 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

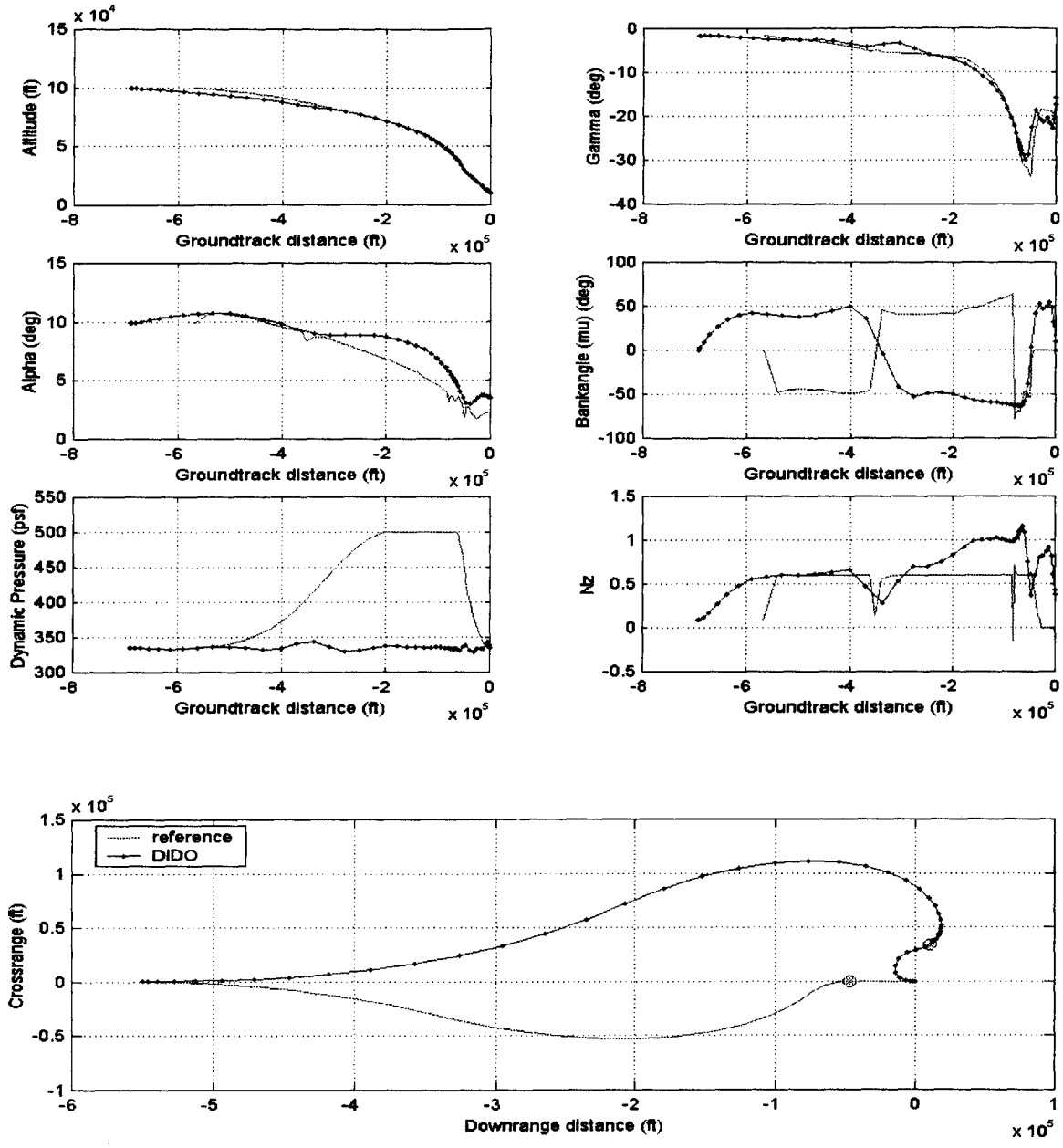


Figure B.15: DIDO Run (qam3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(\dot{\mu})^2 \right] dt$$

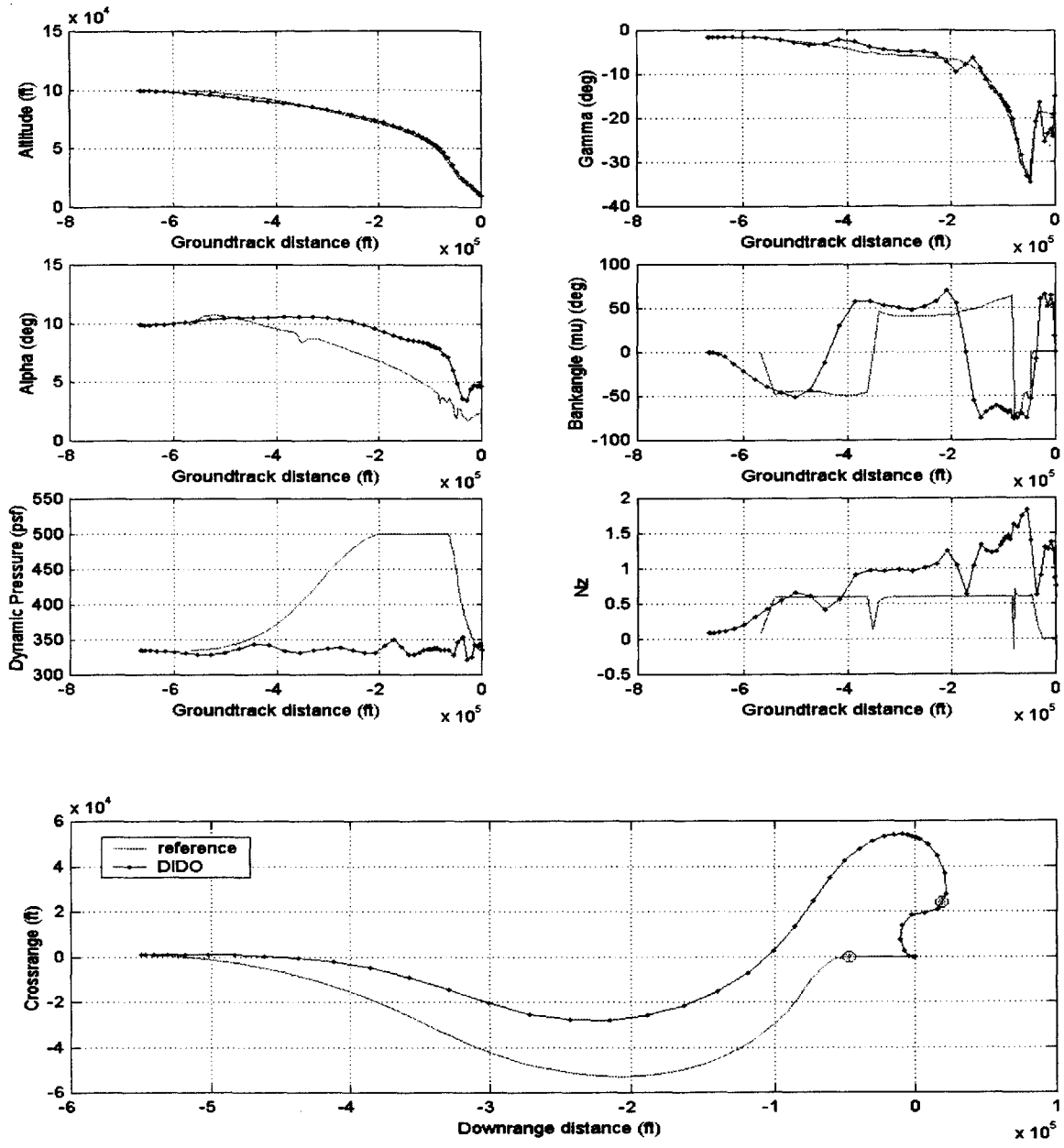


Figure B.16: DIDO Run (qamy3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

