OPTIMAL TRAJECTORIES FOR AUTONOMOUS THERMAL SOARING

TRAJECTOIRES OPTIMALES EN VOL PLANÉ AUTONOME UTILISANT DES ASCENDANTS THERMIQUES

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Thomas R. Connerty, B.Eng., CD, QFTE
Major

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Aeronautical Engineering

September, 2014

For my family, as none of this would have been possible without their on-going guidance, love, and support.

Acknowledgments

I would like to thank my supervisor, Dr. Ruben E. Perez for imparting a great deal of knowledge on aircraft conceptual design and the advanced field of optimization while continually raising the bar to ensure the highest results were achieved. Additionally, Dr. Mark LaViolette provided sound and extremely helpful advice on all aspects of this research.

Significant acknowledgment must be made to Stephen Andrews and Peter Jansen for their extreme patience and a continual willingness to assist with the advanced aircraft design tools and optimization framework.

Finally, I would like to extend my sincere appreciation to the Dean of Graduate Studies and Research of the Royal Military College of Canada for sponsoring my Master's program. I was provided with an excellent and rewarding opportunity to further my educational career, and for that I am eternally grateful.

Abstract

Connerty, Thomas Robert. M.A.Sc. Royal Military College of Canada, September 2014. *Optimal Trajectories for Autonomous Thermal Soaring*. Supervised by Ruben E. Perez, B.Eng., M.A.Sc., Ph.D., P.Eng., Assistant Professor.

Autonomous thermal soaring aircraft exploit the use of naturally occurring updrafts to increase their overall energy state. Thermal soaring strategies for these types of aircraft have been developed and refined to increase aircraft range and/or endurance performance through pilot flying experience and an increased knowledge of gliding flight theory. Current autopilot development for thermal centering control makes use of analytical expressions or accepted soaring techniques for their implementation. The use of trajectory optimization to analyze thermal centering strategies has been very limited given the numerical difficulties encountered when analyzing the problem in a Cartesian coordinate system. In this study, the generalized aircraft equations of motion are restructured in cylindrical coordinates, to reduce numerical computational issues arising during optimization, and are augmented with wind component forces. To generate optimal trajectories for autonomous thermal soaring, the optimal control problem is defined as maximizing either the aircraft's instantaneous or final specific energy state and the trajectories are solved using a direct collocation method using nonlinear programming. Optimal flight trajectories are presented for a small autonomous aircraft operating in multiple thermal profiles of varying strengths. These trajectories are compared to the derived analytic expressions and a thermal centering strategy for controller development is presented. As an application, optimal flight trajectories are presented that maximize thermal energy extraction whilst providing persistent aerial surveillance coverage for a variety of thermal and target locations.

Keywords: Aerial Surveillance, Autonomous Soaring, Direct Collocation, Feedback Control, Optimal Trajectories, Target Acquisition, Thermals, Thermal Centering, Thermal Soaring

Résumé

Connerty, Thomas Robert. M.A.Sc. Collège militaire royal du Canada, Septembre 2014. *Trajectoires optimales en vol plané autonome utilisant des ascendants thermiques*. Thèse dirigée par Ruben E. Perez, B.Eng., M.A.Sc., Ph.D., P.Eng., Professeur adjoint.

Le vol à voile autonome utilise le vol thermique pour accroître leur énergie. Des stratégies de vol thermique ont été développées et améliorées pour accroître la distance franchissable et/ou le temps de vol en utilisant l'expérience des pilotes ou la mécanique du vol plané. La commande de positionnement sur l'ascendant thermique des pilotes automatiques actuelles est basée sur des expressions analytiques ou des techniques de vol connues. L'emploie d'optimisateur de trajectoire pour déterminer la commande de positionnement est prescrit dû aux difficultés numériques lors de l'analyse en coordonnées cartésiens. Pour cette étude, les équations générales du mouvement sont réécrites en coordonnées cylindriques, afin de réduire les difficultés numériques durant l'optimisation et sont accompagnées de composantes de la force du vent. La commande optimale par méthode de point intérieure non linéaire est utilisée pour générer les trajectoires optimales en vol thermique et est définie comme le maxima de l'énergie instantanée ou de l'énergie finale. Les trajectoires optimales sont présentées pour de petit aéronef autonome volant dans de multiple profiles d'ascendant thermique de force différente. Ces trajectoires sont comparées avec les expressions analytiques et stratégies de positionnement. Comme exemple d'application, les trajectoires optimales maximisant l'apport d'énergie de la thermique à l'avion en maintenant une surveillance aérienne d'un territoire pour différents ascendants thermiques et cibles terrestres.

Mots clés: Surveillance aérienne, vol plané autonome, méthode de point intérieure, asservissement, trajectoires optimales, ascendants thermiques, utilisant des ascendants thermiques

Contents

Acknowledgments **iii**

Abstract **iv**

Résumé **v**

List of Tables **ix**

List of Figures **x**

Nomenclature **xiii**

1 Introduction **1**
1.1 Motivation for the Research 1
1.2 Research Objectives 2
1.3 Thesis Layout 3
1.4 Contributions 3

2 Literature Review **4**
2.1 Thermal Mapping 4
2.2 Thermal Centering and Control 5
2.3 Path Planning 11
2.4 Target Acquisition and Surveillance 13
2.5 Knowledge Gaps in the Research 14

3 Aircraft Characterization **15**
3.1 Reference Axis Systems 15
3.2 Point-Mass Model Equations of Motion 15
3.2.1 Equations of Motion with Wind Component Forces 20
3.3 General Aircraft Description 21
3.4 Characterization of the Aircraft Aerodynamics 22

3.4.1 Aircraft Geometric Model . . . 22
3.4.2 Aircraft Aerodynamic Data . . . 22
3.5 Aircraft Gliding Flight Characteristics . . . 26
3.5.1 Minimum Glide Angle . . . 26
3.5.2 Minimum Sink Rate . . . 27
3.5.3 Aircraft Turning Radius . . . 27
3.5.4 Gliding Flight Velocities . . . 29

4 Thermal Models **31**
4.1 Conceptual Thermal Models . . . 31
4.2 Mathematical Thermal Models . . . 32
4.3 Chimney Thermal Model . . . 38
4.4 Bubble Thermal Model . . . 41

5 Optimal Control Problem **46**
5.1 Generalized Optimal Control Problem . . . 46
5.2 Method to Solve the Optimal Control Problem . . . 47
5.3 Optimal Control Problem Example . . . 48
5.4 Objective Function Development . . . 52
5.5 System Dynamics . . . 53
5.5.1 Kinematic Equations of Motion . . . 53
5.5.2 Equations of Motion Restructured in Cylindrical Coordinates 54
Change in Aircraft Radial Position with Time . . . 54
Change in Aircraft Angular Position with Time . . . 56
Change in Aircraft Tangential Tracking Angle with Time . . . 57
5.6 Optimization Problem Definition . . . 58

6 Thermal Soaring Case Studies **60**
6.1 Case Studies Definition . . . 60
6.2 Optimal Trajectories for the Chimney Thermal Model . . . 61
6.2.1 Effect of Thermal Strength and Initial Starting Position . . . 61
6.2.2 Effect of Variations in the Objective Function . . . 70
6.2.3 Effect of Aircraft Tangential Tracking Angle Range . . . 74
6.2.4 Energy Analysis of the Optimal Trajectories . . . 74
6.3 Optimal Trajectories for the Bubble Thermal Model . . . 82
6.3.1 Effect of Thermal Strength on the Trajectories . . . 82
6.3.2 Effect of Initial Starting Location on the Trajectories . . . 82
6.3.3 Energy Analysis of the Optimal Trajectories . . . 87
6.4 Optimal Control Strategy for Thermal Soaring . . . 92

7 Target Localization **97**
7.1 Camera Optics 97
7.2 Camera Reference Axis 100
7.3 Target Localization Optimization Case Studies 102
7.3.1 Problem Formulation 102
7.3.2 Optimal Trajectory Results 106

8 Conclusions and Recommendations **115**
8.1 Conclusions 115
8.2 Recommendations for Future Work 117

Bibliography **118**

Appendices **123**

A Characterization of the Aircraft Flight Dynamics **124**
A.1 Rigid Body Equations of Motion 124
A.2 Dimensionless Stability and Control Coefficients 124
A.3 Dimensional Stability and Control Derivatives 126
A.4 Longitudinal Flight Dynamics 129
A.5 Lateral-Directional Flight Dynamics 130

B Computer Source Code **151**
B.1 Thermal Models 151
B.2 Trajectory Optimization 166

List of Tables

3.1 Performance Equations of Motion Variables 19
3.2 Aircraft Component List and Weights 21
3.3 Aerodynamic Data for the Point-Mass Model of the Cularis D-5223 Glider 24
3.4 Pertinent Glide Velocities for Various Altitudes 30

4.1 Seasonal Variations in Updraft Strength and Altitude 39
4.2 Shape Constants for a Bell-Shaped Vertical Velocity Distribution Developed Experimentally for use in SI System of Units 40

5.1 Definition of the Path Constraints 59
5.2 Definition of the Boundary Conditions for the States and Controls . . . 59

6.1 Definition of the Thermal Strength Cases 61
6.2 Initial Aircraft Flight Conditions 61
6.3 Initial Aircraft Starting Location 65
6.4 Optimization Efficiency Results - Varying Thermal Strengths 71
6.5 Optimization Efficiency Results - Varying Thermal Strengths 86

7.1 Definition of the Additional Path Constraints and Boundary Conditions 106
7.2 Target Locations 106
7.3 Optimization Efficiency Results - Varying Target Locations 114

A.1 Linearized Small-Disturbance Longitudinal Rigid Body State Equations 125
A.2 Linearized Small-Disturbance Lateral-Directional Rigid Body State Equations 125
A.3 Dimensionless State Coefficients 127
A.4 Dimensionless Control Coefficients 128
A.5 Dimensionless State Rate Coefficients 128
A.6 Summary of Longitudinal Derivatives 128
A.7 Summary of Lateral-Directional Derivatives 129

List of Figures

2.1 Reichmann Centering Methods 7
2.2 270° Centering Correction Method 8

3.1 Aircraft Body Axis Orientation 16
3.2 Angular Relationships Between Earth, Body, and Velocity Axes (subscripts e, b, and v respectively) 17
3.3 Free Body Diagram Outlining the Forces Acting on the Aircraft 19
3.4 Multiplex Cularis D-5223 Glider 21
3.5 Geometric Representation of the Cularis D-5223 Glider 23
3.6 HQ 1.0/12 Airfoil 24
3.7 Drag Polar for the Cularis D-5223 Glider at an Altitude of $h = 1000$ ft . 25
3.8 Vertical Velocity as a Function of Airspeed for the Cularis D-5223 Glider at an Altitude of $h = 1000$ ft 28
3.9 An Aircraft in Turning Flight 29

4.1 A Column or Chimney Thermal Model 32
4.2 Typical Life Cycle of a Thermal with Cumulus Cloud 32
4.3 Bubble Model of a Thermal 33
4.4 Graphical Representation of the 3D Trapezoidal Thermal Model 34
4.5 3D Trapezoidal Thermal Model 35
4.6 Gaussian Thermal Model 36
4.7 Gedeon Thermal Model 37
4.8 Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s for the Chimney Thermal Model 41
4.9 Thermal Updraft Speed Profiles at Multiple Altitude Slices for Chimney Thermal Model 42
4.10 Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s for the Lawrance Bubble Model 43

4.11 Thermal Updraft Speed Profiles at Multiple Altitude Slices for Lawrance Bubble Model 44

5.1 Plot of the States for the Analytic Solution to the Breakwell Problem . . 50
5.2 Plot of the States for the Numerical Solution to the Breakwell Problem . 50
5.3 Plot of the Controls for the Analytic Solution to the Breakwell Problem 51
5.4 Plot of the Controls for the Numerical Solution to the Breakwell Problem 51
5.5 Cylindrical Coordinates Definition 54
5.6 Aircraft Tangential Tracking Angle Definition 57

6.1 Optimal Trajectories for Varying Thermal Strengths Defined by the Month Index 62
6.2 Time History of the States for Varying Thermal Strengths 63
6.3 Time History of the Control Inputs for Varying Thermal Strengths . . . 64
6.4 Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 0, January] 66
6.5 Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 3, April] 67
6.6 Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 6, July] 68
6.7 Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 9, October] 69
6.8 Optimal Trajectories for Varying Initial Starting Locations 72
6.9 Effect of Variations in the Objective Function 73
6.10 Effect of Aircraft Tangential Tracking Angle Range on the Trajectory . 75
6.11 Time History of the Effect of Tangential Tracking Angle Range 76
6.12 Time History Energy Analysis for Varying Thermal Strengths 77
6.13 Time History Analysis of Vertical Speed Rates and Acceleration Changes 79
6.14 Time History Analysis of Airspeed Rates and Acceleration Changes . . 80
6.15 Time History Analysis of Thermal Strength and Acceleration 81
6.16 Optimal Trajectories for Varying Thermal Strengths - Bubble Thermal . 83
6.17 Time History of the States for Varying Thermal Strengths - Bubble Thermal 84
6.18 Time History of the Control Inputs for Varying Thermal Strengths - Bubble Thermal 85
6.19 Optimal Trajectories for Varying Initial Starting Locations - Bubble Thermal 88
6.20 Time History Energy Analysis for Varying Thermal Strengths - Bubble Thermal 89
6.21 Plots of Vertical Speed Rates and Acceleration Changes - Bubble Thermal 90
6.22 Plots of Airspeed Rates and Acceleration Changes - Bubble Thermal . . 91

6.23 Angle of Bank versus C_L for a given Radial Distance 95

7.1 Digital Spatial Resolution 98
7.2 Viewing Angle Effect on the Digital Spatial Resolution 98
7.3 Camera Axis System 100
7.4 Field of View Diagrams for the Cularis D-5223 Aircraft 103
7.5 Optimal Trajectory for See-Ability Formulation 105
7.6 Optimal Trajectory for Target Localization - Case 1 107
7.7 Optimal Trajectory for Target Localization - Case 2 108
7.8 Optimal Trajectory for Target Localization - Case 3 109
7.9 Optimal Trajectory for Target Localization - Case 4 110
7.10 Two Dimensional Trajectories for Target Localization 112
7.11 Time History of Airspeed and Radial Distance for Target Localization . 113

A.1 Forces and Moments Acting on an Aircraft in the Body Axes 125

Nomenclature

Roman Symbols

$\bar{c}_w$	Mean Aerodynamic Chord of the Wing	[ft]
$\bar{w}$	Average Updraft Velocity	$[\frac{\text{ft}}{\text{s}}]$
$\dddot{E}_h$	Triple Time Derivative of Energy Height	$[\frac{\text{ft}}{\text{s}^3}]$
$\ddot{E}_h$	Double Time Derivative of Energy Height	$[\frac{\text{ft}}{\text{s}^2}]$
$\dot{E}_h$	Time Derivative of Energy Height	$[\frac{\text{ft}}{\text{s}}]$
$\dot{m}$	Rate of Change of Aircraft Mass	$[\frac{\text{slug}}{\text{s}}]$
$\dot{r}$	Rate of Change of Aircraft Radial Position in a Thermal	$[\frac{\text{ft}}{\text{s}}]$
$\dot{V}$	Rate of Change of Aircraft True Airspeed	$[\frac{\text{ft}}{\text{s}^2}]$
$\dot{V}_{w,x}$	X Component of the Wind Acceleration	$[\frac{\text{ft}}{\text{s}^2}]$
$\dot{V}_{w,y}$	Y Component of the Wind Acceleration	$[\frac{\text{ft}}{\text{s}^2}]$
$\dot{V}_{w,z}$	Z Component of the Wind Acceleration	$[\frac{\text{ft}}{\text{s}^2}]$
$\left(\frac{L}{D}\right)_{max}$	Maximum Lift-to-Drag Ratio	$[-]$
$\boldsymbol{d}_{los}$	Camera Line of Sight Vector	[ft]
$\boldsymbol{F}_a$	Aerodynamic Force	$[\text{lb}_\text{f}]$
$\boldsymbol{F}_g$	Gravitational Force	$[\text{lb}_\text{f}]$
$\boldsymbol{F}_I$	Inertial Force	$[\text{lb}_\text{f}]$
$\boldsymbol{F}_p$	Propulsive Force	$[\text{lb}_\text{f}]$

$\boldsymbol{F}_w$	Wind Force	$[\mathrm{lb_f}]$
b	Wing Span	[ft]
D	Drag of the Aircraft	$[\mathrm{lb_f}]$
d	Camera Charge-Coupled Device Detector Size	[ft]
D_m	Momentum Drag	$[\mathrm{lb_f}]$
E	Aircraft Energy State	$[\mathrm{lb_f\,ft}]$
e	Oswald's Efficiency Factor	$[-]$
E_h	Energy Height	[ft]
f	Camera Focal Length	[ft]
$F_{w,x}$	X Component of the Wind Force	$[\mathrm{lb_f}]$
$F_{w,y}$	Y Component of the Wind Force	$[\mathrm{lb_f}]$
$F_{w,z}$	Z Component of the Wind Force	$[\mathrm{lb_f}]$
G	Pixel Size	[ft]
g	Acceleration due to Gravity	$[\frac{\mathrm{ft}}{\mathrm{s}^2}]$
h	Aircraft Altitude or Height Above the Ground	[ft]
h_i	Thermal Convective Mixing-Layer Thickness	[ft]
I_{xx}	Mass Moment of Inertia about the Aircraft X-axis	$[\mathrm{slug\,ft^2}]$
I_{xy}	Product of Inertia about the Aircraft XY-Plane	$[\mathrm{slug\,ft^2}]$
I_{xz}	Product of Inertia about the Aircraft XZ-Plane	$[\mathrm{slug\,ft^2}]$
I_{yx}	Product of Inertia about the Aircraft YX-Plane	$[\mathrm{slug\,ft^2}]$
I_{yy}	Mass Moment of Inertia about the Aircraft Y-axis	$[\mathrm{slug\,ft^2}]$
I_{yz}	Product of Inertia about the Aircraft YZ-Plane	$[\mathrm{slug\,ft^2}]$
I_{zx}	Product of Inertia about the Aircraft ZX-Plane	$[\mathrm{slug\,ft^2}]$
I_{zy}	Product of Inertia about the Aircraft ZY-Plane	$[\mathrm{slug\,ft^2}]$
I_{zz}	Mass Moment of Inertia about the Aircraft Z-axis	$[\mathrm{slug\,ft^2}]$

J	Performance Index	$[-]$
K	Induced Drag Coefficient	$[-]$
k_1	Thermal Model Shape Constant	$[-]$
k_2	Thermal Model Shape Constant	$[-]$
k_3	Thermal Model Shape Constant	$[-]$
k_4	Thermal Model Shape Constant	$[-]$
L	Aircraft Lift	$[\mathrm{lb_f}]$
L_β	Lateral Stability (Dihedral Effect)	$[\frac{1}{\mathrm{s}^2}]$
L_p	Roll Damping	$[\frac{1}{\mathrm{s}}]$
L_r	Effect of Yaw Rate on Rolling Moment	$[\frac{1}{\mathrm{s}}]$
L_{δ_a}	Roll or Aileron Power	$[\frac{1}{\mathrm{s}^2}]$
L_{δ_r}	Effect of Rudder Deflection on Rolling Moment	$[\frac{1}{\mathrm{s}^2}]$
M	Mach Number	$[-]$
m	Aircraft Mass	[slug]
M_q	Pitch Damping	$[\frac{1}{\mathrm{ft\,s}}]$
M_u	Speed Stability Effect	$[\frac{1}{\mathrm{ft\,s}}]$
M_w	Angle of Attack Stability	$[\frac{1}{\mathrm{ft\,s}}]$
$M_{\dot{w}}$	Downwash Lag	$[\frac{1}{\mathrm{ft}}]$
M_{δ_e}	Elevator Power	$[\frac{1}{\mathrm{s}^2}]$
N_β	Directional Stability (Weathercock Effect)	$[\frac{1}{\mathrm{s}^2}]$
N_p	Effect of Roll Rate on Yawing Moment	$[\frac{1}{\mathrm{s}}]$
N_r	Yaw Damping	$[\frac{1}{\mathrm{s}}]$
N_{δ_a}	Effect of Aileron Deflection on Yawing Moment	$[\frac{1}{\mathrm{s}^2}]$
N_{δ_r}	Rudder Power	$[\frac{1}{\mathrm{s}^2}]$
Q	Dynamic Pressure	$[\frac{\mathrm{lb_f}}{\mathrm{ft}^2}]$

r	Aircraft Radial Position in a Thermal	[ft]
r_1	Thermal Inner Radius	[ft]
r_2	Thermal Outer Radius	[ft]
r_d	Desired Aircraft Radial Distance to Thermal Core	[ft]
S	Wing Reference Area	[ft^2]
S_a	See-ability	[−]
S_{res}	Digital Spatial Resolution of an Image	[−]
T	Aircraft Thrust	[lb_f]
t_0	Initial Time	[s]
t_f	Final Time	[s]
V	Aircraft True Airspeed	[$\frac{\text{ft}}{\text{s}}$]
W	Aircraft Take-off Weight	[lb_f]
w^*	Thermal Convective Velocity Scale	[$\frac{\text{ft}}{\text{s}}$]
w_d	Downdraft Velocity	[$\frac{\text{ft}}{\text{s}}$]
W_h	Thermal Updraft Speed	[$\frac{\text{ft}}{\text{s}}$]
w_l	Intermediate Downdraft Velocity Variable	[$\frac{\text{ft}}{\text{s}}$]
W_x	X Component of the Thermal Updraft Speed	[$\frac{\text{ft}}{\text{s}}$]
W_y	Y Component of the Thermal Updraft Speed	[$\frac{\text{ft}}{\text{s}}$]
w_{peak}	Maximum Vertical Velocity	[$\frac{\text{ft}}{\text{s}}$]
X_u	Effect of Forward Speed on Drag	[$\frac{1}{\text{s}}$]
X_w	Effect of Vertical Speed on Drag	[$\frac{1}{\text{s}}$]
x_{cam}	Camera x-Position on the Aircraft from the Center of Gravity	[ft]
x_{los}	x-Position of the Camera Line of Sight Vector on Earth's Surface	[ft]
x_{tgt}	x-Position on the Target in the Earth Axis System	[ft]
X_{δ_e}	Effect of Elevator Deflection on Drag	[$\frac{\text{ft}}{\text{s}^2}$]

Y	Side Force	$[\mathrm{lb_f}]$
Y_β	Effect of Sideslip on Side Force	$[\frac{\mathrm{ft}}{\mathrm{s}^2}]$
Y_p	Effect of Roll Rate on Side Force	$[\frac{\mathrm{ft}}{\mathrm{s}}]$
y_{cam}	Camera y-Position on the Aircraft from the Center of Gravity	[ft]
y_{los}	y-Position of the Camera Line of Sight Vector on Earth's Surface	[ft]
y_{tgt}	y-Position on the Target in the Earth Axis System	[ft]
Y_{δ_r}	Effect of Rudder Deflection on Side Force	$[\frac{\mathrm{ft}}{\mathrm{s}^2}]$
z	Height in Earth Axis System	[ft]
Z_u	Effect of Forward Speed on Lift	$[\frac{1}{\mathrm{s}}]$
Z_w	Effect of Vertical Speed on Lift	$[\frac{1}{\mathrm{s}}]$
z_{cam}	Camera z-Position on the Aircraft from the Center of Gravity	[ft]
z_{tgt}	z-Position on the Target in the Earth Axis System	[ft]
Z_{δ_e}	Effect of Elevator Deflection on Lift	$[\frac{\mathrm{ft}}{\mathrm{s}^2}]$

Greek Symbols

α	Angle of Attack	[rad]
β	Sideslip Angle	[rad]
δ_a	Aileron Deflection	[rad]
δ_e	Elevator Deflection	[rad]
δ_r	Rudder Deflection	[rad]
$\dot{\beta}$	Rate of Change of Sideslip Angle	$[\frac{\mathrm{rad}}{\mathrm{s}}]$
$\dot{\varepsilon}$	Rate of Change of Aircraft Tangential Tracking Angle	$[\frac{\mathrm{rad}}{\mathrm{s}}]$
$\dot{\gamma}_1$	Roll Rate	$[\frac{\mathrm{rad}}{\mathrm{s}}]$
$\dot{\gamma}_2$	Climb Rate	$[\frac{\mathrm{rad}}{\mathrm{s}}]$
$\dot{\gamma}_3$	Turn Rate	$[\frac{\mathrm{rad}}{\mathrm{s}}]$

$\dot{\phi}$	Roll Rate	$[\frac{\text{rad}}{\text{s}}]$
$\dot{\psi}$	Turn Rate	$[\frac{\text{rad}}{\text{s}}]$
$\dot{\theta_{th}}$	Rate of Change of Aircraft Angular Position in a Thermal	$[\frac{\text{rad}}{\text{s}}]$
$\dot{\theta}$	Pitch Rate	$[\frac{\text{rad}}{\text{s}}]$
ε	Aircraft Tangential Tracking Angle	[rad]
γ_1	Bank Angle	[rad]
γ_2	Climb Angle	[rad]
γ_3	Track Angle	[rad]
κ	Camera Azimuth Rotation Angle	[rad]
λ	Eigenvalue	$[\frac{1}{\text{s}}]$
ζ_k	Defect Constraints	$[-]$
ω_n	Undamped Natural Frequency	$[\frac{\text{rad}}{\text{s}}]$
ϕ	Roll Angle	[rad]
ψ	Aircraft Heading	[rad]
ρ	Air Density	$[\frac{\text{slug}}{\text{ft}^3}]$
τ	Camera Elevation Rotation Angle	[rad]
τ_1	Propulsion Vectoring	[rad]
θ	Pitch Angle	[rad]
θ_v	Camera Viewing Angle	[rad]
θ_{opt}	Optimal Camera Viewing Angle	[rad]
θ_{th}	Aircraft Angular Position in a Thermal	[rad]
ζ	Damping Ratio	$[-]$

Dimensionless Groups

$(C_L)_{max}$	Maximum Lift Coefficient

C_D	Drag Coefficient
C_L	Lift Coefficient
C_{D0}	Parasitic Drag Coefficient
$C_{D\alpha}$	Variation of Drag Coefficient with Angle of Attack
C_{Dq}	Variation of Drag Coefficient with Pitch Rate
C_{Du}	Variation of Drag Coefficient with Speed
$C_{D\delta e}$	Variation of Drag Coefficient with Elevator Deflection
$C_{D\dot{\alpha}}$	Variation of Drag Coefficient with Angle of Attack Rate
C_{L0}	Lift Coefficient at α_0
$C_{l\beta}$	Variation of Rolling Moment Coefficient with Sideslip Angle
C_{lp}	Variation of Rolling Moment Coefficient with Roll Rate
C_{Lq}	Variation of Lift Coefficient with Pitch Rate
C_{lr}	Variation of Rolling Moment Coefficient with Yaw Rate
C_{Lu}	Variation of Lift Coefficient with Speed
$C_{l\delta a}$	Variation of Rolling Moment Coefficient with Aileron Deflection
$C_{L\delta e}$	Variation of Lift Coefficient with Elevator Deflection
$C_{l\delta r}$	Variation of Rolling Moment Coefficient with Rudder Deflection
$C_{L\dot{\alpha}}$	Variation of Lift Coefficient with Angle of Attack Rate
$C_{m\alpha}$	Variation of Pitching Moment Coefficient with Angle of Attack
C_{mq}	Variation of Pitching Moment Coefficient with Pitch Rate
C_{mu}	Variation of Pitching Moment Coefficient with Speed
$C_{m\delta e}$	Variation of Pitching Moment Coefficient with Elevator Deflection
$C_{m\dot{\alpha}}$	Variation of Pitching Moment Coefficient with Angle of Attack Rate
$C_{n\beta}$	Variation of Yawing Moment Coefficient with Sideslip Angle
C_{np}	Variation of Yawing Moment Coefficient with Roll Rate

C_{nr}	Variation of Yawing Moment Coefficient with Yaw Rate
$C_{n\delta a}$	Variation of Yawing Moment Coefficient with Aileron Deflection
$C_{n\delta r}$	Variation of Yawing Moment Coefficient with Rudder Deflection
$C_{y\beta}$	Variation of Side Force Coefficient with Sideslip Angle
C_{y_p}	Variation of Side Force Coefficient with Roll Rate
C_{y_r}	Variation of Side Force Coefficient with Yaw Rate
$C_{y_{\delta a}}$	Variation of Side Force Coefficient with Aileron Deflection
$C_{y_{\delta r}}$	Variation of Side Force Coefficient with Rudder Deflection

Roman Subscripts

$\dot{p}$	Rate of Change of Roll Rate
$\dot{q}$	Rate of Change of Pitch Rate
$\dot{r}$	Rate of Change of Yaw Rate
$\dot{u}$	Rate of Change of Linear Velocity Component
$\dot{w}$	Rate of Change of Vertical Velocity Component
DR	Dutch-Roll
e	Earth Axis
l	Rolling Moment
lp	Long Period (Phugoid)
m	Pitching Moment
n	Yawing Moment
p	Roll Rate
q	Pitch Rate
r	Yaw Rate
sp	Short Period
u	Linear Velocity Component

v	Velocity Axis
w	Vertical Velocity Component
y	Side Force

1 Introduction

1.1 Motivation for the Research

In the modern battle space, the application of an Intelligence, Surveillance, Target Acquisition, and Reconnaissance (ISTAR) architecture is imperative to link multiple battlefield functions together with the aim of assisting a combat group in the application of force. Intelligence information is collected through systematic observation of enemy territories, dispositions, targets, and intents. The information is analyzed by intelligence personnel and is used by the Commander to formulate battle plans. A variety of platforms and sensors from the Air Force, Army and Navy elements are available to collect this information, and it is imperative that these assets provide full coverage of the enemy for extended periods. One of the platforms used to gather intelligence information are Small Unmanned Aerial Vehicles (SUAVs), which are defined as vehicles that can operate up to 600 m (2,000 ft) above ground level (AGL) and have an operating range of 5,000 m (16,400 ft). These air vehicles are rapidly deployed by ground troops to gather timely information of the enemy for rapid decision making. Given their relatively small payload capacity, SUAVs are highly constrained in the types of sensors they can employ and the amount of fuel or energy they can store resulting in a shortened flight endurance.

Improved flight endurance for SUAVs can be achieved through improvements in aerodynamic and propulsive efficiency, and material advancements. However, it is equally important to examine the operational flight envelope to develop manoeuvres and strategies to enhance the performance of the vehicle. Birds have evolved to extract energy from the atmosphere whilst exerting very little of their own energy. As an example, an albatross has developed a highly efficient soaring strategy that enables them to exploit weather systems to perform transoceanic flights without flapping [1].

Extracting energy from the atmosphere is not a new concept and has attracted the attention of researchers, aircraft designers and glider pilots. A number of soaring techniques has been developed to extract the abundant energy that is readily available in the atmosphere while at the same time minimizing the inherent sinking properties of a gliding vehicle. Energy extraction can be achieved through a number of soaring

techniques including thermal, gust, dynamic or wave soaring [2–4]. The application of some of these techniques is limited based on the available topology or climate of the operating environment. The mechanics of thermal soaring in that rising bubbles or columns of air are formed on the ground through warming of the Earth's surface from solar heating, make it an attractive soaring technique and it is quite often employed by birds of prey [5].

For effective thermal soaring, several successful phases are required to ensure efficient energy extraction. Initially, a method of detecting thermal activity is critical and as such a sound understanding of weather that is conducive to thermal soaring is required. Once thermal detection has occurred, the craft must enter and center on the thermal core. Once centered, an optimal climb rate is achieved by commanding an aircraft bank angle and airspeed appropriate for a given thermal strength and capturing altitude. The final phase is defined as the thermal departure. At this point, an aerodynamically efficient flight condition is either adopted or the potential energy gained by the thermal climb can used to extend the range or accomplish other mission tasks. Each distinct phase of the thermal soaring problem forms the basis of the various research areas in this field, which can be defined as thermal mapping, thermal centering strategies, controller development, and path planning.

1.2 Research Objectives

This research will focus primarily on thermal centering strategies and achieving energy efficient climb rates for a thermal that has been detected with a known size and strength. The development of these strategies will be approached numerically using trajectory optimization as the framework while employing an energy approach to analyze aircraft performance and to define the objective function. More specifically, this thesis will address the following research objectives:

i. Determine the optimal flight trajectories for autonomous thermal soaring aircraft for a thermal of known size and strength using numerical methods.
ii. Compare the optimal trajectories with the predictive analytic approaches.
iii. Develop a strategy for thermal centering control using standard flight control surfaces and using sensor information that is available on gliding aircraft.
iv. Investigate the change in the optimal flight trajectories when the aircraft is provided with a concurrent objective of providing persistent aerial surveillance coverage at a predefined minimum image resolution.

1.3 Thesis Layout

The remainder of this thesis dissertation is organized into seven chapters. Chapter 2 presents the literature review with a focus on the current state-of-the-art of the research completed in the autonomous soaring field. A detailed investigation of the performance, and gliding flight characteristics of a representative SUAV used in this research is performed in Chapter 3, including the development of the analysis tools used to generate the point-mass aircraft model used in trajectory optimization. Current mathematical models of the various thermal types, and the state-of-the-art thermal models that have been developed and will be used in this study are detailed in Chapter 4. Chapter 5 introduces the optimal control problem and the application of the direct collocation method using nonlinear programming to solve the optimal trajectories. Furthermore, the generalized aircraft equations of motion are restructured in cylindrical coordinates, and in Chapter 6 optimal flight trajectories are solved to maximize thermal energy extraction and are compared to derived analytic predictions. As an application, trajectories that are concurrently optimized for energy extraction and providing persistent aerial surveillance are presented in Chapter 7. Chapter 8 provides concluding arguments and recommendations for future areas of research on this topic. Given that the standard in aviation is to represent units of measure in the Imperial Measurement System, all units will be presented herein using this same standard.

1.4 Contributions

- The performance equations of motion are restructured in cylindrical coordinates and are augmented with wind component forces to provide a framework to numerically solve the optimal soaring problem.
- Optimal flight trajectories that maximize thermal energy extraction are presented for a small autonomous aircraft operating in multiple thermal profiles of varying strengths.
- Analytical expressions are presented and their predictions are compared to the optimal flight trajectories.
- A thermal centering strategy for controller development is presented based on an energy approach and the optimal soaring trajectories.
- As an application, optimal flight trajectories are presented that maximize thermal energy extraction whilst providing persistent aerial surveillance coverage for a variety of thermal and target locations.

2 Literature Review

Autonomous soaring for an SUAV is a challenging endeavour given its limited payload capacity and on-board decision making capabilities. The overall challenge is to locate a thermal sufficiently strong to enable energy extraction, while at the same time position the SUAV to use the potential energy gained to accomplish mission tasks. In this chapter, an investigation of the current research that has been conducted for each phase of the thermal soaring problem is presented. As previously identified, the various research areas in this field were defined as thermal mapping, thermal centering strategies, controller development, and path planning.

2.1 Thermal Mapping

The first major obstacle to soaring flight is locating a thermal within a defined operating environment. Thermal detection can either be accomplished by using a single aircraft or using a cooperative team working together to search, detect and map a particular area. Cheng and Langelaan presented an approach to use a group of aircraft to explore regions that were associated with both a high likelihood of thermal activity and areas with an elevated level of uncertainty in vertical wind estimates [6]. The operating environment was discretized into a grid and a Kalman filter was used to estimate vertical wind speed in each cell. An exploration priority function was computed for each of these cells, that was calculated primarily based on a relationship between the operating topography, vertical wind covariance and the solar incidence angle. The search algorithm was guided by this exploration priority and simulations were performed to investigate the trade off between group size and mapping altitude. Even though this approach was effective in mapping thermal activity in a defined area, it required *a priori* knowledge of the topography and was computationally intensive for the pre-flight planning of the mission. In the case of target reassignment or rapid redeployment of the SUAV to another location outside the currently mapped area, this technique would be impractical to employ operationally.

An alternative approach to thermal mapping is to use a single aircraft for simul-

taneous mapping and utilizing a detected wind field for soaring as was proposed by Lawrance and Sukkarieh [7]. In this case, an overarching global planner was employed to facilitate full wind field velocity mapping of the operating region by defining global target locations. Additionally, a low-level planning algorithm used an energy-based heuristic to identify control actions that maximized local energy capture while concurrently reducing local area map uncertainty and achieving the global target. Wind estimation models were generated using a Gaussian process regression technique from the wind field data that was measured as point observations throughout the flight. The approach was tested in simulation with wind fields consisting of single and multiple stationary thermal bubbles and it was reported that a single aircraft could effectively explore and exploit an unknown wind field. The main advantage of the Gaussian process regression technique was that it provided a variance estimate at each test point. The variance estimate allowed the path planner to identify regions of low or noisy information in the field, which could be targeted for further exploration. However, this comes at the cost of high computational and storage requirements, which may be significant depending on the application.

Alternatively, Andersson *et al.* used an energy approach as the steering criteria for aircraft guidance and control to investigate the possible benefits of using a cooperating team of small UAVs to increase the probability of finding thermal lift [8]. More specifically, an algorithm was developed to define the switching logic between soaring and search modes. The main switching criteria was based on a minimum threshold value of the rate of change of energy height, $\dot{E}_h$ encountered by the aircraft. The cooperative portion of the algorithm consisted of guiding vehicles that were operating in the search mode toward the estimated location of a thermal center where another vehicle was soaring. The trajectories were corrected for the drift of the thermal and taking into consideration collision avoidance. The algorithm and collision avoidance system were operated from a ground control station, which greatly increased the support requirements to implement this strategy. Additionally, $\dot{E}_h$ was obtained by differentiating the signal from the pitot static system and required multiple filters to avoid significant latency errors in the controller. Using a minimum threshold value of the rate of change of energy height as a switching logic was a sound approach, but performance would be improved if it could be measured directly.

2.2 Thermal Centering and Control

Once the location of a thermal updraft has been detected, whether through a detailed mapping of the environment provided by a mission planning architecture or by an on-board sensing system, the next challenge is to quickly center on and circle the

thermal core. Cross-wind components can drift the thermal core from the detected or predicted location. As such, continual flight control adjustments are required to achieve optimal climb rates and remain centered on the core. For decades, glider pilots have developed and continue to refine heuristic techniques to center thermals. Reichmann outlined the following three centering methods, which are shown in Figure 2.1 and that were based on changes in rate of climb vice the absolute climb rate [4]:

i. **Method 1** - When lift is encountered, level out the aircraft and fly straight for a short duration before starting to circle again,
ii. **Method 2** - Once the lift appears weaker, fly a half circle as tightly as possible until the rate of climb begins to increase at which time resume the original bank angle, or
iii. **Method 3** - The final method combines the features of the first two methods and is more methodical in its approach, namely:
    - As the climb rate improves, flatten the circle by decreasing the angle of bank to within the range of $\phi \in [\pi/12, \pi/9]$ rad.
    - As the climb rate deteriorates, tighten the circle by increasing the angle of bank to $\phi = 5\pi/18$ rad.
    - If climb remains constant, maintain a constant bank angle within a range of $\phi \in [5\pi/36, \pi/6]$ rad.

In the Glider Flying Handbook that was produced by the Federal Aviation Administration (FAA) of the U.S. Department of Transportation, the following centering technique was detailed as a correction method that a pilot could employ when rolling into a thermal and immediately encountered a sink rate [2]:

- Once a sink rate is encountered, complete a $3\pi/2$ rad turn.
- Following the turn, straighten out for a few seconds and determine if positive lift is encountered.
- If positive lift is confirmed, initiate a turn into the direction that was originally commanded.
- If the aircraft is still sinking, repeat the procedure until a climb rate is generated. Avoid reversing the direction of turn, as the distance flown while reversing turns can lead the aircraft completely away from the thermal.

The flight path of this technique is shown in Figure 2.2. Additionally, the following three variations to the aforementioned correction method were also presented [2]:

i **Variation 1** - When weaker lift is encountered, reduce the turn slightly by $\phi = \pi/36 \rightarrow \pi/8$ rad until a stronger lift force is felt. If the turn is too shallow, it is possible to fly completely away from the updraft. Once positive lift is felt, the original bank angle should be commanded.

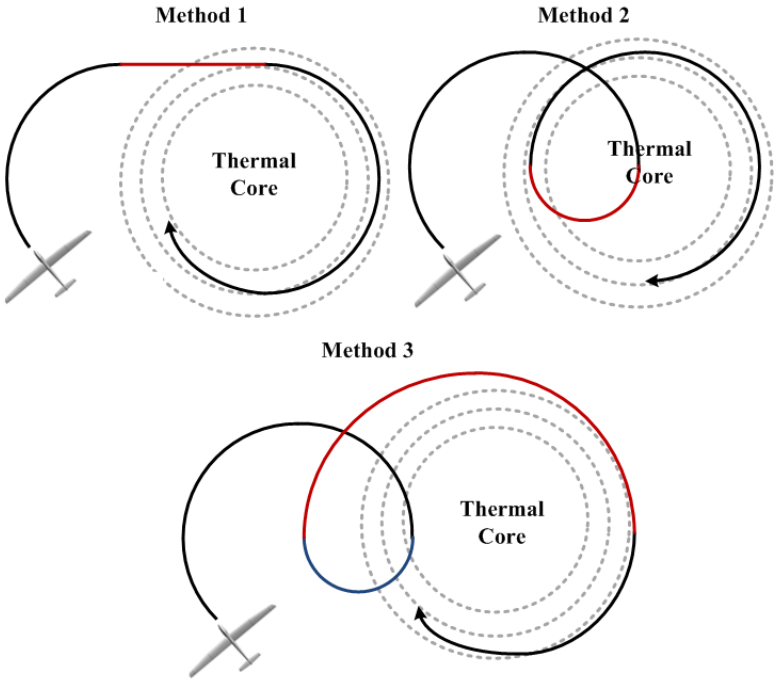


Figure 2.1: Reichmann Centering Methods [4]

ii **Variation 2** - Following a heading change of $\pi/3$ rad after the worst sink rate had been indicated by the variometer, straighten or shallow the turn for a few seconds. This accounts for the lag in the variometer, as the worst sink rate will have occurred a couple of seconds earlier than was indicated. Resume the original bank angle.

iii **Variation 3** - Straighten or shallow the turn for a few seconds when the strongest surge is felt and then resume the original bank angle.

Most of the aforementioned centering strategies were developed through glider piloting experience and an analytical knowledge of gliding flight. These heuristic strategies have formed the foundation of many of the centering algorithms that have been currently developed. Several of these outer-loop soaring controllers were loosely based on Reichmann's third method for thermal centering, which was previously defined above [4]. Allen [9] was one of the first to attempt to use analytical simulation as an approach to quantify the increase in endurance as a result of autonomous soaring.

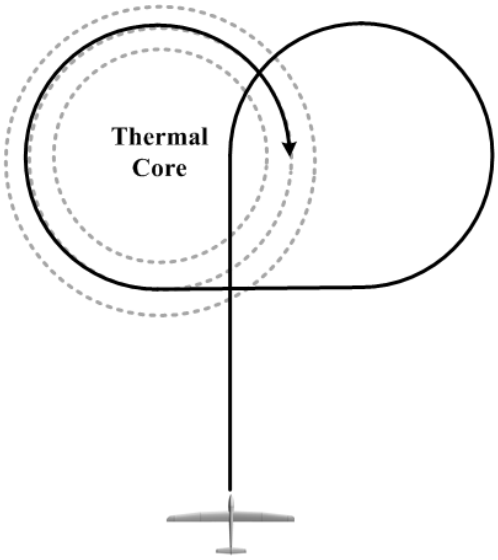


Figure 2.2: 270° Centering Correction Method [2]

Allen and Lin furthered the research by investigating whether the aircraft total energy state, E could be used to detect and soar within thermals, which was defined as [10, 11]:

$$E = mgh + \frac{1}{2}mV^2 \tag{2.1}$$

where m is the aircraft mass, W is the aircraft weight, h is the height above the ground, V is the aircraft true airspeed, and g is the acceleration due to gravity. Additionally, the aircraft energy state was normalized with respect to the aircraft weight and represented an energy height, E_h. The normalized energy state of the aircraft was used to estimate thermal position, drift rate, strength and radius. A switching mode logic was developed to determine when the aircraft should search for thermals or enter the centering mode. The switching criteria was a threshold change in the aircraft normalized energy state. The circle guidance calculates the turn rate, position and velocity error for tracking a circular path inside the thermal. The controller attempts to drive position and velocity errors to zero and respond to changes in the rate of change of energy height. The turn rate to fly at a given aircraft velocity, V was given as [10, 11]:

$$\dot{\psi} = \frac{V}{r_{cmd}} \tag{2.2}$$

where $\dot{\psi}$ is the aircraft turn rate and r_{cmd} was chosen as 65% of the thermal radius. This radius command ensured a flight path radius that was small enough to remain within the thermal core, but large enough to avoid high aircraft sink rates associated with large bank angles. Latency in the rate of change of energy height estimates was reported to be the primary cause of reduced controller performance in weak thermals. Furthermore, it was suggested that soaring algorithms must incorporate aircraft acceleration information to improve the energy state estimations. An additional contribution was a chimney thermal updraft model that was derived from surface and rawinsonde balloon measurements taken at Desert Rock, Nevada.

Andersson *et al.* also used Reichmann's third method as a foundation for their thermal centering controller [12, 13]. Reichmann's method was implemented by applying a feedback control law where the change in specific energy acceleration was used to represent a change in the aircraft's climb rate to generate the turn rate commands. A thermal centering controller producing a turn rate command to the onboard autopilot was proposed as [13]:

$$\dot{\psi} = \frac{V}{r_d} - k_1 \ddot{E}_h \tag{2.3}$$

where r_d was the desired aircraft radial distance to the core of the updraft once centered on the thermal, k_1 was the feedback coefficient, and $\ddot{E}_h$ was used to represent changes in the time rate of change of energy height. Additionally, a measure of the aircraft's tangential track with respect to the thermal (time derivatives of η) and the thermal radius, r were used to develop the following kinematic equations that described the centering problem [13]:

$$\begin{aligned} \dot{\eta} &= \dot{\psi} - \frac{V \cos \eta}{r} \\ \dot{r} &= -V \sin \eta \end{aligned} \tag{2.4}$$

The control objective was to drive η to zero, and r to r_d by controlling the turn rate, $\dot{\psi}$, of the glider. The overall objective of the controller was to adjust the flight path of the UAV so that it's centered on the updraft by driving the $\ddot{E}_h$ to zero, in which case $\dot{E}_h$, over time, converges to a positive constant value. Following simulation and initial flight testing, it was proposed that the controller be updated to incorporate a term containing $\dddot{E}_h$ (the time derivative of $\ddot{E}_h$) [12]:

$$\dot{\psi} = \frac{V}{r_d} - k_1 \ddot{E}_h - k_2 \dddot{E}_h \tag{2.5}$$

This was done to improve performance by better handling lag caused by the filtering of feedback signals. The controller was shown to be asymptotically stable, and a region of attraction was established with the size dependent on the updraft's size

and strength as well as the controller feedback gain and desired range to the thermal center.

Akhtar *et al.* used a combination of an analytical approach using the simplified aircraft performance equations of motion and a soaring strategy to develop a thermal centering algorithm [14]. The basis for the centering algorithm was defined as follows:

- If a thermal updraft is detected, an angle of $\phi = 7\pi/30$ rad is commanded.
- If a sink rate is measured, the sailplane continues to circle. After a heading change of $5\pi/6$ rad, the bank angle is reduced for 3 s before resuming $\phi = 7\pi/30$ rad turns.

This method assumed that the center of the thermal core was known and used the aircraft positional information with respect to the thermal core to correct for a deviation in the target aircraft turn radius. The target airspeed and angle of bank were determined graphically from the relationship between aircraft airspeed, turn radius, minimum sink rate, and the profile of the thermal encountered. An arbitrary average turn radius of 200 ft was targeted during all simulations regardless of the thermal characteristics, and a Proportional-Plus-Integral (PI) structure was used for the airspeed and bank angle controllers.

Much of the research on thermal centering controllers has focused on centering the aircraft's trajectory around the core of the thermal by adjusting the turn rate, and using Reichmann's centering method as its foundation [15–17]. It was reported that this thermal centering strategy offers the following advantages for thermal soaring [16]:

i. The algorithm does not require an estimate of the thermal profile.
ii. Use of the aircraft's sink polar is not required, and as such the algorithm works for multiple aircraft without the need for additional tuning.
iii. The strategy is simple to implement, with minimal computational and memory requirements.
iv. The controller easily adapts to changing thermal conditions.

Additionally, the research has focused on how to either estimate or measure total aircraft energy. In some cases, estimations of $\dot{E}_h$ were made directly using sensor measurements for local updraft strength using a netto variometer [15]. Other techniques involved using an asymmetric Savitzky-Golay filter that computes estimates of total energy, rate of change of total energy and the second derivative of total energy using polynomial approximations over a moving time window [16]. Despite the differences in measuring techniques, successful simulations with stable thermal centering were reported and in some instances, a flight range of over 25 nmi was achieved using only the energy from a 450 ft high launch [15].

Liu *et al.* pursued a more advanced control strategy in that a nonlinear predictive control system was used to harvest the energy from the atmospheric updrafts [18]. In this framework, an online estimation of the updraft distribution using a two-layer *Generalized Regression Neural Network* (GRNN) was combined with a heuristic search method to obtain an optimal trajectory for the UAV. To allow for real-time computation of the control commands, the optimal control problem was solved in a Cartesian coordinate system using a 3DOF model and the control inputs were applied to a more realistic 6DOF model. Simulations showed that the control system succeeded in energy extraction in a challenging dynamic atmospheric environment while satisfying its real-time constraints.

Robust adaptive control and anti-windup design tools were used by Kahveci *et al.* to develop an adaptive control scheme based on Linear Quadratic (LQ) control with disturbance rejection [19]. In this investigation, the presence of actuator saturation nonlinearities and large parametric uncertainties were considered only for the linearized longitudinal dynamics of the aircraft to determine an optimal horizontal velocity for static soaring. The saturation-type nonlinearity was addressed by an adaptive version of a Linear Matrix Inequality (LMI) based anti-windup design.

In this research area, a number of heuristic and analytical approaches have been investigated to determine an effective method to center thermals. Most of the outer-loop controller strategies used a turn rate command that was easy to implement in real-time and produced decent results. However, these controllers were primarily based on heuristic strategies and did not consider the full set of aircraft states and controls nor actuator limitations. Additionally, the resulting trajectory from the centering algorithm was not analyzed to determine if the aircraft was in fact optimally extracting energy from the thermal updraft. As such and before developing a controller for thermal centering, the optimal trajectory for soaring flight in a given thermal updraft strength must first be solved and compared to analytic derivations to better understand the flight condition required for thermal soaring.

2.3 Path Planning

Once a thermal updraft has been detected and its available energy has been exploited, the next phase in the autonomous soaring problem is path planning or determining how the aircraft should best utilize the potential energy gained. One of the potential options that was previously discussed is thermal mapping. Lawrance and Sukkarieh extended their technique of thermal energy mapping of the wind field to develop a path planning and dynamic target assignment algorithm to generate energy-gain paths from the current wind estimate [20, 21].

Path planning for an autonomous vehicle becomes a bit more of a challenge as

typical visual cueing aides are not available or apparent to the SUAV. Chakrabarty and Langelaan developed a graph-based method for planning energy-efficient trajectories over a set of waypoints [22, 23]. This approach introduced an energy map, which provided the path to the destination as a sequence of waypoints, the optimal speeds to fly for each segment between waypoints, and the heading required to fly along a segment while accounting for a known three-dimensional wind field. The target airspeed was computed by minimizing the energy expenditure for that segment. As such, the total energy required to reach the destination goal defined the energy map. The major constraint in generating the energy map was that each path had to always proceed toward the goal. As a result, lower-energy trajectories may have existed that proceeded away from the goal for some portion of the path. The computation of minimum energy to fly between waypoints was used with a heuristic planner to determine the effect of the progress-to-goal constraint. Although there may have existed paths that were more efficient from an energy perspective in some cases, the cost was often an increased distance flown to the destination.

Langelaan also developed a tree-based approach to find a feasible trajectory between the start position and an objective that was well beyond the simple gliding range of an aircraft [24]. Trajectory branches were predefined to decrease computational efforts in expanding the tree during optimization. Given the high nonlinearities and non-convexity of the problem, tree paths were expanded randomly to find feasible solutions. Wind speed and direction were known quantities but were not included in the set of predefined trajectory branches. Nodes were defined within the branches and encoded aircraft position, heading, airspeed, its energy state, and distance from the objective. A weighted random approach was used for node selection, with the weight dependent on specific energy height and the distance from the objective. This directed the search towards the goal but allowed for exploration of the operating environment. Upon the selection of a node, this set of branches was added to the tree and checked for feasibility based on height above the ground and the allowable divergence from the goal. Infeasible nodes were pruned and tree expansion terminated when a node was within gliding distance of the goal.

As an extension to the maximizing range mission objective, Kahveci and Ioannou proposed using Genetic Algorithms (GAs) for path planning in an attempt to reach the assigned destination objective in the shortest amount of time while minimizing use of the on-board power supply [25]. Thermal updrafts were given a predefined location in an operating area on a map and were represented by the genes on the chromosomes describing a potential thermal soaring route. For a given population size, the chromosomes were randomly initialized denoting a proposed flight path. Based on the natural evolution paradigm, near-optimal solutions were obtained for a determinant number of generations. Additionally, GA simulation optimal route results were consistent with those obtained by iterative style algorithms.

One of the main underlining assumptions made for the development of these energy-based path planning approaches was having knowledge of the wind field in the operating region. In practice, the thermal locations and strengths are unknown and will evolve, shift, and dissipate making it extremely difficult to map out an entire trajectory. The dynamic nature of thermal activity and the highly nonlinear behaviour of the updraft fields make it extremely difficult, if not impossible, to implement these approaches in practice.

2.4 Target Acquisition and Surveillance

An operational use of the energy gained through thermal energy exploitation is target acquisition and surveillance. Guo *et al.* [26] investigated the basic patterns as well as tradeoffs in UAV flights that maximized see-ability while concurrently minimized power consumptions. A see-ability model was established that peaked when the UAV was flying at a certain angle from the normal vector perpendicular to the surface of the target and was defined as [26]:

$$S_a = \frac{\cos(\theta - \theta_{op})}{1 + \kappa(R/d_c)^2} \tag{2.6}$$

where θ_{op} was the optimal viewing angle, $d_c = V_c^2/g$ was the normalizing quantity for distance for a commanded airspeed V_c, and κ was a scaling factor. UAV flights were formulated as nonlinear periodic optimal control problems that were subject to various motion constraints and these trajectories were solved using a gradient-based optimizer.

Nguyen *et al.* also explored the same objective of searching for a ground target while simultaneously collecting energy from known thermal updraft sources [27]. However, the objective was formulated as a tree search problem by dividing the mission into similar segments of flying between and climbing in thermals. This algorithm attempted to maximize the probability of detecting a target by exploring a tree of the possible thermal-to-thermal transitions and solving the trajectories using a gradient-based optimizer.

Makovkin and Langelaan considered a coordinated soaring approach where a flock of SUAVs relied on thermal exploitation to maximize endurance for monitoring missions [28]. More specifically, each SUAV repeated a round-trip between a thermal and a target of interest so as to maintain persistent surveillance of the target. The research focused on minimizing the number of agents required to maintain continuous, complete surveillance of the target for a given thermal strength, location, and distance between the thermal and target. It was shown that the optimal cruising speed for maximizing the endurance of monitoring-type missions varied between

the minimum sink airspeed and speed for best aerodynamic efficiency defined as the MacCready speed. Additionally, an examination of multiple-thermal exploitation was also presented to determine the flock size needed to provide full aerial coverage.

Another application that SUAVs can perform between thermal energy extraction flight phases, is to perform target localization. Since target localization is highly dependent on the vehicle trajectory, Ponda *et al.* explored the development of optimal trajectories for this usage [29].

2.5 Knowledge Gaps in the Research

Based on the literature review, knowledge gaps exist in this research field and to fill some of these gaps the following areas will be explored in this thesis dissertation:

i. Obtain a better understanding of the flight trajectory required for optimal thermal soaring flight.
ii. Determine the relationship between the optimal flight trajectory and a particular thermal updraft strength and aircraft geometric characteristics.
iii. Compare the full aircraft states and controls from the trajectories with the analytic derivations to better understand the flight condition required for thermal soaring.
iv. Develop a control strategy that can be implemented real-time with no prior knowledge of the wind field in a given operating environment.
v. Investigate whether an aircraft can simultaneously observe a target with good image resolution and efficiently extract energy from a thermal updraft.

3 Aircraft Characterization

The first step in trying to solve for an optimal thermal soaring trajectory is to fully understand and characterize the dynamics of an aircraft system. For the purposes of this research project, the Cularis D-5223 aircraft was acquired for the experimental implementation of a thermal soaring algorithm. In this chapter, a development of the aircraft point-mass and a full characterization of the Cularis aircraft are presented. Additionally, a study of the effect of the wind updraft component contributions on an aircraft dynamic system is also covered.

3.1 Reference Axis Systems

The analysis of the flight path of an autonomous soaring vehicle requires the definition of a system of axes to which its relative motion can be referred. All axis systems follow right-hand rule methodology to define their orientation. The Earth axis system is selected as the fixed frame with its origin located on the surface with the axes denoted with a subscript e. The translational and rotational transformations from the Earth to the body axes of the aircraft (denoted subscript b) is outlined in Figure 3.1. Additionally, detailed angular relationships are presented in Figure 3.2 to show the relationship between the Earth's axes and the aircraft's body and velocity (denoted subscript v) axis system. The origins of these additional reference axis systems are located at the center of gravity of the aircraft. For the purposes of this study, the point-mass model has been assumed to be accurate enough to simulate aircraft trajectories while its simplicity helps to improve algorithm efficiency for the optimization of such trajectories. The point-mass model aircraft equations of motions are derived in the velocity axis system with respect to the Earth's axes.

3.2 Point-Mass Model Equations of Motion

The aircraft is modeled as a point-mass system and its motion is related to the Earth's fixed reference frame by its velocity vector. A free-body diagram showing the forces

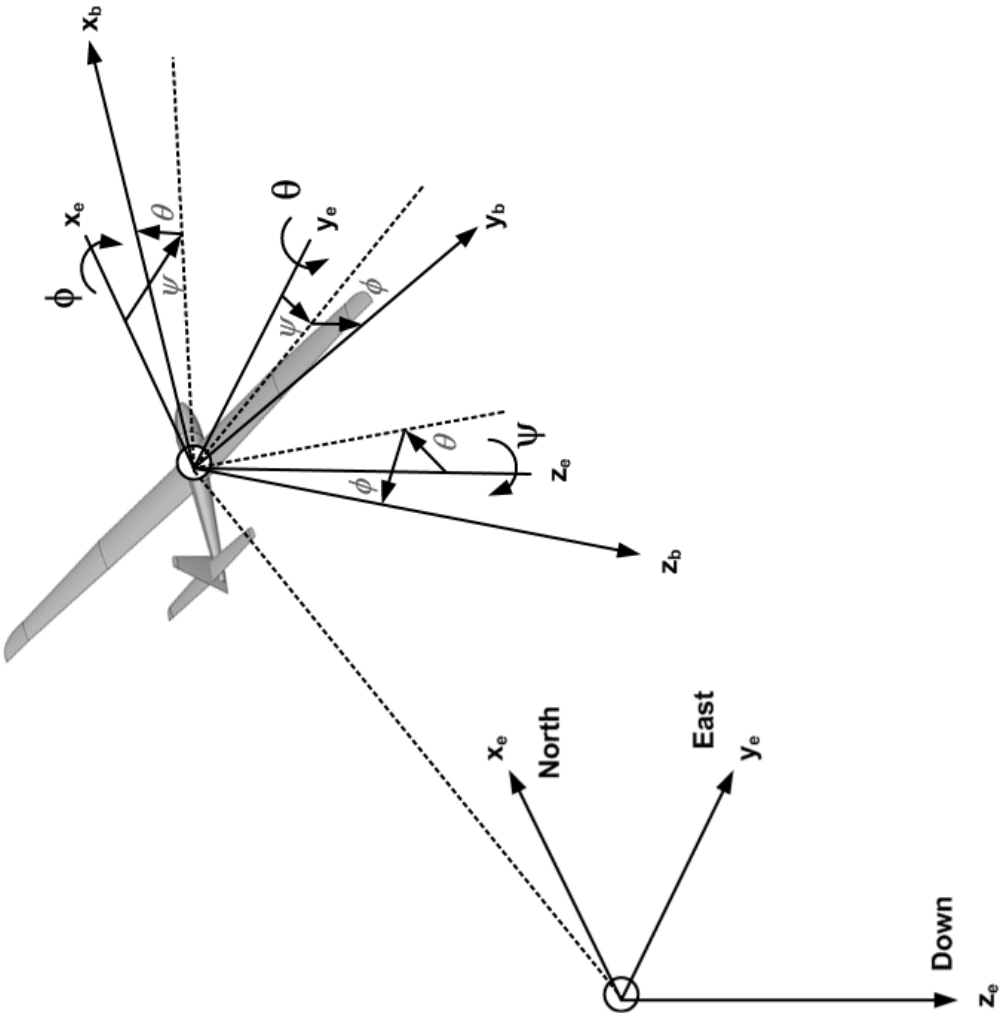
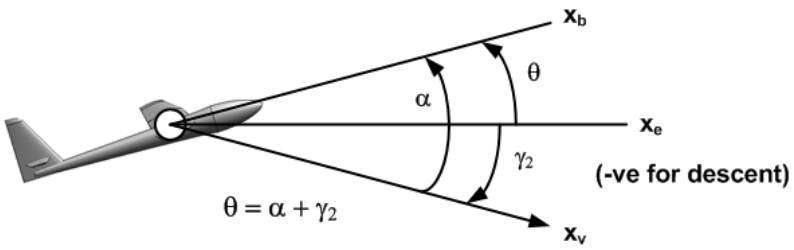
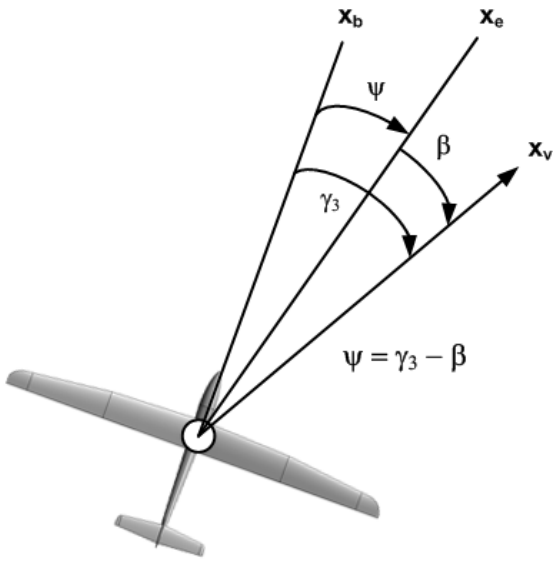


Figure 3.1: Aircraft Body Axis Orientation [30]



(a) Pitch Angle



(b) Aircraft Heading

Figure 3.2: Angular Relationships Between Earth, Body, and Velocity Axes (subscripts e, b, and v respectively) [30]

acting on an aircraft is shown in Figure 3.3. Using Newton's second law of motion, the general equations of motion for an aircraft are derived in terms of its resultant inertial force vector, $\boldsymbol{F}_I$ that is resolved in the velocity axis system [30]. The forces acting on the aircraft are defined as gravitational, $\boldsymbol{F}_g$ aerodynamic, $\boldsymbol{F}_a$ and propulsive, $\boldsymbol{F}_p$, which act in the Earth, wind, and body axis systems respectively.

$$[\boldsymbol{F}_I]_v = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_v = \frac{\mathrm{d}(mV)_v}{\mathrm{d}t} \tag{3.1}$$

$$[\boldsymbol{F}_g]_e = \begin{bmatrix} F_{g,x} \\ F_{g,y} \\ F_{g,z} \end{bmatrix}_e = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}_e \tag{3.2}$$

$$[\boldsymbol{F}_a]_w = \begin{bmatrix} F_{a,x} \\ F_{a,y} \\ F_{a,z} \end{bmatrix}_w = \begin{bmatrix} -D \\ Y \\ -L \end{bmatrix}_w \tag{3.3}$$

The full development of the generic equations of motion for an aircraft are summarized as follows:

$$\begin{aligned} -D + [T\cos(\alpha+\tau_1)\cos\beta - D_M] - mg\sin\gamma_2 &= m\dot{V} + \dot{m}V \\ Y\cos\gamma_1 + L\sin\gamma_1 + T[-\cos(\alpha+\tau_1)\sin\beta\cos\gamma_1 + \sin(\alpha+\tau_1)\sin\gamma_1] &= mV\dot{\gamma}_3\cos\gamma_2 \\ Y\sin\gamma_1 - L\cos\gamma_1 + T[-\cos(\alpha+\tau_1)\sin\beta\sin\gamma_1 - \sin(\alpha+\tau_1)\cos\gamma_1] + mg\cos\gamma_2 &= -mV\dot{\gamma}_2 \end{aligned}$$

where the description of each variable is contained in Table 3.1. Assuming the aircraft is trimmed, the following assumptions can be made for gliding flight:

i. The rate of change of aircraft mass is negligible, $\dot{m} = 0$,
ii. The aircraft is in symmetric flight, such that $\beta = 0$ and $Y = 0$, and
iii. The sailplane is not powered and as such there is no net thrust available from the propulsive system, $T = 0$ hence no momentum drag is generated, $D_m = 0$.

Using the aforementioned assumptions, the equations of motion are reduced to:

$$\begin{aligned} -D - mg\sin\gamma_2 &= m\dot{V} \\ L\sin\gamma_1 &= mV\dot{\gamma}_3\cos\gamma_2 \\ L\cos\gamma_1 - mg\cos\gamma_2 &= mV\dot{\gamma}_2 \end{aligned} \tag{3.4}$$

These equations represent the dynamics of the aircraft system in the velocity axes. In order to use the aforementioned dynamics for thermal soaring, they must be augmented with wind component forces to fully characterize the motion of the system.

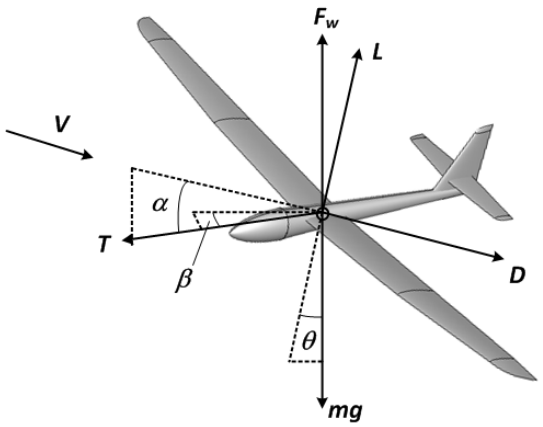


Figure 3.3: Free Body Diagram Outlining the Forces Acting on the Aircraft

Table 3.1: Performance Equations of Motion Variables

Variable	Description	Units
D	Drag of the Aircraft	$\mathrm{lb_f}$
D_M	Momentum Drag	$\mathrm{lb_f}$
L	Aircraft Lift	$\mathrm{lb_f}$
T	Aircraft Thrust	$\mathrm{lb_f}$
Y	Side Force	$\mathrm{lb_f}$
m	Aircraft Mass	slug
$\dot{m}$	Rate of Change of Aircraft Mass	slug/s
V	Aircraft True Airspeed	ft/s
$\dot{V}$	Rate of Change of Aircraft True Airspeed	$\mathrm{ft/s^2}$
α	Angle of Attack	rad
τ_1	Propulsion Vectoring	rad
β	Sideslip Angle	rad
γ_1	Bank Angle	rad
γ_2	Climb Angle	rad
γ_3	Track Angle	rad
$\dot{\gamma}_1$	Roll Rate	rad/s
$\dot{\gamma}_2$	Climb Rate	rad/s
$\dot{\gamma}_3$	Turn Rate	rad/s

3.2.1 Equations of Motion with Wind Component Forces

The wind force, $\boldsymbol{F}_w$ acts in the Earth axis system and can be expressed in the following matrix form:

$$[\boldsymbol{F}_w]_e = \begin{bmatrix} F_{w,x} \\ F_{w,y} \\ F_{w,z} \end{bmatrix}_e = \begin{bmatrix} m\dot{V}_{w,x} \\ m\dot{V}_{w,y} \\ m\dot{V}_{w,z} \end{bmatrix}_e \tag{3.5}$$

where $\dot{V}_{w,x}$, $\dot{V}_{w,y}$, and $\dot{V}_{w,z}$ are the components of the wind acceleration. In order to include the effect of the wind on the aircraft, the wind force must be transferred from the Earth to velocity axes using a rotational matrix as follows:

$$[\boldsymbol{F}_w]_v = \begin{bmatrix} \cos\gamma_2 & 0 & -\sin\gamma_2 \\ 0 & 1 & 0 \\ \sin\gamma_2 & 0 & \cos\gamma_2 \end{bmatrix} \begin{bmatrix} \cos\gamma_3 & \sin\gamma_3 & 0 \\ -\sin\gamma_3 & \cos\gamma_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_{w,x} \\ F_{w,y} \\ F_{w,z} \end{bmatrix}_e$$

$$[\boldsymbol{F}_w]_v = \begin{bmatrix} \cos\gamma_2\cos\gamma_3 & \cos\gamma_2\sin\gamma_3 & -\sin\gamma_2 \\ -\sin\gamma_3 & \cos\gamma_3 & 0 \\ \sin\gamma_2\cos\gamma_3 & \sin\gamma_2\sin\gamma_3 & \cos\gamma_2 \end{bmatrix} \begin{bmatrix} m\dot{V}_{w,x} \\ m\dot{V}_{w,y} \\ m\dot{V}_{w,z} \end{bmatrix}_e$$

$$[\boldsymbol{F}_w]_v = \begin{bmatrix} m[\dot{V}_{w,x}]_e\cos\gamma_2\cos\gamma_3 + m[\dot{V}_{w,y}]_e\cos\gamma_2\sin\gamma_3 - m[\dot{V}_{w,z}]_e\sin\gamma_2 \\ -m[\dot{V}_{w,x}]_e\sin\gamma_3 + m[\dot{V}_{w,y}]_e\cos\gamma_3 \\ m[\dot{V}_{w,x}]_e\sin\gamma_2\cos\gamma_3 + m[\dot{V}_{w,y}]_e\sin\gamma_2\sin\gamma_3 + m[\dot{V}_{w,z}]_e\cos\gamma_2 \end{bmatrix} \tag{3.6}$$

Using the result from Equation 3.6, the wind force contribution can be added to the reduced equations of motion defined in Equation 3.4 to yield the soaring flight equations of motion for a point-mass aircraft model:

$$\begin{aligned} -D - mg\sin\gamma_2 + m[\dot{V}_{w,x}]_e\cos\gamma_2\cos\gamma_3 + m[\dot{V}_{w,y}]_e\cos\gamma_2\sin\gamma_3 - m[\dot{V}_{w,z}]_e\sin\gamma_2 &= m\dot{V} \\ L\sin\gamma_1 - m[\dot{V}_{w,x}]_e\sin\gamma_3 + m[\dot{V}_{w,y}]_e\cos\gamma_3 &= mV\dot{\gamma}_3\cos\gamma_2 \\ L\cos\gamma_1 - mg\cos\gamma_2 + m[\dot{V}_{w,x}]_e\sin\gamma_2\cos\gamma_3 + m[\dot{V}_{w,y}]_e\sin\gamma_2\sin\gamma_3 + m[\dot{V}_{w,z}]_e\cos\gamma_2 &= mV\dot{\gamma}_2 \end{aligned}$$

From the equations of motion, it can observed that a definition of a point-mass model of an aircraft is required. More specifically, the geometrics, aerodynamics, and weight of the aircraft need to be defined. The contribution of the thermal updraft acceleration terms will be covered in the development of the thermal models that introduced in Chapter 4.

3.3 General Aircraft Description

As part of a long-term research project, the Cularis D-5223 aircraft was acquired with the aim of implementing a real-time thermal soaring algorithm on an SUAV. The Multiplex Cularis D-5223 was an electric powered radio-controlled glider that was fitted with a single Himax HC3522-0700 brushless outrunner motor and two folding propeller blades. The main aircraft structures were manufactured from an expanded polypropylene foam, or *Elapor* foam that was designed to be crash resistant. It had a maximum take-off weight of 4.81 lb_f and a wing span of 8.53 ft. The aircraft is shown in Figure 3.4, while a breakdown of the installed components and their contribution to the overall aircraft weight is outlined in Table 3.2. A more detailed description of the aircraft can be obtained from the building and operating instruction manual [31].

Figure 3.4: Multiplex Cularis D-5223 Glider [32]

Table 3.2: Aircraft Component List and Weights

Component Description	Value	Units
Fuselage and HC3522-0700 Engine	1.62	lb_f
Canopy	0.40	lb_f
Wings	1.44	lb_f
3 Cell 11.1V LiPo Battery (2)	0.88	lb_f
Electronic Speed Controller	0.15	lb_f
Autopilot (Ardupilot 2.5)	0.07	lb_f
GPS and Antenna	0.04	lb_f
Power Module, Telemetry and Pitot Static System	0.22	lb_f
Total Aircraft Weight	4.81	lb_f

3.4 Characterization of the Aircraft Aerodynamics

3.4.1 Aircraft Geometric Model

The Cularis glider was modeled in python^{TM} using the *Geometry* class within the *pyACDT* [33] object-oriented framework. *pyACDT* is a tool that was developed by Perez *et al.* to facilitate the definition, analysis, and optimization during conceptual design of aircraft. Within this framework, individual aircraft components were treated as objects and were classified, characterized, and associated based on their configuration and utility. These components were created from common physical and geometric attributes, and were combined into full-aircraft configurations that could be analyzed using the disciplinary analysis tools contained within *pyACDT*. Additionally, airfoil coordinate data that was unit-normalized as compared to the chord length could be uploaded to model the wing, vertical and horizontal tail profile shapes. The geometric model of the Cularis aircraft is shown in Figure 3.5 and the code used to generate the model is contained in Listing A.1. Additionally, the profile shapes of the aircraft's airfoils were digitized using a FaroArm Coordinate Measuring Machine (CCM) and were compared to existing airfoil data to determine an appropriate airfoil designation. The wing airfoil was approximated as an HQ 1.0/12 airfoil, and its profile shape is plotted in Figure 3.6 using airfoil data provided by Quabeck [34]. Both the horizontal and vertical stabilizers were estimated as NACA 0012 and 0009 series airfoils at the root respectively.

3.4.2 Aircraft Aerodynamic Data

Using the geometric model presented in Figure 3.5, the *Aerodynamic* module of *pyACDT* was used to generate aircraft aerodynamic data for the Cularis for use in the point-mass aircraft model. The drag profile of the aircraft was calculated from the geometric model, at different flow conditions, and at a known aircraft center of gravity location. Parasite drag was calculated using the interaction between the individual components within the overall aircraft configuration and considering viscous separation effects, while the skin friction drag was modeled using the Sommer & Short formulation [33, 35]. Induced drag was calculated using a semi-empirical approach based on a *Leading-Edge-Suction Method* as described by Raymer [36]. The drag polar of the aircraft varied little with operating altitude and is shown in Figure 3.7 for an altitude of 1000 ft. The drag polar was calculated using a quadratic polar of the form [37]:

$$C_D = C_{D0} + K{C_L}^2 \tag{3.7}$$

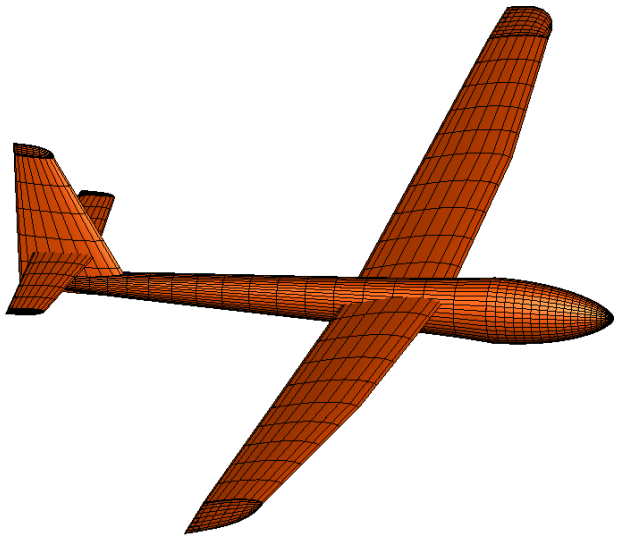


Figure 3.5: Geometric Representation of the Cularis D-5223 Glider

where C_D is the drag coefficient, C_{D0} is the parasitic drag coefficient, K is the induced drag coefficient, and C_L is the lift coefficient. Since the aircraft drag polar varied little with operating altitude, a single polar was generated for the aircraft and is detailed in Table 3.3.

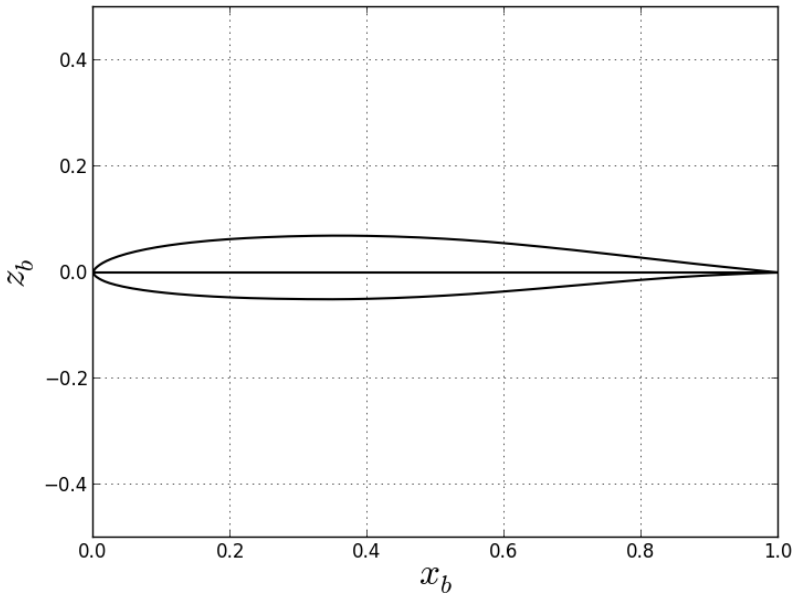


Figure 3.6: HQ 1.0/12 Airfoil

Table 3.3: Aerodynamic Data for the Point-Mass Model of the Cularis D-5223 Glider

Symbol	Description	Value	Units
b	Wing Span	8.54	ft
S	Wing Reference Area	4.57	ft^2
W	Aircraft Take-off Weight	4.81	lb_f
C_{L0}	Lift Coefficient at α_0	0.261	-
$C_{L\alpha}$	Lift Curve Slope	5.865	-
$(C_L)_{max}$	Maximum Lift Coefficient	1.674	-
C_{D0}	Parasitic Drag Coefficient	0.0223	-
$\left(\frac{L}{D}\right)_{max}$	Maximum Lift-to-Drag Ratio	23.09	-
K	Induced Drag Coefficient	0.021	-
e	Oswald's Efficiency Factor	0.95	-

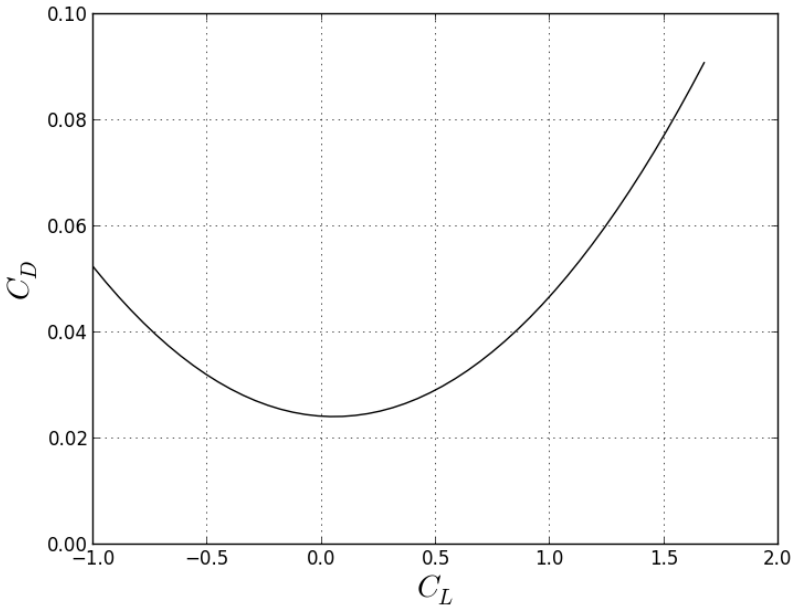


Figure 3.7: Drag Polar for the Cularis D-5223 Glider at an Altitude of $h = 1000$ ft

3.5 Aircraft Gliding Flight Characteristics

In order to validate the results from the optimal trajectories, the gliding flight characteristics of the aircraft must be investigated. As such, analytic expressions for the minimum glide angle, sink rate, turning radius, and pertinent gliding flight velocities are presented. To evaluate the gliding flight characteristics of the Cularis, a *Gliding Flight* class was created in *pyACDT* and is shown in Listing A.2. The class was initialized with a point-mass model of the aircraft, and could output the minimum sink angle, vertical velocity, and turn radius for a given velocity, bank and climb angle. Additionally, the pertinent gliding flight velocities could also be generated for a given altitude and bank angle.

3.5.1 Minimum Glide Angle

For an aircraft flying at a constant velocity, $\dot{V} = 0$ and at a constant climb angle, $\dot{\gamma}_2 = 0$ the performance equations of motion for a point-mass aircraft model that are detailed in Equation 3.4 can be further reduced to the following:

$$D = -mg\sin\gamma_2 \tag{3.8}$$

$$L = \frac{mg\cos\gamma_2}{\cos\gamma_1} \tag{3.9}$$

Expressing Equations 3.8 and 3.9 in terms of a lift-to-drag ratio, we have:

$$\frac{C_L}{C_D} = \frac{L}{D} = \frac{\frac{mg\cos\gamma_2}{\cos\gamma_1}}{-mg\sin\gamma_2} = \frac{-\cos\gamma_2}{\cos\gamma_1\sin\gamma_2}$$

$$\sin\gamma_2 = \frac{-\cos\gamma_2}{\frac{C_L}{C_D}\cos\gamma_1} \tag{3.10}$$

The minimum equilibrium glide angle is achieved at the highest aerodynamic efficiency flying while flying straight and level, $\gamma_1 = 0$ and can be calculated as:

$$\tan(\gamma_2)_{min} = \frac{-1}{\left(\frac{L}{D}\right)_{max}} \tag{3.11}$$

For the Cularis aircraft in level flight, $\left(\frac{L}{D}\right)_{max} = 23.09$ and the minimum glide angle is given as:

$$(\gamma_2)_{min} = -0.043\,\text{rad} \tag{3.12}$$

The minimum equilibrium glide angle will be used as the initial starting condition for the climb angle in the optimization problem formulation.

3.5.2 Minimum Sink Rate

The vertical velocity of an aircraft is given as [37]:

$$\dot{h} = V \sin \gamma_2 \tag{3.13}$$

where $h = -z$ and represents the aircraft altitude, $\dot{h}$ is the vertical velocity, and the velocity, V is calculated for a given air density, ρ as:

$$V = \left[\frac{L \cos \gamma_2}{\frac{1}{2} \rho S C_L} \right]^{\frac{1}{2}} = \left[\frac{mg \cos \gamma_2}{\frac{1}{2} \rho S C_L \cos \gamma_1} \right]^{\frac{1}{2}} \tag{3.14}$$

Substituting Equations 3.10 and 3.14 into 3.13, the following expression is obtained for the vertical velocity of a gliding aircraft:

$$\begin{aligned} \dot{h} &= \left[\frac{mg \cos \gamma_2}{\frac{1}{2} \rho S C_L \cos \gamma_1} \right]^{\frac{1}{2}} \left[\frac{-\cos \gamma_2}{\frac{C_L}{C_D} \cos \gamma_1} \right] \\ \dot{h} &= -\left(\frac{W}{S} \right)^{\frac{1}{2}} \left(\frac{2}{\rho} \right)^{\frac{1}{2}} \left(\frac{\cos \gamma_2}{C_L \cos \gamma_1} \right)^{\frac{1}{2}} \left(\frac{\cos \gamma_2}{\frac{C_L}{C_D} \cos \gamma_1} \right) \end{aligned}$$

Expressing the the vertical velocity of the aircraft in terms of general aircraft design characteristics (i.e. wing loading, W/S where $W = mg$):

$$\dot{h} = -\left(\frac{W}{S} \right)^{\frac{1}{2}} \left(\frac{2}{\rho} \right)^{\frac{1}{2}} \left(\frac{C_D}{C_L^{\frac{3}{2}}} \right) \sec^{\frac{3}{2}} \gamma_1 \cos^{\frac{3}{2}} \gamma_2 \tag{3.15}$$

Using this expression, the vertical velocity profile as a function of airspeed can be generated for the Cularis aircraft and is shown for a representative operational altitude of $h = 1000$ ft ($\gamma_1 = 0$) in Figure 3.8. It can be observed that for a minimum sink rate condition, the aircraft would need to fly at an airspeed of approximately $V = 22.5$ ft/s.