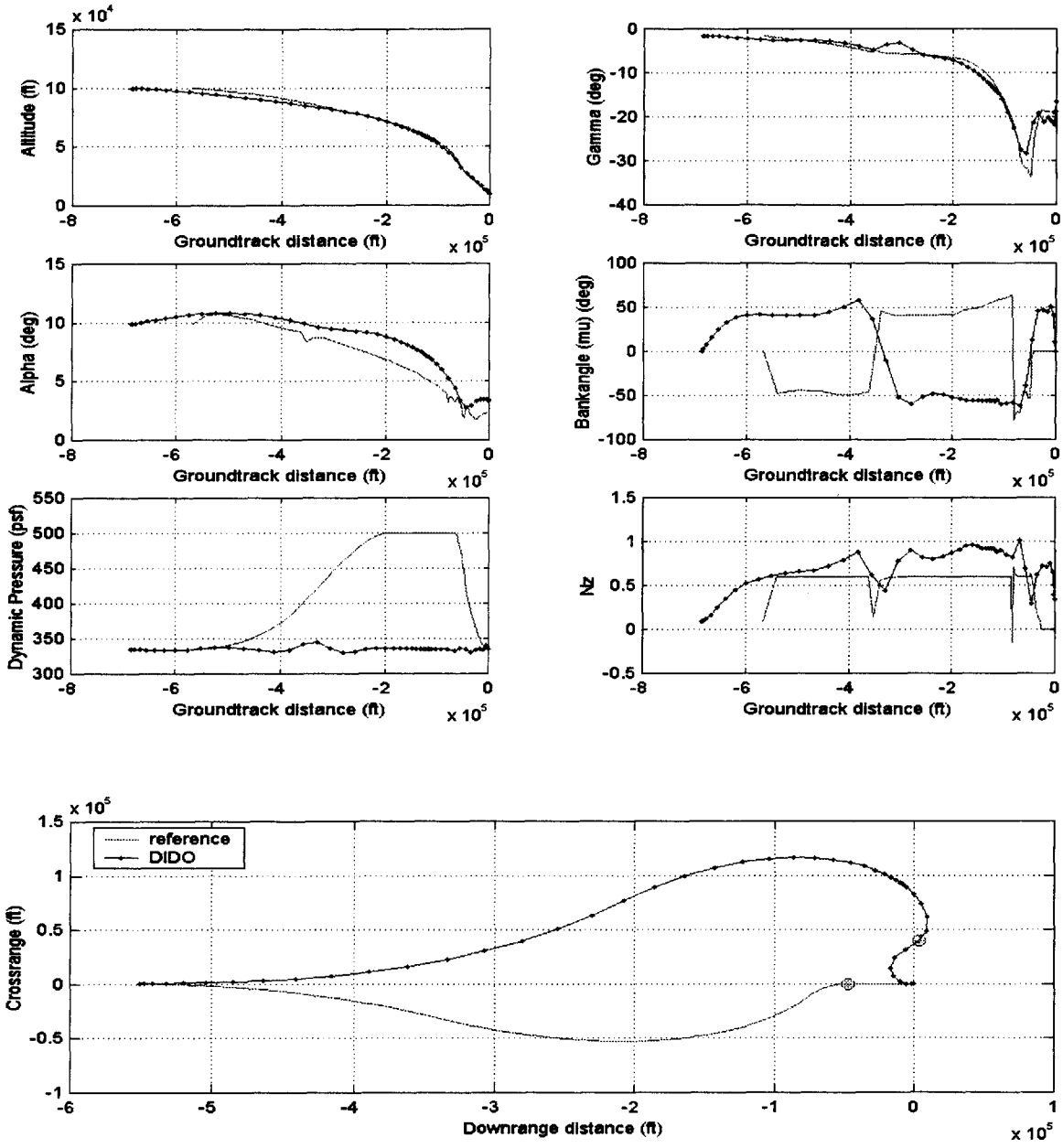


Figure B.17: DIDO Run (qamc3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

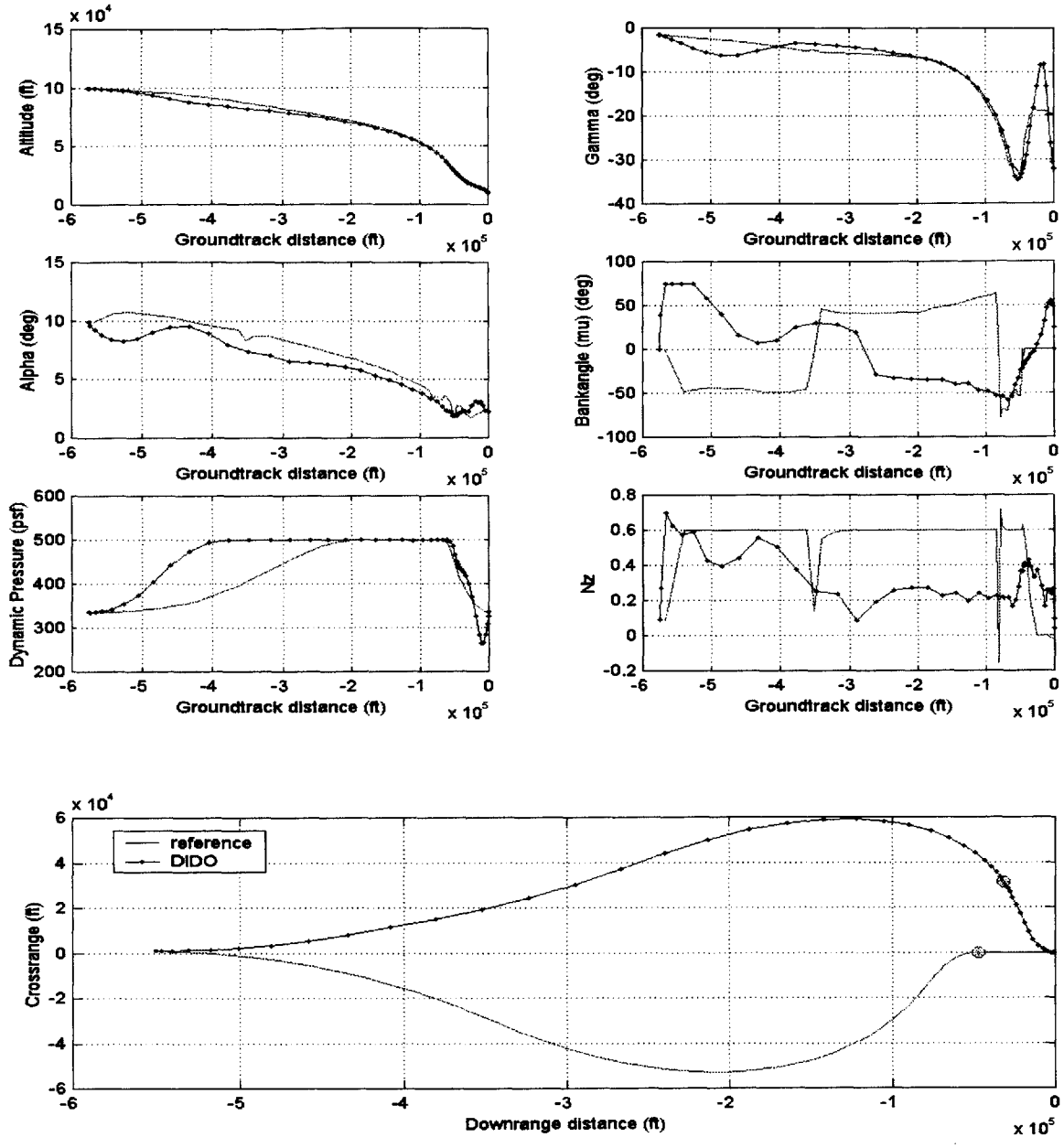


Figure B.18: DIDO Run (minEamnz3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(Nz_b)^2\right) \right] dt$$

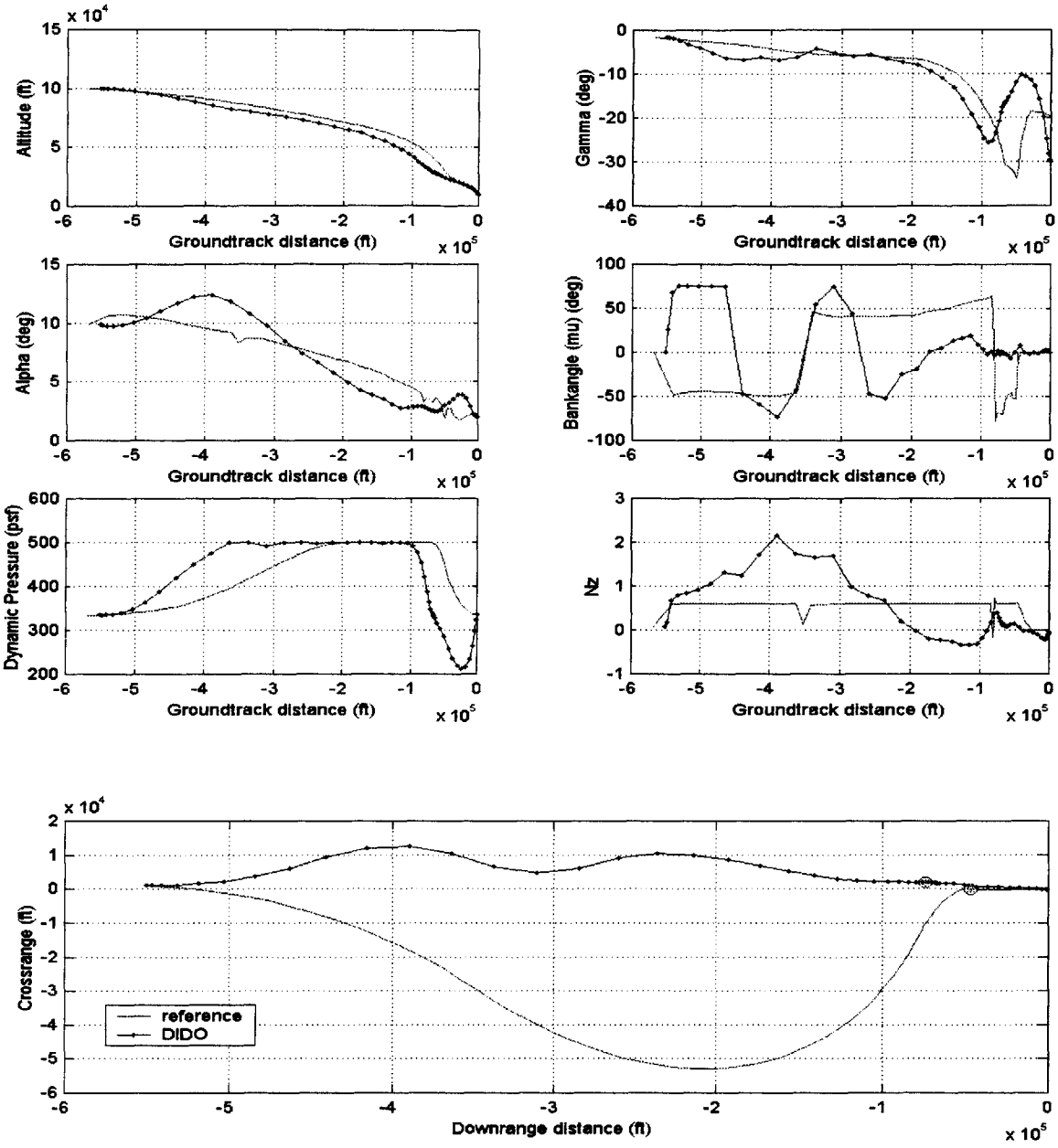


Figure B.19: DIDO Run (minEamy3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

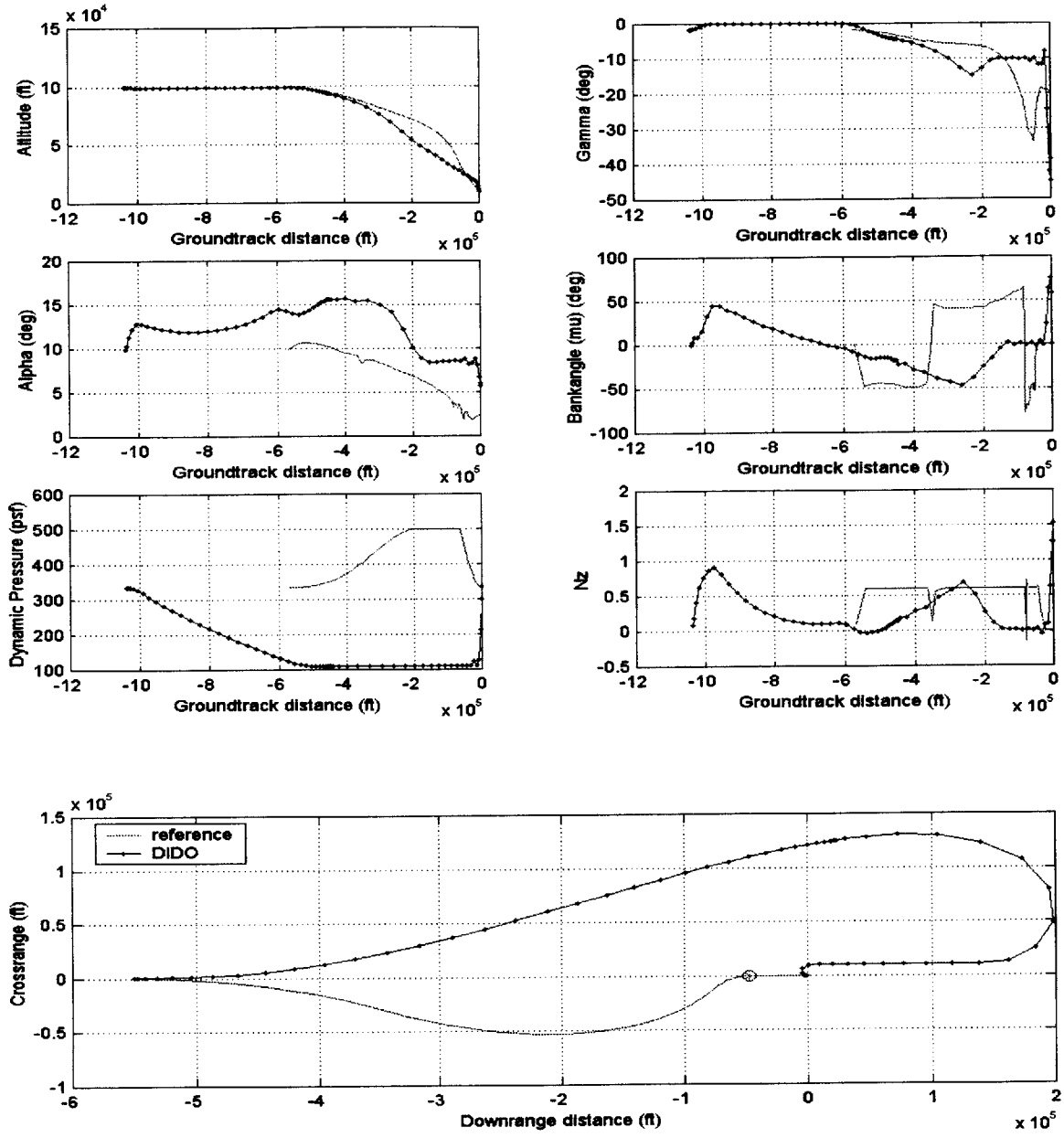


Figure B.20: DIDO Run (maxEamc3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