3.5.3 Aircraft Turning Radius

The flight path for an aircraft in turning flight is shown in Figure 3.9. The angular velocity, or *turn rate* along a curved flight path is determined as follows [37]:

$$V \cos \gamma_2 = R \frac{\mathrm{d}\gamma_3}{\mathrm{d}t} = R\dot{\gamma}_3 \Rightarrow \dot{\gamma}_3 = \frac{V \cos \gamma_2}{R} \tag{3.16}$$

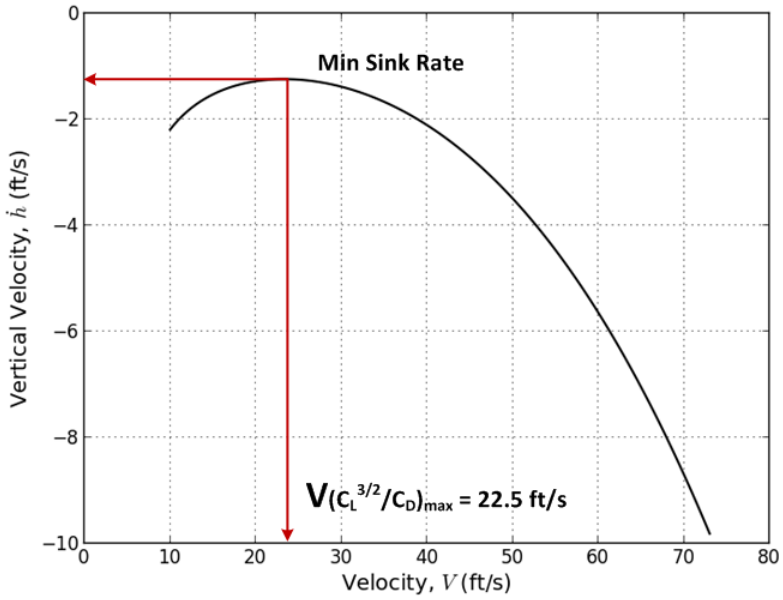


Figure 3.8: Vertical Velocity as a Function of Airspeed for the Cularis D-5223 Glider at an Altitude of $h = 1000$ ft

Using the reduced equation of motion defined in Equation 3.4 and rearranging for the turn rate, $\dot{\gamma}_3$ results in:

$$\dot{\gamma}_3 = \frac{L \sin \gamma_1}{mV \cos \gamma_2} \tag{3.17}$$

Substituting Equation 3.16 into 3.17 and solving for the turn radius, R yields:

$$\begin{aligned} \frac{V \cos \gamma_2}{R} &= \frac{L \sin \gamma_1}{mV \cos \gamma_2} \\ R &= \frac{mV^2 \cos^2 \gamma_2}{L \sin \gamma_1} \\ R &= \frac{W}{g} \left(\frac{V^2 \cos^2 \gamma_2}{L \sin \gamma_1} \right) \end{aligned}$$

Using the following expression for the lift [37]:

$$L = \frac{1}{2}\rho V^2 S C_L \tag{3.18}$$

the expression for the turn radius becomes:

$$R = \left(\frac{W}{S}\right)\left(\frac{2}{\rho g}\right)\left(\frac{\csc\gamma_1}{C_L}\right)\cos^2\gamma_2 \tag{3.19}$$

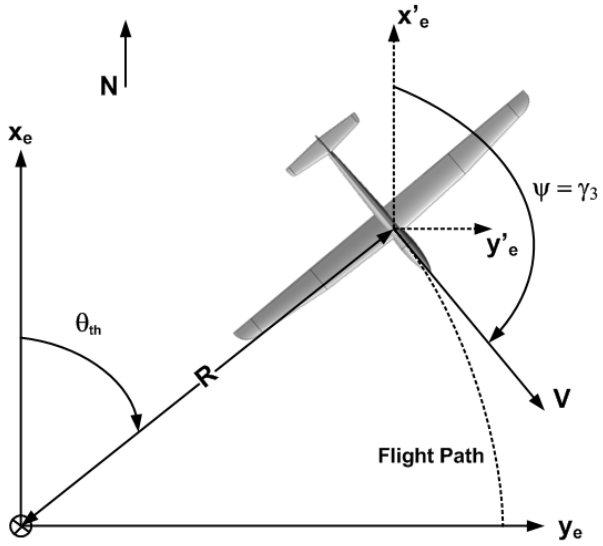


Figure 3.9: An Aircraft in Turning Flight

For a given aircraft, it can be observed that the turn radius is a function of the lift coefficient and correspondingly the airspeed, the commanded angle of bank, and the aircraft climb angle. This expression will be used further in the analysis phase and the development of a control strategy when trying to predict the optimal angle of bank to fly for a given thermal strength.

3.5.4 Gliding Flight Velocities

The pertinent velocities associated with gliding flight were developed by Anderson for an aircraft flying in level flight [37]. For the case where the aircraft is in turning

flight where $L \neq W$, the aircraft's load factor must be considered. As such, the stall speed, V_{stall} can be calculated as follows:

$$V_{stall} = \sqrt{\left(\frac{2}{\rho}\right)\left(\frac{W}{S}\right)\frac{1}{(C_L)_{max}}\sec\gamma_1} \tag{3.20}$$

In gliding flight, the minimum drag condition corresponds to the speed to fly to obtain the maximum glide range. When adjusted for turning flight, the maximum glide range speed, $V_{(L/D)_{max}}$ is calculated as [37]:

$$V_{(L/D)_{max}} = \sqrt{\left(\frac{2}{\rho}\right)\left(\frac{W}{S}\right)\sqrt{\frac{K}{C_{D0}}}\sec\gamma_1} \tag{3.21}$$

To achieve the minimum sink rate, the ratio of $(C_L^{3/2}/C_D)$ is a maximum and corresponds to the minimum power required condition. The resulting minimum sink speed, $V_{(C_L^{3/2}/C_D)_{max}}$ adjusted for bank angle is determined from [37]:

$$V_{(C_L^{3/2}/C_D)_{max}} = \sqrt{\left(\frac{2}{\rho}\right)\left(\frac{W}{S}\right)\sqrt{\frac{K}{3C_{D0}}}\sec\gamma_1} \tag{3.22}$$

The gliding flight velocities of the Cularis D-5223 aircraft for various altitudes in level flight are shown in Table 3.4. It is interesting to note that the stall speed of the aircraft is greater than the speed to fly for a minimum sink rate condition. As such, the Cularis will not be capable of operating at that condition due to the stall limitation.

Table 3.4: Pertinent Glide Velocities for Various Altitudes

Altitude	V_{stall}	$V_{(L/D)_{max}}$	$V_{(C_L^{3/2}/C_D)_{max}}$	Units
Sea Level	22.98	29.31	22.27	ft/s
500 ft	23.15	29.52	22.43	ft/s
1000 ft	23.32	29.74	22.60	ft/s
1500 ft	23.50	29.96	22.76	ft/s
2000 ft	23.67	30.18	22.93	ft/s

As a summary, a point-mass model was presented for the Cularis D-5223 aircraft that will be subsequently used during the generation of optimal trajectories. In addition, the flight dynamic characteristics of the aircraft were obtained using a geometric model and analytical prediction methods.

4 Thermal Models

To determine the optimal thermal soaring trajectories, thermal models need to be developed to provide the wind force contributions that will be used in the point-mass model equations of motion. In this chapter, a brief overview of the conceptual thermal models and the associated atmospheric conditions conducive to thermal soaring are detailed. A development of analytical and heuristic thermal representations based on the conceptual models are also presented. For the subsequent analysis, it is assumed that thermal detection has occurred, its center location is known, and wind shear effects are not considered.

4.1 Conceptual Thermal Models

The FAA defined a thermal as a rising mass of buoyant air and listed it as the most common updraft used by glider pilots in soaring flight [2]. Thermal updrafts are generated due to solar surface heating, which results in localized increased ground temperatures. Consequently, a thermal gradient is created and heats the surrounding air mass. As the density of the heated air is decreased, the air mass is forced upward.

Two conceptual models have been developed to characterize the overall complexity of a thermal updraft. The chimney or column model was defined as a continuous column of rising air that extended from the ground surface right up to the mixing-layer maximum altitude [38]. An illustration of the chimney model is shown in Figure 4.1 while the development of its typical life cycle is provided in Figure 4.2.

The second conceptual model was defined as a bubble or toroidal model. This model was described as an individual rising thermal segment that resembled a vortex ring [2]. Unlike the chimney model, bubble thermals were initially formed on the surface of the Earth when heated areas were smaller. Additionally, they were characterized with having rising air in its core and descending air on the boundaries of the bubble. Figure 4.3 provides a detailed sketch of the bubble thermal model. Regardless of the thermal type, the air in the middle of the thermal rises faster than the air at its outer radius, where typically descent air is present. Additionally, a certain

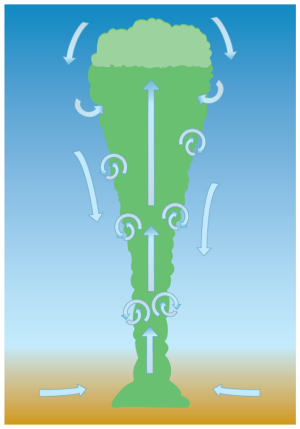


Figure 4.1: A Column or Chimney Thermal Model [2]

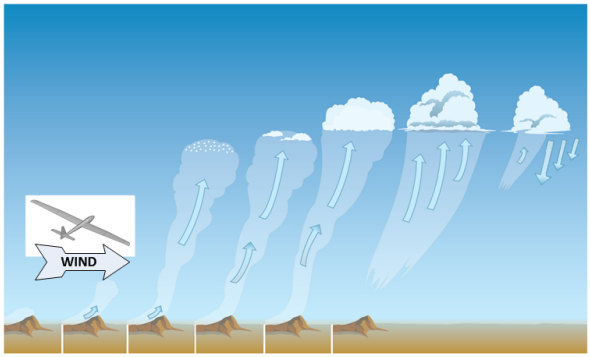


Figure 4.2: Typical Life Cycle of a Thermal with Cumulus Cloud [2]

level of atmospheric thermal instability is required for the development of thermals and their resulting vertical motion. Given the two conceptual thermal types that can develop in the atmosphere, analytical expressions can now be developed to model their behaviour.

4.2 Mathematical Thermal Models

To represent chimney type thermals, basic thermal models were developed as mathematical variations of a 3-dimensional trapezoidal shape. The simplest model is

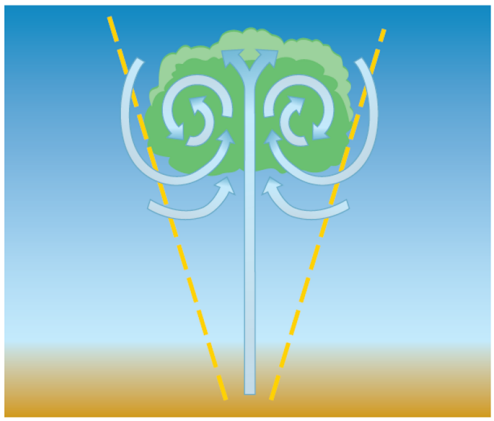


Figure 4.3: Bubble Model of a Thermal [2]

known as a conical trunk, and was defined by the following analytic expression [38]:

$$W_h(r) = \begin{cases} W_{hmax}, & r < r_1, \\ W_{hmax}\frac{r_2 - r}{r_2 - r_1}, & r \in [r_1, r_2], \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

where r is the distance from the thermal center, W_{hmax} is the maximum updraft velocity (or thermal core updraft speed), r_1 and r_2 are the inner and outer thermal radii. A graphical representation of this model is shown in Figure 4.4. In this model, the updraft velocity decreased linearly from the inner to the outer thermal radius and there was no vertical velocity component outside the outer radius. A contour plot and a thermal updraft speed profile at an altitude of $h = 500$ ft for this model are shown in Figures 4.5(a) and 4.5(b) respectively.

An alternative mathematical representation of a thermal updraft is the 2-dimensional Gaussian profile. In this model, the center of the thermal was also taken as the center of the Gaussian profile and is represented by [38]:

$$W_h(r) = W_{hmax} \cdot \mathrm{e}^{-(r/R_T)^2} \tag{4.2}$$

where R_T is the thermal radius. The updraft velocity was scaled to be at a maximum in the thermal core and decreased exponentially towards at the outer radius. Similar to the conical updraft model, the 2-dimensional Gaussian model did not include any exterior downdraft velocity. A contour plot and a thermal updraft speed profile

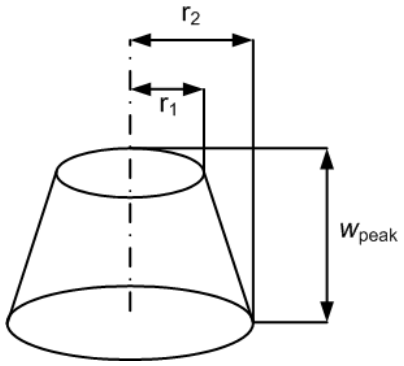


Figure 4.4: Graphical Representation of the 3D Trapezoidal Thermal Model

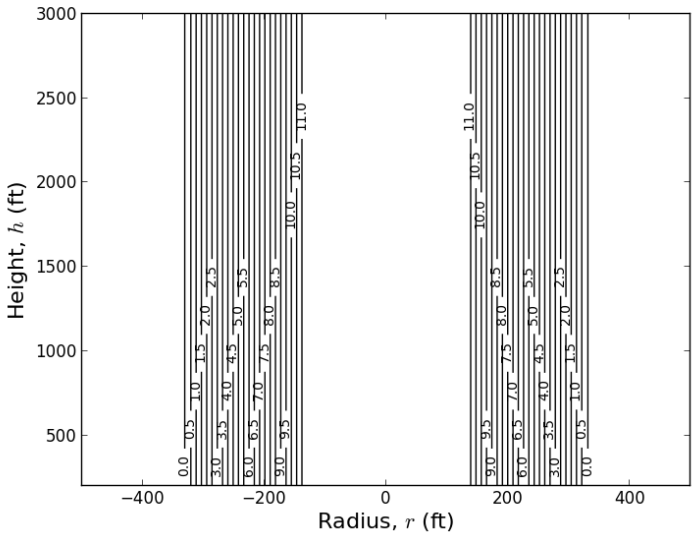
at an altitude of $h = 500$ ft for this model are shown in Figures 4.6(a) and 4.6(b) respectively.

Gedeon developed an extension to the Gaussian model to incorporate a downdraft velocity that would be present outside the thermal radius. The thermal strength in this model is defined as [38]:

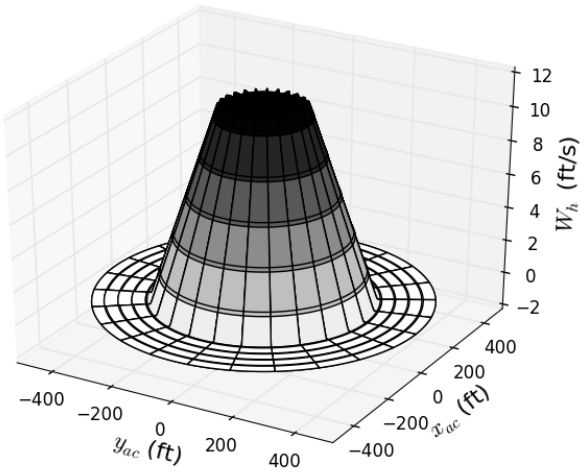
$$W_h(r) = W_{hmax} \cdot \mathrm{e}^{-(r/R_T)^2} \cdot [1 - (r/R_T)^2] \tag{4.3}$$

A contour plot and a thermal updraft speed profile at an altitude of $h = 500$ ft for this model is shown in Figures 4.7(a) and 4.7(b) respectively.

Each of the aforementioned mathematical models were easy to implement and required minimal computational effort; however, they did not have a direct dependency on altitude for calculating updraft strength. As such, a maximum updraft velocity, W_{hmax} for each altitude must be included as an additional component of the model in order to account for these effects. Even though these models were not directly used to determine the wind force contribution, the state-of-the-art thermal models were largely based on their formulation and the notion of a trapezoidal updraft distribution.

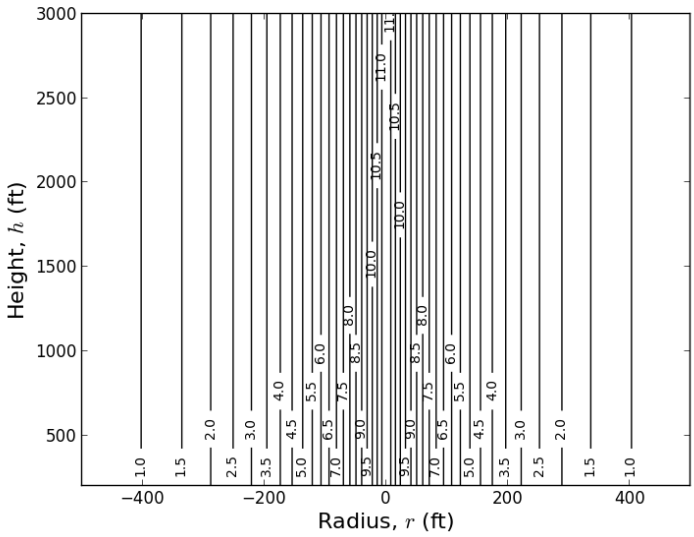


(a) Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s

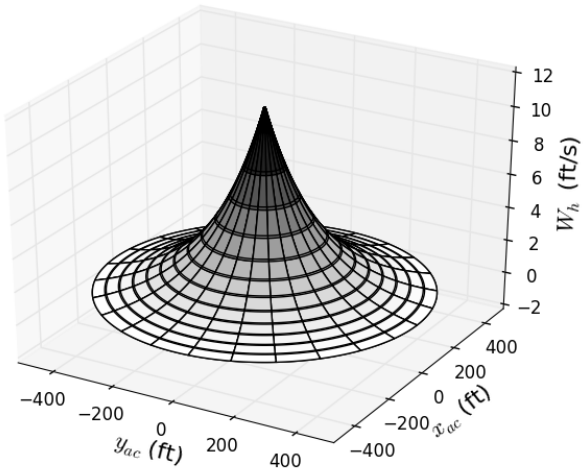


(b) Thermal Updraft Speed Profile at an Altitude of $h = 500$ ft

Figure 4.5: 3D Trapezoidal Thermal Model

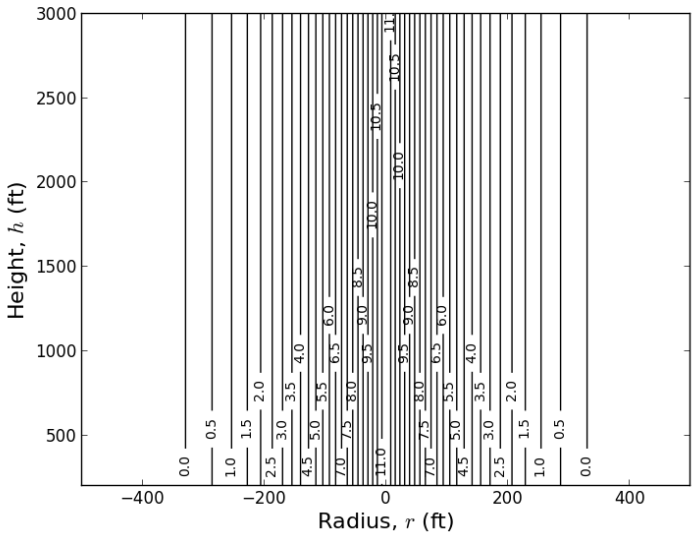


(a) Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s

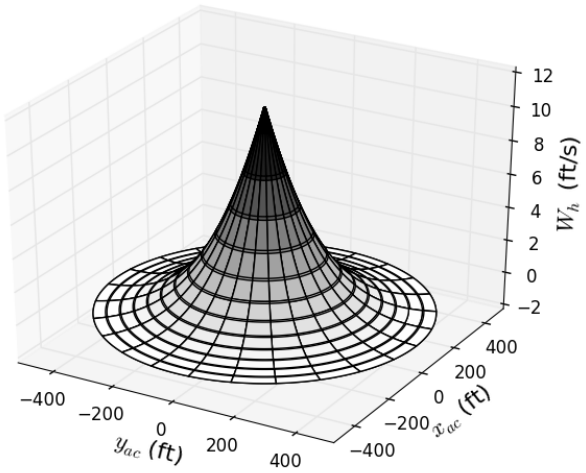


(b) Thermal Updraft Speed Profile at an Altitude of $h = 500$ ft

Figure 4.6: Gaussian Thermal Model



(a) Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s



(b) Thermal Updraft Speed Profile at an Altitude of $h = 500$ ft

Figure 4.7: Gedeon Thermal Model

4.3 Chimney Thermal Model

In contrast to the mathematical approaches, Allen [39] used an experimental method in Desert Rock, Nevada to develop a the current state-of-the-art chimney or column thermal model that was based on surface and altitude measurements. The measured data was used to create a statistical representation of the convective velocity scale, w^* and the convective mixing-layer thickness, h_i. The results of this statistical analysis are shown in Table 4.1 as seasonal variations of updraft strength and altitude. This model and all the corresponding constants were developed in SI units and will be presented as such.

Using the scaling parameters from Table 4.1, and the height, h, the average updraft velocity, $\bar{w}$ is calculated as [39]:

$$\bar{w} = w^* \left(\frac{h}{h_i}\right)^{\frac{1}{3}} \left(1 - 1.1\frac{h}{h_i}\right) \tag{4.4}$$

while the updraft outer radius, r_2 and inner radius, r_1 are determined from [39]:

$$r_2 = \max\left(10,\ 0.102\left(\frac{h}{h_i}\right)^{\frac{1}{3}} \left(1 - 0.25\frac{h}{h_i}\right) h_i\right) \tag{4.5}$$

$$\frac{r_1}{r_2} = \begin{cases} 0.0011 r_2 + 0.14, & r_2 < 600\text{ m} \\ 0.8, & \text{otherwise.} \end{cases} \tag{4.6}$$

Similar to the pure mathematical representations, it was assumed that the vertical velocity profile followed a revolved trapezoid updraft distribution. Additionally and for a given r_1/r_2 ratio, shape constants were used to modify the velocity profile to provide a bell-shaped distribution to model the revolved trapezoidal shape and are detailed in Table 4.2.
Using the shape constants from Table 4.2, the updraft velocity, W_h at a particular radial distance from the thermal center, r, is given as [39]:

$$W_h = w_{peak}\left(\frac{1}{1 + \left|k_1 \frac{r}{r_2} + k_3\right|^{k_2}} + k_4\frac{r}{r_2} + w_d\right) \tag{4.7}$$

where the peak value of the updraft velocity, w_{peak} is given as:

$$w_{peak} = \frac{3\bar{w}\left(r_2^3 - r_2^2 r_1\right)}{r_2^3 - r_1^3} \tag{4.8}$$

Table 4.1: Seasonal Variations in Updraft Strength and Altitude [39]

Parameters	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Units
mean w^*	3.74	4.86	5.38	6.46	8.30	7.81	8.83	8.01	7.38	5.87	4.30	4.13	ft/s
h_i for mean w^*	1,654	2,185	2,792	3,980	6,191	5,669	6,480	5,823	4,534	2,930	2,057	1,447	ft
max w^*	11.8	13.0	16.0	18.1	18.0	18.1	20.7	18.5	19.6	15.0	14.9	13.5	ft/s
h_i for max w^*	5,906	6,463	12,795	7,808	12,575	13,212	12,999	16,207	8,071	10,778	5,850	5,512	ft

Table 4.2: Shape Constants for a Bell-Shaped Vertical Velocity Distribution Developed Experimentally for use in SI System of Units [39]

r_1/r_2	k_1	k_2	k_3	k_4
0.14	1.5352	2.5826	-0.0113	0.0008
0.25	1.5265	3.6054	-0.0176	0.0005
0.36	1.4866	4.8354	-0.0320	0.0001
0.47	1.2042	7.7904	0.0848	0.0001
0.58	0.8816	13.972	0.3404	0.0001
0.69	0.7067	23.994	0.5689	0.0002
0.80	0.6189	42.797	0.7157	0.0001

The downdraft velocity, w_d was included to simulate a toroidal-like downward velocity profile that was said to be present on the outer edge of the updraft [39]:

$$w_l = \begin{cases} \frac{-\pi}{6} \sin\left(\frac{\pi r}{r_2}\right), & r_1 < r < 2r_2 \\ 0, & \text{otherwise.} \end{cases} \tag{4.9}$$

$$w_d = \begin{cases} 2.5 w_l \left(\frac{h}{h_i} - 0.5\right), & 0.5 < \frac{h}{h_i} < 0.9 \\ 0, & \text{otherwise.} \end{cases} \tag{4.10}$$

A plot showing constant contour lines of the thermal updraft speeds, W_h as a function of aircraft radial distance and altitude is shown in Figure 4.8, while the code used to generate the model is contained in Listing B.1. Additionally, thermal updraft speed profiles for multiple altitude slices are shown in Figure 4.9. It was observed that a discontinuity was present slightly above $h = 1000$ ft, which added a significant non-linearity to the gradients around this location. This thermal model was predominately used to generate the wind force contributions during trajectory optimization.

The equations of motion required an acceleration component to characterize the wind force. As such, a sensitivity analysis method was performed in order to evaluate the partial derivatives at any location within the thermal. These sensitivities were estimated using the *Complex Step* method. In this technique, the following expression was used to estimate a particular sensitivity for a given imaginary step size, $i\Delta_h$ [40]:

$$\nabla f(x) = \frac{\Im[f(x + i\Delta_h)]}{\Delta_h} \tag{4.11}$$

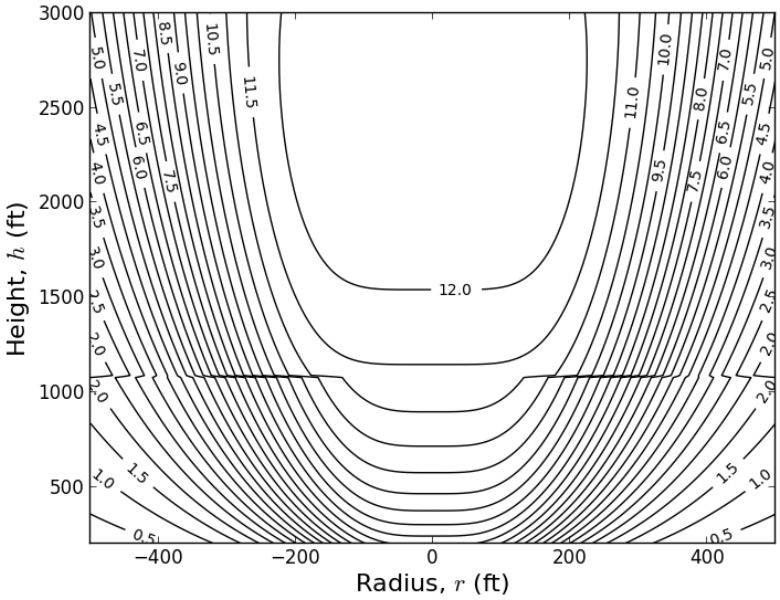


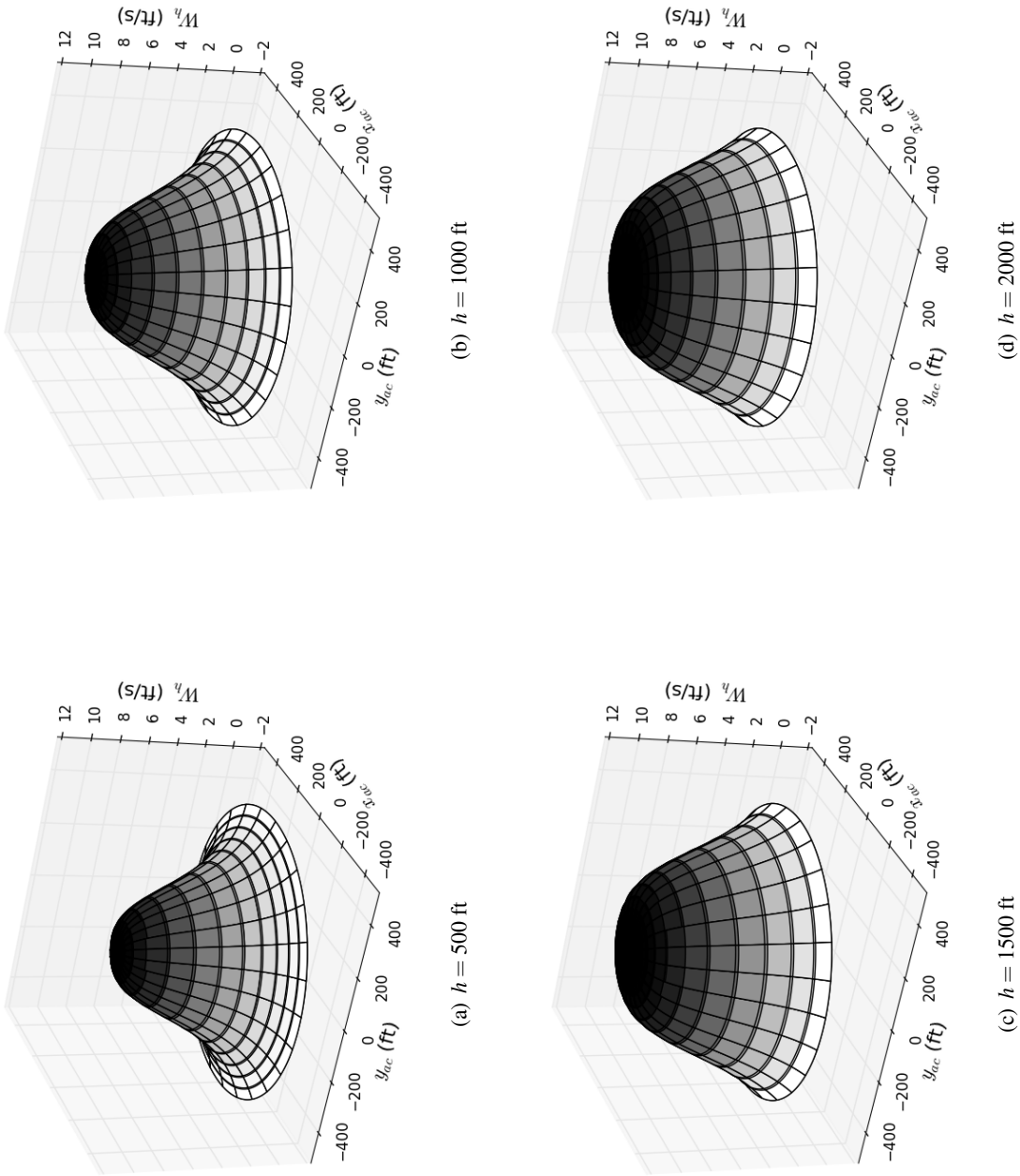
Figure 4.8: Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s for the Chimney Thermal Model

The *Complex Step* method provides sensitivity information with machine accuracy as it is not affected by step-size selection that causes subtractive cancellation errors [40].

4.4 Bubble Thermal Model

Lawrance [21] presented a toroidal thermal model, where the hot air mass rose up in a bubble-like structure and was disconnected from the ground. In this model, downdrafts were constrained to the outer thermal rim and the flow-field was 3-dimensional, continuous, but not mass conservative. The updraft profile for the bubble thermal model is given as [38]:

$$W_h = \begin{cases} w_{core}, & d_H = 0, \\ \frac{\cos\left(1+\frac{\pi h}{kR}\right)}{2} \frac{R w_{core}}{\pi d_H} \sin\left(\frac{\pi d_H}{R}\right), & d_H \in (0, 2R], \\ 0, & \text{otherwise.} \end{cases} \tag{4.12}$$



(a) $h = 500$ ft (b) $h = 1000$ ft

(c) $h = 1500$ ft (d) $h = 2000$ ft

Figure 4.9: Thermal Updraft Speed Profiles at Multiple Altitude Slices for Chimney Thermal Model

$$W_x = -W_h \frac{h}{(d_H - R)k^2} \frac{x}{d_H} \tag{4.13}$$

$$W_y = -W_h \frac{h}{(d_H - R)k^2} \frac{y}{d_H} \tag{4.14}$$

where $d_H = \sqrt{x^2 + y^2}$, w_{core} is the bubble core updraft speed, R is the radial distance that limits the updraft area, x, y, and z are the positions relative to the thermal center, and k is the bubble eccentricity factor that is a function of bubble height and thermal radius. A plot showing constant contour lines of the thermal updraft speeds, W_h as a function of aircraft radial distance and altitude is shown in Figure 4.10, while the code used to generate the model is contained in Listing B.2. Additionally, thermal updraft speed profiles for multiple altitude slices are shown in Figure 4.11. This thermal model was also used to generate the wind force contributions during trajectory optimization to permit a comparison of the trajectories for different thermal types.

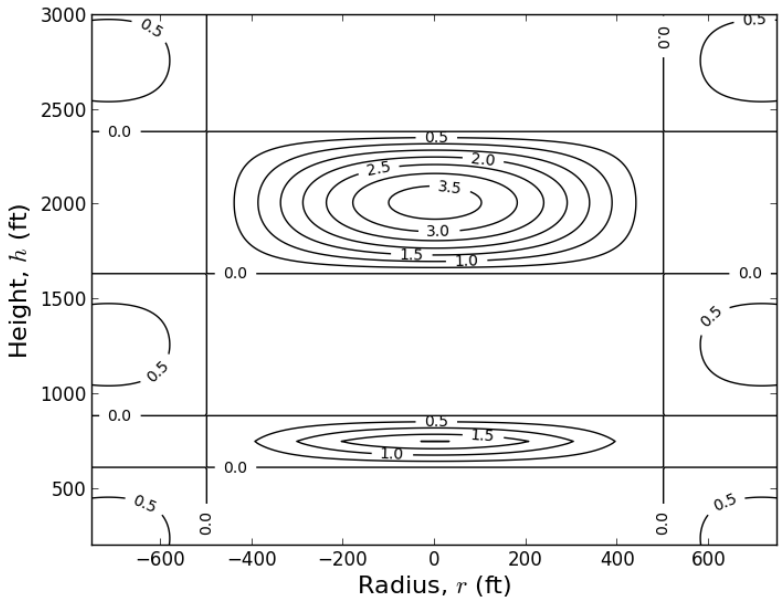
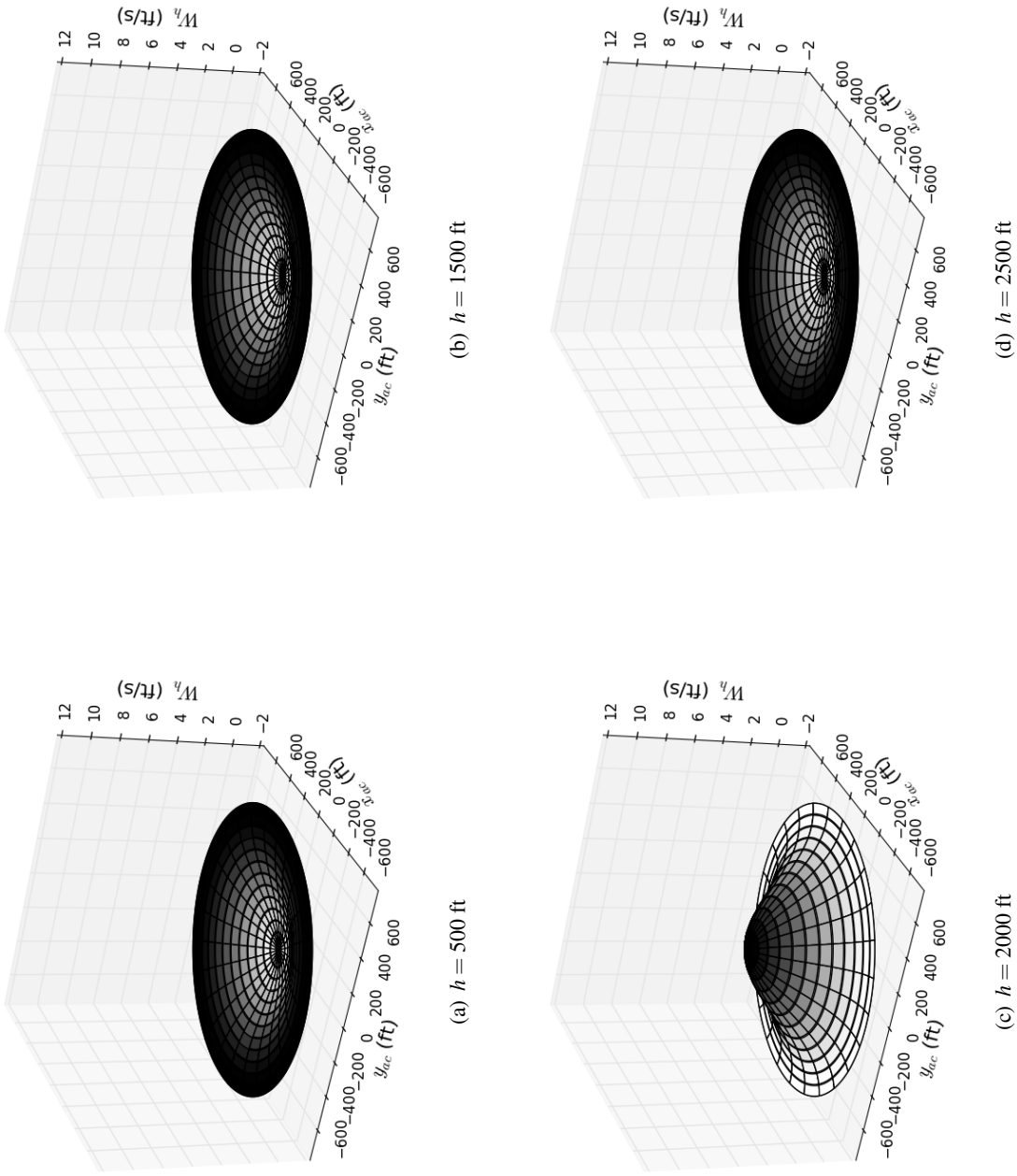


Figure 4.10: Plot of Constant Contour Lines of the Thermal Updraft Speeds, W_h in ft/s for the Lawrance Bubble Model

Unlike the previous thermal model, the trigonometric functions in the updraft estimation presents numerical difficulties in estimating the sensitivities using *Complex*



(a) $h = 500$ ft

(b) $h = 1500$ ft

(c) $h = 2000$ ft

(d) $h = 2500$ ft

Figure 4.11: Thermal Updraft Speed Profiles at Multiple Altitude Slices for Lawrance Bubble Model

Step. As such, the sensitivity analysis was performed using a *Central-Difference* approximation to evaluate the partial derivatives at any location for a given step size, Δ_h. The *Central-Difference* approximation can be expressed as [40]:

$$\nabla f(x) = \frac{f(x+\Delta_h) - f(x-\Delta_h)}{2\Delta_h} + \mathcal{O}(\Delta_h^2) \tag{4.15}$$

where $\mathcal{O}(\Delta_h^2)$ is the truncation error. This sensitivity approximation provided higher truncation error accuracy as compared to a *Forward Difference* method, but was at the expense of twice the number of function evaluations.

In summary, two state-of-the-art thermal models were developed to provide the wind force contributions and sensitivities that will be used in the point-mass model equations of motion for subsequent thermal soaring trajectory generation. For these models, it was assumed that the thermal core or center location is known, and wind shear effects were not considered.

5 Optimal Control Problem

In this chapter, the generalized optimal control problem is presented and is further developed specifically for the autonomous thermal soaring case. The trajectories are obtained by optimizing the aircraft's instantaneous or final energy height along the flight path and also at the terminal time. An energy approach is taken for the development of the cost or objective function used during optimization. The optimization problem is solved numerically using a direct collocation method using nonlinear programming, and this technique is presented herein. Additionally, the previously developed generalized aircraft equations of motion are restructured in cylindrical coordinates to improve the numerical performance during optimization. Finally, the full optimization problem definition is presented, including the dynamic and path constraints and the boundary conditions.

5.1 Generalized Optimal Control Problem

The generalized optimal control problem is defined as optimizing a continuous-time cost functional, J defined as [41, 42]:

$$J = \Phi[\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_f), t_f] + \int_{t_0}^{t_f} \mathscr{L}[\boldsymbol{x}(t), \boldsymbol{u}(t), t]\,\mathrm{d}t \tag{5.1}$$

where Φ are the boundary conditions at both t_0 and t_f, while $\mathscr{L}$ is the Lagrangian that defines the dynamic performance index. The above cost-functional is minimized with respect to the control inputs, $\boldsymbol{u}(t)$ to obtain an optimal path or trajectory subject to the dynamic constraints of the system that are defined by a set of Ordinary Differential Equations (ODEs) and represented in the following form as the state equations:

$$\dot{\boldsymbol{x}}(t) = f[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \tag{5.2}$$

where $\boldsymbol{x}(t) \in \Re^n$ are the state variables, $\boldsymbol{u}(t) \in \Re^m$ are the control inputs, and $t \in [t_0, t_f]$ is the time and is defined as the independent variable. Additionally, the path

constraints must also be satisfied along the trajectory and can either be represented as equality or inequality constraints such that:

$$\boldsymbol{C}[\boldsymbol{x}(t),\boldsymbol{u}(t),t] \leq 0 \tag{5.3}$$

5.2 Method to Solve the Optimal Control Problem

The techniques used to solve the optimal control problem are either classified as *indirect methods* or *direct methods*. The *indirect method* is an analytic method that uses the calculus of variations to determine the first order optimality conditions and are typically derived using the augmented Hamiltonian, $\mathscr{H}$ so as to transform the optimal control problem into an unconstrained optimization process and is defined as [41]:

$$\mathscr{H}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{u},t) = \mathscr{L} + \boldsymbol{\lambda}^T\boldsymbol{f} - \boldsymbol{\mu}^T\boldsymbol{C} \tag{5.4}$$

where $\boldsymbol{\lambda}(t) \in \Re^n$ is the *co-states* for the continuous differential equation constraints and $\boldsymbol{\mu}(t) \in \Re^C$ is the Lagrange multiplier associated with the path constraint. Given the complexity of the dynamics of thermal soaring flight, there are a large number of design variables and constraints, and as such the optimal control problem becomes extremely difficult to solve. Alternatively, for this research we use a direct collocation using nonlinear programming approach [41, 42], which is a *direct method* that relies on a numerical approach that involves discretizing the states and controls for a continuous trajectory. This discretizational approach transcribes the optimal control problem defined in Equation 5.1 into a nonlinear programming problem. This method is a state and control parameterization approach where a fixed time interval $[t_0, t_f]$ is divided into N sub-intervals or collocation points:

$$t_0 < t_1 < t_2 < \ldots < t_{k-1} < t_k < \ldots < t_N = t_f \tag{5.5}$$

In order to ensure continuity of each state across the sub-intervals, the following compatibility constraint must be enforced [41]:

$$\boldsymbol{x}(t_k^-) = \boldsymbol{x}(t_k^+),\ k = 2,3,\ldots,k-1 \tag{5.6}$$

Additionally, the derivatives of the state must also be matched to ensure continuity. At each collocation point, the dynamics are written as *defect constraints*, $\boldsymbol{\zeta}_k$ to ensure that the matching criteria are met and so that they can be solved simultaneously. A trapezoidal method will be used to estimate the solution to the ODEs for a given

interval size h_k, which is a second-order numerical approximation technique of the form [42]:

$$\boldsymbol{\zeta}_k = \boldsymbol{y}_{k+1} - \boldsymbol{y}_k - \frac{h_k}{2}\left[\boldsymbol{f}_{k+1} + \boldsymbol{f}_k\right] = 0 \tag{5.7}$$

For the collocation method, the objective is to find a trajectory that ensures that all of the defect constraints are zero. The optimal control problem can now be reformulated as the following non-linear programming problem:

$$\begin{aligned}
\text{minimize} \quad & \int_{t_0}^{t_k} \mathscr{L}[\boldsymbol{x}(t_k), \boldsymbol{u}(t_k), t_k]\,\mathrm{d}t \\
\text{with respect to} \quad & \boldsymbol{x_k},\ \boldsymbol{u_k},\ k = 1, 2, \ldots, N_{coll} \\
\text{subject to} \quad & \boldsymbol{\zeta}_k = \boldsymbol{y}_{k+1} - \boldsymbol{y}_k - \frac{h_k}{2}\left[\boldsymbol{f}_{k+1} + \boldsymbol{f}_k\right] = 0 \\
& \boldsymbol{C}[\boldsymbol{x}(t_k), \boldsymbol{u}(t_k), t_k] \leq 0 \\
& \boldsymbol{\Phi}[\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_N), t_N] = 0
\end{aligned} \tag{5.8}$$

A gradient-based method will be used to solve the nonlinear programming problem. The problem will be solved using *pyOpt*, which is an object-oriented framework developed for formulating and efficiently solving nonlinear constrained optimization problems [43]. One of the optimization algorithms that was integrated into the framework was *SNOPT*, which was a Sparse Nonlinear Optimizer that was particularly useful for solving large-scale constrained problems with smooth objective functions and constraints. The algorithm consisted of a Sequential Quadratic Programming (SQP) algorithm that used a smooth augmented Lagrangian merit function, and approximated the Hessian of the Lagrangian using a quasi-Newton method. It also made explicit provision for infeasibility in the original problem and in the quadratic programming sub-problems.

5.3 Optimal Control Problem Example

Consider the following optimal control problem, which is defined as the *Breakwell* problem and has an analytical solution for comparison and validation with the numerically generated results [44]. The problem is defined as minimize the cost functional:

$$J = \int_0^{t_f} u(t)^2\,\mathrm{d}t \tag{5.9}$$

subject to the dynamic constraints:

$$\begin{aligned}
\dot{x} &= v \\
\dot{v} &= u
\end{aligned} \tag{5.10}$$

the state dependent constraint:

$$x(t) \leq l \tag{5.11}$$

where $l = 0.1$, $t_f = 1$, and subject to the boundary conditions that are given as:

$$\begin{aligned} x(0) &= 0 \\ v(0) &= 1 \\ x(t_f) &= 0 \\ v(t_f) &= -1 \end{aligned} \tag{5.12}$$

The analytical solution of the problem, which is only valid for $0 \leq l \leq 1/6$ was given as [44]:

$$u(t) = \begin{cases} -\frac{2}{3l}\left(1 - \frac{t}{3l}\right), & 0 \leq t < 3l \\ 0, & 3l \leq t < 1 - 3l \\ -\frac{2}{3l}\left(1 - \frac{1-t}{3l}\right), & 1 - 3l \leq t \leq 1 \end{cases} \tag{5.13}$$

$$x(t) = \begin{cases} l\left(1 - \left(1 - \frac{t}{3l}\right)^3\right), & 0 \leq t < 3l \\ l, & 3l \leq t < 1 - 3l \\ l\left(1 - \left(1 - \frac{1-t}{3l}\right)^3\right), & 1 - 3l \leq t \leq 1 \end{cases} \tag{5.14}$$

$$v(t) = \begin{cases} \left(1 - \frac{t}{3l}\right)^2, & 0 \leq t < 3l \\ 0, & 3l \leq t < 1 - 3l \\ -\left(1 - \frac{1-t}{3l}\right)^2, & 1 - 3l \leq t \leq 1 \end{cases} \tag{5.15}$$

$$\lambda_x(t) = \begin{cases} \frac{2}{9l^2}, & 0 \leq t < 3l \\ 0, & 3l \leq t < 1 - 3l \\ -\frac{2}{9l^2}, & 1 - 3l \leq t \leq 1 \end{cases} \tag{5.16}$$

$$\lambda_v(t) = \begin{cases} \frac{2}{3l}\left(1 - \frac{t}{3l}\right), & 0 \leq t < 3l \\ 0, & 3l \leq t < 1 - 3l \\ -\frac{2}{3l}\left(1 - \frac{1-t}{3l}\right), & 1 - 3l \leq t \leq 1 \end{cases} \tag{5.17}$$

where $\lambda_x(t)$ and $\lambda_v(t)$ are the co-states. The optimal value of the objective function was given as $J = 4/(9l) = 4.44\bar{4}$. The optimal control problem was approached using the direct collocation method using nonlinear programming and was solved for $N = 100$ collocation points using *pyOpt*. The numerical results shown in Figures 5.2 and 5.4 demonstrate the capability of the method to find a solution with a high degree of accuracy as compared to the analytic solution shown graphically in Figures 5.1 and 5.3.

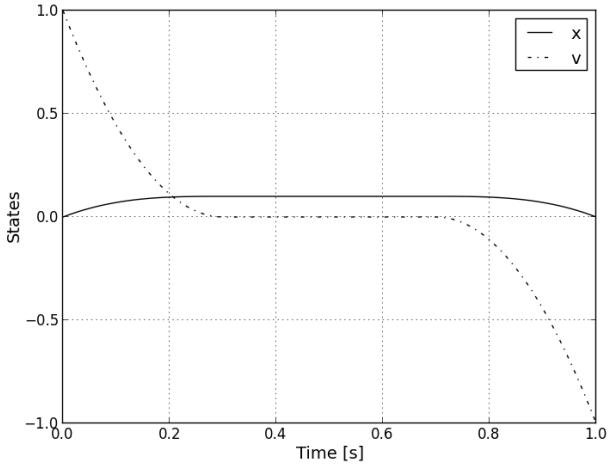


Figure 5.1: Plot of the States for the Analytic Solution to the Breakwell Problem

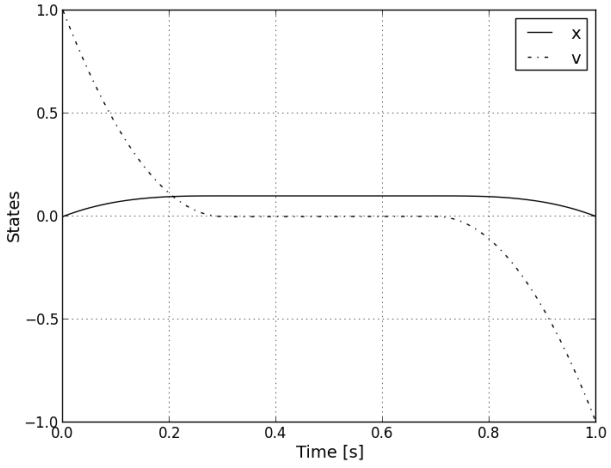


Figure 5.2: Plot of the States for the Numerical Solution to the Breakwell Problem

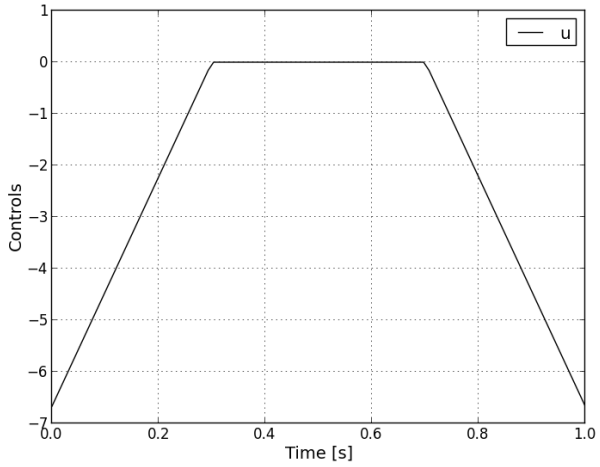


Figure 5.3: Plot of the Controls for the Analytic Solution to the Breakwell Problem

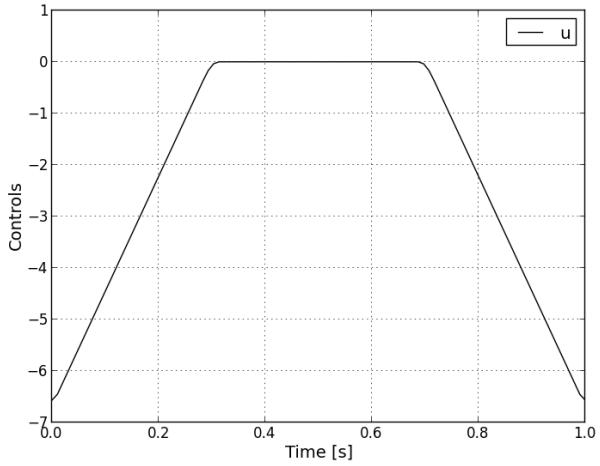


Figure 5.4: Plot of the Controls for the Numerical Solution to the Breakwell Problem

5.4 Objective Function Development

The objective of thermal soaring flight is to extract as much energy as possible out of a given thermal updraft in order to increase the potential energy state of the aircraft and as a result gain altitude. As such, an energy approach is taken to develop an objective function for use in determining optimal thermal soaring trajectories. The total energy state of an aircraft can be represented by the sum of its potential and kinetic energy, which take the form of altitude, h and airspeed, V respectively [30]:

$$E = PE + KE = mgh + \frac{1}{2}mV^2 \tag{5.18}$$

Given that the energy loss due to control actuation is significantly less than the energy that is required to overcome aerodynamic and gravitational forces associated with gliding flight, it is assumed that the internally stored energy of the system is neglected and therefore is not considered in the objective function formulation. The energy height is defined as the altitude an aircraft could attain if it converted all of its kinetic energy into potential energy assuming that no losses would occur. More specifically, it can be represented as a normalized quantity with respect to the aircraft weight, mg and is defined as the Energy Height, E_h [30]:

$$E_h = \frac{E}{mg} = h + \frac{V^2}{2g} \tag{5.19}$$

The primary objective of thermal soaring is to maximize thermal energy extraction, which is directly related to achieving the highest aircraft energy height. As such, the optimization problem can be formulated as maximizing the energy height of an aircraft at the end of a fixed time period, $t_N = t_f$ and with respect to the aircraft's angle of attack α, and bank angle γ_1:

$$\begin{aligned}
\text{maximize} \quad & E_h(t_f) = h + \frac{V^2}{2g} \\
\text{with respect to} \quad & \boldsymbol{x_k},\ \boldsymbol{u_k} = [\boldsymbol{\alpha}, \boldsymbol{\gamma_1}]^T,\ k = 1, 2, \ldots, N_{coll} \\
\text{subject to} \quad & \boldsymbol{\zeta}_k = \boldsymbol{y}_{k+1} - \boldsymbol{y}_k - \frac{h_k}{2}\left[\boldsymbol{f}_{k+1} + \boldsymbol{f}_k\right] = 0 \\
& \boldsymbol{C}[\boldsymbol{x}(t_k), \boldsymbol{u}(t_k), t_k] \leq 0 \\
& \boldsymbol{\Phi}[\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_N), t_N] = 0
\end{aligned} \tag{5.20}$$

Alternatively, the following formulation represents an alternative optimization problem where the instantaneous energy height is maximized throughout the complete flight trajectory:

$$\begin{aligned} \text{minimize} \quad & \int_{t_0}^{t_f} E_h \, \mathrm{d}t = \int_{t_0}^{t_f} h + \frac{V^2}{2g} \, \mathrm{d}t \\ \text{with respect to} \quad & \boldsymbol{x_k},\ \boldsymbol{u_k} = [\boldsymbol{\alpha}, \boldsymbol{\gamma_1}]^T,\ k = 1, 2, \ldots, N_{coll} \\ \text{subject to} \quad & \boldsymbol{\zeta}_k = \boldsymbol{y}_{k+1} - \boldsymbol{y}_k - \frac{h_k}{2} \left[\boldsymbol{f}_{k+1} + \boldsymbol{f}_k \right] = 0 \\ & \boldsymbol{C}[\boldsymbol{x}(t_k), \boldsymbol{u}(t_k), t_k] \leq 0 \\ & \boldsymbol{\Phi}[\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_N), t_N] = 0 \end{aligned} \tag{5.21}$$

5.5 System Dynamics

In order to evaluate and optimize the continual change of the aircraft's energy state within a trajectory, it is necessary to establish the full system dynamics to be used in the optimization problem. Furthermore, the development of the system dynamics will identify both the state and control vectors. To fully characterize the system, both the aircraft point-mass model equations of motion previously developed in Chapter 3 and the kinematic expressions must be considered.

5.5.1 Kinematic Equations of Motion

The kinematic equations of motion related the motion of the aircraft with respect to the Earth axes. The kinematic equations of motion augmented with wind component velocities are defined in the Earth axes using the following rotational transformation:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_e = \begin{bmatrix} \cos\gamma_2 \cos\gamma_3 & -\sin\gamma_3 & \sin\gamma_2 \cos\gamma_3 \\ \cos\gamma_2 \sin\gamma_3 & \cos\gamma_3 & \sin\gamma_2 \sin\gamma_3 \\ -\sin\gamma_2 & 0 & \cos\gamma_2 \end{bmatrix} \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}_v + \begin{bmatrix} V_{w,x} \\ V_{w,y} \\ V_{w,z} \end{bmatrix}_e \tag{5.22}$$

Performing the rotations results in the following kinematic equations of motion in Cartesian coordinates:

$$\dot{x} = [V]_v \cos\gamma_2 \cos\gamma_3 + [V_{w,x}]_e \tag{5.23}$$

$$\dot{y} = [V]_v \cos\gamma_2 \sin\gamma_3 + [V_{w,y}]_e \tag{5.24}$$

$$\dot{z} = -[V]_v \sin\gamma_2 + [V_{w,z}]_e \tag{5.25}$$

5.5.2 Equations of Motion Restructured in Cylindrical Coordinates

Previous heuristic and analytic strategies have identified that the most efficient and effective methods to extract energy from a thermal involves centering and circling around the thermal core [2,4,9–11,13]. As such, these trajectories are cyclic in nature and can cause computational issues when trying to solve them numerically using the standard equations of motion in Cartesian coordinates leading to the generation of non-converged sub-optimal results that do not reflect a realistic flight trajectory. As such, the kinematic equations of motion were restructured in cylindrical coordinates as shown in Figure 5.5 and represented by the following coordinate transformation:

$$x = r\cos\theta_{th} \tag{5.26}$$

$$y = r\sin\theta_{th} \tag{5.27}$$

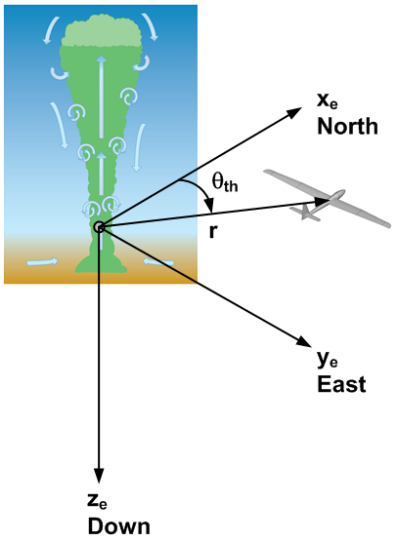


Figure 5.5: Cylindrical Coordinates Definition

It is assumed that the location of the thermal core on the Earth's surface corresponds to the origin of the cylindrical coordinate axes.

Change in Aircraft Radial Position with Time

Taking the time derivative of x and y and equating them to the corresponding kinematic expressions derived in Cartesian coordinates yields:

$$\dot{x} = \dot{r}\cos\theta_{th} - r\sin\theta_{th}\dot{\theta}_{th} = [V]_v\cos\gamma_2\cos\gamma_3 + [V_{w,x}]_e \quad (5.28)$$

$$\dot{y} = \dot{r}\sin\theta_{th} + r\cos\theta_{th}\dot{\theta}_{th} = [V]_v\cos\gamma_2\sin\gamma_3 + [V_{w,y}]_e \quad (5.29)$$

Solving and equating each expression for the aircraft's radial location, r:

$$r = \frac{-[V]_v\cos\gamma_2\cos\gamma_3 - [V_{w,x}]_e + \dot{r}\cos\theta_{th}}{\sin\theta_{th}\dot{\theta}_{th}} = \frac{[V]_v\cos\gamma_2\sin\gamma_3 + [V_{w,y}]_e - \dot{r}\sin\theta_{th}}{\cos\theta_{th}\dot{\theta}_{th}}$$

For the above expression, it is only true for the conditions when $\dot{\theta}_{th} \neq 0$. When $\dot{\theta}_{th} = 0$, the flight condition is defined as the aircraft tracking directly towards or away from the center of the thermal core. As such, this formulation assumes that the thermal has been detected, the soaring mode has been engaged, the aircraft is in turning flight and actively engaged in centering the thermal core. Expanding the above result and solving the expression for $\dot{r}$ yields:

$$\dot{r}\left(\sin^2\theta_{th} + \cos^2\theta_{th}\right) = [V]_v\cos\gamma_2\cos\gamma_3\cos\theta_{th} + [V]_v\cos\gamma_2\sin\gamma_3\sin\theta_{th} + [V_{w,x}]_e\cos\theta_{th} + [V_{w,y}]_e\sin\theta_{th}$$

Using the following trigonometric identities:

$$\sin^2\theta_{th} + \cos^2\theta_{th} = 1$$

$$\cos(\gamma_3 - \theta_{th}) = \cos\gamma_3\cos\theta_{th} + \sin\gamma_3\sin\theta_{th}$$

the expression for $\dot{r}$ reduces to:

$$\dot{r} = [V]_v\cos\gamma_2\cos(\gamma_3 - \theta_{th}) + [V_{w,x}]_e\cos\theta_{th} + [V_{w,y}]_e\sin\theta_{th} \quad (5.30)$$

Using the coordinate transformations to obtain a radial component for the wind speed:

$$[V_{w,x}]_e = [V_{w,r}]_e\cos\theta_{th}$$
$$[V_{w,y}]_e = [V_{w,r}]_e\sin\theta_{th}$$

a concise form of the expression for $\dot{r}$ is obtained:

$$\dot{r} = [V]_v\cos\gamma_2\cos(\gamma_3 - \theta_{th}) + [V_{w,r}]_e \quad (5.31)$$

Change in Aircraft Angular Position with Time

Recall the time derivative expression obtained above for x and y:

$$\begin{aligned} \dot{x} &= \dot{r}\cos\theta_{th} - r\sin\theta_{th}\dot{\theta}_{th} \\ \dot{y} &= \dot{r}\sin\theta_{th} + r\cos\theta_{th}\dot{\theta}_{th} \end{aligned}$$

Multiplying the time derivative of x by $\sin\theta_{th}$, and the time derivative of y by $\cos\theta_{th}$ results in:

$$\dot{x}\sin\theta_{th} = \dot{r}\sin\theta_{th}\cos\theta_{th} - r\sin^2\theta_{th}\dot{\theta}_{th} \tag{5.32}$$

$$\dot{y}\cos\theta_{th} = \dot{r}\sin\theta_{th}\cos\theta_{th} + r\cos^2\theta_{th}\dot{\theta}_{th} \tag{5.33}$$

Subtracting the resulting time derivative of x by the time derivative of y yields:

$$\dot{y}\cos\theta_{th} - \dot{x}\sin\theta_{th} = r\cos^2\theta_{th}\dot{\theta}_{th} + r\sin^2\theta_{th}\dot{\theta}_{th} = r\dot{\theta}_{th}$$

Solving for $\dot{\theta}_{th}$:

$$\dot{\theta}_{th} = \frac{\dot{y}\cos\theta_{th} - \dot{x}\sin\theta_{th}}{r} \tag{5.34}$$

Replacing $\dot{x}$ and $\dot{y}$ with the corresponding kinematic expressions derived in Cartesian coordinates, the expression for $\dot{\theta}_{th}$ is expanded to:

$$\dot{\theta}_{th} = \frac{[V]_v\cos\gamma_2\sin\gamma_3\cos\theta_{th} + [V_{w,y}]_e\cos\theta_{th} - [V]_v\cos\gamma_2\cos\gamma_3\cos\theta_{th} - [V_{w,x}]_e\cos\theta_{th}}{r}$$

Using the following trigonometric identity:

$$\sin(\gamma_3 - \theta_{th}) = \sin\gamma_3\cos\theta_{th} - \cos\gamma_3\sin\theta_{th}$$

and the previously defined coordinate transformations to obtain a radial component for the wind speed, a concise form of the expression for $\dot{\theta}_{th}$ is obtained:

$$\dot{\theta}_{th} = \frac{[V]_v}{r}\cos\gamma_2\sin(\gamma_3 - \theta_{th}) \tag{5.35}$$

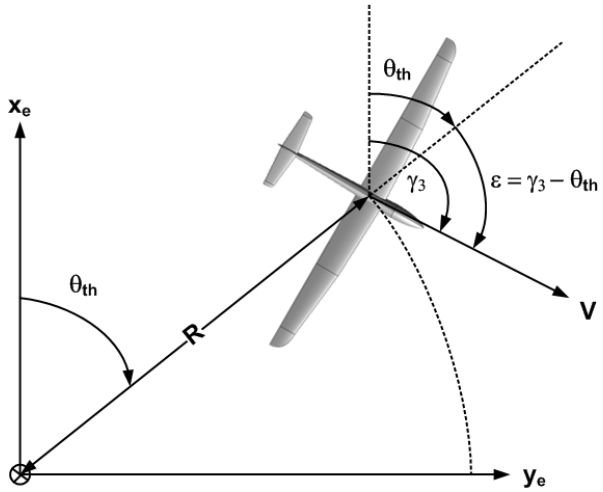


Figure 5.6: Aircraft Tangential Tracking Angle Definition

Change in Aircraft Tangential Tracking Angle with Time

A new variable, ε is introduced that represents the aircraft tangential tracking angle and is shown in Figure 5.6 and is defined as:

$$\varepsilon = \gamma_3 - \theta_{th} \tag{5.36}$$

It follows that the time derivative of the aircraft tangential tracking angle is given as:

$$\dot{\varepsilon} = \dot{\gamma}_3 - \dot{\theta}_{th} \tag{5.37}$$

Recall the expression for the rate of change in aircraft track, $\dot{\gamma}_3$ that was previously derived in Chapter 3:

$$\dot{\gamma}_3 = \frac{L\sin\gamma_1 - m[\dot{V}_{w,x}]_e \sin\gamma_3 + m[\dot{V}_{w,y}]_e \cos\gamma_3}{m[V]_v \cos\gamma_2} \tag{5.38}$$

Substituting the expressions for $\dot{\gamma}_3$ and $\dot{\theta}_{th}$ results in the following equation for the change in aircraft tangential tracking angle with time:

$$\dot{\varepsilon} = \frac{L\sin\gamma_1 - m[\dot{V}_{w,x}]_e \sin\gamma_3 + m[\dot{V}_{w,y}]_e \cos\gamma_3}{m[V]_v \cos\gamma_2} - \frac{[V]_v}{r}\cos\gamma_2 \sin(\gamma_3 - \theta_{th}) \tag{5.39}$$

5.6 Optimization Problem Definition

The autonomous soaring optimal control problem is defined as being a multi-variable constrained optimization problem and is defined as:

$$\begin{aligned} &\text{maximize} \;\; E_h = h + \frac{V^2}{2g} \\ &\text{with respect to} \;\; \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \;\; \boldsymbol{u} = (\alpha, \gamma_1)^T \end{aligned}$$

subject to the defect constraints, ζ_k that are defined by the following set of ODEs:

$$\dot{V} = -\frac{D}{m} - g\sin\gamma_2 + [\dot{V}_{w,x}]_e \cos\gamma_2 \cos\gamma_3 + [\dot{V}_{w,y}]_e \cos\gamma_2 \sin\gamma_3 - [\dot{V}_{w,z}]_e \sin\gamma_2 \tag{5.40}$$

$$\dot{\gamma_2} = \frac{L\cos\gamma_1 - mg\cos\gamma_2 + m[\dot{V}_{w,x}]_e \sin\gamma_2 \cos\gamma_3 + m[\dot{V}_{w,y}]_e \sin\gamma_2 \sin\gamma_3 + m[\dot{V}_{w,z}]_e \cos\gamma_2}{m[V]_v} \tag{5.41}$$

$$\dot{r} = [V]_v \cos\gamma_2 \cos(\gamma_3 - \theta_{th}) + [V_{w,r}]_e \tag{5.42}$$

$$\dot{\theta}_{th} = \frac{[V]_v}{r} \cos\gamma_2 \sin(\gamma_3 - \theta_{th}) \tag{5.43}$$

$$\dot{z} = -[V]_v \sin\gamma_2 + [V_{w,z}]_e \tag{5.44}$$

$$\dot{\varepsilon} = \frac{L\sin\gamma_1 - m[\dot{V}_{w,x}]_e \sin\gamma_3 + m[\dot{V}_{w,y}]_e \cos\gamma_3}{m[V]_v \cos\gamma_2} - \frac{[V]_v}{r} \cos\gamma_2 \sin(\gamma_3 - \theta_{th}) \tag{5.45}$$

The initial flight conditions are defined as:

$$\left.\begin{array}{lll} V(t_0) = V_0, & r(t_0) = r_0, & \gamma_2(t_0) = \gamma_{20} \\ \theta_{th}(t_0) = \theta_{th0}, & z(t_0) = z_0, & \varepsilon(t_0) = \varepsilon_0 \end{array}\right\} \tag{5.46}$$

The path constraints, $\boldsymbol{C}$ are represented as inequalities and are shown in Table 5.1, while the boundary conditions, $\boldsymbol{\Phi}$ for the states and controls are detailed in Table 5.2. The constraints and boundary conditions represent either aerodynamic, structural, or control limitations of the Cularis aircraft that were either outlined in the flight manual or were generated in Chapter 3 during the characterization.

Table 5.1: Definition of the Path Constraints

Minimum Control Speed, V_{mc}	$g_1(\boldsymbol{x}) \geq V_{mc} = 1.1V_{stall}$
Maximum Operating Speed, V_{mo}	$g_2(\boldsymbol{x}) \leq V_{mo}$
Minimum Load Factor, n_{min}	$g_3(\boldsymbol{x}) \geq n_{min} = -1.5$
Maximum Load Factor, n_{max}	$g_4(\boldsymbol{x}) \leq n_{max} = 4.5$
Maximum Lift Coefficient, C_{Lmax}	$g_5(\boldsymbol{x}) \leq C_{Lmax} = 1.674$
Maximum Pitch Rate, $\dot{\gamma}_{2max}$	$g_6(\boldsymbol{x}) \leq \dot{\gamma}_{2max} = \frac{\pi}{12}$
Maximum Roll Rate, $\dot{\gamma}_{1max}$	$g_7(\boldsymbol{x}) \leq \dot{\gamma}_{1max} = \frac{\pi}{6}$

Table 5.2: Definition of the Boundary Conditions for the States and Controls

Lower Bound	State or Control Variable	Upper Bound
V_{stall}	V	V_{mo}
$\frac{-\pi}{2}$	γ_2	$\frac{\pi}{2}$
25 ft	r	1000 ft
0.0 rad	θ_{th}	∞ rad
50 ft	z	9000 ft
0.2 rad	ε	1.8 rad
$\frac{-\pi}{10}$	α	$\frac{\pi}{10}$
$\frac{-\pi}{3}$	γ_1	$\frac{\pi}{3}$

Using the point-mass model of the aircraft that was described in Chapter 3, the thermal models that were presented in Chapter 4, and the optimal control problem formulation detailed above, optimal trajectories can be generated for various case studies and will be presented in the succeeding chapter.

6 Thermal Soaring Case Studies

In this chapter, optimal flight trajectories are presented for a small autonomous aircraft, operating in multiple thermal profiles of varying strengths using the optimal control problem formulation developed in Chapter 5, the point-mass model of the Cularis D-5223 aircraft that was described in Chapter 3, and the thermal models that were presented in Chapter 4. These trajectories are compared to derived analytic expressions for gliding flight, and a thermal centering strategy for controller development is presented.

6.1 Case Studies Definition

Optimal soaring trajectories are generated and evaluated by varying the following five objectives within the problem formulation:

i. Variations of the objective function (Instantaneous Energy or Energy at t_f).
ii. Thermal type.
iii. Thermal strength.
iv. Initial starting conditions.
v. Effect of the allowable aircraft tangential tracking angle, ε range.

Trajectories are first shown for the chimney thermal model described in Section 4.3, and then followed by the bubble thermal model detailed in Section 4.4 for the thermal strength conditions outlined in Table 6.1. For each simulation case, the aircraft is positioned to fly tangential to the thermal core at the initial flight conditions detailed in Table 6.2 and the various starting positions defined in Table 6.3. Apart from the analysis of the trajectories at different starting conditions, all simulations are generated for start position 4 as outlined in Table 6.2. Results are presented for $N = 200$ collocation points and for a fixed time interval of $t = 120$ s. The specified collocation leads to an optimization problem with 1,600 design variables (6 states and 2 controls for $N = 200$ collocation points) and 2,600 constraints, and as such 4,160,00 constraint gradients and 1,600 objective gradients are required. The feasibility and optimality tolerance levels defining the convergence criteria for all cases was selected

as $\upsilon = 1.0e^{-5}$. To improve computational efficiency, the optimal trajectories were solved using a *Beowulf* cluster using either 48 or 60 processors to enable parallel gradient calculations. The computer code used to generate the optimal trajectories for both the chimney and bubble thermal models are contained in Listings B.3 and B.4 respectively.

Table 6.1: Definition of the Thermal Strength Cases

Chimney Thermal Model		
Month Index	w^*_{max} [ft/s]	h_i [ft]
0	11.8	5,906
3	18.1	7,808
6	20.7	12,999
9	15.0	10,778
Bubble Thermal Model		
w_{core} [ft/s]	Bubble Height, h_t [ft]	Updraft Area Limit, R [ft]
7.5	1500	500
10.0	2000	750
12.5	2500	1000

Table 6.2: Initial Aircraft Flight Conditions

Description	Parameter	Value	Units
Airspeed	V	$V_{L/D_{max}} = 29.5$	ft/s
Climb Angle	γ_2	$(\gamma_2)_{min} = -0.043$	rad
Angle of Attack	α	0.0	rad
Bank Angle	γ_1	0.0	rad
Tangential Tracking Angle	ε	$\pi/2$	rad

6.2 Optimal Trajectories for the Chimney Thermal Model

6.2.1 Effect of Thermal Strength and Initial Starting Position

Using the chimney thermal model described in Section 4.3, optimal trajectories for varying thermal strengths were generated and are shown in Figure 6.1. Time histories of the states and controls for each trajectory are provided in Figures 6.2 and 6.3 respectively. Following the initial turn towards and the capture of the thermal, the trajectories were smooth and both the states and controls achieved steady-state conditions and thermal centering occurred within 20 seconds.

(a) Month Index = 0, January

(b) Month Index = 3, April

(c) Month Index = 6, July

(d) Month Index = 9, October

Figure 6.1: Optimal Trajectories for Varying Thermal Strengths Defined by the Month Index

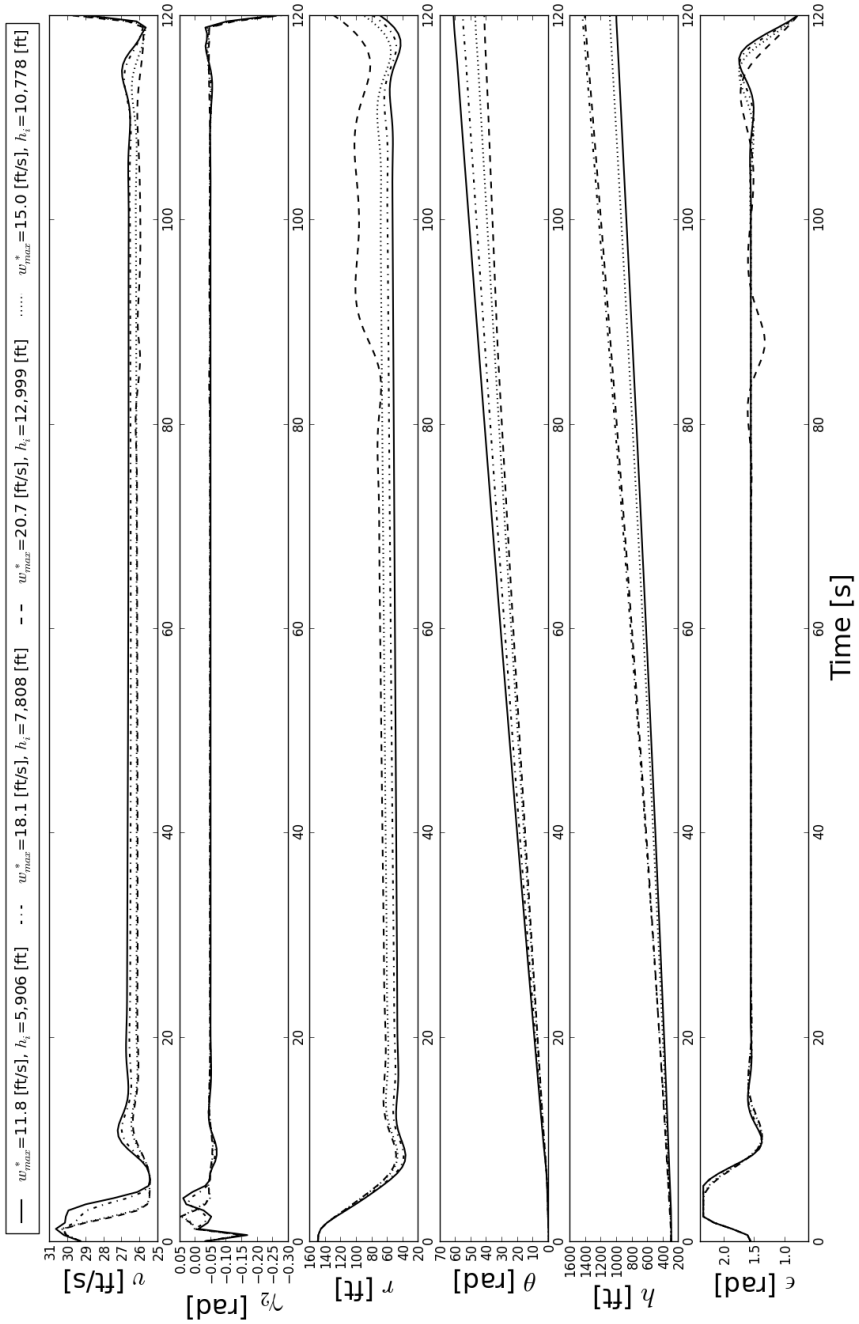


Figure 6.2: Time History of the States for Varying Thermal Strengths

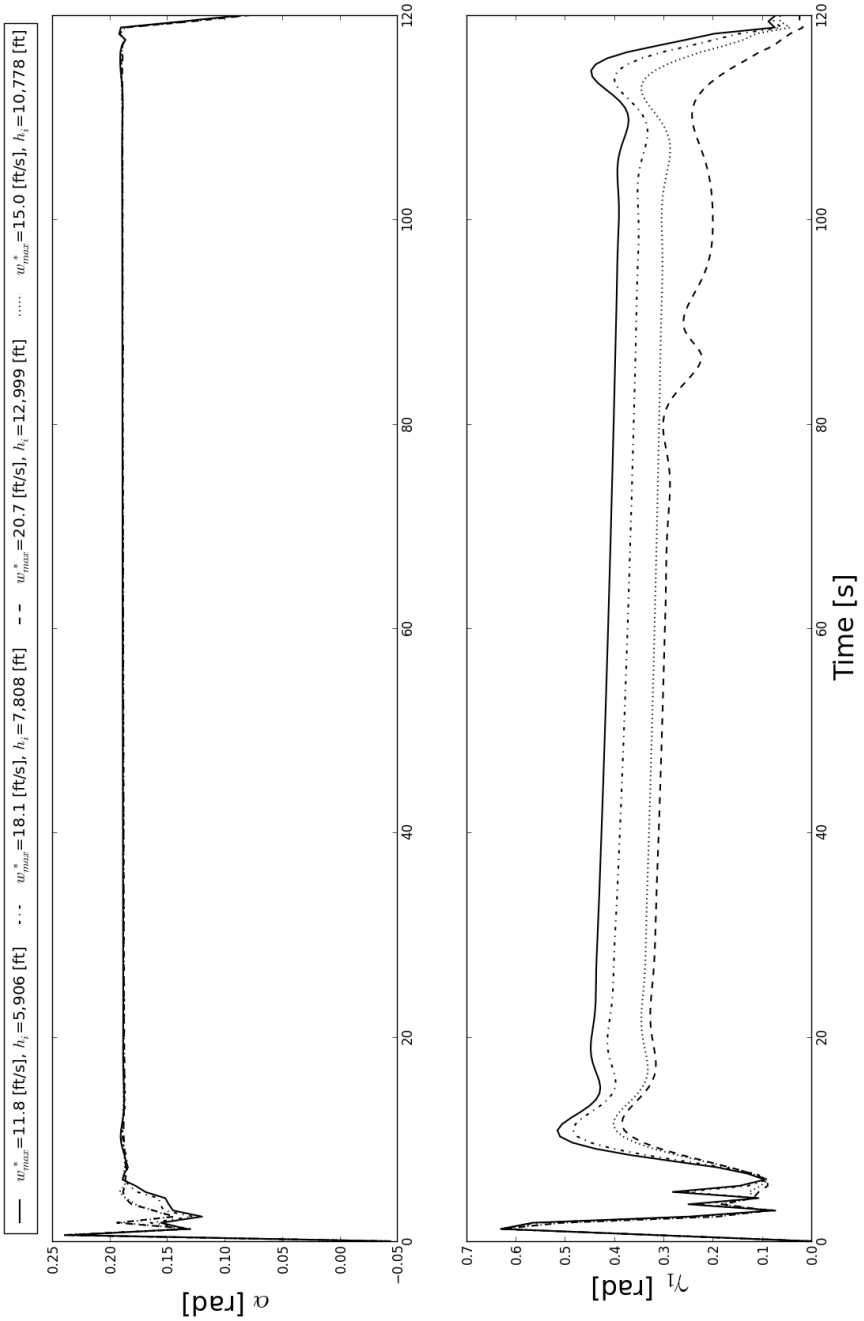


Figure 6.3: Time History of the Control Inputs for Varying Thermal Strengths

Table 6.3: Initial Aircraft Starting Location

Location	Radial Position, r [ft]	Angular Position, θ_{th} [rad]	Altitude, h [ft]
1	100	0.0	300
2	100	0.0	500
3	100	0.0	700
4	150	0.0	300
5	150	0.0	500
6	150	0.0	700
7	300	0.0	300
8	300	0.0	500
9	300	0.0	700

Following a slight acceleration that was initiated to quickly enter the thermal to avoid energy losses in the downdraft, the airspeed was reduced and remained almost constant throughout the climb. It was observed that the commanded airspeed was minimally effected by the thermal strength, and was driven by the flight altitude and the aerodynamic characteristics to obtain a minimum sink rate condition. Additionally, the flight path angle was also constant and was found to be independent of thermal strength upon thermal centering. Once centered, the aircraft settled at a value of $\gamma_{2_{min}} = -0.049$ rad $[-2.8^\circ]$, which was a slightly steeper trajectory then when flying at the minimum glide angle of $\gamma_2 = -0.043$ rad $[-2.5^\circ]$. A high angle of attack was generated in an attempt to reach the $(C_L^{3/2}/C_D)$ aerodynamic condition required for the minimum sink rate; however, due to the stall speed being greater than the minimum sink airspeed, an angle of attack of $\alpha \approx 0.18$ rad $[10.5^\circ]$ was selected to obtain the minimum control speed. A comparison of the optimization results to the analytic predictions for airspeed at varying thermal strengths is shown in Figures 6.4, 6.5, 6.6, and 6.7. Following the initial transient conditions, the airspeed was reduced to the minimum control speed, V_{mc} and flew at that speed throughout the climb phase of the trajectory. Given that the minimum sink rate speed, $V_{(C_L^{3/2}/C_D)_{max}}$ was less than the stall speed, it was not possible for the aircraft to fly at that speed due to the constraint of the minimum control speed.

For lower thermal strength conditions, the aircraft performed tighter turns to take more advantage of the higher updraft strengths in the thermal core and made more complete circular passes around the core. As such, angle of bank command was a function of thermal strength. Once centered, an angle of bank of no more than $\gamma_1 = 5\pi/36$ $[25^\circ]$ was commanded, and it was decreased linearly at a rate of $4.36e^{-4}$ rad/s $[0.025^\circ/\text{s}]$ during the climb. Following the initial capture, the aircraft flew a circular flight path that was tangential to the thermal core, i.e. $\varepsilon = \pi/2$ $[90^\circ]$.