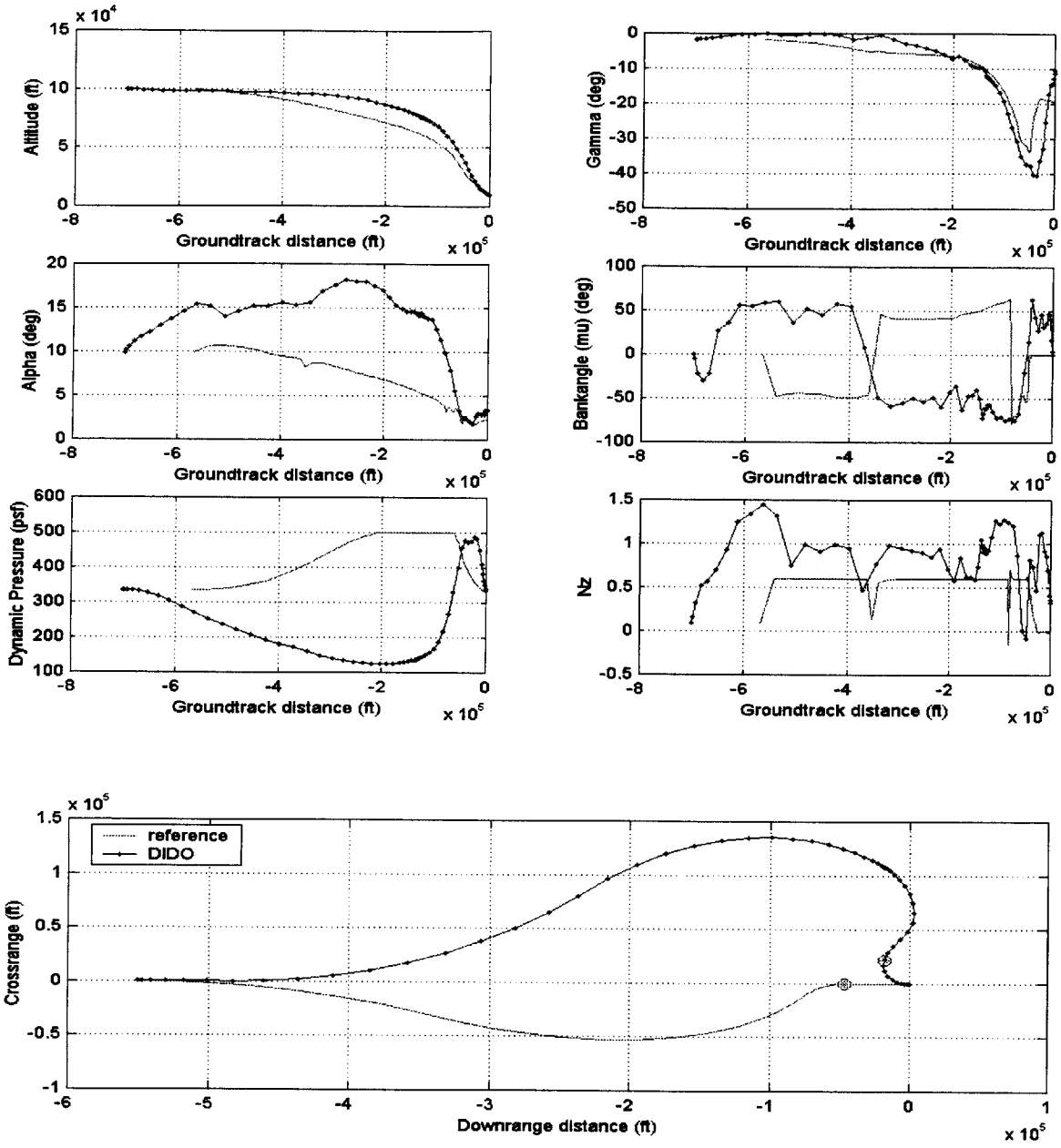


Figure B.21: DIDO Run (maxEamy3) versus Point 3 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

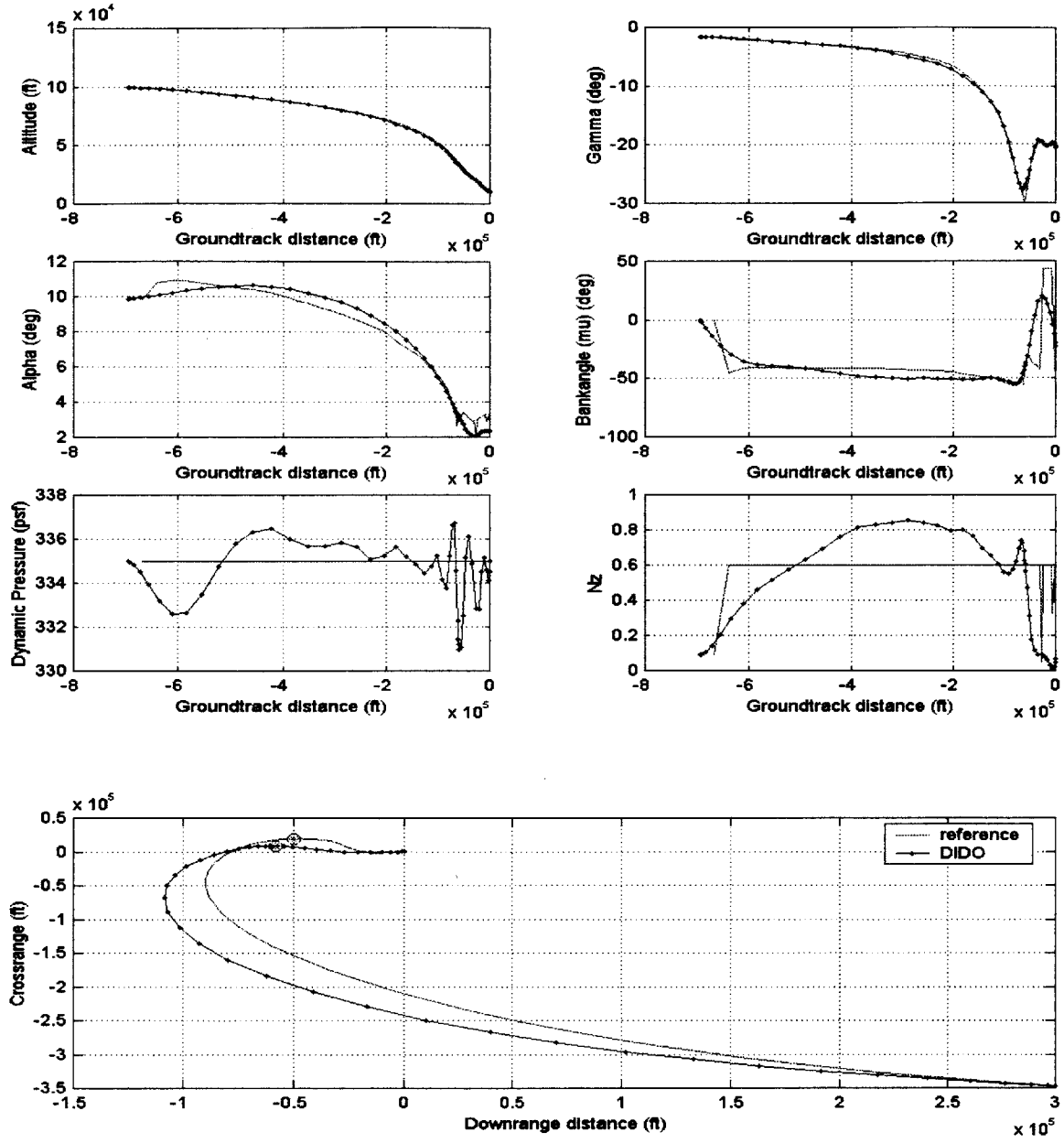


Figure B.22: DIDO Run (qam4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2(\dot{\mu})^2 \right] dt$$