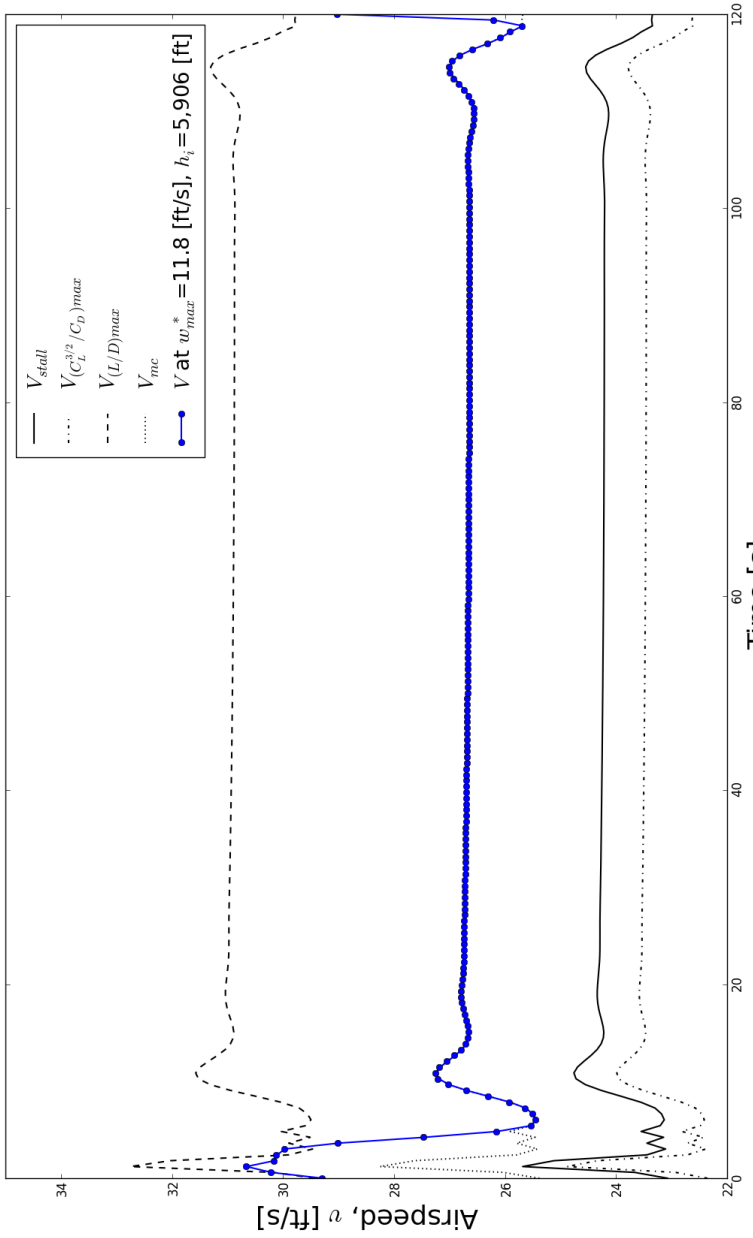


Figure 6.4: Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 0, January]

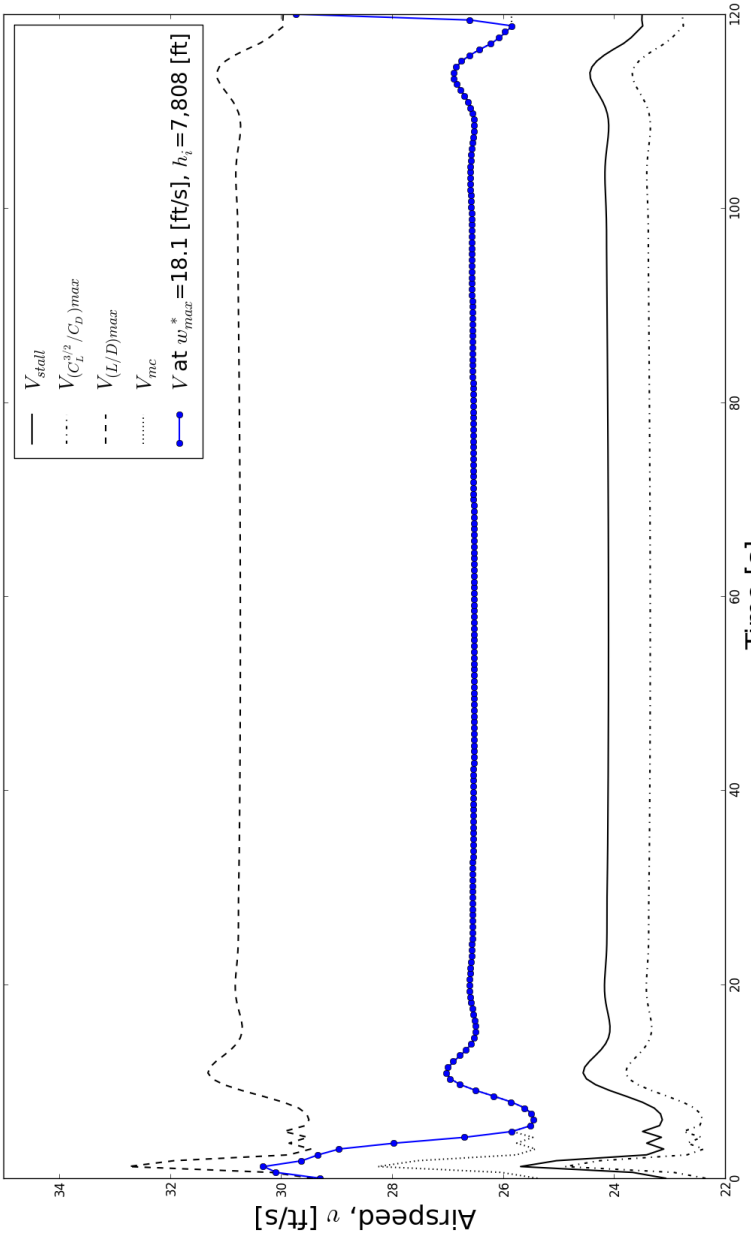


Figure 6.5: Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 3, April]

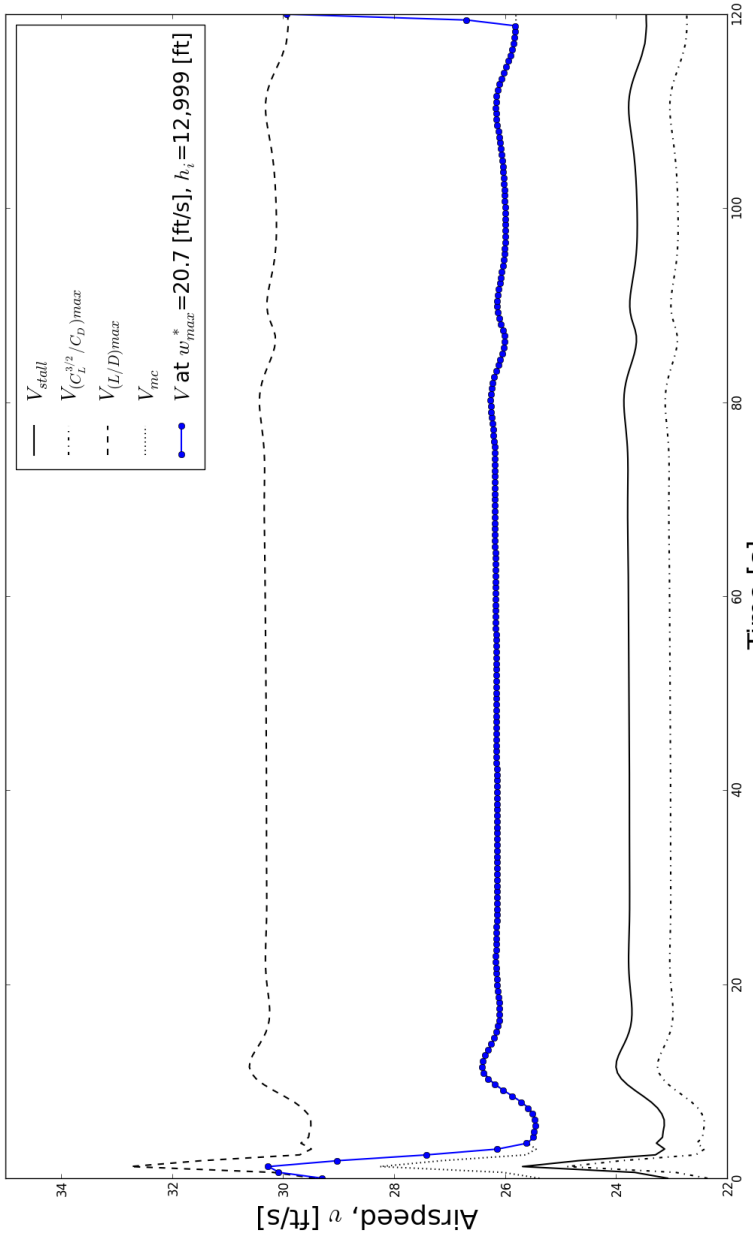


Figure 6.6: Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 6, July]

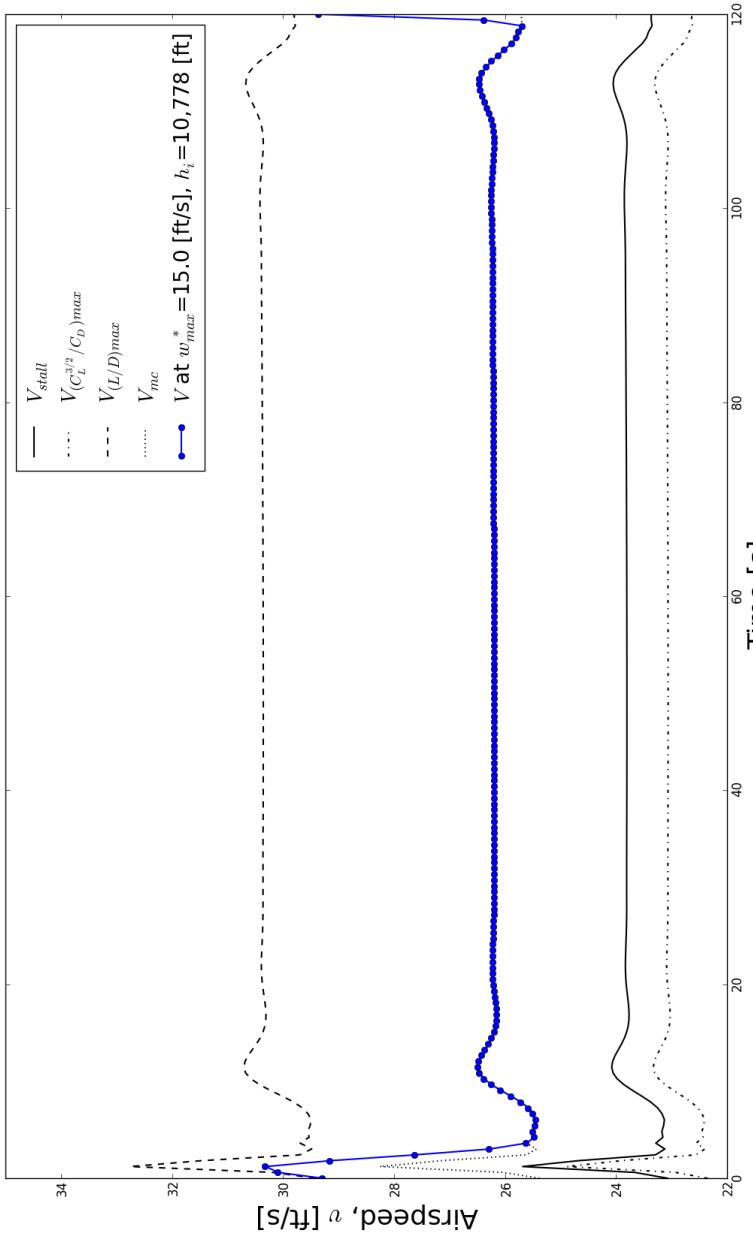


Figure 6.7: Airspeed Comparison to Analytical Gliding Flight Velocities [Month Index = 9, October]

The resulting objective function value, or maximized energy height, the computational effort required to solve the trajectories, and the overall optimization efficiency are presented in Table 6.4. The increase in energy height due to thermal soaring for varying thermal strengths ranged from 721.8 to 1158.5 ft. Due to the large scale of the optimization problem, it took an average of 5.40 hrs and 1715 function evaluations to solve for the optimal trajectory.

Three dimensional plots of the optimal trajectories for varying initial starting locations are shown in Figure 6.8. The thermal strength was consistent for all starting locations and a month index = 6, or the month of July was selected for the analysis. Akin to the analysis for varying thermal strengths, a summary of the objective function values obtained and computational requirements is presented in Table 6.4. The increase in energy height due to thermal soaring for different initial starting locations ranged from 1077.8 to 1259.2 ft. It was observed that for an increasing starting radius, less energy extraction occurred and that effect was slightly reduced with increasing altitude starting points. Conversely, for an increasing starting altitude, more energy extraction occurred. However, the flight trajectories were consistent regardless of the starting condition and the states and controls behaviour was similar to what was observed in the previous case.

6.2.2 Effect of Variations in the Objective Function

As detailed in Chapter 5, the optimal control problem was also formulated for the following two representations of the objective function:

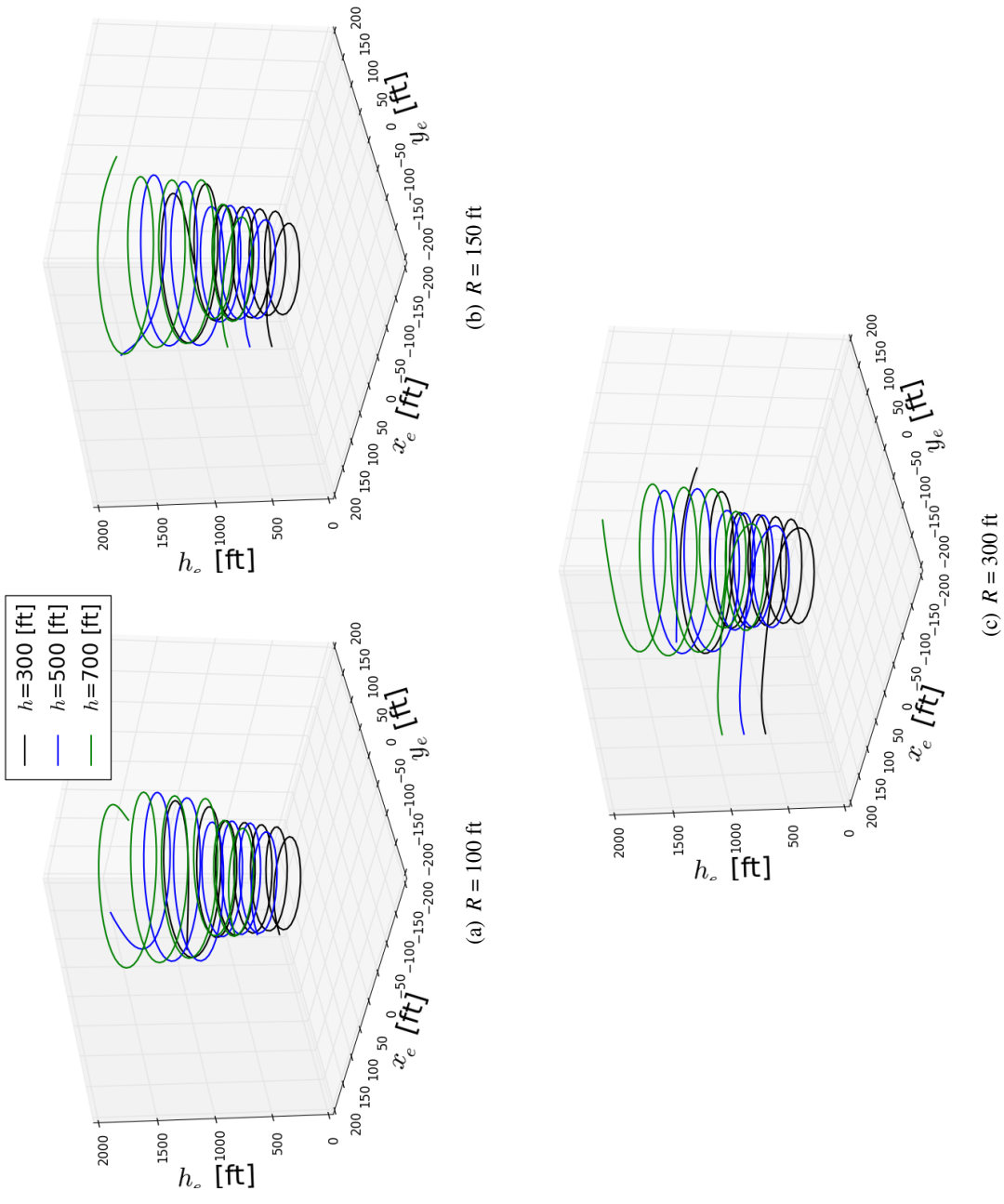
$$\begin{aligned} &\text{maximize} \;\; E_h(t_f) \\ &\text{with respect to} \;\; \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \;\; \boldsymbol{u} = (\alpha, \gamma_1)^T \end{aligned} \tag{6.1}$$

$$\begin{aligned} &\text{maximize} \;\; \int_{t_0}^{t_f} E_h \, \mathrm{d}t \\ &\text{with respect to} \;\; \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \;\; \boldsymbol{u} = (\alpha, \gamma_1)^T \end{aligned} \tag{6.2}$$

Optimal trajectories showing the effect of the variations in the objective function for three different starting positions are shown in Figure 6.9. The thermal strength was consistent for all starting locations and again was selected as month index = 6 from Table 6.4. It can be observed that the variation in the formulation of the objective function has little to no effect on the trajectory and the overall energy extraction. As such, all further trajectories will be solved by maximizing energy at the terminal time vice using the instantaneous energy approach.

Table 6.4: Optimization Efficiency Results - Varying Thermal Strengths

Condition	$f(x^*)$ [ft]	No. of Function Calls	No. of Major Iterations	Time to Solve [hr]
Effect of Varying Thermal Strengths				
Month Index = 0	1021.8	1,386	456	4.35
Month Index = 3	1458.5	1,735	684	5.58
Month Index = 6	1428.8	1,875	554	6.00
Month Index = 9	1105.7	1,858	583	5.68
Effect of the Initial Starting Condition				
Location = 1	1434.1	1,207	425	3.82
Location = 2	1709.9	1,064	568	3.47
Location = 3	1959.2	823	340	2.14
Location = 4	1428.8	1,875	554	6.00
Location = 5	1706.8	550	240	2.84
Location = 6	1957.2	1,273	446	4.36
Location = 7	1377.8	931	406	2.18
Location = 8	1667.6	1,136	418	4.34
Location = 9	1925.4	1,148	526	2.69

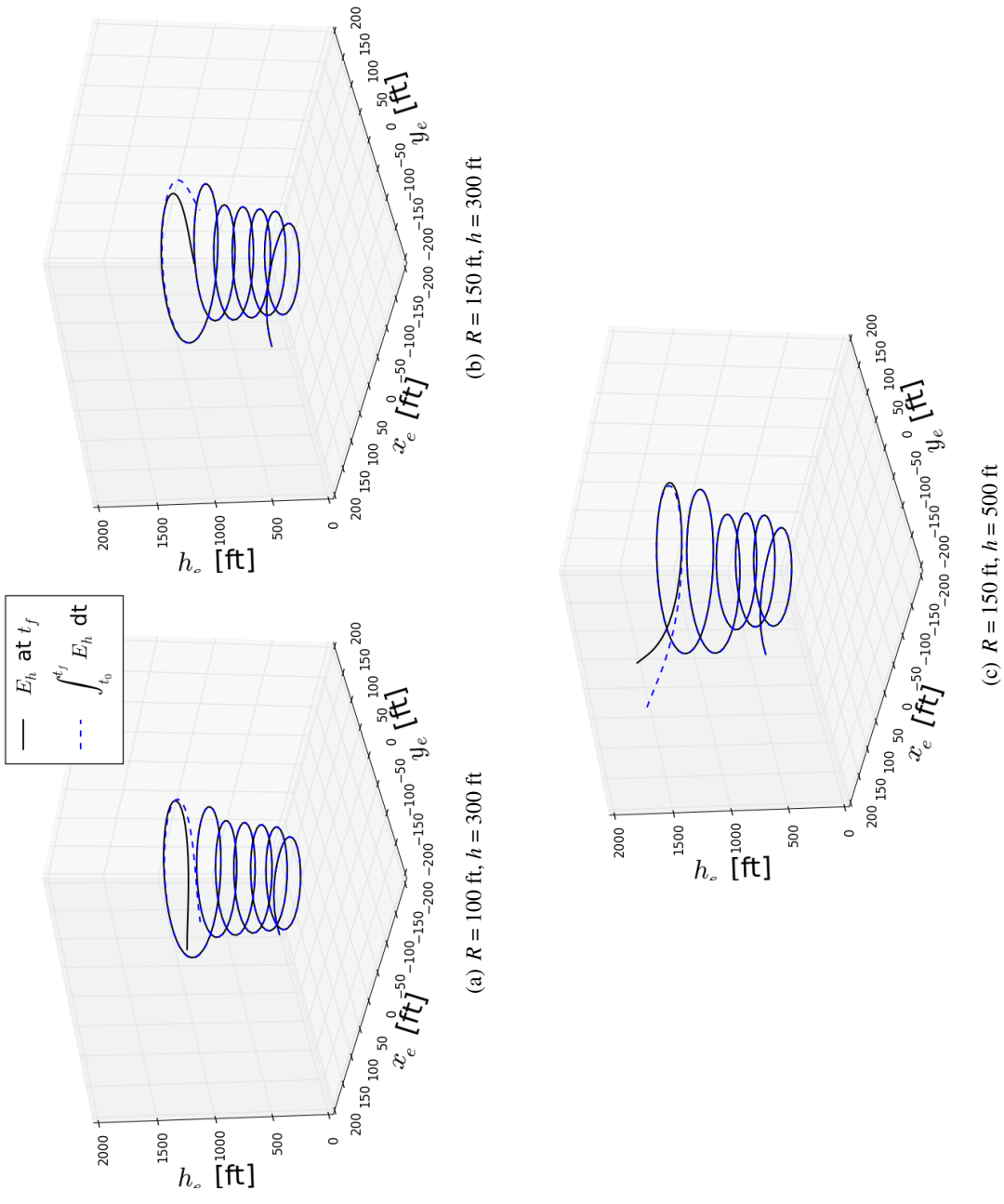


(a) R = 100 ft

(b) R = 150 ft

(c) R = 300 ft

Figure 6.8: Optimal Trajectories for Varying Initial Starting Locations



(a) $R = 100$ ft, $h = 300$ ft

(b) $R = 150$ ft, $h = 300$ ft

(c) $R = 150$ ft, $h = 500$ ft

Figure 6.9: Effect of Variations in the Objective Function

6.2.3 Effect of Aircraft Tangential Tracking Angle Range

The effect of aircraft tangential tracking angle range was investigated for the following three cases:

$$\begin{aligned} \frac{\pi}{10} \leq\ & \varepsilon \leq \frac{9\pi}{10}, \\ \frac{\pi}{4} \leq\ & \varepsilon \leq \frac{3\pi}{4}, \text{ and} \\ \frac{2\pi}{5} \leq\ & \varepsilon \leq \frac{3\pi}{5}. \end{aligned}$$

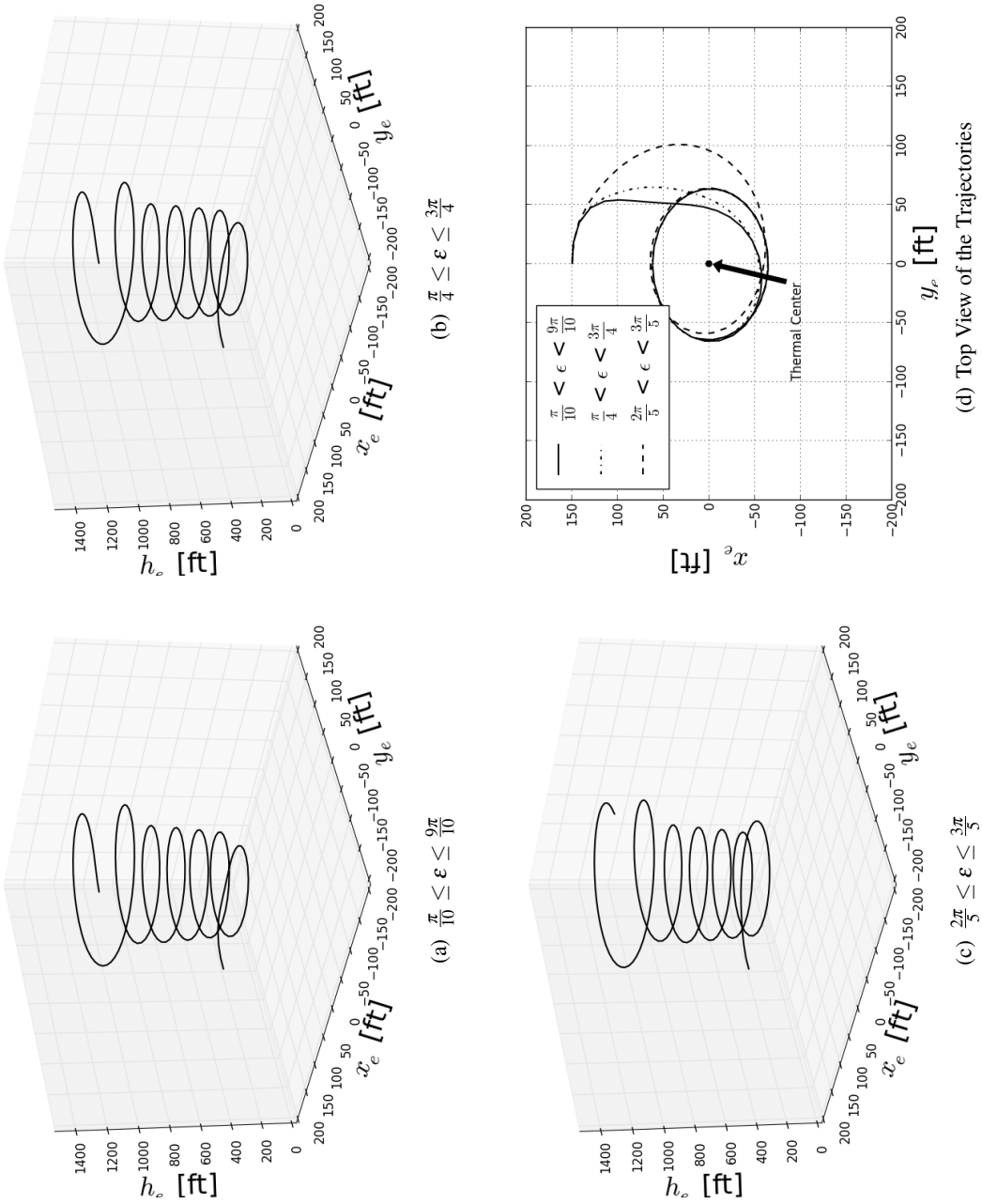
Optimal trajectories showing the effect of the aircraft tangential tracking angle range for the same initial starting point are shown in Figure 6.10. Additionally, traces of aircraft radial distance to the thermal core and tangential tracking angle as they vary with time are shown in Figure 6.11. The initial capture of the thermal varied greatly with a change in the allowable tangential tracking angle range, in that the initial turn towards the thermal was much sharper. However, the steady-state conditions for the aircraft radial distance and tracking angle when the aircraft was centered on the thermal were identical as shown in the time histories. Additionally, the soaring trajectories and final energy heights were consistent and as such a nominal range of $\frac{\pi}{4} \leq \varepsilon \leq \frac{3\pi}{4}$ was selected and used throughout the remaining analysis to provide an increased flexibility when calculating optimal captures while at the same time ensuring a smooth initial trajectory.

6.2.4 Energy Analysis of the Optimal Trajectories

The following energy analysis of the optimal trajectories was performed for the case where the effect of the varying thermal strengths was conducted and detailed in Section 6.2.1. Traces of energy height, E_h, the time rate of change of energy height, $\dot{E}_h$, and the double time derivative of energy height, $\ddot{E}_h$ as the varied with time are shown in Figure 6.12. It was observed that the energy height increases at a constant rate throughout the climb once the aircraft is centered on the thermal. Furthermore, a key indication that the aircraft is fully centered on the thermal core is the condition when $\ddot{E}_h \to 0$. Recall from Chapter 5 the expression for energy height:

$$E_h = h + \frac{V^2}{2g} \tag{6.3}$$

Taking the first and second time derivatives we have:



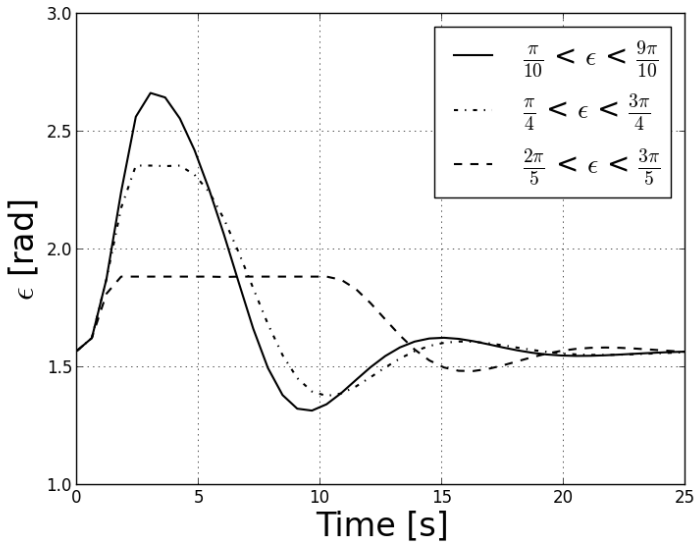
(a) $\frac{\pi}{10} \leq \varepsilon \leq \frac{9\pi}{10}$

(b) $\frac{\pi}{4} \leq \varepsilon \leq \frac{3\pi}{4}$

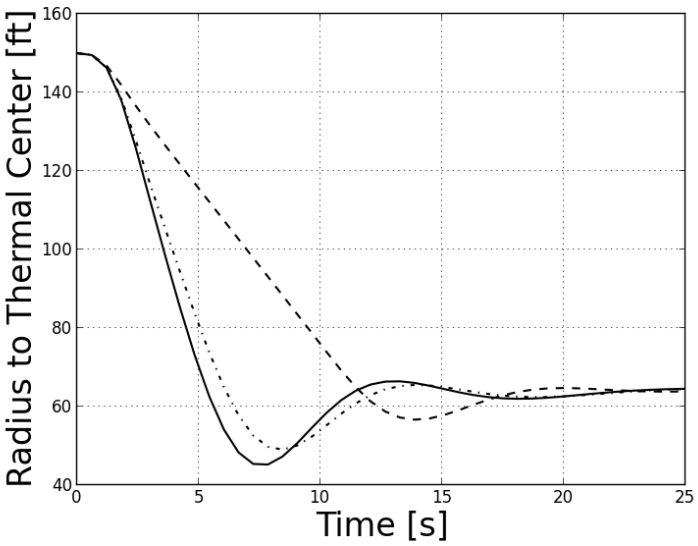
(c) $\frac{2\pi}{5} \leq \varepsilon \leq \frac{3\pi}{5}$

(d) Top View of the Trajectories

Figure 6.10: Effect of Aircraft Tangential Tracking Angle Range on the Trajectory

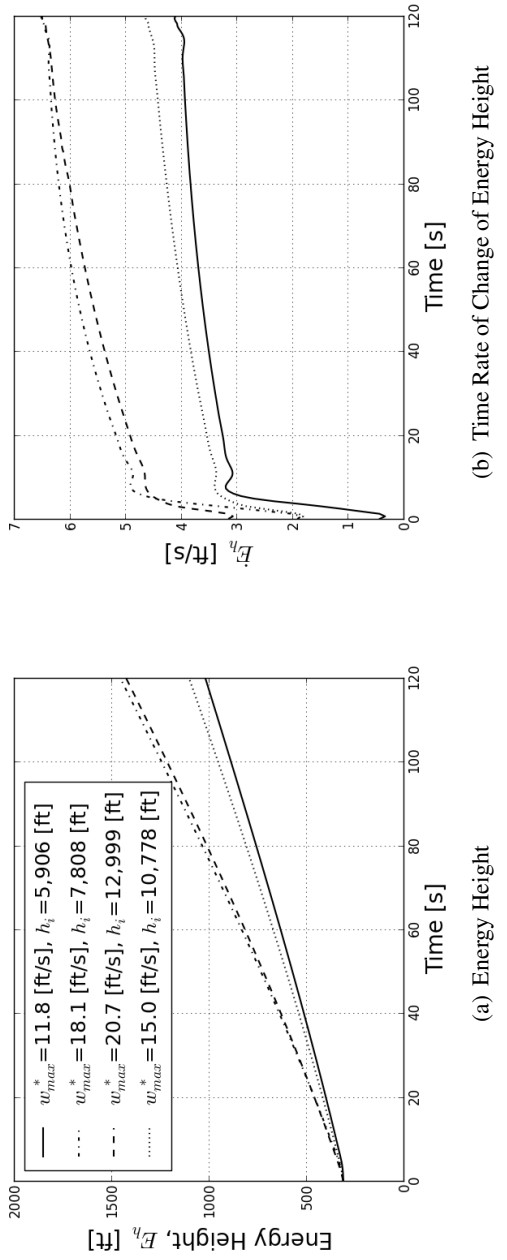


(a) Aircraft Tangential Tracking Angle



(b) Aircraft Radial Distance to Thermal Core

Figure 6.11: Time History of the Effect of Tangential Tracking Angle Range



(a) Energy Height

(b) Time Rate of Change of Energy Height

(c) Double Time Derivative of Energy Height

Figure 6.12: Time History Energy Analysis for Varying Thermal Strengths

$$\dot{E}_h = \dot{h} + \frac{V\dot{V}}{g} \tag{6.4}$$

$$\ddot{E}_h = \ddot{h} + \frac{\dot{V}^2}{g} + \frac{V\ddot{V}}{g} \tag{6.5}$$

From Equation 6.5, an understanding of the time derivatives of vertical speed and airspeed is required. As such, plots were generated to show the change in airspeed and vertical speed rates and acceleration with time and are shown in Figures 6.13 and 6.14 respectively. Given that $\ddot{E}_h$, $\ddot{h}$, $\dot{V}$, and $\ddot{V}$ all approach zero when the aircraft was centered on a thermal, the above first and second time derivatives of energy height reduce to:

$$\dot{E}_h = \dot{h} \tag{6.6}$$

$$\ddot{E}_h = \frac{\dot{V}^2}{g} = 0 \Rightarrow \dot{V} = 0 \tag{6.7}$$

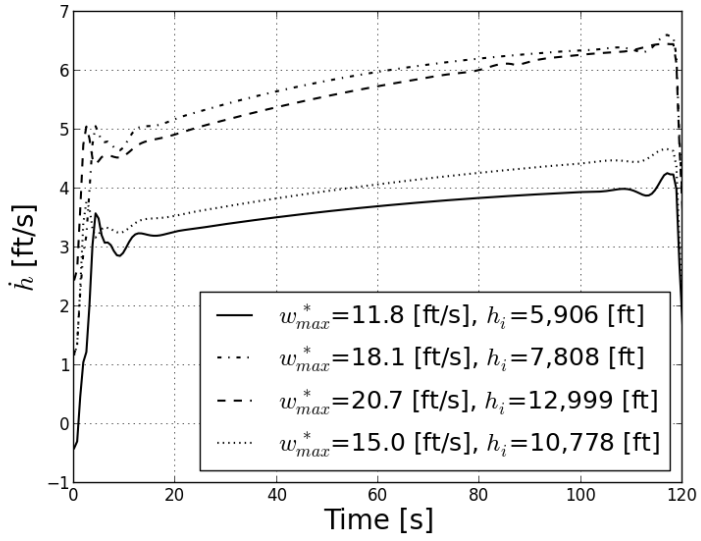
Replacing $\dot{V}$ with the previously derived equation of motion and assuming contributions from only the vertical wind component, we have:

$$\dot{V} = -\frac{D}{m} - g\sin\gamma_2 - [\dot{V}_{w,z}]_e \sin\gamma_2 = 0$$

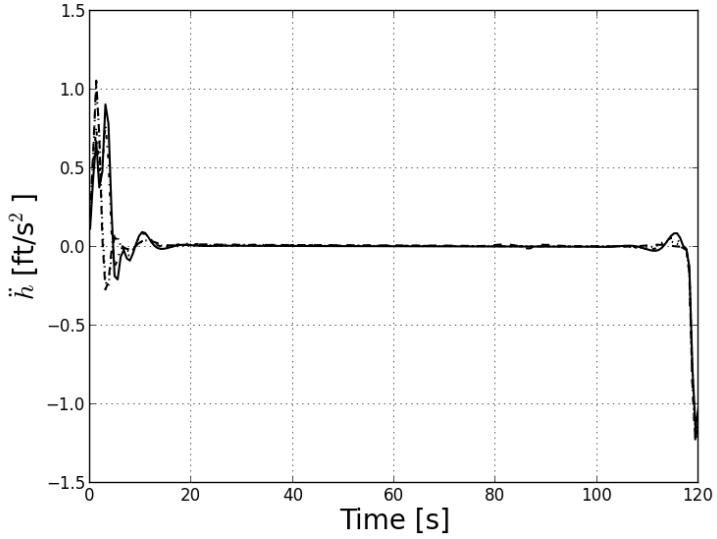
Solving the expression for the climb angle yields:

$$\sin\gamma_2 = \frac{-D}{W + m[\dot{V}_{w,z}]_e} \tag{6.8}$$

Since the weight of the aircraft is fixed, $\dot{V} = 0$ and the angle of attack does not vary with time during the centered thermal climb, the climb angle is a function of the thermal vertical velocity acceleration, aircraft weight and aerodynamic characteristics. The thermal updraft strength and acceleration were also plotted at each aircraft position along the trajectory and are shown in Figure 6.15. Since the contribution of the thermal acceleration was shown to be small for all trajectories (0.2-0.5 ft/s^2) as compared to the aircraft weight, W, it will have a negligible effect on the selection of the climb angle. As such, the climb angle required for an optimal centered thermal climb is a function of the particular aircraft's aerodynamic and weight characteristics and would remain constant within the climb.

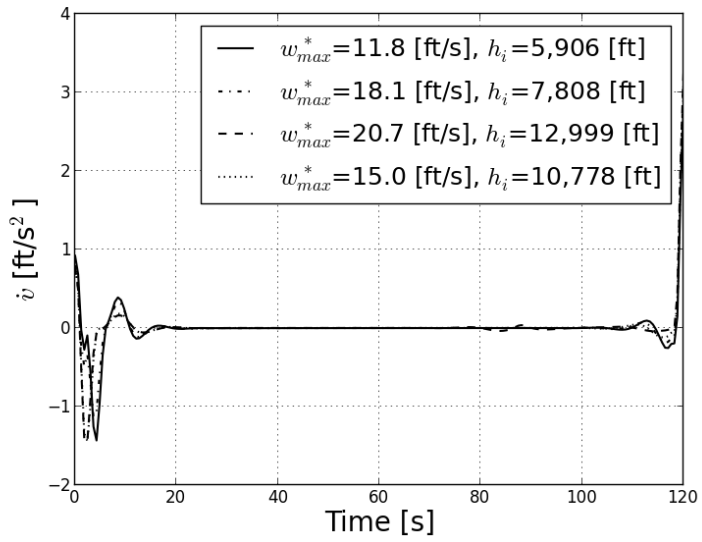


(a) Vertical Speed

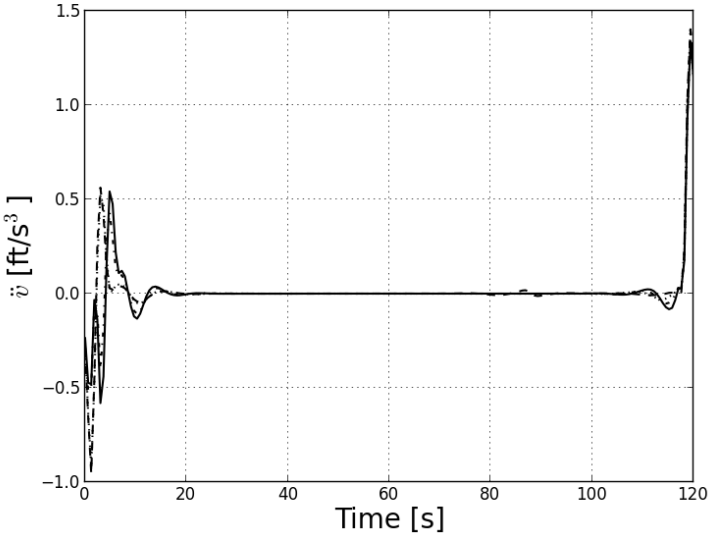


(b) Rate of Change of Vertical Speed

Figure 6.13: Time History Analysis of Vertical Speed Rates and Acceleration Changes

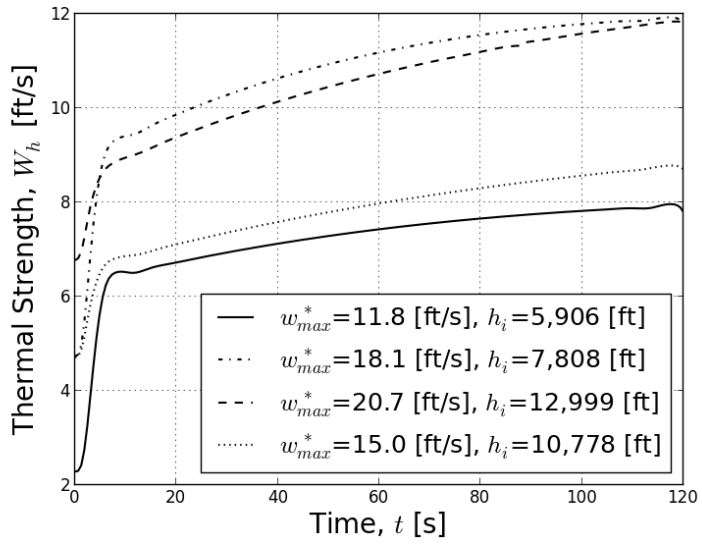


(a) Airspeed Acceleration

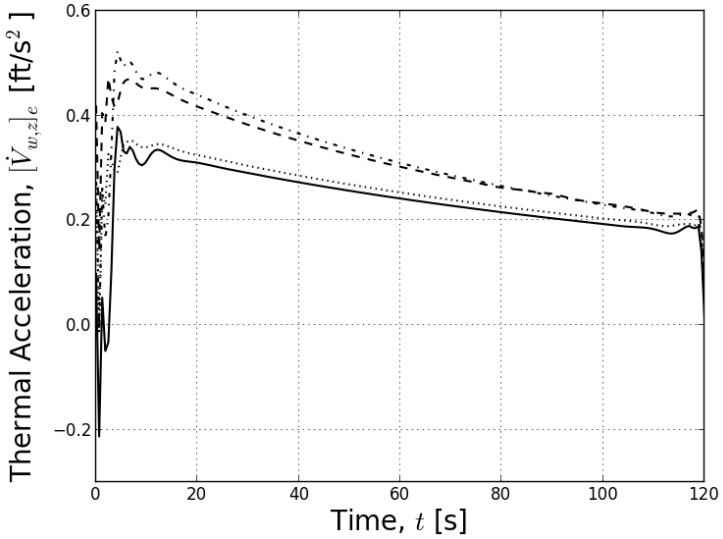


(b) Rate of Change of Airspeed Acceleration

Figure 6.14: Time History Analysis of Airspeed Rates and Acceleration Changes



(a) Thermal Strength



(b) Thermal Acceleration

Figure 6.15: Time History Analysis of Thermal Strength and Acceleration

6.3 Optimal Trajectories for the Bubble Thermal Model

6.3.1 Effect of Thermal Strength on the Trajectories

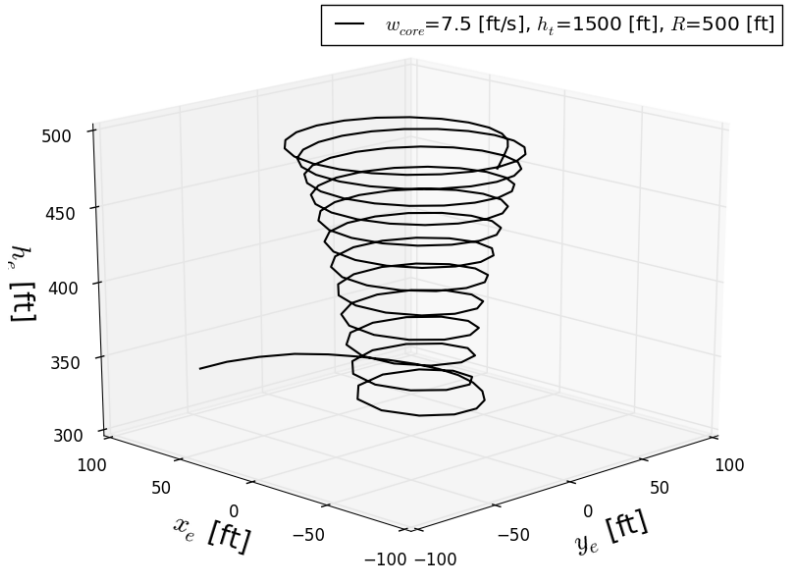
Using the bubble thermal model detailed in Section 4.4, optimal trajectories for varying thermal strengths were generated and are shown in Figure 6.16. These trajectories were generated to confirm the results obtained for the chimney model in an attempt to develop a controller that can be applied to multiple thermal types. Time histories of the states and controls for each trajectory are provided in Figures 6.17 and 6.18 respectively. Similar to the results obtained for the chimney model, the aircraft centered on the thermal after approximately 20 seconds of flight time.