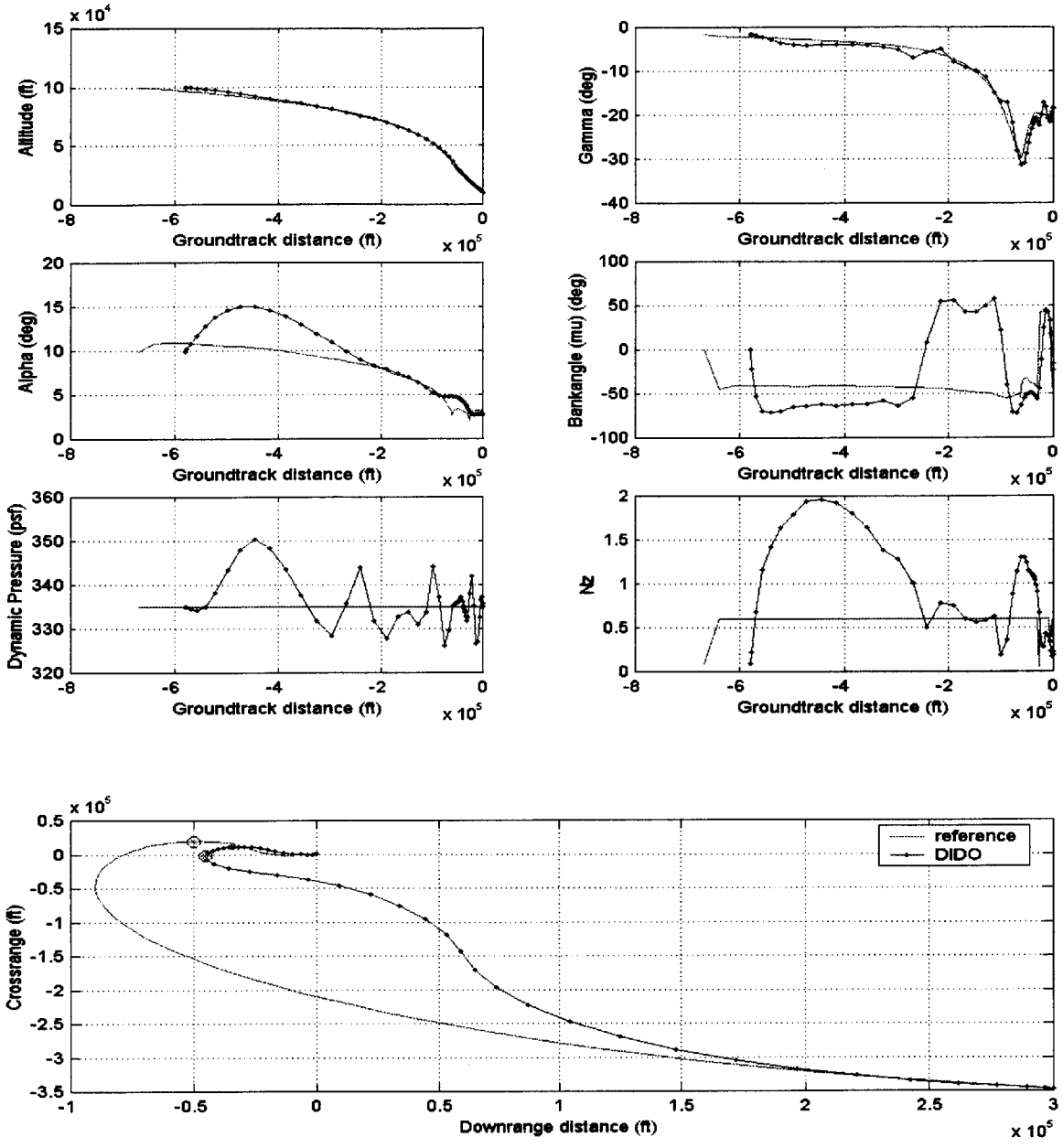


Figure B.23: DIDO Run (qamy4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\overline{q} - \overline{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

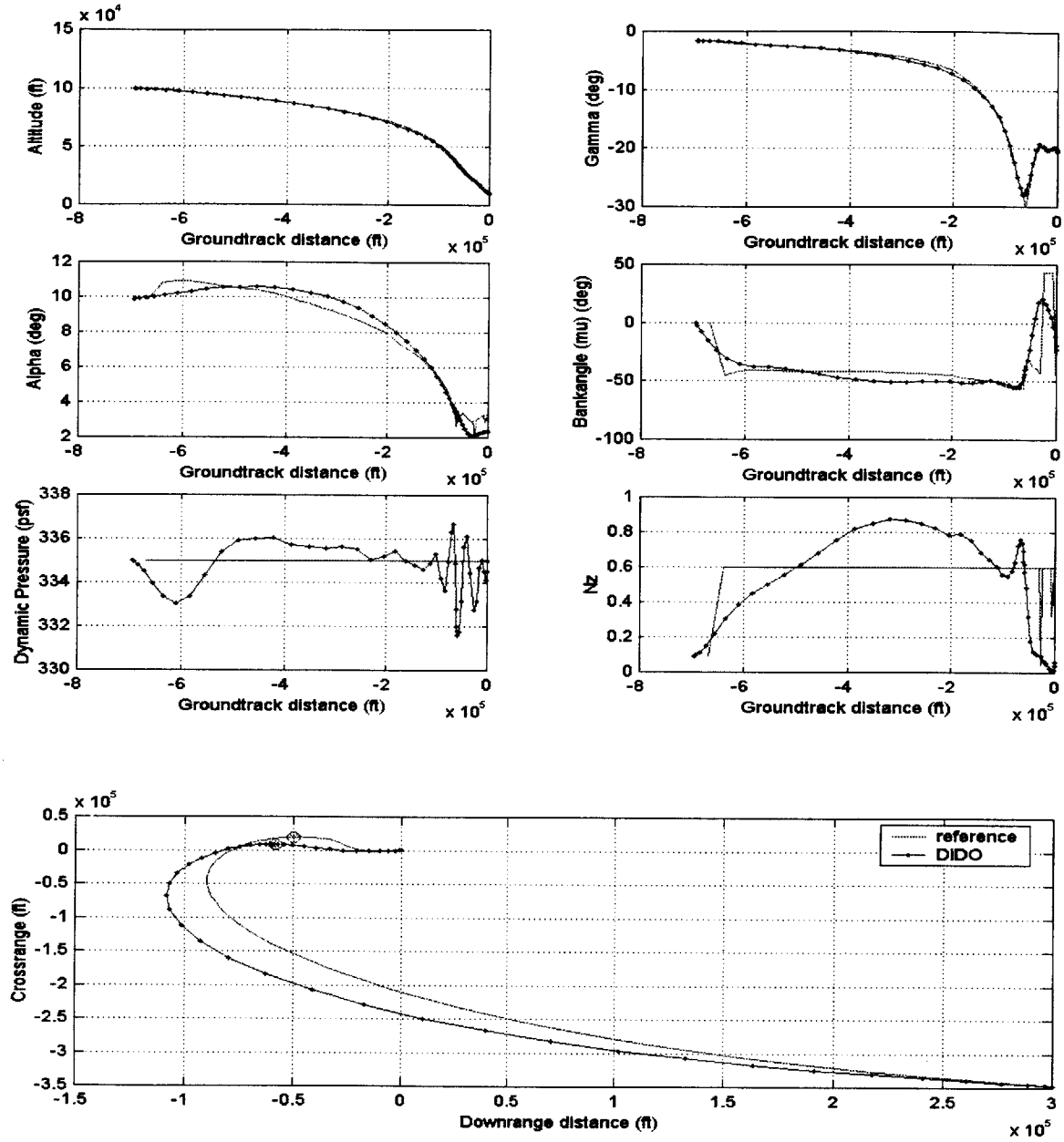


Figure B.24: DIDO Run (qamc4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(\bar{q} - \bar{q}_{ref})^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