Following the initial transient conditions required to steer the aircraft for thermal centering, the states and controls were analyzed for their behaviour and were similar to the results obtained for the chimney model. However, the turns performed by the aircraft were significantly tighter and as a result higher sink rates were generated and less energy was extracted. Additionally and upon thermal centering, the aircraft flew a circular flight path that was tangential to the thermal core. Once the aircraft entered the bubble core, angle of bank commands were reduced to less than $\gamma_1 = \pi/6$ rad [30°] and the aircraft radial distances from the thermal core were more consistent with the previous results obtained for the other thermal model. Towards the end of the trajectory, the steady-state conditions for both the states and controls were commensurate with those obtained with the chimney model. The airspeed was slowly reduced to the minimum control speed, V_{mc} while the flight path angle was also reduced and settled at a value of $\gamma_2 = -0.044$ rad [$-2.5°$].

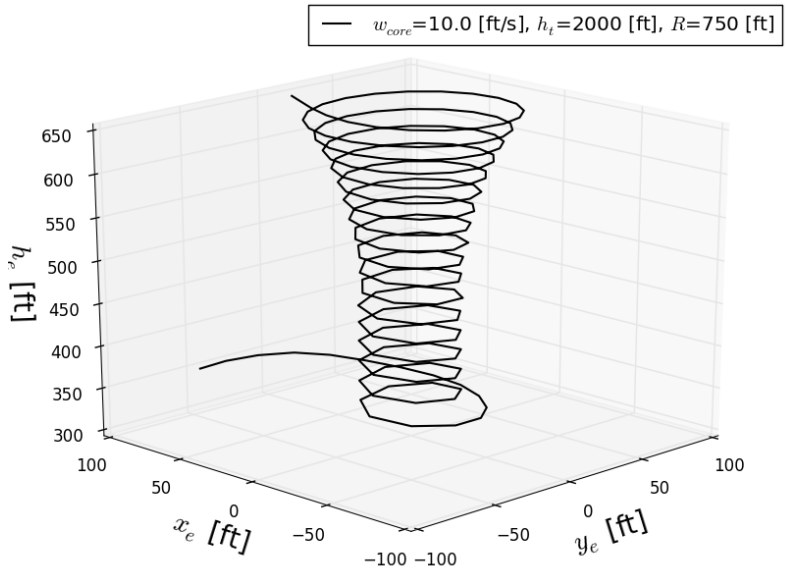
The resulting objective function value, or maximized energy height, the computational effort required to solve the trajectories, and the overall optimization efficiency are presented in Table 6.5. The increase in energy height due to thermal soaring for varying thermal strengths ranged from 208.4 to 387.8 ft. It took over double the amount of time to solve for the optimal trajectories using the bubble thermal model as compared to the chimney model.

6.3.2 Effect of Initial Starting Location on the Trajectories

Three dimensional plots of the optimal trajectories for varying initial starting locations are shown in Figure 6.19. The thermal strength was consistent for all starting locations and a core updraft velocity of $w_{core} = 7.5$ ft/s was selected for the analysis. A summary of the objective function values obtained and computational requirements is presented in Table 6.5. Given the fact that the bubble thermal model did not permit the bubble to detach from the ground and rise with time, the trajectories obtained were dissimilar for the varying initial starting locations. Depending on the



(a)



(b)

Figure 6.16: Optimal Trajectories for Varying Thermal Strengths - Bubble Thermal

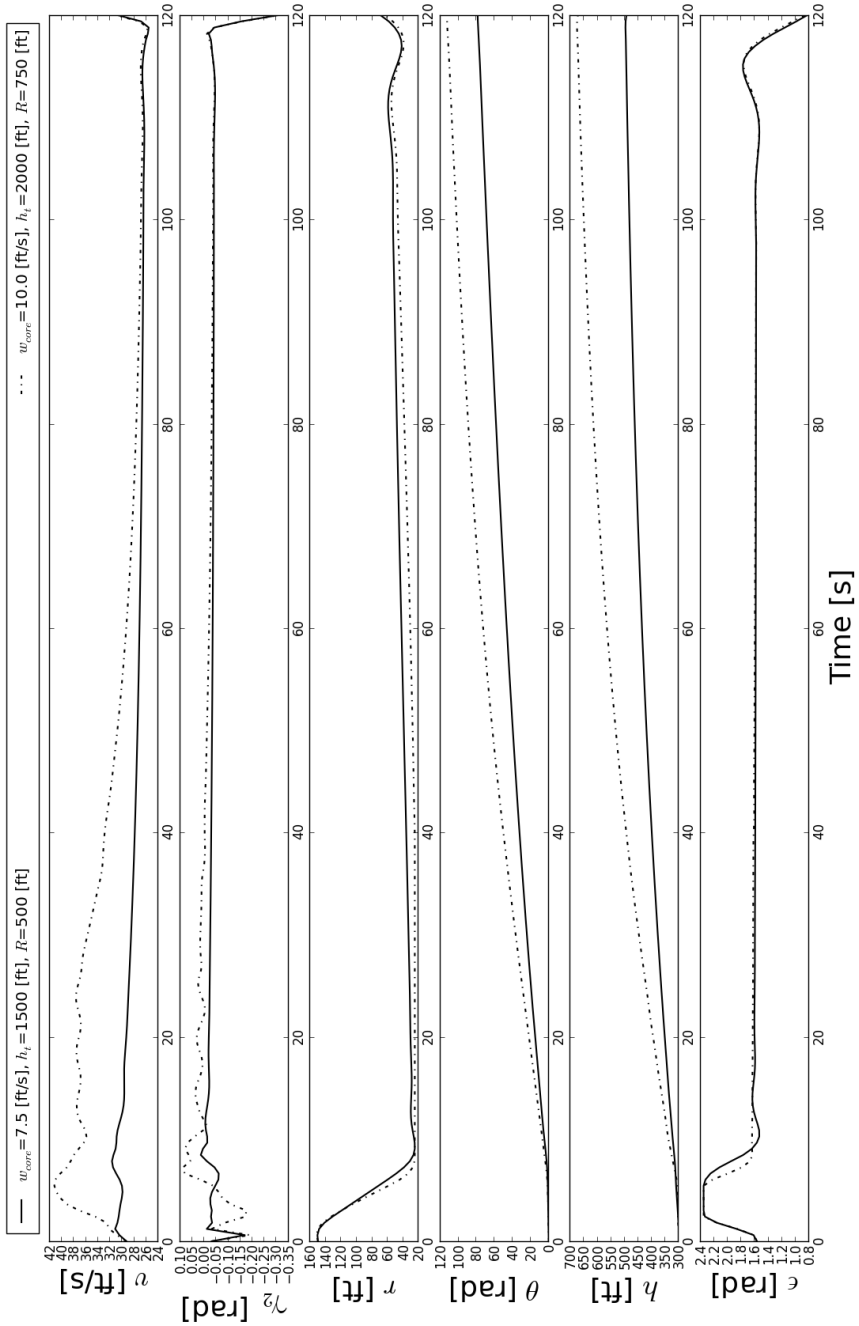


Figure 6.17: Time History of the States for Varying Thermal Strengths - Bubble Thermal

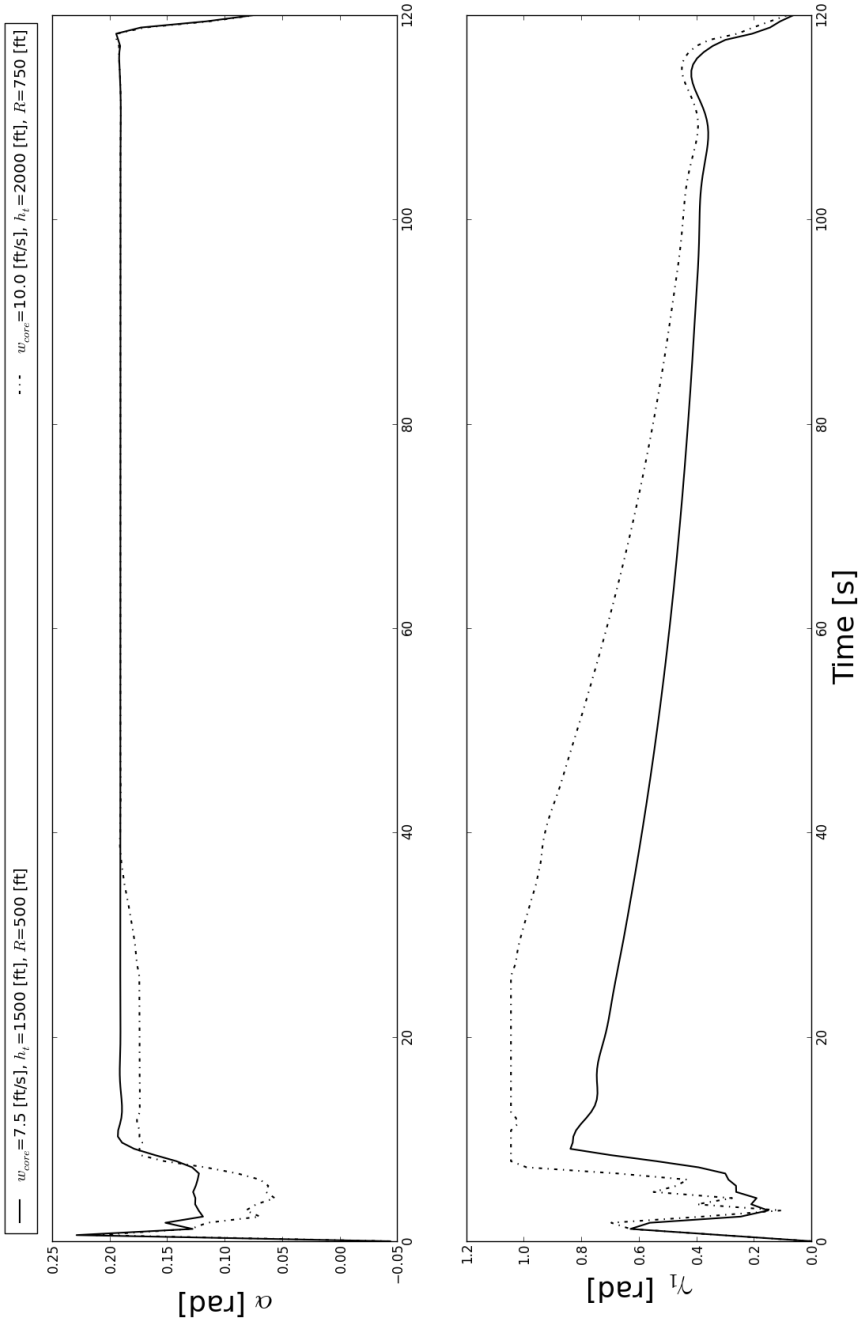


Figure 6.18: Time History of the Control Inputs for Varying Thermal Strengths - Bubble Thermal

Table 6.5: Optimization Efficiency Results - Varying Thermal Strengths

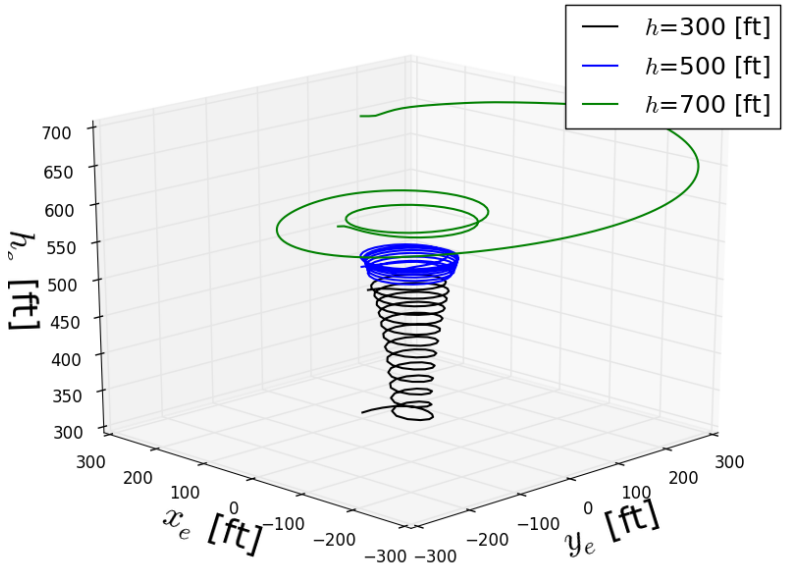
Condition	$f(x^*)$ [ft]	No. of Function Calls	No. of Major Iterations	Time to Solve [hr]
Effect of Varying Thermal Strengths				
$w_{core} = 7.5$ [ft/s]	508.4	3,790	1,386	8.85
$w_{core} = 10.0$ [ft/s]	687.8	7,176	2,976	15.75
Effect of the Initial Starting Condition				
Location = 1	509.0	5,687	1,640	13.30
Location = 2	542.3	1,730	929	3.96
Location = 3	578.8	7,243	2,216	16.16
Location = 4	508.4	3,790	1,386	8.85
Location = 5	542.2	10,145	3,352	21.56
Location = 6	579.5	967	324	2.32

height of the initial starting point as compared to the bubble core, the aircraft either ascended or descended to take advantage of the updraft strength contained within the core. As such, the value obtained for the energy height at the terminal time were similar and within 70 ft for all starting locations.

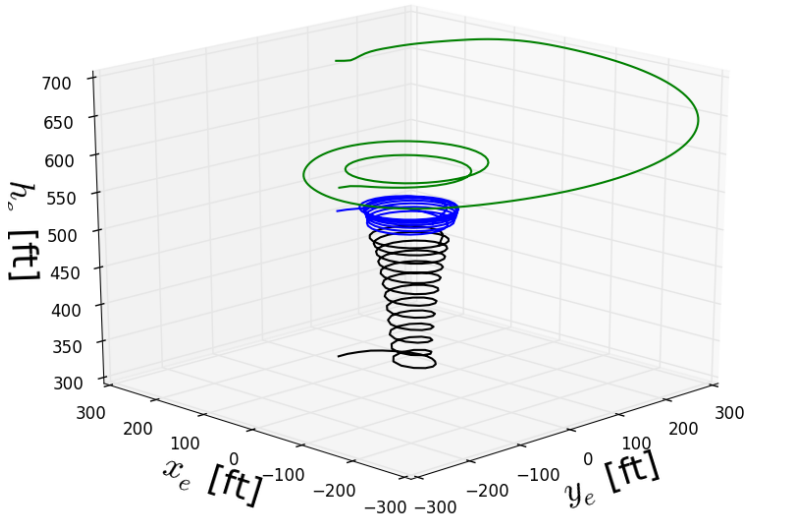
6.3.3 Energy Analysis of the Optimal Trajectories

Recall that for the trajectories obtained for the chimney thermal model, $\ddot{E}_h$, $\ddot{h}$, $\dot{V}$, and $\ddot{V}$, all approached zero when the aircraft was centered on a thermal. Additionally, a reduced expressed for the first and second time derivatives of energy height was obtained and stated in Equation 6.7. As such, a similar energy analysis was performed on the optimal trajectories to confirm that similar results were obtained for a bubble thermal type. Traces of energy height, E_h, the time rate of change of energy height, $\dot{E}_h$, and double time derivative of energy height, $\ddot{E}_h$ as the varied with time are shown in Figure 6.20. It was observed that the energy height increased with time throughout the climb once the aircraft was centered on the thermal. In addition, a positive rate of energy was present throughout the trajectory but decreased with time at a non-linear rate. Furthermore, $\ddot{E}_h \rightarrow 0$, which confirmed that it was indeed fully centered on the thermal core.

To complete the analysis, traces of the change in airspeed and vertical speed rates and acceleration with time were generated and are shown in Figures 6.21 and 6.22 respectively. Consistent with results previously obtained $\ddot{h}$, $\dot{V}$, and $\ddot{V}$ all approached zero when the aircraft was centered on a thermal.

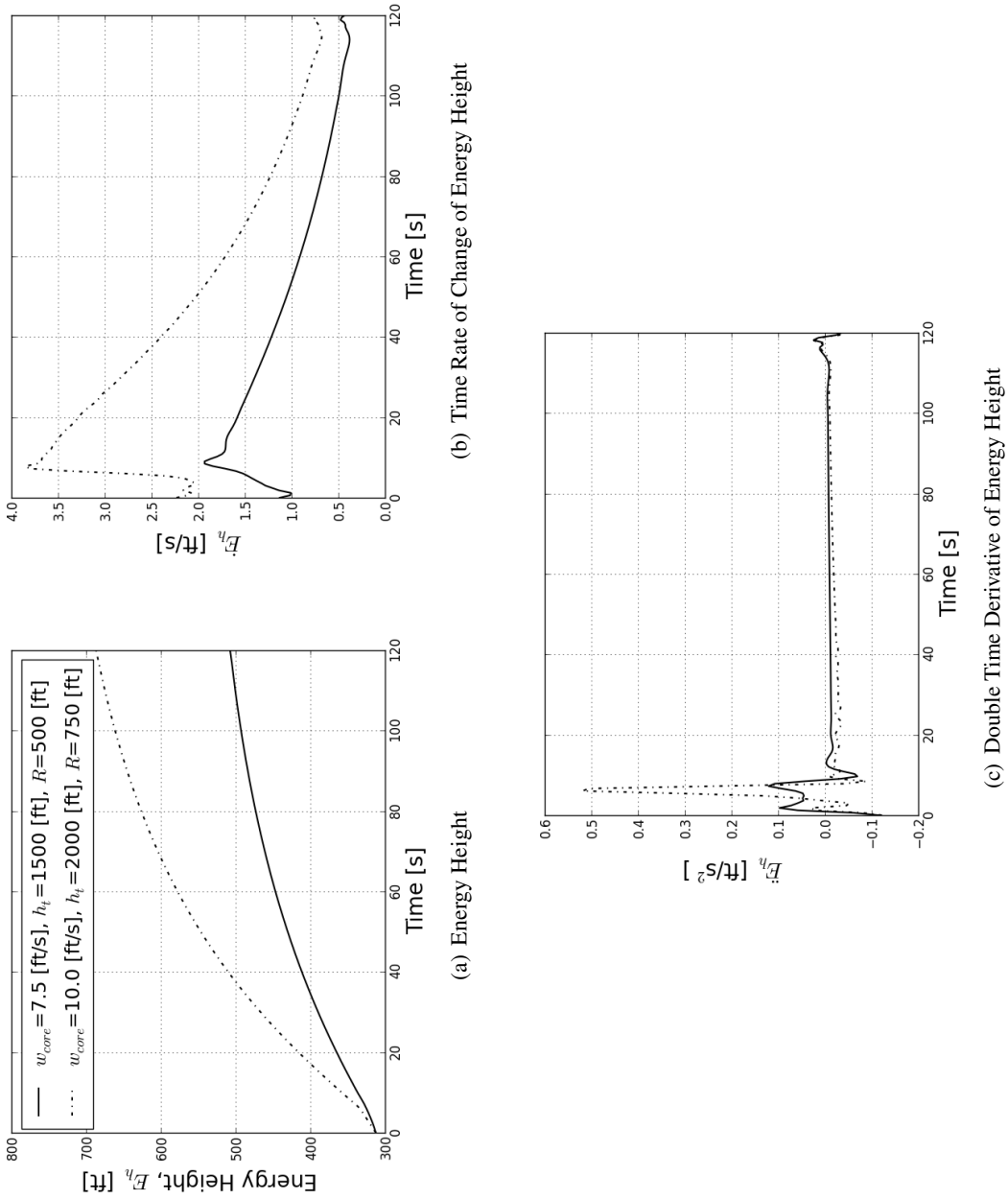


(a) R = 100 ft



(b) R = 150 ft

Figure 6.19: Optimal Trajectories for Varying Initial Starting Locations - Bubble Thermal

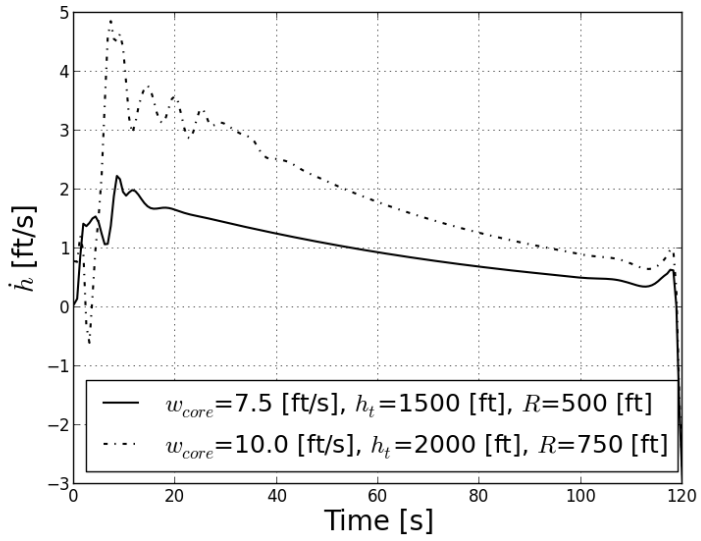


(a) Energy Height

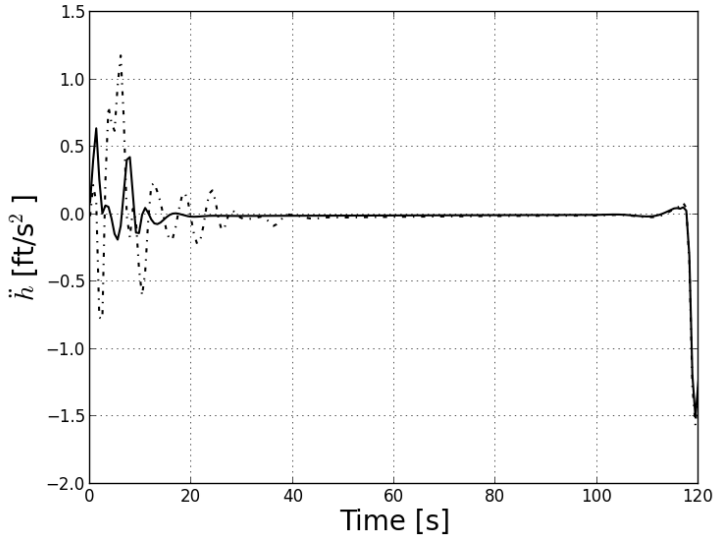
(b) Time Rate of Change of Energy Height

(c) Double Time Derivative of Energy Height

Figure 6.20: Time History Energy Analysis for Varying Thermal Strengths - Bubble Thermal

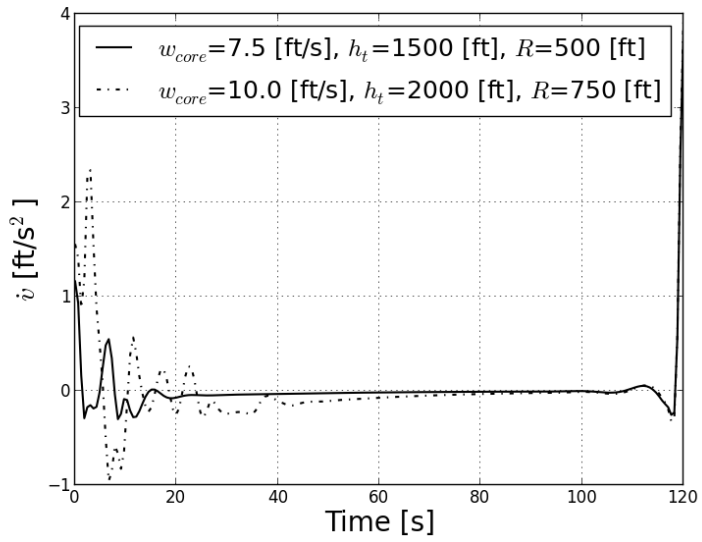


(a) Vertical Speed

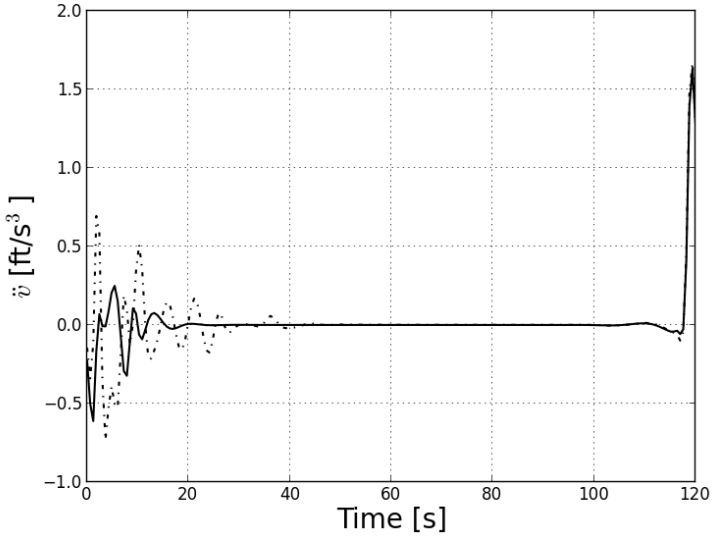


(b) Rate of Change of Vertical Speed

Figure 6.21: Plots of Vertical Speed Rates and Acceleration Changes - Bubble Thermal



(a) Airspeed Acceleration



(b) Rate of Change of Airspeed Acceleration

Figure 6.22: Plots of Airspeed Rates and Acceleration Changes - Bubble Thermal

6.4 Optimal Control Strategy for Thermal Soaring

Given the results and analysis conducted on the optimal trajectories that were generated for autonomous thermal soaring, an optimal control strategy is formulated and presented herein. For either hardware-in-the-loop simulation or actual test flights, the following assumptions made during the optimization problem no longer hold true:

i. The location of the thermal updraft is not known and as such a method to detect thermal activity is required.
ii. The location of the thermal center is not easily determined, so any centering strategies must not rely on having knowledge of the thermal center location.
iii. Thermal activity is inherently characterized by atmospheric thermal instability and the conditions can change rapidly.

For thermal detection, it is recommended that a method to determine or measure the time rate of change of energy height, $\dot{E}_h$ be developed. Recall from Section 6.2.4, the expression for $\dot{E}_h$ was given as:

$$\dot{E}_h = \dot{h} + \frac{V\dot{V}}{g}$$

A complicated method to determine $\dot{E}_h$ would be to measure the individual variables directly, differentiate and filter the signals, and combine them to get a value of the instantaneous $\dot{E}_h$. This approach would require the information from various sensors that are normally installed on glider aircraft (airspeed indicator, altimeter, and variometer); however, time lags in sensing and processing would adversely affect the centering results. Another method would be to develop a sensor that measures $\dot{E}_h$ directly. The concept of a simple total energy sensor was initially proposed and tested by Nicks [45] with good results, and as such a prototype was constructed for its use on the Cularis aircraft. Similar to the approach used by Allen [9], it is proposed that a minimum threshold value of $\dot{E}_h$ be used as a switching criteria from a searching to a soaring mode.

The following observations were made from the results and analysis conducted on the optimal trajectories:

i. For thermal centering to occur, $\ddot{E}_h \to 0$.
ii. The climb angle is directly related to the condition where $\ddot{E}_h = 0$ and can be calculated using the relationship established in Equation 6.8 for a given aircraft weight, wing geometry, and flight condition.
iii. For aircraft where the stall speed is greater than the minimum sink rate speed, the aircraft should fly at the minimum control airspeed and the angle of attack should be commanded to reflect this condition. Otherwise and consistent with

gliding flight theory, the aircraft should fly at a speed for minimum sink rate but adjusted using the derivation shown below.

Recall the following equations of motion that were previously detailed in the optimization problem formulation in Section 5.6. Considering only the vertical component of the thermal updraft yields:

$$\dot{V} = -\frac{D}{m} - g\sin\gamma_2 - [\dot{V}_{w,z}]_e \sin\gamma_2$$
$$\dot{\gamma}_2 = \frac{L\cos\gamma_1 - mg\cos\gamma_2 + m[\dot{V}_{w,z}]_e \cos\gamma_2}{m[V]_v}$$

Expressing the equations of motion again in terms of a lift-to-drag ratio, we have:

$$\frac{C_L}{C_D} = \frac{L}{D} = \frac{\frac{-mg\cos\gamma_2 + m[\dot{V}_{w,z}]_e \cos\gamma_2}{\cos\gamma_1}}{-mg\sin\gamma_2 - m[\dot{V}_{w,z}]_e \sin\gamma_2}$$

Solving the expression for the climb angle:

$$\sin\gamma_2 = \frac{(-W + m[\dot{V}_{w,z}]_e)\cos\gamma_2}{\frac{C_L}{C_D}(-W - m[\dot{V}_{w,z}]_e)\cos\gamma_1} \tag{6.9}$$

As previously seen in Section 3.5, the vertical velocity of an aircraft was given as [37]:

$$\dot{h} = V\sin\gamma_2 \tag{6.10}$$

where the velocity, V was calculated as:

$$V = \left[\frac{W\cos\gamma_2}{\frac{1}{2}\rho S C_L \cos\gamma_1}\right]^{\frac{1}{2}} \tag{6.11}$$

Using the result from Equation 6.9, the following expression is obtained for the vertical velocity of a gliding aircraft:

$$\dot{h} = \left[\frac{W\cos\gamma_2}{\frac{1}{2}\rho S C_L \cos\gamma_1}\right]^{\frac{1}{2}} \left[\frac{(m[\dot{V}_{w,z}]_e - W)\cos\gamma_2}{(m[\dot{V}_{w,z}]_e + W)\cos\gamma_1}\right] \left[\frac{-1}{\frac{C_L}{C_D}}\right]$$
$$\dot{h} = -\left(\frac{W}{S}\right)^{\frac{1}{2}} \left(\frac{2}{\rho}\right)^{\frac{1}{2}} \left(\frac{\cos\gamma_2}{C_L\cos\gamma_1}\right)^{\frac{1}{2}} \left[\frac{(m[\dot{V}_{w,z}]_e - W)}{(m[\dot{V}_{w,z}]_e + W)}\right] \left(\frac{\cos\gamma_2}{\cos\gamma_1}\right) \left(\frac{C_D}{C_L}\right)$$

$$\dot{h} = -\left(\frac{W}{S}\right)^{\frac{1}{2}}\left(\frac{2}{\rho}\right)^{\frac{1}{2}}\left(\frac{C_D}{C_L^{\frac{3}{2}}}\right)\sec^{\frac{3}{2}}\gamma_1\cos^{\frac{3}{2}}\gamma_2\left[\frac{(m[\dot{V}_{w,z}]_e - W)}{(m[\dot{V}_{w,z}]_e + W)}\right] \tag{6.12}$$

It can be observed that the minimum sink rate velocity is adjusted based on the thermal updraft acceleration term, $[\dot{V}_{w,z}]_e$. As was shown in Figure 6.15(b), the thermal updraft acceleration varied between 0.2 to 0.5 ft/s^2. When combining the thermal updraft acceleration term with the aircraft mass (0.149 slug) and the aircraft weight (4.81 lb$_f$), it will scale the vertical velocity of the aircraft by the following factor:

$$-0.988 < \frac{m[\dot{V}_{w,z}]_e - W}{m[\dot{V}_{w,z}]_e + W} < -0.969$$

Furthermore, this contribution will increase the speed-to-fly for the minimum sink rate condition by 1.2 to 3.2%.

One of the aspects of the centering strategy that was not clearly apparent, was the specific relationship between the optimal angle of bank command that was required for a particular thermal strength. Qualitatively, it was shown that the aircraft radial distance to the thermal core varied between 45 and 75 ft with the larger radii observed for updrafts with higher thermal strengths. A graphical approach was devised by Cone [46] to determine an optimum combination of the lift coefficient, C_L and angle of bank, γ_1 such that the sink rate would be a minimum at each radius of turn. Furthermore, this approach used the analytical expressions that were previously developed for the turn radius and vertical velocity of a gliding aircraft in Equations 3.19 and 3.15. The resulting graph for the Cularis D-5223 aircraft is shown in Figure 6.23. The optimum point was defined as where lift coefficient was a maximum for a particular turn radius. For a turn radius between 40 to 100 ft, the optimal angle of bank range is between $\pi/9$ to $5\pi/36$ rad [20 to 25°] and matches the results obtained in the optimization results.

Once thermal detection has occurred (i.e. a threshold value of $\dot{E}_h$ is detected), the following centering strategy is proposed for controller development based on the aforementioned observations:

i. Using the results that were presented in Figure 6.23, schedule the angle of bank for a given thermal strength based on a measurement of $\dot{E}_h$,
ii. Perform and maintain a right hand turn based on the angle of bank schedule and slow the aircraft to the minimum control speed,
iii. If a continual increase in the measurement of $\dot{E}_h$ is obtained, reduce the turn slightly but not less than $\gamma_1 = \pi/6$ rad [20°], else
iv. If a downdraft is encountered, perform the 270° correction method described in Section 2.2,

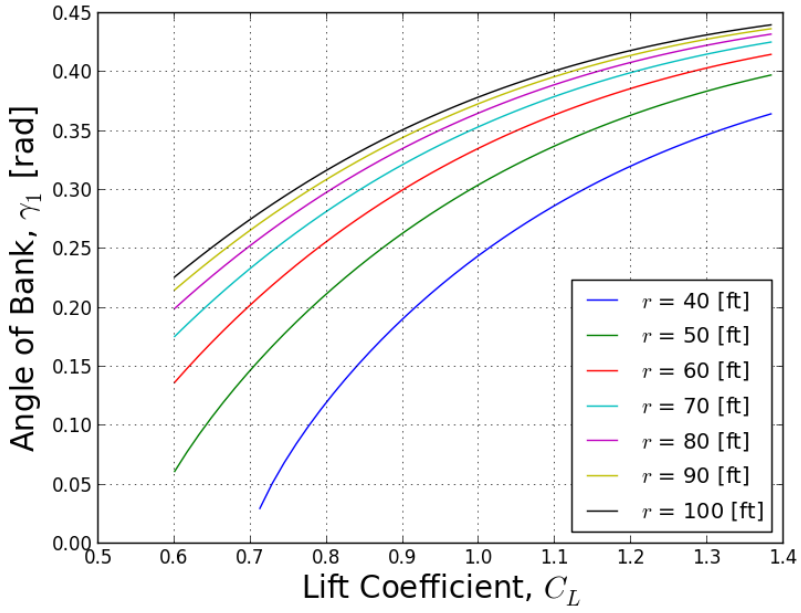


Figure 6.23: Angle of Bank versus C_L for a given Radial Distance

v. Once a continuous positive measurement of $\dot{E}_h$ is obtained either initially or following the correction method, the controller must achieve a flight condition where $\ddot{E}_h = 0$,
vi. Use a higher order flight control system (i.e. autopilot) to implement the desired flight condition, and
vii. Feedback should be performed as an input to the elevator and/or aileron to ensure the aircraft is continually trimmed and achieves the desired flight condition.

Furthermore, an inner loop flight control system based on classical aircraft control techniques should be in place to correct for the undesirable dynamics present in the raw aircraft which were previously discussed in Chapter 3.

In summary, this chapter presents the optimal flight trajectories for a small autonomous aircraft, operating in multiple thermal profiles of varying strengths using the optimal control problem formulation developed in Chapter 5. It was shown that a key condition for thermal centering was when $\ddot{E}_h \to 0$. Another key observation, was that the stall speed of the Cularis D-5223 aircraft was greater than the predicted minimum sink rate speed. As such, the aircraft flew at the minimum control airspeed throughout the soaring climb. It is recommended that a further investigation be con-

ducted for other glider aircraft to determine if they are also stall limited. Finally, an optimal control strategy was presented for efficient thermal centering and soaring flight. However, further research should be performed to determine an analytic expression to predict the angle of bank to command for a given aircraft and thermal strength. Additionally, a controller should be developed to implement the thermal centering strategy and verify its validity through simulation, hardware-in-the-loop testing, and experimental flight tests.

7 Target Localization

In this chapter, the autonomous soaring optimal control problem is extended for simultaneous optimization of thermal energy extraction and target localization using an Electro-Optical (EO) camera that is fixed at the aircraft's center of gravity. Optimal trajectories are generated for various target locations relative to the thermal center, and the problem is further constrained to ensure that a minimum digital spatial resolution is achieved along the full trajectory.

7.1 Camera Optics

The digital spatial resolution of an image is a function of the camera focal length, individual Charge-Coupled Device (CCD) detector size, and the distance the camera is located from the object. A visual representation of this relationship is shown in Figure 7.1 while the effect of the change in viewing angle is shown in Figure 7.2. It follows that the relationship is determined simply by equating the equivalent ratios that are defined as pixel area G to altitude h and focal length f to individual CCD detector size d, namely [47]:

$$\frac{G}{h} = \frac{d}{f} \Rightarrow G = h\left(\frac{d}{f}\right) \tag{7.1}$$

When accounting for the effect of the change in the viewing angle, the effective pixel size, G' projected on the ground is given as:

$$G' = \frac{G}{\cos\theta_v} \tag{7.2}$$

The viewing angle, θ_v can be expressed in terms of the aircraft altitude, h and the line of sight vector, $\boldsymbol{d}_{los}$ as follows:

$$\cos\theta_v = \frac{h}{\boldsymbol{d}_{los}} \tag{7.3}$$

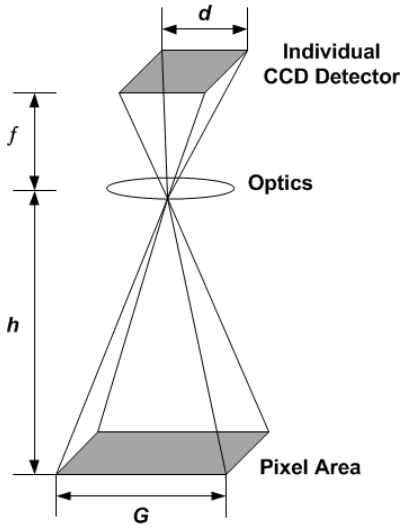


Figure 7.1: Digital Spatial Resolution

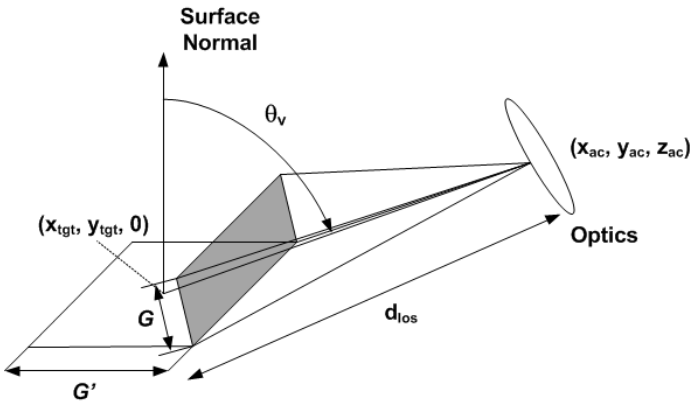


Figure 7.2: Viewing Angle Effect on the Digital Spatial Resolution

where the line of sight vector is the vector denoting the line of sight between the camera and the projected point on the ground and is determined from:

$$\boldsymbol{d}_{los} = \sqrt{(x_{cam} - x_{tgt})^2 + (y_{cam} - y_{tgt})^2 + (z_{cam} - z_{tgt})^2} \tag{7.4}$$

where x_{cam}, y_{cam}, and z_{cam} are the camera coordinate positions with respect to the aircraft's center of gravity, and x_{tgt}, y_{tgt}, and z_{tgt} are the target coordinate positions with respect to the origin of the fixed frame in the Earth's axis system.