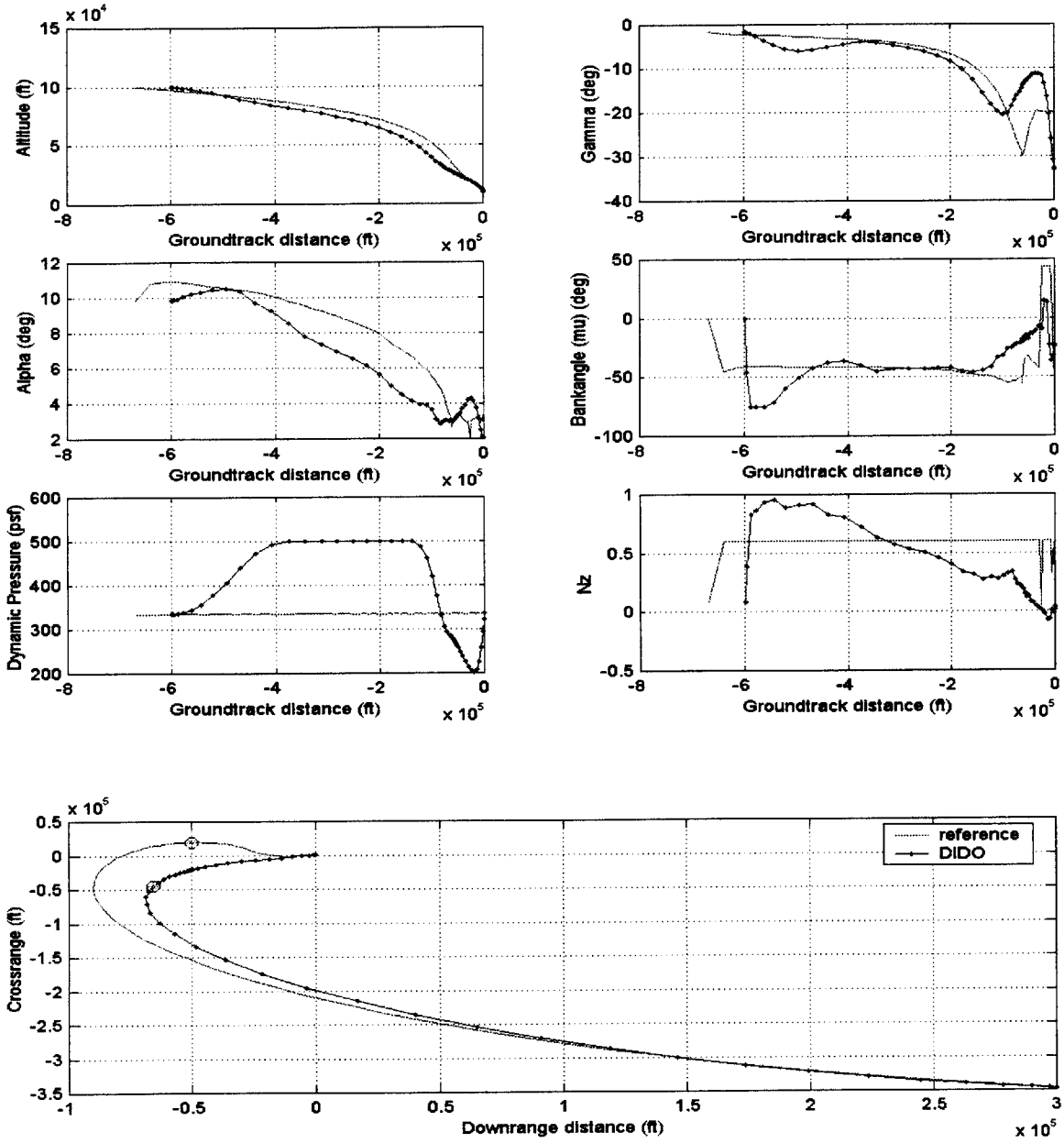


Figure B.25: DIDO Run (minEamnz4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(Nz_b)^2\right) \right] dt$$

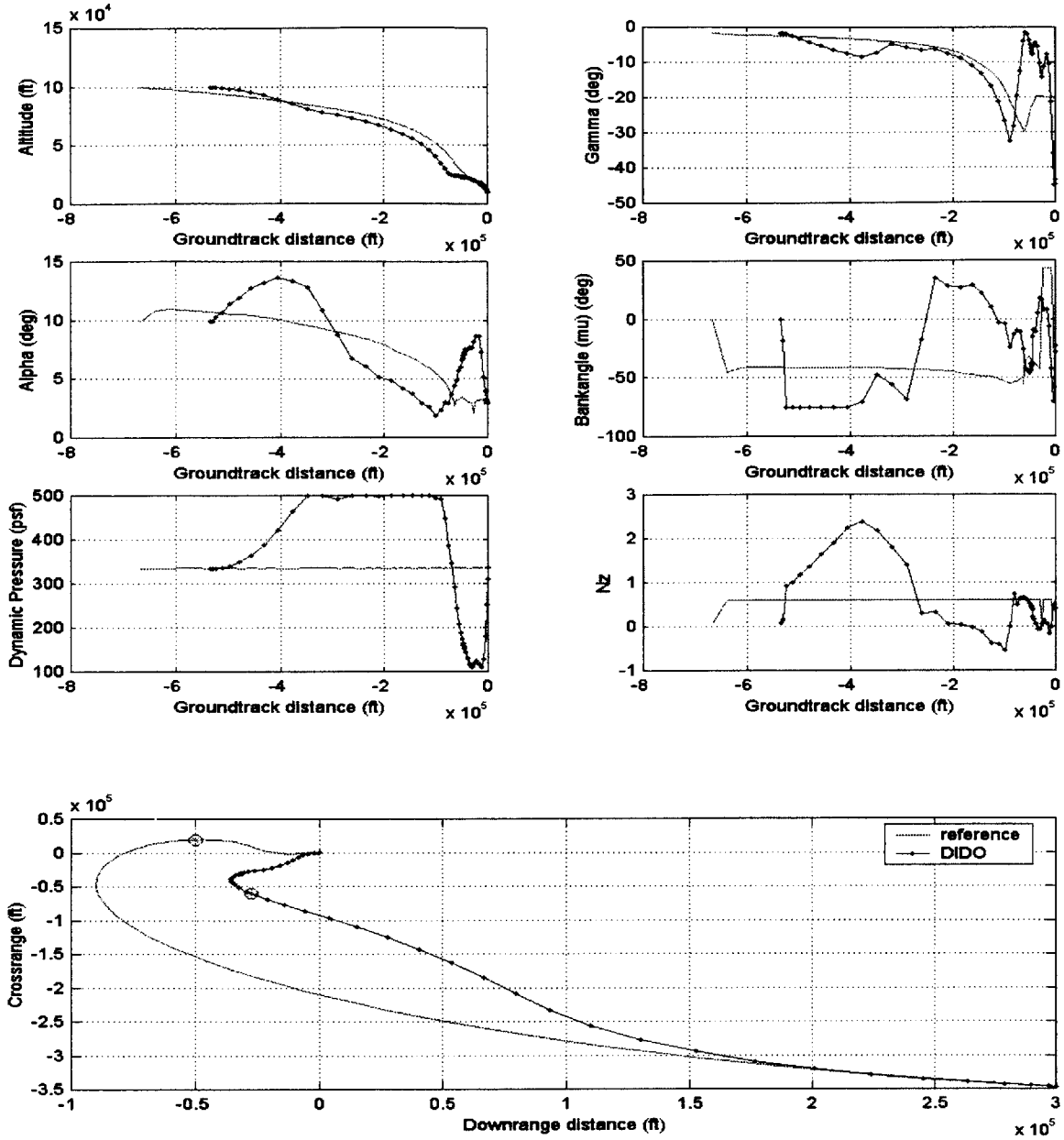


Figure B.26: DIDO Run (minEamy4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

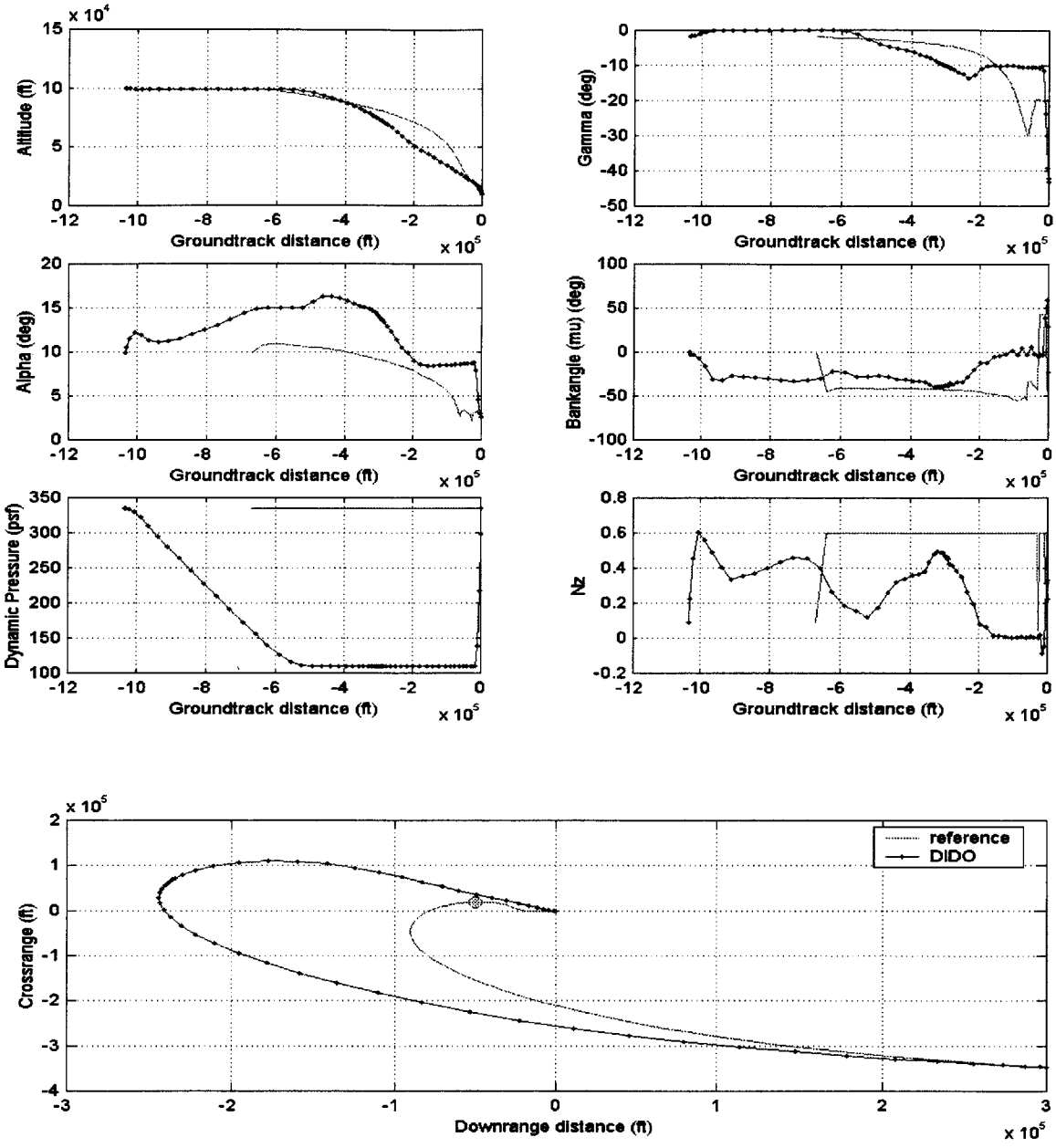


Figure B.27: DIDO Run (maxEamc4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(\dot{\chi})^2\right) \right] dt$$

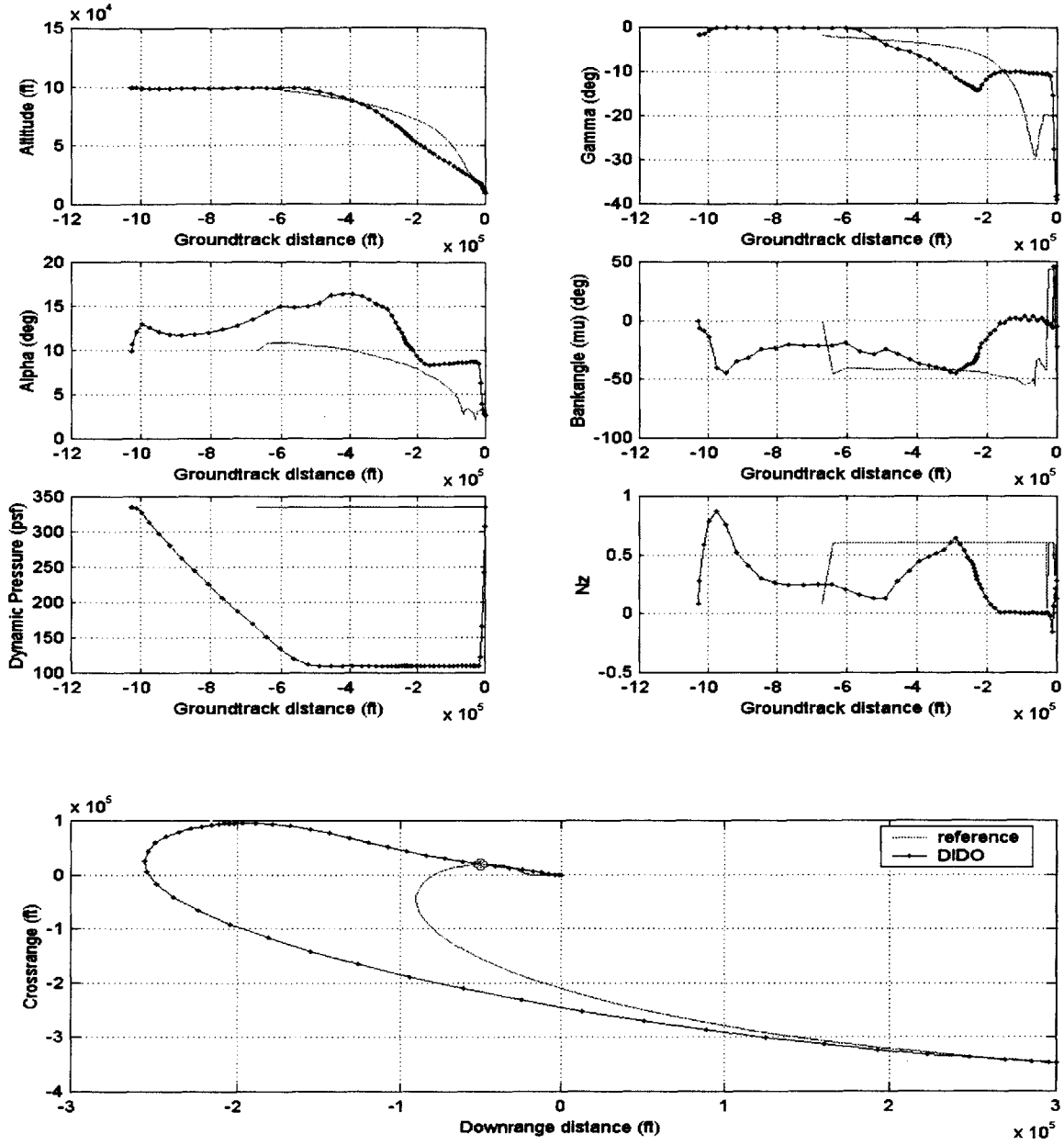


Figure B.28: DIDO Run (maxEamy4) versus Point 4 Reference

Cost Function:

$$J = \int_{t_0}^{t_f} \left[-0.6(E/W)^2 + 0.2(50\dot{\alpha})^2 + 0.2\left(0.5(\dot{\mu})^2 + 0.5(y)^2\right) \right] dt$$

[This page intentionally left blank]

Appendix C

Scoring Results

C.1 Tables of Parameter Scores

The tables of parameter scores are listed below. Across the top of the tables are the trajectory names. The scores for each of the parameters of the trajectories are listed along with the raw total and final grade, which is made up of a weighted sum of the parameter scores.