The digital spatial resolution of an image, S_{res} was defined as a measure of the ratio between the image resolution and the individual CCD element size [47]:

$$S_{res} = \frac{d}{G} = \frac{f}{h} = \frac{\cos\theta_v}{\boldsymbol{d}_{los}} f \tag{7.5}$$

To formulate the digital spatial resolution of an image expression into a useful form for an objective function for its use in optimization, the following normalization is performed:

$$S_{res} = \frac{d}{G} = \frac{\cos\theta_v}{\boldsymbol{d}_{los}} f \left(\frac{f/\boldsymbol{d}_{los} + 1}{f/\boldsymbol{d}_{los} + 1} \right)$$
$$S_{res} = \frac{(f/\boldsymbol{d}_{los} + 1)\cos\theta_v}{1 + f/\boldsymbol{d}_{los}}$$

Given that the focal length is much less than the line of sight vector (i.e. $f \ll \boldsymbol{d}_{los}$), the expression for digital spatial resolution of an image reduces to:

$$S_{res} = \frac{\cos\theta_v}{1 + \boldsymbol{d}_{los}/f} \tag{7.6}$$

For a given optimal viewing angle, θ_{opt} the expression for digital spatial resolution becomes:

$$S_{res} = \frac{\cos(\theta_v - \theta_{opt})}{1 + \boldsymbol{d}_{los}/f}, \quad S_{res} \in [0, 1] \tag{7.7}$$

Finally, the overall digital spatial resolution throughout a flight trajectory can be considered as a probability distribution of individual digital spatial resolution elements, S_{resi} and is represented as follows:

$$S_{res} = 1 - \prod_i (1 - S_{resi}) \tag{7.8}$$

7.2 Camera Reference Axis

In order to determine the camera viewing angle and the line of sight vector, a new axis system is developed and is shown in Figure 7.3. Given that the camera is installed on the aircraft, it will be represented as an extension to the aircraft body axis system and is assigned rotational components κ and τ about the y and z body axes respectively, and a translational component $(x_{cam}, y_{cam}, z_{cam})$. The rotational components were added to simply create an arbitrary field of view on the ground by moving the line of sight vector within a small envelope. Similar to the previously defined aircraft reference frames, the camera axis system will also follow right-hand rule methodology to define its orientation and the origin will be located at the center of gravity of the aircraft. For target localization, the most important piece of information besides the target location is knowing where the camera is looking in the Earth axis system. This point will be represented by the intersection between the end of the line of sight vector and the Earth's surface.

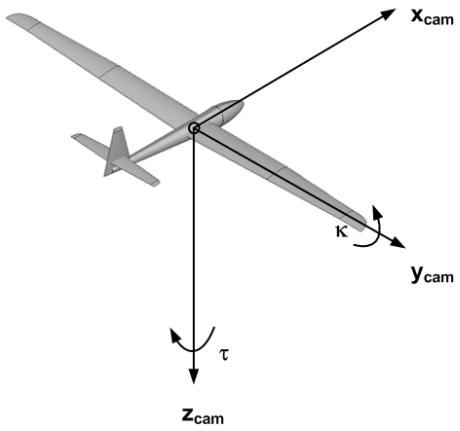


Figure 7.3: Camera Axis System

The Earth-to-Body transformation can be represented as a basic set of homogeneous transformations as follows [30, 48]:

$$H_b^e = Trans_{x,y,z} Rot_{z,\psi} Rot_{y,\theta} Rot_{x,\phi} \tag{7.9}$$

where,

$$Trans_{x,y,z} = \begin{bmatrix} 1 & 0 & 0 & x_{ac} \\ 0 & 1 & 0 & y_{ac} \\ 0 & 0 & 1 & z_{ac} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Rot_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Rot_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, the Body-to-Camera transformation can be represented as follows:

$$H_c^b = Trans_{x,y,z} Rot_{z,\tau} Rot_{y,\kappa} \tag{7.10}$$

where,

$$Trans_{x,y,z} = \begin{bmatrix} 1 & 0 & 0 & x_{cam} \\ 0 & 1 & 0 & y_{cam} \\ 0 & 0 & 1 & z_{cam} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Rot_{z,\tau} = \begin{bmatrix} \cos\tau & -\sin\tau & 0 & 0 \\ \sin\tau & \cos\tau & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{y,\kappa} = \begin{bmatrix} \cos\kappa & 0 & \sin\kappa & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\kappa & 0 & \cos\kappa & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The overall transformation from Earth-to-Camera is the product of the two individual transformations:

$$H_c^e = H_b^e H_c^b \tag{7.11}$$

In order to use the overall homogeneous transformation matrix to determine the point on the ground the camera is looking, consider the most general homogeneous transformation form [48]:

$$H_1^0 = \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $n = [n_x, n_y, n_z]^T$, $s = [s_x, s_y, s_z]^T$, $a = [a_x, a_y, a_z]^T$ represents the direction of x_1, y_2, and z_3 expression in the $o_0x_0y_0z_0$ frame. Additionally, $d = [d_x, d_y, d_z]^T$ represents the vector from the origin o_0 to the origin of o_1.

Given that there is no rotation about the x_b axis and it is assumed that the camera is located at the aircraft center of gravity, the location of the coordinates of the line of sight vector are solved by:

$$x_{los} = d_x + \Delta x$$
$$y_{los} = d_y + \Delta y$$

where,

$$\Delta x = -d_z \tan\zeta, \quad \tan\zeta = \frac{n_x}{n_z}$$
$$\Delta y = -d_z \tan\eta, \quad \tan\eta = \frac{n_y}{n_z}$$

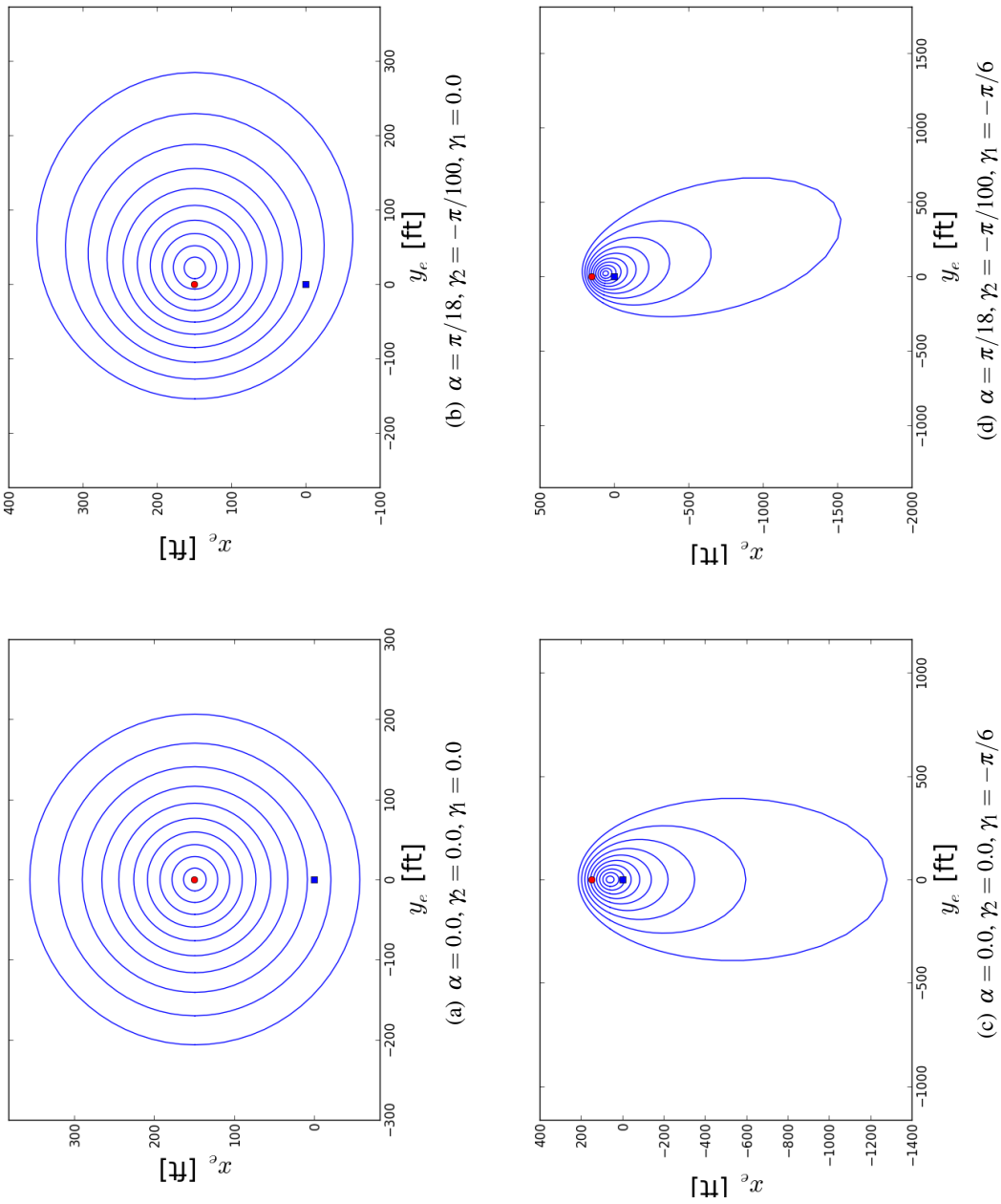
where ζ and η are intermediate angles used to determine Δx and Δy, and correspondingly the location of the line of sight vector on the Earth's surface with respect to its origin. To ensure the accuracy of the formulation, numerous Field of View (FOV) plots were generated for a number of aircraft attitudes for the Cularis D-5223 and are shown in Figure 7.4. In each FOV plot, the aircraft (red dot) is flying tangential to the target (blue square) located at [0,0] heading due East at an altitude of 150 ft AGL. The camera rotations used to define the field of view on the Earth's surface were limited to:

$$-\frac{\pi}{2} \leq \kappa \leq -\frac{\pi}{5},$$
$$-\pi \leq \tau \leq \pi$$

7.3 Target Localization Optimization Case Studies

7.3.1 Problem Formulation

The target localization optimal control problem can be defined in a variety of different manners. Given that a gradient-based optimizer is used to solve for the trajectories, it is imperative to formulate the objective function to be convex so as to ensure convergence and to avoid numerical difficulties during optimization. Consider the following



(a) $\alpha = 0.0, \gamma_2 = 0.0, \gamma_1 = 0.0$

(b) $\alpha = \pi/18, \gamma_2 = -\pi/100, \gamma_1 = 0.0$

(c) $\alpha = 0.0, \gamma_2 = 0.0, \gamma_1 = -\pi/6$

(d) $\alpha = \pi/18, \gamma_2 = -\pi/100, \gamma_1 = -\pi/6$

Figure 7.4: Field of View Diagrams for the Cularis D-5223 Aircraft

visibility error function that describes the distance between the target location and the end of the line of sight vector:

$$erf(\boldsymbol{x}) = \mu((x_{tgt} - x_{los})^2 + (y_{tgt} - y_{los})^2) \tag{7.12}$$

where μ is a constant used to increase the convexity of the function. Additionally, recall the energy approach objective function formulation used for the generation of previous trajectories:

$$E_h(\boldsymbol{x}) = h + \frac{V^2}{2g} \tag{7.13}$$

For simultaneous target localization and energy extraction, the problem can be defined as either a single or multi-objective constrained optimization problem, and was formulated for the subsequent cases as follows:

Case Study 1 - Maximize Energy and Minimize Visibility Error Function

$$\begin{aligned} &\text{minimize} \quad \int_{t_0}^{t_f} -E_h(\boldsymbol{x})\,\mathrm{d}t + \int_{t_0}^{t_f} erf(\boldsymbol{x})\,\mathrm{d}t \\ &\text{with respect to} \quad \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \quad \boldsymbol{u} = (\alpha, \gamma_1, \kappa, \tau)^T \end{aligned} \tag{7.14}$$

Case Study 2 - Minimize Visibility Error Function

$$\begin{aligned} &\text{minimize} \quad \int_{t_0}^{t_f} erf(\boldsymbol{x})\,\mathrm{d}t \\ &\text{with respect to} \quad \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \quad \boldsymbol{u} = (\alpha, \gamma_1, \kappa, \tau)^T \end{aligned} \tag{7.15}$$

Case Study 3 - Minimize the Ratio of the Error Function to Energy Extraction

$$\begin{aligned} &\text{minimize} \quad \frac{\int_{t_0}^{t_f} erf(\boldsymbol{x})\,\mathrm{d}t}{\int_{t_0}^{t_f} E_h(\boldsymbol{x})\,\mathrm{d}t} \\ &\text{with respect to} \quad \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \quad \boldsymbol{u} = (\alpha, \gamma_1, \kappa, \tau)^T \end{aligned} \tag{7.16}$$

Case Study 4 - Maximize a Weighted Energy and See-Ability Function

$$\begin{aligned} &\text{maximize} \quad K_{E_h} \int_{t_0}^{t_f} E_h(\boldsymbol{x})\,\mathrm{d}t + K_S \int_{t_0}^{t_f} S(\boldsymbol{x})\,\mathrm{d}t \\ &\text{with respect to} \quad \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \quad \boldsymbol{u} = (\alpha, \gamma_1, \kappa, \tau)^T \end{aligned} \tag{7.17}$$

Case Study 5 - Maximize Energy and use See-Ability as a Constraint

$$\begin{aligned} &\text{maximize} \quad \int_{t_0}^{t_f} E_h(\boldsymbol{x})\,\mathrm{d}t \\ &\text{with respect to} \quad \boldsymbol{x} = (V, \gamma_2, r, \theta_{th}, z, \varepsilon)^T \quad \boldsymbol{u} = (\alpha, \gamma_1, \kappa, \tau)^T \end{aligned} \tag{7.18}$$

Numerical difficulties were encountered for the first two problem formulations because of the design space becoming non-convex, due to the requirement of simultaneously optimizing two conflicting objective functions. As a result, the optimizer terminated early with numerical difficulties and optimal trajectories were not generated. The formulation of the final two case studies generated optimal trajectories but yielded trivial results. The see-ability formulation developed in Equation 7.7 as an optimization objective will force the aircraft to fly directly on top of the target at the minimum altitude constraint. As an example, Figure 7.5 shows a optimal trajectory for Case Study 4 that was weighted equally for see-ability and energy extraction. Using see-ability as a constraint produced optimal trajectories that were identical to those obtained for maximizing energy height.

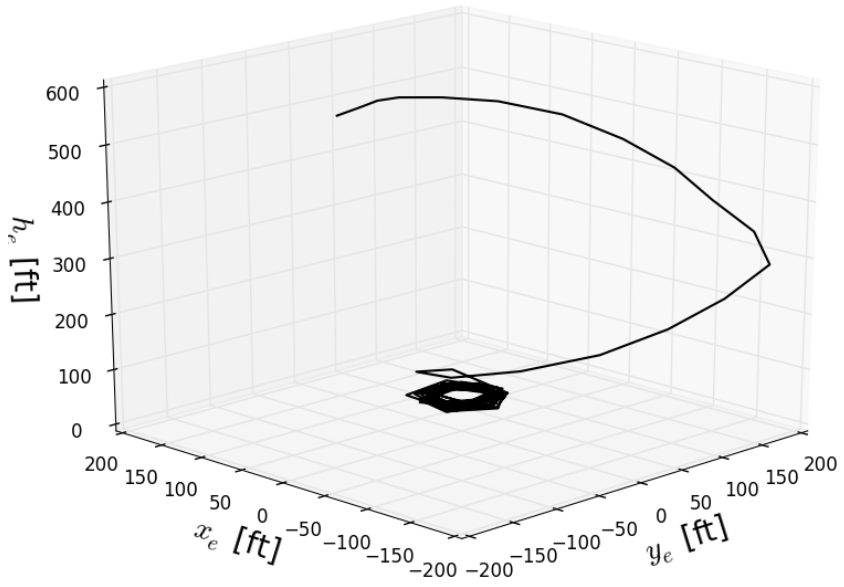


Figure 7.5: Optimal Trajectory for See-Ability Formulation

The only formulation that produced practically useable results was for Case Study 3. In addition to the previously defined path constraints, $\boldsymbol{C}$ and boundary conditions, $\boldsymbol{\Phi}$ found in Tables 5.1 and 5.2, additional requirements are outlined in Table 7.1. Results are presented for the chimney thermal model described in Section 4.3, N = 120 collocation points and for a fixed time interval of t = 120 s. Optimal trajectories will be generated for varying target locations, which will be specified with respect to the thermal center and are detailed in Table 7.2. For each target location, the air-

craft is positioned to fly tangential to the thermal core at the initial flight conditions detailed in Table 6.2. The initial position of the camera will be fixed facing forward with a depression angle of $\kappa_0 = -\pi/3$. The feasibility and optimality tolerance levels defining the convergence criteria for all cases was selected as $\upsilon = 1.0e^{-5}$.

Table 7.1: Definition of the Additional Path Constraints and Boundary Conditions

Camera Focal Length, f	0.138 [ft]
Individual CCD Element Size, d	1.64e^{-5} [ft]
Effective Pixel Resolution, G	1.64 [ft]
Length of Line of Sight Vector , $(d_{los})_{max}$	$g_8(\boldsymbol{x}) \leq d_{losmax} = 13,780$ [ft]
Camera Azimuth Rotation Angle Range, τ	$-\pi \leq \tau \leq \pi$
Camera Elevation Rotation Angle Range, κ	$-\frac{\pi}{2} \leq \kappa \leq -\frac{\pi}{5}$

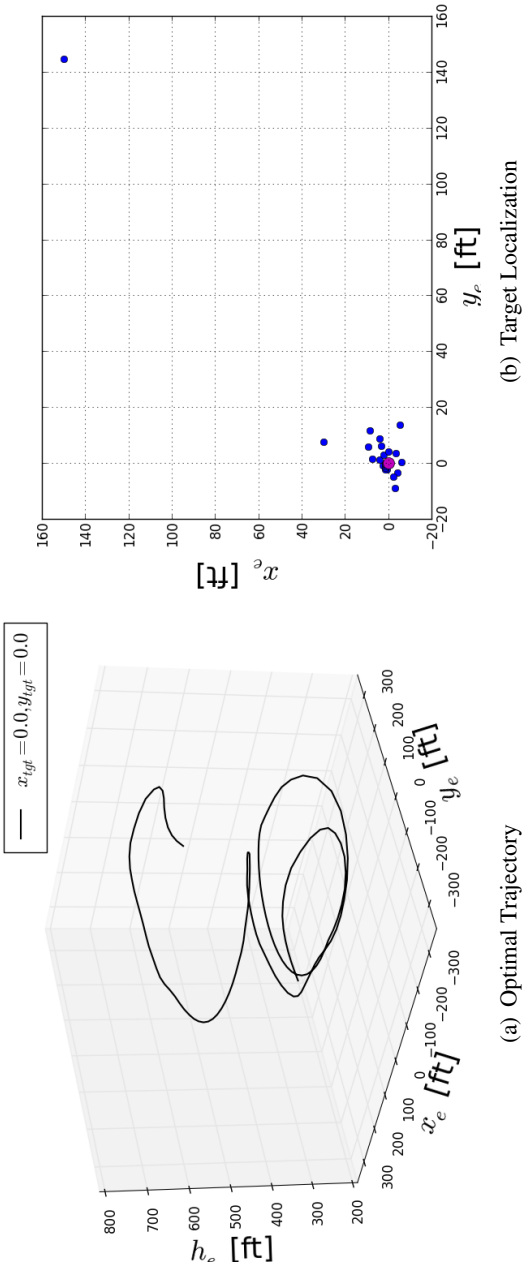
Table 7.2: Target Locations

Case	x_{tgt} [ft]	y_{tgt} [ft]
1	0.0	0.0
2	100.0	100.0
3	300.0	300.0
4	500.0	500.0

7.3.2 Optimal Trajectory Results

Using the chimney thermal model described in Section 4.3 with a thermal center located at $[x_t = 0.0, y_t = 0.0]$, optimal trajectories for varying target locations were generated and are shown in Figures 7.6(a), 7.7(a), 7.8(a), and 7.9(a). The thermal strength was consistent for all target locations and a month index = 6, or the month of July was selected for the analysis (see Table 6.1). Additionally, the location of the line of sight vector (blue dots) as compared to the target location (magenta dot) for each collocation point is shown in Figures 7.6(b), 7.7(b), 7.8(b), and 7.9(b).

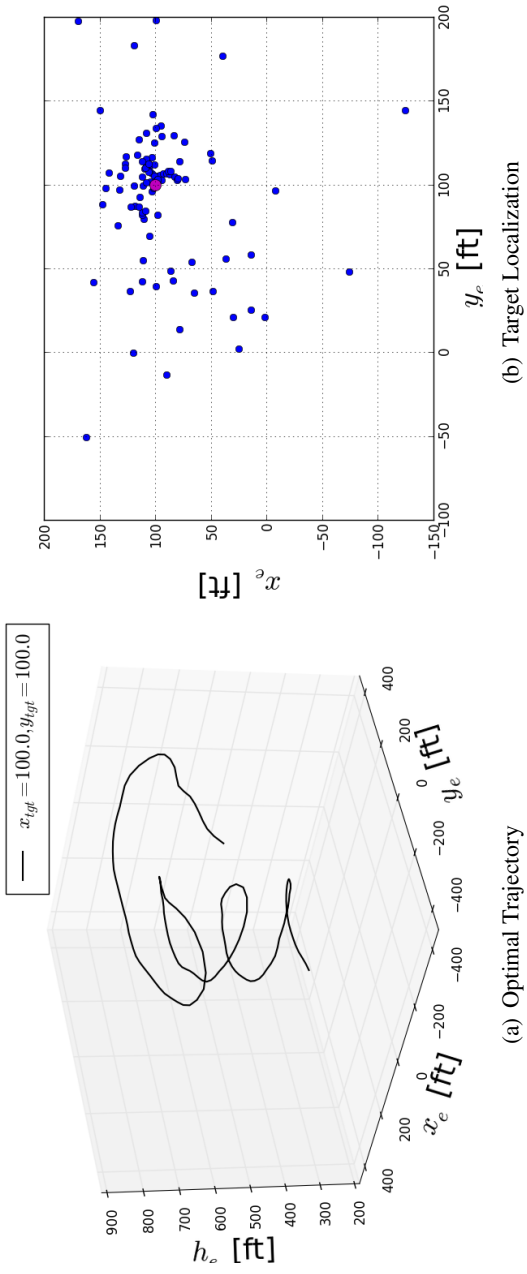
For each target location, the aircraft was able to use the thermal energy to perform a soaring climb and at the same time made an attempt to position itself to observe the target. The trajectories were not as smooth as those obtained for the thermal soaring climb; however and due to the conflicting objectives, the aircraft remained centered on the thermal core for the climb portion of the trajectory and extended further away from the core to better observe the target, which resulted in a trajectory that was oval in shape as shown in Figure 7.10. Additionally, the trajectories were extended towards the target location in an area with less thermal activity, at which time the



(a) Optimal Trajectory

(b) Target Localization

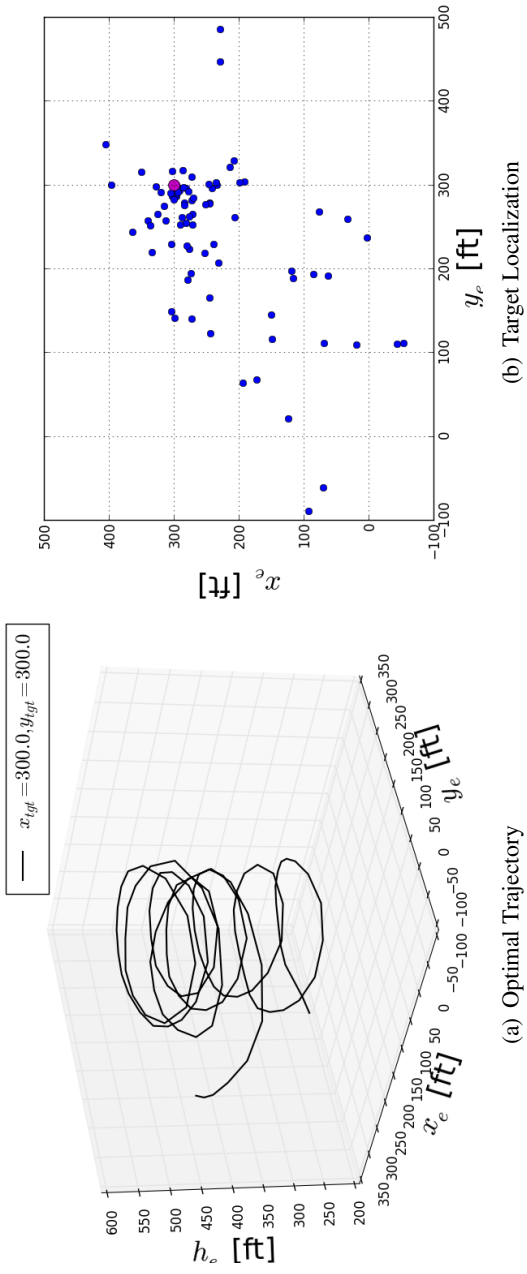
Figure 7.6: Optimal Trajectory for Target Localization - Case 1



(a) Optimal Trajectory

(b) Target Localization

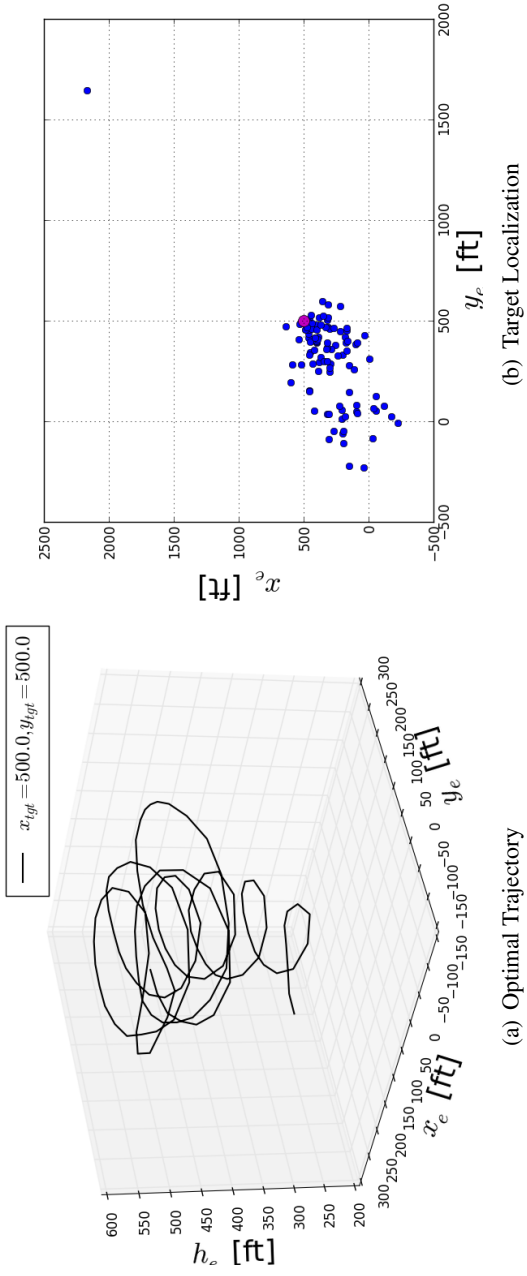
Figure 7.7: Optimal Trajectory for Target Localization - Case 2



(a) Optimal Trajectory

(b) Target Localization

Figure 7.8: Optimal Trajectory for Target Localization - Case 3



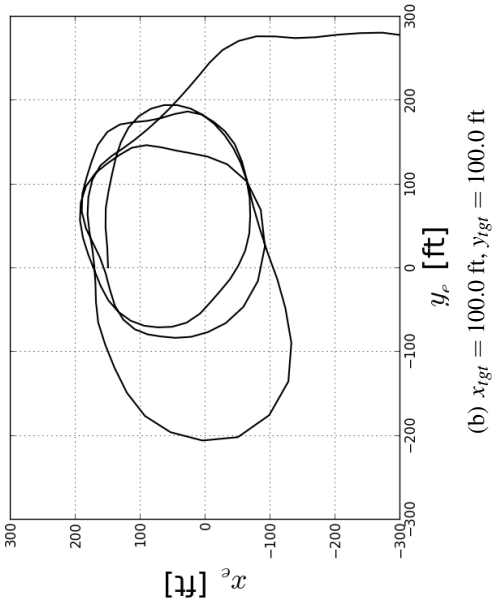
(a) Optimal Trajectory

(b) Target Localization

Figure 7.9: Optimal Trajectory for Target Localization - Case 4

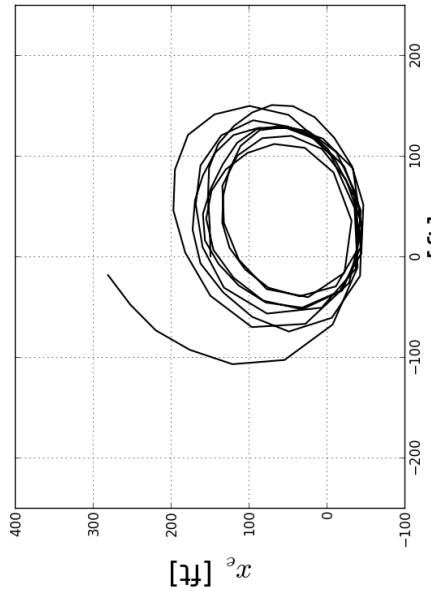
aircraft decelerated to maximize time on target. Upon reaching the target or an area with little to no thermal activity, the aircraft turned back and accelerated toward the thermal core to extract energy. The velocity profile and aircraft radial distance from the thermal center outlining the cyclic behaviour throughout the trajectory for various target locations is shown in Figure 7.11. The resulting objective function value, the computational effort required to solve the trajectories, and the overall optimization efficiency are presented in Table 7.3.

In this chapter, the autonomous soaring optimal control problem was extended for simultaneous optimization of thermal energy extraction and target localization for various target locations relative to the thermal center. For each target location, the aircraft was able to use the thermal energy to perform a soaring climb and at the same time made an attempt to position itself to observe the target. Despite obtaining somewhat predictable results, the objective function needs additional refinement to reduce the complexity of the optimization problem and increase its convexity. Additionally, the camera field of view also requires some further tuning to avoid numerical discontinuities that could arise in certain flight conditions during optimization.

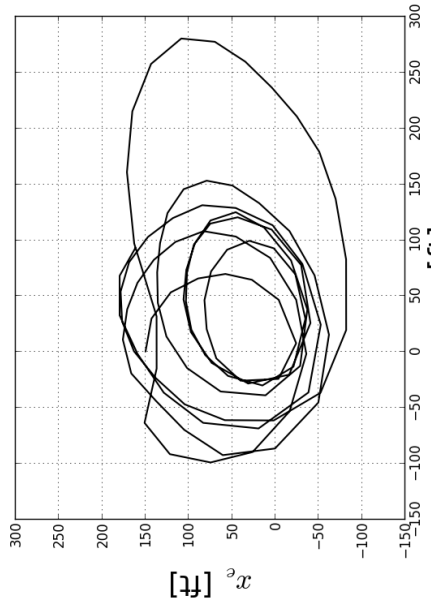


(a) $x_{tgt} = 0.0$ ft, $y_{tgt} = 0.0$ ft

(b) $x_{tgt} = 100.0$ ft, $y_{tgt} = 100.0$ ft

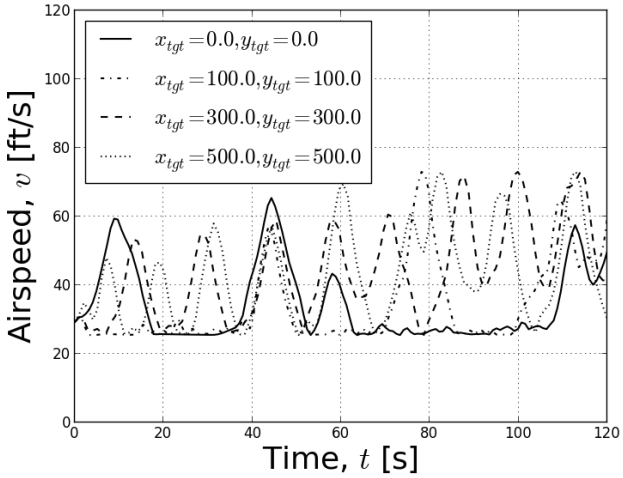


(c) $x_{tgt} = 300.0$ ft, $y_{tgt} = 300.0$ ft

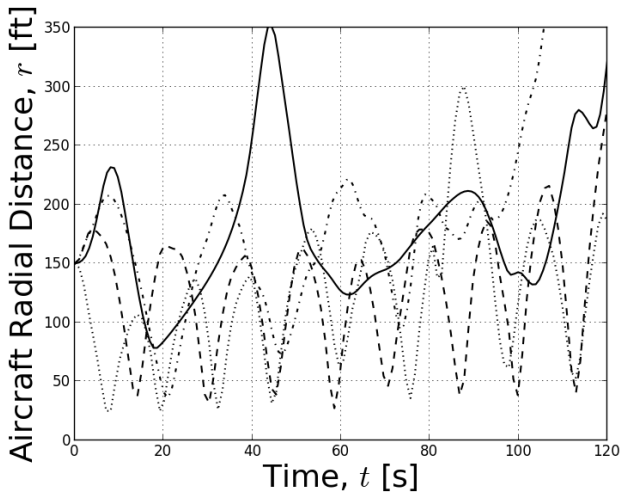


(d) $x_{tgt} = 500.0$ ft, $y_{tgt} = 500.0$ ft

Figure 7.10: Two Dimensional Trajectories for Target Localization



(a) Velocity Profile



(b) Aircraft Radial Distance from Thermal Core

Figure 7.11: Time History of Airspeed and Radial Distance for Target Localization

Table 7.3: Optimization Efficiency Results - Varying Target Locations

Target Location	$f(x^*)$ [ft]	No. of Function Calls	No. of Major Iterations	Time to Solve [hr]
$x_{tgt} = 0.0$ [ft], $y_{tgt} = 0.0$ [ft]	60.834	12,628	11,202	43.5
$x_{tgt} = 100.0$ [ft], $y_{tgt} = 100.0$ [ft]	490.69	2,672	2,148	5.47
$x_{tgt} = 300.0$ [ft], $y_{tgt} = 300.0$ [ft]	4526.3	1,883	1,520	3.47
$x_{tgt} = 500.0$ [ft], $y_{tgt} = 500.0$ [ft]	38951	2,101	607	2.76

8 Conclusions and Recommendations

The main objective of this thesis was to determine optimal flight trajectories for autonomous thermal soaring aircraft in an operating environment with a thermal of a known size and strength. This final chapter will provide concluding remarks, areas for improvement, and will also provide recommendations for future areas of research on this topic.

8.1 Conclusions

This thesis dissertation focused primarily on thermal centering strategies and achieving energy efficient climb rates within a chimney thermal that had been previously detected. The development of these strategies was approached numerically using trajectory optimization as the framework while employing energy methods to analyze aircraft performance and to define the objective function. To solve the autonomous soaring optimal control problem, the equations of motion were restructured in cylindrical coordinates and were augmented with wind component forces. A direct collocation with nonlinear programming approach for trajectory optimization was used to solve the autonomous thermal soaring optimal control problem. Additionally, a gradient-based optimizer that employed a Sequential Quadratic Programming (SQP) algorithm was used to solve the aforementioned nonlinear programming problem.

As a test case, a point-mass model of the Cularis D-5223 aircraft was developed as the primary test vehicle to be used in determining the optimal soaring trajectories. Additionally, a rigid-body model of the aircraft was created to understand the aerodynamic and dynamic characteristics of the raw aircraft and to ensure that it was fully controllable and could be trimmed.

Modeling the updraft strength was an additional component that was required to complete the formulation of the optimization problem. Using the conceptual notions of the structure of thermals, analytical and heuristic thermal representations were

developed to be used in the optimization formulation. The output of these developed thermal models were the wind force contributions that were used in the equations of motion to fully characterize the dynamic motion of the aircraft.

Optimal trajectories were generated and evaluated by varying the objective function, the thermal type and strength, and the initial starting locations of the aircraft. It was shown that the resulting optimal trajectories matched the predictive analytic approaches. As such, a strategy for thermal centering control was presented that could be implemented in real-time on Small Unmanned Aerial Vehicles (SUAVs). More specifically, it was observed that the commanded airspeed was minimally effected by the thermal strength, and was driven by the flight altitude and the aerodynamic characteristics of the aircraft to obtain a minimum sink rate condition. For the specific case of the Cularis aircraft, the stall speed was greater than that of the airspeed required for the minimum sink condition, and as such the minimal control speed constraint was active throughout most of the trajectory.

A key indication that the aircraft was fully centered on the thermal core was the condition where the specific energy acceleration approached zero. It was shown that the climb angle was directly related to the condition where $\ddot{E}_s = 0$ and could be calculated using a relationship that was presented for a given aircraft weight, wing geometry, and flight condition.

One of the aspects of the centering strategy that was not clearly apparent, was the specific relationship between the optimal angle of bank command that was required for a particular thermal strength. Qualitatively, it was shown that the aircraft radial distance to the thermal core varied between 45 and 75 ft with the larger radii observed for updrafts with higher thermal strengths. A graphical approach based on the analytical expressions was implemented to determine an optimum combination of the lift coefficient, C_L and angle of bank, γ_1 such that the sink rate would be a minimum at each radius of turn. For a turn radius between 40 to 100 ft, the optimal angle of bank range was between $\pi/9$ to $5\pi/36$ rad [20 to 25^{circ}] and matched the values obtained in the optimization results.

As an application, optimal flight trajectories were presented that maximized thermal energy extraction whilst providing persistent aerial surveillance coverage for a variety of thermal and target locations. Multiple formulations of the optimization problem were developed and considered both single and multi-objective constrained objective functions. For most of the formulations, either numerical difficulties were encountered and solutions did not converge, or trivial flight trajectories were produced. Expressing the objective function as a ratio of the target localization error function to the energy extracted produced converged and meaningful trajectories.

Optimal trajectories were generated for various target locations relative to the thermal center ensuring that a minimum digital spatial resolution was achieved along the full trajectory. A full development of the camera optics, and the homogeneous

transformations that were required to determine the camera viewing angle and the line of sight vector were also presented. For each target location, the aircraft was able to use the thermal energy to perform a soaring climb and at the same time made an attempt to position itself to observe the target. For simultaneous thermal soaring and persistent surveillance, the objective function needs additional refinement to reduce the complexity of the optimization problem and increase its convexity. Additionally, the camera field of view also requires some further tuning to avoid numerical discontinuities that could arise in certain flight conditions during optimization.

8.2 Recommendations for Future Work

Throughout the research, a number of potential areas for further advancement were identified and if implemented would contribute to the overall objective of operational autonomous thermal soaring flight. The thermal models should be refined to incorporate cross-wind components to investigate the skewing effects on the vertical updraft profile and correspondingly the optimal trajectory. Additionally, a more realistic model of the bubble thermal is required to better reflect its true atmospheric behaviour such that provisions are included to permit the bubble to detach from the Earth's surface, rise, and mix with other atmospheric layers.

Given that a thermal centering strategy was presented with the aim of achieving energy efficient climb rates, a controller should be developed to implement the strategy and verify its validity in a simulated environment. This work should also be extended to conduct hardware-in-the-loop testing using the avionics of the Cularis aircraft with the ultimate goal of performing flight tests. Additional refinement of the centering strategy is also required, in that an analytic prediction must be derived to predict the angle of bank to command for a given aircraft and thermal strength. Furthermore, it is also recommended that a further investigation be conducted for other glider aircraft to determine if they are also stall limited and are not permitted to fly at the minimum sink condition.

The final recommendation for future areas of research is in the field of path planning. In this investigation, optimal flight trajectories were generated that simultaneously maximized thermal energy extraction whilst providing persistent aerial surveillance coverage. It might be more prudent to take a phased approach of first maximizing energy and followed by observing a target to determine the optimal trajectories. Additionally, it should also be considered to redefine the mission strategy in that multiple cooperative SUAVs could be employed in the aforementioned phased approach to provide full enemy and target coverage. This problem could be further extended to investigate and solve for the trajectories using multiple thermal updraft of varying strength in a fixed operational zone with surveillance target reassignment.

Bibliography

[1] M. Denny. Dynamic soaring: Aerodynamics for albatrosses. *European Journal of Physics*, 30:75–84, 2009.

[2] FAA. Glider flying handbook. Flight Manual Handbook FAA-H-8083-13, U.S. Department of Transportation, 2003.

[3] Frank Irving. *The Paths of Soaring Flight*. Imperial College Press, Imperial College, London, 1999.

[4] H. Reichmann. *Cross-Country Soaring: A Handbook for Performance and Competition Soaring*. Thomson Publications, 1978.

[5] Z. Akos, M. Nagy, and T. Vicsek. Comparing bird and human soaring strategies. *PNAS - Proceedings of the National Academy of Sciences*, 105(11):4139–4143, March 2008.

[6] K. Cheng and J.W. Langelaan. Guided exploration for coordinated autonomous soaring flight. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2014-0969, National Habor, Maryland, United States, January 13-17 2014.

[7] N. Lawrance and S. Sukkarieh. Autonomous exploration of a wind field with a gliding aircraft. *Journal of Guidance Control and Dynamics*, 34(3):719–733, May-June 2011.

[8] K. Andersson, I. Kaminer, K.D. Jones, V. Dobrokhodov, and D.J. Lee. Cooperating uavs using thermal lift to extend endurance. In *AIAA Infotech Aerospace Conference*, number AIAA 2009-2043, Seattle, Washington, April 2009.

[9] M.J. Allen. Autonomous soaring for improved endurance of a small uninhabited air vehicle. In *43rd AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA 2005-1025, Reno, NV, January 10-13 2005.

[10] M.J. Allen. Guidance and control of an autonomous soaring uav. Technical Memorandum NASA / TM-2007-214611, NASA Dryden Flight Research Center, February 2007.

[11] M.J. Allen and V. Lin. Guidance and control of an autonomous soaring vehicle with flight test results. In *45th AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA 2007-867, Reno, Nevada, January 8 - 11 2007.

[12] K. Andersson, I. Kaminer, and V. Dobrokhodov. Thermal centering control for autonomous soaring: Stability analysis and flight test results. *Journal of Guidance Control and Dynamics*, 35(3):963 – 975, May - June 2012.

[13] K. Andersson and I. Kaminer. On stability of a thermal centering controller. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2009-6114, Chicago, Illinois, August 10 - 13 2009.

[14] N. Akhtar, A.K. Cooke, and J.F. Whidborne. Positioning algorithm for autonomous thermal soaring. *Journal of Aircraft*, 49(2):472 – 482, March - April 2012.

[15] D.J. Edwards. Implementation details and flight test results of an autonomous soaring controller. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, number AIAA 2008-7244, Honolulu, Hawaii, August 18 - 21 2008.

[16] S.C. Daugherty and J.W Langelaan. Improving autonomous soaring via energy state estimation and extremum seeking control. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2014-0260, National Harbor, Maryland, United States, January 13-17 2014.

[17] I.D. Cowling, Willcox. S., Y. Patel, P. Smith, and M. Roberts. Increasing persistence of uavs and mavs through thermal soaring. *The Aeronautical Journal*, 113(1145):479–489, July 2009.

[18] Y. Liu, S. Longo, and E. Kerrigan. Nonlinear predictive control of autonomous soaring uavs using 3dof models. In *European Control Conference*, pages 1365–1370, Zurich, Switzerland, July 17-19 2013.

[19] N.E. Kahveci, P.A. Ioannou, and D. Mirmirani. Adaptive lq control with anti-windup augmentation to optimize uav performance in autonomous soaring applications. *IEEE Transactions On Control Systems Technology*, 16(4):691–707, 2008.