Table C.1: Point 1 Trajectory Results

	qam1	qamy1	qamc1	minEamnz1	minEamy1	maxEamc1	maxEamy1	Proto-snake
$\overline{q}$ *score*	0.0000	0.0006	0.0000	0.2139	0.2918	0.2901	0.7485	0.0000
Nz_h *score*	0.3097	0.6246	0.2897	0.0000	0.1459	0.0774	0.1390	0.2198
χ *score*	0.4589	0.3257	0.5636	0.0789	0.0169	0.1145	0.7186	0.4031
y *score*	0.7320	0.1749	0.6995	0.0934	0.0500	0.2380	0.8750	0.5499
$\dot{\alpha}$ *score*	0.1141	0.3315	0.1042	0.2023	0.2036	0.4752	0.3150	0.2967
$\dot{\mu}$ *score*	0.1298	0.3827	0.1037	0.1387	0.1408	0.1133	0.0945	0.0883
Raw total	1.7445	1.8400	1.7607	0.7272	0.8490	1.3085	2.8906	1.5578
Grade	17.2110	23.3490	17.4155	13.6560	20.3785	21.7175	54.5085	14.2005

Table C.2: Point 2 Trajectory Results

	qam2	qamy2	qamc2	minEamnz2	minEamy2	maxEamc2	maxEamy2	Proto-snake
$\overline{q}$ *score*	0.0026	0.0290	0.0025	0.2758	0.3680	0.7953	0.7941	0.6466
Nz_h *score*	0.0000	0.0142	0.0000	0.0000	0.0097	0.0393	0.0899	0.1166
χ *score*	0.2875	0.1057	0.2928	0.2501	0.1688	0.3556	0.7040	0.0000
y *score*	0.7237	0.3068	0.7450	0.7244	0.6045	1.0000	1.0000	0.0893
$\dot{\alpha}$ *score*	0.0704	0.1543	0.0702	0.1273	0.1491	0.2772	0.1878	0.1886
$\dot{\mu}$ *score*	0.0121	0.0559	0.0169	0.0829	0.0920	0.0636	0.0693	0.0379
Raw total	1.0963	0.6659	1.1274	1.4605	1.3921	2.5310	2.8451	1.0790
Grade	7.0360	5.4470	7.2135	20.9640	24.5585	51.0075	55.2780	36.8240

Table C.3: Point 3 Trajectory Results

	qam3	qamy3	qamc3	minEamnz3	minEamy3	maxEamc3	maxEamy3	Proto-snake
$\overline{q}$ *score*	0.0000	0.0005	0.0000	0.5164	0.4243	0.7265	0.5419	0.5646
Nz_b *score*	0.5257	0.6837	0.4707	0.0500	0.2464	0.1287	0.6152	0.2025
χ *score*	0.5997	0.6600	0.5711	0.4355	0.0109	0.6952	0.5601	0.0607
y *score*	0.6645	0.7220	0.6512	0.7827	0.0466	0.7808	0.6435	0.0161
$\dot{\alpha}$ *score*	0.1799	0.1420	0.1520	0.3374	0.2889	0.2334	0.5872	0.3329
$\dot{\mu}$ *score*	0.1007	0.1553	0.0978	0.1817	0.2814	0.0772	0.2659	0.1041
Raw total	2.0705	2.3635	1.9428	2.3037	1.2985	2.6418	3.2138	1.2809
Grade	23.8650	28.8116	21.9835	37.9340	30.5685	51.9515	55.5590	36.1650

Table C.4: Point 4 Trajectory Results

	qam4	qamy4	qamc4	minEamnz4	minEamy4	maxEamc4	maxEamy4	Proto-snake
$\overline{q}$ *score*	0.0000	0.0000	0.0000	0.4199	0.6163	0.7737	0.7866	0.0000
Nz_b *score*	0.2344	0.6106	0.2386	0.1646	0.2736	0.0485	0.0714	0.2391
χ *score*	0.1021	0.2692	0.1019	0.1772	0.4234	0.2963	0.1831	0.2463
y *score*	0.0639	0.4820	0.0633	0.5028	0.6926	0.5816	0.5144	0.5800
$\dot{\alpha}$ *score*	0.1357	0.3374	0.1358	0.1890	0.5451	0.2742	0.2590	0.2524
$\dot{\mu}$ *score*	0.0432	0.1981	0.0480	0.0954	0.1919	0.0616	0.0772	0.0570
Raw total	0.5793	1.8973	0.5876	1.5489	2.7429	2.0359	1.8917	1.3748
Grade	8.0950	23.0465	8.2195	30.8180	49.0370	47.4475	47.1990	12.8875

References

[1] Grubler, A. C., *New Methodologies for Onboard Generation of Terminal Area Energy Management Trajectories for Autonomous Reusable Launch Vehicles.* S.M. Thesis, Department of Aeronautics and Astronautics, MIT, June 2001.

[2] Aron, E., Barton, G., and Bottkol, M., "RADX-34 – A Future-X Demonstration of Autonomous Robust Abort Technologies on the X-34," American Astronautical Society Paper 01-015, February 2001.

[3] Orbital Sciences X-34 Homepage, "X-34 Reusable Rocketplane," URL: http://www.orbital.com/LaunchVehicles/X34/x-34.htm [cited 18 April 2002].

[4] Marshall Space Flight Center Fact Sheets, "X-34: Demonstrating Reusable Launch Vehicle Technologies," URL: http://www1.msfc.nasa.gov/NEWSROOM/background/facts/x-34.htm [cited 12 February 2001].

[5] Anderson, J. D., *Aircraft Performance and Design*, WCB/McGraw-Hill, New York, NY, 1999.

[6] Etkin, B. and Reid L. D., *Dynamics of Flight: Stability and Control,* 3rd ed., John Wiley and Sons, Inc., New York, NY, 1996.

[7] Girerd, A. R., and Barton, G. H., "Next Generation Entry Guidance—Onboard Trajectory Generation for Unpowered Drop Tests," AIAA Paper 2000-3960, 2000.

[8] Moore, T., "Space Shuttle Entry Terminal Area Energy Management," NASA Johnson Space Center, NASA Technical Memorandum 104744, Nov. 1991.

[9] James, J., *Entry Guidance Training Manual.* NASA Johnson Space Center, Flight Training Branch, TD357, Houston, TX, July 1988.

[10] Tsikalas, G. M., "Space Shuttle Autoland Design," AIAA Paper 82-1604-CP, presented at AIAA Guidance and Control Conference, August 9-11, 1982.

[11] Barton, G. H., and Tragresser, S. G., "Autolanding Trajectory Design for the X-34," AIAA Paper 99-4161, August 1999.

[12] Barton, G. H., "New Methodologies for Assessing the Robustness of the X-34 Autolanding Trajectories," American Astronautical Society Paper 01-014, 2001.

[13] Girerd, A. R., *Onboard Trajectory Generation for the Unpowered Landing of Autonomous Reusable Launch Vehicles.* S.M. Thesis, Department of Aeronautics and Astronautics, MIT, June 2001.

[14] Ross, I. M. and Fahroo, F., "User's Manual for DIDO 2001: A MATLAB Application Package for Dynamic Optimization," NPS Technical Report AA-01-003, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA, October 2001.

[15] Fahroo, F. and Ross, I. M., "Costate Estimation by a Lengendre Psuedospectral Method," *Journal of Guidance, Control and Dynamics*, Vol. 24, No. 2, Mar-Apr. 2001, pp.270-277.

[16] Rea, J. R., *A Legendre Pseudospectral Method for Rapid Optimization of Launch Vehicle Trajectories*. S.M. Thesis, Department of Aeronautics and Astronautics, MIT, June 2001.

[17] Ross, I. M. and Fahroo, F., "A Direct Method for Solving Nonsmooth Optimal Control Problems," to appear in the Proceedings of the 15th IFAC World Congress, July 2002.

[18] Ross, I. M. and Fahroo, F., "A Pseudospectral Transformation of the Covectors of Optimal Control Systems," Proceedings of the First IFAC Symposium on System Structure and Control, August 2001.

[19] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H., "User's Guide for NPSOL 5.0: A FORTRAN Package for Nonlinear Programming," Tech. rep., Stanford Business Software, Inc., Palo Alto, California, July 1998.

[20] Gill, P. E., Murray, W., and Saunders, M. A., "User's Guide for SNOPT 5.3: A FORTRAN Package for Large-Scale Nonlinear Programming," Tech. rep., December 1998.

3231-14