[20] N. Lawrance and S. Sukkarieh. Path planning for autonomous soaring flight in dynamic wind fields. In *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9-13 2011.

[21] N.R.J. Lawrance and S. Sukkarieh. Wind energy based path planning for a small gliding unmanned aerial vehicle. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2009-6112, Chicago, Illinois, August 10 - 13 2009.

[22] A. Chakrabarty and J. Langelaan. Energy-based long-range path planning for soaring-capable unmanned aerial vehicles. *Journal of Guidance Control and Dynamics*, 4(4):1002 – 1015, July - August 2011.

[23] A. Chakrabarty and J.W. Langelaan. Energy maps for long-range path planning for small-and micro-uavs. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2009-6113, Chicago, Illinois, August 10 - 13 2009.

[24] J.W. Langelaan. Tree-based trajectory planning to exploit atmospheric energy. In *American Control Conference*, number ThA16.2, Seattle, Washington, June 11-13 2008.

[25] N.E. Kahveci and P.A. Ioannou. Genetic algorithms for shortest path routing of autonomous gliders. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, Hawaii, August 18 - 21 2008.

[26] W. Guo, Y.J. Zhao, and B. Capozzi. Optimal unmanned aerial vehicle flights for seeability and endurance in winds. *Journal of Aircraft*, 48(1):305–314, January-February 2011.

[27] J. Nguyen, N. Lawrance, R. Fitch, and S. Sukkarieh. Energy-constrained motion planning for information gathering with autonomous aerial soaring. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 6-10 2013.

[28] D. Makovkin and J.W. Langelaan. Optimal persistent surveillance using coordinated soaring. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2014-0261, National Habor, Maryland, United States, January 13-17 2014.

[29] S.S. Ponda, R.M. Kolacinski, and E. Frazzoli. Trajectory optimization for target localization using small unmanned aerial vehicles. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2009-6015, Chicago, Illinois, August 10-13 2009.

[30] M.E. Eshelby. *Aircraft Performance: Theory and Practice*. American Institute of Aeronautics and Astronautics, Inc., 2000.

[31] Anym. *Cularis: Building Instructions*. Multiplex Modellsport GmbH & Co. KG, 1 edition, 2007.

[32] Multiplex. RR Cularis. `http://www.multiplex-rc.de/en/home.html`, June 2014.

[33] R. Perez and J.R.R.A. Martins. pyACDT: An object-oriented framework for aircraft design modelling and multidisciplinary optimization. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, BC, September 2008. AIAA.

[34] H. Quabeck. HQ Original Profile. `http://www.hq-modellflug.de/koordinatenframe.htm`, July 2014.

[35] F. White. *Viscous Flow*. McGraw Hill, New York, NY, 1974.

[36] D.P. Raymer. *Aircraft Design: A Conceptual Approach*. AIAA Education Series. American Institute of Aeronautics and Astronautics, 3rd edition, 1999.

[37] J.D. Anderson. *Aircraft Performance and Design*. WCB McGraw-Hill, 1999.

[38] R. Bencatel, J. Tasso de Sousa, and A. Girard. Atmospheric flow field models applicable for aircraft endurance extension. *Progress in Aerospace Sciences*, 61:01–25, August 2013.

[39] M.J. Allen. Updraft model for development of autonomous soaring uninhabited air vehicles. In *44th AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA 2006-1510, Reno, NV, January 9-12 2006.

[40] A.J. Keane and P.B. Nair. *Computational Approaches for Aerospace Design*. John Wiley & Sons, Ltd., 2005.

[41] A.V. Rao. A survey of numerical methods for optimal control. In *AAS/AIAA Astrodynamics Specialist Conference*, number AAS 09-334, Pittsburgh, Pennsylvania, August 10-13 2009.

[42] J.T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2001.

[43] R. Perez, P. Jansen, and J. Martins. pyOpt: A python-based object-oriented framework for nonlinear constrained optimization. *Structural and Multidisciplinary Optimization*, 45:101–118, May 2011.

[44] A.E. Bryson and Y.-C. Ho. *Applied Optimal Control, Optimization, Estimation, and Control*. John Wiley & Sons, Ltd., 1975.

[45] O.W. Nicks. A simple total energy sensor. Technical Memorandum NASA TM X-73928, NASA Langley Research Center, March 1976.

[46] C.D. Cone. The design of sailplanes for optimum thermal soaring performance. Technical Note TN D-2052, NASA Langley Research Center, January 1964.

[47] W.G. Rees. *Physical Principles of Remote Sensing*. Cambridge University Press, 2001.

[48] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Ltd., 2006.

[49] R.C. Nelson. *Flight Stability and Automatic Control*. McGraw Hill, 2 edition, 1998.

[50] D.E. Hoak. The USAF stability and control DATCOM. Technical Report TR-83-3048, Air Force Wright Aeronautical Laboratories, October 1960 (Revised 1978).

[51] J. Roskam. *Airplane Design - Part V: Component Weight Estimation*. Design, Analysis and Research Corporation, 2003.

[52] R.L. Williams and D.A. Lawrence. *Linear State-Space Control Systems*. John Wiley & Sons, Ltd., 2007.

Appendices

A Characterization of the Aircraft Flight Dynamics

To complete the characterization of the Cularis D-5223 aircraft, an evaluation must be conducted on its stability and control characteristics and its dynamic response to a variety of perturbations. Additionally, to validate the assumptions made in the derivation of the point-mass model, it must be shown that the aircraft is fully controllable and can be trimmed.

A.1 Rigid Body Equations of Motion

To determine the flight dynamic characteristics of an aircraft, the aircraft must be treated as a rigid body vice a point-mass model. As such, rigid body equations of motion must be derived for both the longitudinal and lateral-directional cases. Unlike the point-mass model, the rigid body or flight dynamics equations of motion are developed in the wind axis system with respect to the body axes that is related by the aircraft's angle of attack, α. Newton's second law was used to derive these equations, which states that the summation of all external forces acting on a body is equal to the time rate of change of the momentum of the body in a fixed inertial reference frame [49]. Additionally, *Small-Disturbance Theory* was used to linearize the equations of motion about a trimmed flight condition. The forces and moments acting on an aircraft in the body axes are shown in Figure A.1. The full derivation of the rigid body equations of motion can be obtained from Reference [49], while a summary of the small-disturbance longitudinal and lateral-directional state equations are shown in Tables A.1 and A.2 respectively.

A.2 Dimensionless Stability and Control Coefficients

The stability coefficients are predicted using an analytic approach described in the *USAF Stability and Control DATCOM* [50]. The stability coefficients for the Cularis

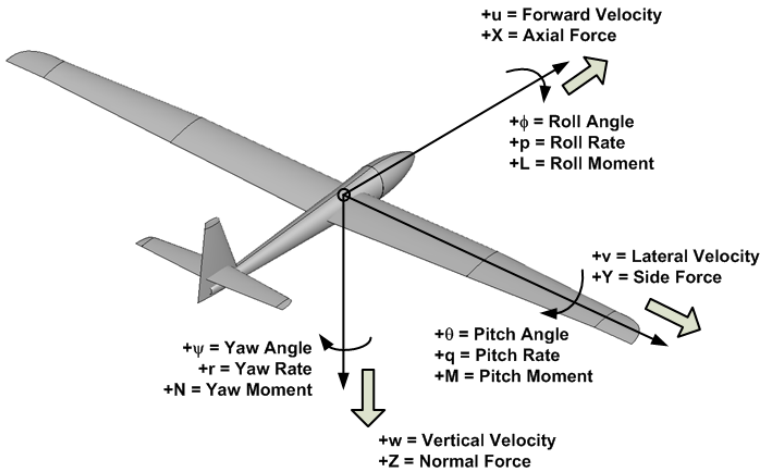


Figure A.1: Forces and Moments Acting on an Aircraft in the Body Axes [49]

Table A.1: Linearized Small-Disturbance Longitudinal Rigid Body State Equations [49]

$$\left(\frac{\mathrm{d}}{\mathrm{d}t}-X_u\right)\Delta u - X_w\Delta w + (g\cos\theta_0)\,\Delta\theta = X_{\delta_e}\Delta\delta_e + X_{\delta_T}\Delta\delta_T$$
$$-Z_u\Delta u + \left[(1-Z_{\dot{w}})\frac{\mathrm{d}}{\mathrm{d}t}-Z_w\right]\Delta w - \left[(u_0+Z_q)\frac{\mathrm{d}}{\mathrm{d}t} - g\sin\theta_0\right]\Delta\theta = Z_{\delta_e}\Delta\delta_e + Z_{\delta_T}\Delta\delta_T$$
$$-M_u\Delta u - \left(M_{\dot{w}}\frac{\mathrm{d}}{\mathrm{d}t}+M_w\right)\Delta w + \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} - M_q\frac{\mathrm{d}}{\mathrm{d}t}\right)\Delta\theta = M_{\delta_e}\Delta\delta_e + M_{\delta_T}\Delta\delta_T$$

Table A.2: Linearized Small-Disturbance Lateral-Directional Rigid Body State Equations [49]

$$\left(\frac{\mathrm{d}}{\mathrm{d}t}-Y_v\right)\Delta v - Y_p\Delta p + (u_0 - Y_r)\,\Delta r - (g\cos\theta_0)\,\Delta\phi = Y_{\delta_r}\Delta\delta_r$$
$$-L_v\Delta v + \left(\frac{\mathrm{d}}{\mathrm{d}t}-L_p\right)\Delta p - \left(\frac{I_{yz}}{I_{xx}}\frac{\mathrm{d}}{\mathrm{d}t}+L_r\right)\Delta r = L_{\delta_a}\Delta\delta_a + L_{\delta_r}\Delta\delta_r$$
$$-N_v\Delta v - \left(\frac{I_{xz}}{I_{zz}}\frac{\mathrm{d}}{\mathrm{d}t}+N_p\right)\Delta p + \left(\frac{\mathrm{d}}{\mathrm{d}t}-N_r\right)\Delta r = N_{\delta_a}\Delta\delta_a + N_{\delta_r}\Delta\delta_r$$

aircraft at a Mach number of $M = 0.05$ are presented for the state, control and rate coefficients in Tables A.3, A.4, and A.5 respectively. Minimal change was observed in the stability coefficients over the operating altitudes and for a given lift coefficient, C_L. Additionally, the aircraft moment of inertia matrix was estimated using Roskam's Class I method that assumed a radius of gyration could be identified for the aircraft [51]. The non-dimensional radius gyration for the Cularis is given as:

$$\bar{\boldsymbol{R}}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},} = \begin{bmatrix} \bar{R}_x \\ \bar{R}_y \\ \bar{R}_z \end{bmatrix} = \begin{bmatrix} 0.248 \\ 0.377 \\ 0.402 \end{bmatrix} \tag{A.1}$$

and the resulting inertia matrix is estimated as:

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} 0.167 & 0.000 & 0.005 \\ 0.000 & 0.087 & 0.000 \\ 0.005 & 0.000 & 0.238 \end{bmatrix} \text{slug ft}^2 \tag{A.2}$$

A.3 Dimensional Stability and Control Derivatives

In order to evaluate the flight dynamic characteristics of an aircraft, the non-dimensional coefficients need to be converted into a dimensional form. An analytic approach defined by Nelson [49] was used to perform this conversion, which is summarized for the longitudinal and lateral-directional derivatives in Tables A.6 and A.7 respectively. Additionally, the analytic approach could also be used to provide an approximation of the stability and control characteristics of the aircraft around a defined trim condition. The source code demonstrating this functionality is contained in Listing A.3 and it was developed as a *Dynamics* class to be used as part of the *pyACDT* framework [33].

Once dimensionalized, the derivatives are assembled in a linear state-space representation using the general form [52]:

$$\Delta \dot{\boldsymbol{x}} = A\Delta \boldsymbol{x} + B\Delta \boldsymbol{u} \tag{A.3}$$

where $\Delta \boldsymbol{x}$ is the change of state vector about trim, $\Delta \boldsymbol{u}$ is the change of input vector from trim, A is the evolution matrix, and B is the input matrix. For the longitudinal set

Table A.3: Dimensionless State Coefficients

Longitudinal	C_L	C_D	$C_{L\alpha}$	$C_{D\alpha}$	$C_{m\alpha}$	C_{Lu}	C_{Du}	C_{mu}	C_{Lq}	C_{Dq}	C_{mq}
1000 ft	0.3500	0.0257	5.8646	0.1333	-2.3974	0.0007	-0.0020	0.0008	6.2819	0.0000	-13.6796
2000 ft	0.3500	0.0265	5.8646	0.1374	-2.3974	0.0009	-0.0021	0.0010	6.2819	0.0000	-13.6796
1000 ft	0.6500	0.0332	5.8646	0.1969	-2.3974	0.0016	-0.0021	0.0018	6.2819	0.0000	-13.6796
2000 ft	0.6500	0.0333	5.8646	0.1969	-2.3974	0.0016	-0.0021	0.0018	6.2819	0.0000	-13.6796
1000 ft	0.9500	0.0446	5.8646	0.2752	-2.3974	0.0024	-0.0021	0.0026	6.2819	0.0000	-13.6796
2000 ft	0.9500	0.0447	5.8646	0.2752	-2.3974	0.0024	-0.0021	0.0026	6.2819	0.0000	-13.6796
Lateral	$C_{y\beta}$	$C_{l\beta}$	$C_{n\beta}$	C_{yp}	C_{lp}	C_{np}	C_{yr}	C_{lr}	C_{nr}		
1000 ft	-0.2943	-0.0635	0.0512	-0.0286	-0.6733	-0.0390	0.1013	0.1298	-0.0333		
2000 ft	-0.2943	-0.0634	0.0514	-0.0277	-0.6731	-0.0453	0.1087	0.1410	-0.0339		
1000 ft	-0.2943	-0.0626	0.0533	-0.0221	-0.6711	-0.0826	0.1529	0.2083	-0.0384		
2000 ft	-0.2943	-0.0626	0.0533	-0.0221	-0.6711	-0.0826	0.1529	0.2083	-0.0384		
1000 ft	-0.2943	-0.0618	0.0562	-0.0165	-0.6675	-0.1200	0.1968	0.2754	-0.0448		
2000 ft	-0.2943	-0.0618	0.0562	-0.0165	-0.6675	-0.1200	0.1968	0.2754	-0.0449		

Table A.4: Dimensionless Control Coefficients

Longitudinal	C_L	$C_{L\delta e}$	$C_{D\delta e}$	$C_{m\delta e}$		
1000 ft	0.3500	0.6726	0.0000	-2.7618		
2000 ft	0.3500	0.6726	0.0000	-2.7618		
1000 ft	0.6500	0.6726	0.0000	-2.7618		
2000 ft	0.6500	0.6726	0.0000	-2.7618		
1000 ft	0.9500	0.6726	0.0000	-2.7618		
2000 ft	0.9500	0.6726	0.0000	-2.7618		
Lateral	$C_{y\delta a}$	$C_{l\delta a}$	$C_{n\delta a}$	$C_{y\delta r}$	$C_{l\delta r}$	$C_{n\delta r}$
1000 ft	0.0000	0.2341	0.0023	0.1601	0.0112	-0.0423
2000 ft	0.0000	0.2341	0.0026	0.1601	0.0108	-0.0424
1000 ft	0.0000	0.2341	0.0049	0.1601	0.0086	-0.0429
2000 ft	0.0000	0.2341	0.0049	0.1601	0.0086	-0.0429
1000 ft	0.0000	0.2341	0.0071	0.1601	0.0064	-0.0433
2000 ft	0.0000	0.2341	0.0071	0.1601	0.0064	-0.0433

Table A.5: Dimensionless State Rate Coefficients

Longitudinal	C_L	$C_{L\dot{\alpha}}$	$C_{D\dot{\alpha}}$	$C_{m\dot{\alpha}}$
1000 ft	0.3500	1.0709	0.0000	-4.3977
2000 ft	0.3500	1.0709	0.0000	-4.3977
1000 ft	0.6500	1.0709	0.0000	-4.3977
2000 ft	0.6500	1.0709	0.0000	-4.3977
1000 ft	0.9500	1.0709	0.0000	-4.3977
2000 ft	0.9500	1.0709	0.0000	-4.3977

Table A.6: Summary of Longitudinal Derivatives [49]

$X_u = \frac{-(C_{Du}+C_{D0})QS}{mV}$	$X_w = \frac{-(C_{D\alpha}-C_{L0})QS}{mV}$
$Z_u = \frac{-(C_{Lu}+2C_{L0})QS}{mV}$	$X_{\delta_e} = \frac{C_{x\delta e}QS}{m}$
$Z_w = \frac{-(C_{L\alpha}+C_{D0})QS}{mV}$	$Z_{\dot{w}} = -C_{z\dot{\alpha}}\frac{\bar{c}_w}{2V}\frac{QS}{mV}$
$Z_\alpha = VZ_w$	$Z_{\dot{\alpha}} = VZ_{\dot{w}}$
$Z_q = -C_{zq}\frac{\bar{c}_w}{2V}\frac{QS}{m}$	$Z_{\delta_e} = -C_{z\delta e}\frac{QS}{m}$
$M_u = C_{mu}\frac{QS\bar{c}_w}{VI_{yy}}$	
$M_w = C_{m\alpha}\frac{QS\bar{c}_w}{VI_{yy}}$	$M_{\dot{w}} = C_{m\dot{\alpha}}\frac{\bar{c}_w}{2V}\frac{QS\bar{c}_w}{VI_{yy}}$
$M_\alpha = VM_w$	$M_{\dot{\alpha}} = VM_{\dot{w}}$
$M_q = C_{mq}\frac{\bar{c}_w}{2V}\frac{QS\bar{c}_w}{I_{yy}}$	$M_{\delta_e} = C_{m\delta e}\frac{QS\bar{c}_w}{I_{yy}}$

Table A.7: Summary of Lateral-Directional Derivatives [49]

$Y_\beta = \frac{QSC_{y_\beta}}{m}$	$N_\beta = \frac{QSbC_{n_\beta}}{I_{zz}}$	$L_\beta = \frac{QSbC_{l_\beta}}{I_{xx}}$
$Y_p = \frac{QSbC_{y_p}}{2mV}$	$N_p = \frac{QSb^2C_{n_p}}{2I_{xx}V}$	$L_p = \frac{QSb^2C_{l_p}}{2I_{xx}V}$
$Y_r = \frac{QSbC_{y_r}}{2mV}$	$N_r = \frac{QSb^2C_{n_r}}{2I_{xx}V}$	$L_r = \frac{QSb^2C_{l_r}}{2I_{xx}V}$
$Y_{\delta_a} = \frac{QSC_{y_{\delta_a}}}{m}$	$Y_{\delta_r} = \frac{QSC_{y_{\delta_r}}}{m}$	
$N_{\delta_a} = \frac{QSbC_{n_{\delta_a}}}{I_{zz}}$	$N_{\delta_r} = \frac{QSbC_{n_{\delta_r}}}{I_{zz}}$	
$L_{\delta_a} = \frac{QSbC_{l_{\delta_a}}}{I_{xx}}$	$L_{\delta_r} = \frac{QSbC_{l_{\delta_r}}}{I_{xx}}$	

of linearized equations about the trim condition detailed in Table A.1, the state-space representation for an aircraft in gliding flight (i.e. $\Delta\delta_T = 0$) is given as [49]:

$$\begin{bmatrix} \Delta\dot{u} \\ \Delta\dot{w} \\ \Delta\dot{q} \\ \Delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} X_u & X_w & 0 & -g \\ Z_u & Z_w & u_0 & 0 \\ M_u + M_{\dot{w}}Z_u & M_w + M_{\dot{w}}Z_w & M_q + M_{\dot{w}}u_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta w \\ \Delta q \\ \Delta\theta \end{bmatrix} + \begin{bmatrix} X_{\delta_e} \\ Z_{\delta_e} \\ M_{\delta_e} + M_{\dot{w}}Z_{\delta_e} \\ 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_e \end{bmatrix} \tag{A.4}$$

while the state-space representation for the lateral-directional set of linearized equations about the trim condition defined in Table A.2 is given as [49]:

$$\begin{bmatrix} \Delta\dot{\beta} \\ \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{Y_\beta}{u_0} & \frac{Y_p}{u_0} & -\left(1 - \frac{Y_r}{u_0}\right) & \frac{g\cos\theta}{u_0} \\ L_\beta & L_p & L_r & 0 \\ N_\beta & N_p & N_r & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\beta \\ \Delta p \\ \Delta r \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} 0 & \frac{Y_{\delta_r}}{V} \\ L_{\delta_a} & L_{\delta_r} \\ N_{\delta_a} & N_{\delta_r} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_a \\ \Delta\delta_r \end{bmatrix} \tag{A.5}$$

A.4 Longitudinal Flight Dynamics

Using the non-dimensional coefficients outlined in Tables A.3 and A.4, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices for the state-space representation of the longitudinal flight dynamics of the unaugmented Cularis aircraft is given as:

$$\boldsymbol{A}_{long} = \begin{bmatrix} -0.139 & 1.031 & 0.000 & -32.200 \\ -2.935 & -11.608 & 55.630 & 0.000 \\ 0.125 & -4.031 & -9.449 & 0.000 \\ 0.000 & 0.000 & 1.000 & 0.000 \end{bmatrix} \tag{A.6}$$

$$\boldsymbol{B}_{long} = \begin{bmatrix} 0.000 \\ -73.600 \\ -286.039 \\ 0.000 \end{bmatrix} \tag{A.7}$$

The longitudinal flight dynamic system is defined as fully controllable and observable. The eigenvalues, damping ratio, and undamped natural frequency for the short period mode are calculated as:

$$\lambda_{sp} = -10.531 \pm 14.996\text{i}$$

$$\zeta_{sp} = 0.299 \quad \omega_{nsp} = 15.843 \text{ rad/s}$$

The short period response is characterized as being stable and underdamped with a period of 0.4s. The eigenvalues, damping ratio, and natural frequency for the phugoid mode are:

$$\lambda_{lp} = -0.067 \pm 1.127\text{i}$$

$$\zeta_{lp} = 0.053 \quad \omega_{nlp} = 1.303 \text{ rad/s}$$

The phugoid has a period 4.8s, where its long term dynamic response would be highly oscillatory and marginally stable.

A.5 Lateral-Directional Flight Dynamics

Using the non-dimensional coefficients outlined in Tables A.3 and A.4, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices for the state-space representation of the lateral-directional flight dynamics of the unaugmented Cularis aircraft is given as:

$$\boldsymbol{A}_{lat,dir} = \begin{bmatrix} -0.579 & -0.003 & -0.975 & 0.577 \\ -52.047 & -42.902 & 14.699 & 0.000 \\ 31.654 & -4.237 & -1.806 & 0.000 \\ 0.000 & 1.000 & 0.000 & 0.000 \end{bmatrix} \tag{A.8}$$

$$\boldsymbol{B}_{lat,dir} = \begin{bmatrix} 0.000 & 0.315 \\ 195.382 & 6.613 \\ 3.274 & -25.148 \\ 0.000 & 0.000 \end{bmatrix} \tag{A.9}$$

The lateral-direction flight dynamic system is defined as fully controllable and observable. The eigenvalues for both the spiral and roll modes are calculated as:

$$\lambda_{spiral} = 0.128$$

$$\lambda_{roll} = -41.518$$

The spiral mode is unstable, while the roll mode is stable. The eigenvalues, damping ratio, and natural frequency for the Dutch-Roll mode are:

$$\lambda_{DR} = -1.947 \pm 6.049\text{i}$$

$$\zeta_{DR} = 0.211 \quad \omega_{nDR} = 5.648 \text{ rad/s}$$

The Dutch-Roll mode has a period 1.1s, and its long term dynamic response would be oscillatory and stable.

Most of the dynamic modes were found to be stable and would converge when disturbed from trim. However, it is recommended that a stability augmentation system should be designed and implemented to dampen out undesirable disturbances and improve the flying qualities in all axes.

Listing A.1: Model of the Aircraft Geometry

```
#!/usr/bin/env python
'''
Multiplex Cularis D-5223


Developers:
-----------
- Major Thomas R. Connerty (TC)

History
-------
    v. 1.0 - Initial Creation (TC, 2013)
'''

__version__ = '$Revision: $'

'''
To Do:
    -
'''

# ==============================================================================
# Standard Python modules
# ==============================================================================
import os, sys

# ==============================================================================
# External Python modules
# ==============================================================================
import numpy

# ==============================================================================
# Extension modules
# ==============================================================================
sys.path.append(os.path.abspath('../../../..'))
sys.path.append(os.path.abspath('../../../../pyACDT/Aerodynamics'))
sys.path.append(os.path.abspath('../../../../pyACDT/Atmosphere/pyAtmos'))
sys.path.append(os.path.abspath
('../../../../pyACDT/Atmosphere/pyAtmos/pyICAO'))
sys.path.append(os.path.abspath('../../../../pyACDT/Fuel'))
sys.path.append(os.path.abspath('../../../../pyACDT/Geometry'))
sys.path.append(os.path.abspath('../../../../pyACDT/Payload'))
sys.path.append(os.path.abspath('../../../../pyACDT/Propulsion'))
sys.path.append(os.path.abspath('../../../../pyACDT/Systems'))
sys.path.append(os.path.abspath('../../../../pyACDT/Tools/Interpolation'))
sys.path.append(os.path.abspath('../../../../pyACDT/Tools/Search'))
sys.path.append(os.path.abspath
('../../../../pyACDT/Visualization/pySciPlot/'))

from pyGeometry_aircraft import Aircraft
from pyGeometry_bodysurface import BodySurface
from pyGeometry_liftingsurface import LiftingSurface
from pyGeometry_liftingsurface import LiftingSegment
from pyGeometry_wingtip import Wingtip
```

```
# ============================================================================
# Aircraft Geometric Model
# ============================================================================

input = {
    'Name':'Cularis D-5223 - scale 1:1 <ft>',
    '_components':{

        0:BodySurface({
            'Name':'',
            'Group':'Fuselage',
            'Type':'Closed',
            'xLoc':0.0,'yLoc':0.0,'zLoc':0.0,
            'xRot':0.0,'yRot':0.0,'zRot':0.0,
            '_components':{
                0:{'Name':'Section 0','Length':0.8229,'fwd_Radius':0.00000,
                'fwd_Ratio':1.000,'fwd_rShape':1.0,'fwd_lShape':1.2,
                'fwd_vOffset':0.0,'aft_Radius':0.20189,'aft_Ratio':1.265,
                'aft_lShape':1.0,'aft_rShape':1.0,'aft_vOffset': 0.000000},
                1:{'Name':'Section 1','Length':3.0208,'fwd_Radius':0.20189,
                'fwd_Ratio':1.235,'fwd_rShape':1.0,'fwd_lShape':1.0,
                'fwd_vOffset':0.0,'aft_Radius':0.06272,'aft_Ratio':1.182,
                'aft_lShape':1.4,'aft_rShape':0.8,'aft_vOffset': 0.000000,
                'dySlope':0.4,'dzSlope':0.35}
            },
        }),

        1:LiftingSurface({
            'Name':'',
            'Group':'Wing',
            'Symmetry':True,
            'xrLE':1.125,'yrLE':0.0,'zrLE':0.0833,
            'xRot':0.0,'yRot':0.0,'zRot':0.0,
            '_components':{
                0:LiftingSegment({'Name':'Segment 1','Type':'external',
                'Area':1.30512,'Span':2.0208333,'Taper':0.8788,
                'SweepLE':2.36,'Dihedral':2.5,'xc_offset':0.0,
                'root_Incidence':5.0,'root_Thickness':0.12,
                'root_Airfoil_type':'datafile',
                'root_Airfoil_ID':'HQ_1_0_12.dat','tip_Incidence':4.0,
                'tip_Thickness':0.12,'tip_Airfoil_type':'datafile',
                'tip_Airfoil_ID':'HQ_1_0_12.dat',
                    '_components':{
                        0:{'Name':'TE Flap','Group':'TE Flap',
                        'Type':'Plain','Deflection':0.0,
                        'sym_Deflection':True,
                        'max_Deflection': +3.612,'min_Deflection': -4.818,
                        'inboard_Station':0.155,'outboard_Station':1.00,
                        'inboard_fwd_Station':0.71875,
                        'inboard_aft_Station':1.00,
                        'outboard_fwd_Station':0.75862,
                        'outboard_aft_Station':1.00}},
                    }),
                1:LiftingSegment({'Name':'Segment 2','Type':'external',
                'Area':0.87885,'Span':1.7395833,'Taper':0.6724,
                'SweepLE':6.53,'Dihedral':2.5,'xc_offset':0.0,
```

```
            'root_Incidence':4.0,'root_Thickness':0.12,
            'root_Airfoil_type':'datafile',
            'root_Airfoil_ID':'HQ_1_0_12.dat','tip_Incidence':3.0,
            'tip_Thickness':0.12,'tip_Airfoil_type':'datafile',
            'tip_Airfoil_ID':'HQ_1_0_12.dat',
                '_components':{
                    0:{'Name':'Aileron','Group':'Aileron',
                    'Type':'Plain','Deflection':0.0,
                    'sym_Deflection':False,
                    'max_Deflection':+31.664,'min_Deflection':-15.217,
                    'inboard_Station':0.000,'outboard_Station':1.00,
                    'inboard_fwd_Station':0.75862,
                    'inboard_aft_Station':1.00,
                    'outboard_fwd_Station':0.76923,
                    'outboard_aft_Station':1.00}},
                }),
            2:Wingtip({'Type':'wingtip','shape':1.0,'bias':1.0,
            'scale':1.25,'location':1.0,'curvature':1.0})
        },
    }),

    2:LiftingSurface({
        'Name':'Horizontal Stabilizer',
        'Group':'Horizontal Tail',
        'Symmetry':True,
        'xrLE':3.5208,'yrLE':0.0,'zrLE':0.1979,
        'xRot':0.0,'yRot':0.0,'zRot':0.0,
        '_components':{
            0:{'Name':'','Type':'external','Area':0.30556,
            'Span':0.91667,'Taper':0.600,'SweepLE':6.48,'Dihedral':0.0,
            'xc_offset':0.0,'root_Incidence':0.0,'root_Thickness':0.12,
            'root_Airfoil_type':'naca4','root_Airfoil_ID':'00xx',
            'tip_Incidence':0.0,'tip_Thickness':0.09,
            'tip_Airfoil_type':'naca4','tip_Airfoil_ID':'00xx',
                '_components':{
                    0:{'Name':'Elevator','Group':'Elevator',
                    'Deflection':0.0,'sym_Deflection':True,
                    'max_Deflection':+14.18,
                    'min_Deflection':-14.18,'inboard_Station':0.00,
                    'outboard_Station':1.00,'inboard_fwd_Station':0.0,
                    'inboard_aft_Station':1.0,
                    'outboard_fwd_Station':0.0,
                    'outboard_aft_Station':1.0}
                },
            },
            1:Wingtip({'Type':'wingtip','shape':1.0,'bias':0.5,
            'scale':0.33,'location':1.0,'curvature':1.0})
        },
    }),

    3:LiftingSurface({
        'Name':'Vertical Stabilizer',
        'Group':'Vertical Tail',
        'Symmetry':False,
        'xrLE':3.25,'yrLE':0.0,'zrLE':-0.06771,
        'xRot':0.0,'yRot':0.0,'zRot':0.0,
```

```
            '_components':{
                0:LiftingSegment({'Name':'','Type':'external',
                'Area':0.50911,'Span':0.95833,'Taper':0.3421,
                'SweepLE':28.52,'Dihedral':90.0,'xc_offset':0.0,
                'root_Incidence':0.0,'root_Thickness':0.09,
                'root_Airfoil_type':'naca4','root_Airfoil_ID':'00xx',
                'tip_Incidence':0.0,'tip_Thickness':0.16,
                'tip_Airfoil_type':'naca4','tip_Airfoil_ID':'00xx',
                    '_components':{
                        0:{'Name':'Rudder','Group':'Rudder',
                        'Deflection':0.0,'sym_Deflection':True,
                        'max_Deflection':+24.26,'min_Deflection':-24.26,
                        'inboard_Station':0.00,'outboard_Station':1.00,
                        'inboard_fwd_Station':0.68421,
                        'inboard_aft_Station':1.0,
                        'outboard_fwd_Station':0.5,
                        'outboard_aft_Station':1.0,},
                    },
                }),
                1:Wingtip({'Type':'wingtip','shape':1.0,'bias':0.5,
                'scale':0.3077,'location':1.0,'curvature':1.0})
            },
        }),
    },
}
ac = Aircraft(input)

#====================================================================================
#
#====================================================================================
if __name__ == '__main__':

    #ac.plotPlanform()
    fig = ac.plotSurface()
    #ac[1].Equivalent_Surface.plotSurface(fig)
    #ac.writeDAT('Cularis_D5223')
```

Listing A.2: Gliding Module within *pyACDT*

```
#!/usr/local/bin/python
'''
pyGliding_gliding

Analytic Prediction of Gliding Flight Characteristics for an Aircraft

Copyright (c) 2004-2014 by pyACDT Developers
All rights reserved.
Revision: 1.0  $Date: 02/04/2014 21:00$

References:
-----------



Developers:
-----------
```

```
- Major Thomas R. Connerty (TC)

History:
--------
    v. 1.0 - Initial Creation in Python (TC, 2014)

'''

__version__ = '$Revision: $'

'''
To Do:
    -
'''

# ============================================================================
# Standard Python modules
# ============================================================================
import os, math, sys, time
import pdb # pdb.set_trace()

# ============================================================================
# External Python modules
# ============================================================================

import numpy

# ============================================================================
# Extension modules
# ============================================================================

sys.path.append(os.path.abspath('..'))
sys.path.append(os.path.abspath('../Trajectory_Optimization'))

from pyAtmosphere_US1976 import atmosphere
from pyGliding_object import glidingObject

# ============================================================================
# Misc Definitions
# ============================================================================


# ============================================================================
# Define the Function to Determine Gliding Flight Characteristics of Cularis
# ============================================================================

class GlidingFlight(glidingObject):

    def __init__(self,aircraft,*args, **kwargs):

        '''
        Gliding Model Class Initialization

        *Arguments:*

        - aircraft -> DICTIONARY: Define the Aircraft Parameters
```

```
        Documentation last updated:  Apr 16, 2014 - Thomas R. Connerty
        '''

        #
        name = 'Gliding Flight Class'
        def_opts = {}

        # Initialize Abstract Class
        glidingObject.__init__(self,name, def_opts, *args, **kwargs)

        #
        self.b = aircraft['b']
        self.S = aircraft['S']
        self.AR = (self.b ** 2.0) / self.S
        self.W = aircraft['W']
        self.CLo = aircraft['CLo']
        self.CLa = aircraft['CLa']
        self.CLmax = aircraft['CLmax']
        self.e = aircraft['e']
        self.CDo = aircraft['CDo']
        self.K = 1.0 / (math.pi * self.e * self.AR)
        self.LDmax = math.sqrt(1/(4.0 * self.CDo * self.K))
        self.n_max = aircraft['n_max']
        self.n_min = aircraft['n_min']
        self.Vmo = aircraft['Vmo']


    def __call__(self, altitude=0.0, V=0.0, gamma_1=0.0, gamma_2=0.0,
        *args, **kwargs):

        '''
        Gliding Flight Class (Calling Routine)

        **Keyword Arguments:**

        - altitude -> FLOAT: Aircraft Altitude (ft), *Default* = 0.0
        - V -> FLOAT: Aircraft True Airspeed (ft/s), *Default* = 0.0
        - gamma_1 -> FLOAT: Aircraft Bank Angle (rad), *Default* = 0.0
        - gamma_2 -> FLOAT: Aircraft Climb Angle (rad), *Default* = 0.0

        *Outputs:*

        - data -> DICTIONARY: Outputs the Gliding Flight Characteristics

        Documentation last updated:  Apr 16, 2014 - Thomas R. Connerty
        '''

        # ======================================================================
        # Calculate the Atmospheric and Aerodynamic Characteristics
        # ======================================================================

        # Atmospheric Conditions
        conditions = atmosphere(altitude,'ENG')
        rho = conditions['Density']
```

```
        # Acceleration due to Gravity                          ft/s ** 2.0
        g = 32.174

        # Lift-to-Drag Ratio for Minimum Sink                  Dimensionless
        LDmax_minSink = 0.25 * ((3.0 / (self.K * \
        (self.CDo ** (1.0/3.0)))) ** 0.75)
        self.LDmax_minSink = LDmax_minSink

        # Determine the Best Glide Angle                       rad
        theta_min = -math.atan(1.0 / self.LDmax)

        # Calculate the Lift Coefficient                       Dimensionless
        CL = (self.W / self.S) * (2.0 / rho) * (1.0 / \
        (math.cos(gamma_1) * (V ** 2.0)))

        # Calculate the Drag Coefficient                       Dimensionless
        CD = self.CDo + self.K * (CL ** 2.0)

        # Calculate the Angle of Attack                        rad
        alpha = (CL - self.CLo) / self.CLa

        # ======================================================================
        # Calculate the Aircraft Sink Rate and Flight Radius
        # ======================================================================

        # Calculate the Aircraft Sink Rate                     ft/s
        h_dot = -((self.W / self.S) ** 0.5) * ((2.0 / rho) ** 0.5) * \
        ((1.0 / math.cos(gamma_1)) ** 1.5) * (CD / (CL ** 1.5)) * \
        ((math.cos(gamma_2)) ** 1.5)

        if gamma_1 == 0.0:

            # Calculate the Radius to Fly in the Thermal
            R = (self.W / self.S) * (2.0 / (rho * g)) * (1.0 / CL) * \
            ((math.cos(gamma_2)) ** 2.0)

        else:

            # Calculate the Radius to Fly in the Thermal
            R = (self.W / self.S) * (2.0 / (rho * g)) * (1.0 / \
            (math.sin(gamma_1) * CL)) * ((math.cos(gamma_2)) ** 2.0)
        #end

        data = {'LDmax_minSink': LDmax_minSink, 'theta_min': theta_min,
        'CL': CL, 'CD': CD, 'alpha': alpha, 'h_dot': h_dot, 'R': R}

        return data

    def getVelocities(self, altitude=0.0, gamma_1=0.0, *args, **kwargs):

        '''
        Computes Velocities of the Aircraft

        **Keyword Arguments:**

        - altitude -> FLOAT: Aircraft Altitude (ft), *Default* = 0.0
```

```
        - gamma_1 -> FLOAT: Aircraft Bank Angle (rad), *Default* = 0.0

        *Outputs:*

        - velocities -> DICTIONARY: Outputs the various
            Velocities for Gliding Flight

        Documentation last updated:  Apr 16, 2014 - Thomas R. Connerty
        '''

        # Atmospheric Conditions
        conditions = atmosphere(altitude,'ENG')
        rho = conditions['Density']

        # Calculate the Stall Speed                          ft/s
        Vstall = math.sqrt((2.0 / rho) * (self.W / self.S) * \
        (1.0 / self.CLmax) * (1.0 / math.cos(gamma_1)))

        # Determine the Best Glide Speed                     ft/s
        Vmd = math.sqrt((2.0 / rho) * (self.W / self.S) * \
        (math.sqrt(self.K / self.CDo)) * (1.0 / math.cos(gamma_1)))

        # Speed to Fly for Minimum Sink Rate                 ft/s
        V_minSink = math.sqrt((2.0 / rho) * (self.W / self.S) * \
        (math.sqrt(self.K / (3.0 * self.CDo))) * (1.0 / math.cos(gamma_1)))

        velocities = {'Vstall': Vstall, 'Vmd': Vmd,
        'V_minSink': V_minSink}

        return velocities
```

Listing A.3: Dynamics Module within *pyACDT*

```
#!/usr/local/bin/python
'''
pyDynamics_dynamics

Holds the pyACDT Flight Dynamics Analysis Class.

Copyright (c) 2004-2013 by pyACDT Developers
All rights reserved.
Revision: 1.1  $Date: 04/11/2011 21:00$

References:
-----------
- B. Etkin and L.D. Reid. Dynamics of Flight: Stability and Control,
    Third Edition. John Wiley & Sons, Inc., 1996.
- R.C. Nelson. Flight Stability and Automatic Control, Second Edition.
    The McGraw-Hill Companies, 1998.
- L.V. Schmidt. Introduction to Aircraft Flight Dynamics. American
    Institute of Aeronautics and Astronautics, Inc., 1998.

Developers:
-----------
- Dr. Ruben E. Perez (RP)
- Major Thomas R. Connerty (TC)
```

```
- Mr. Peter W. Jansen (PJ)

History:
--------
    v. 1.0 - Initial Creation in Matlab (RP, 2004)
    v. 1.1 - Initial Code Migration to Python (TC, 2013)

'''

__version__ = '$Revision: $'

'''
To Do:
    - Create more abstract FDM and expand to other models like BEM based
'''

# ==============================================================================
# Standard Python modules
# ==============================================================================
import os, math, sys
import pdb

# ==============================================================================
# External Python modules
# ==============================================================================
import matplotlib.pyplot as plt
import numpy

# ==============================================================================
# Extension modules
# ==============================================================================
sys.path.append(os.path.abspath('../'))
sys.path.append(os.path.abspath('../pyInertia'))
sys.path.append(os.path.abspath('../Atmosphere/pyAtmos'))
sys.path.append(os.path.abspath('../Atmosphere/pyAtmos/pyICAO'))
sys.path.append(os.path.abspath('../Aerodynamics'))
sys.path.append(os.path.abspath('../Payload'))
sys.path.append(os.path.abspath('../Geometry'))
sys.path.append(os.path.abspath('../Fuel'))
sys.path.append(os.path.abspath('../Tools/Interpolation'))
sys.path.append(os.path.abspath('../Tools/Search'))
sys.path.append(os.path.abspath('../Weights/pyInertia'))
sys.path.append(os.path.abspath('../Weights/pyWeight'))

from pyAero_flow import Flow
from pyAero_reference import Reference
from pyAero_solver import AeroSolver
from pyDynamics_object import DynObject
from pyFuel_fuel import Fuel
from pyGeometry_aircraft import Aircraft
from pyGeometry_bodysurface import BodySurface
from pyGeometry_liftingsurface import LiftingSurface
from pyGeometry_system import System
from pyInertia_model import InertiaModel
from pyPayload_lumped import Lumped
from pyPayload_payload import Payload
```

```
from pyWeight_model import WeightModel

# ==============================================================================
# Misc Definitions
# ==============================================================================


# ==============================================================================
# FDM Class
# ==============================================================================
class FDM(DynObject):

    '''
    Flight Dynamics Model Class
    '''

    def __init__(self, aircraft=None, aero=None, flow=None, fuel=None,
        ref=None, inertia=None, payload=None, weight=None, CL=None,
        *args, **kwargs):

        '''
        Flight Dynamics Model Class Initialization

        **Keyword Arguments:**

        - aircraft -> OBJECT: Aircraft Geometry Object, *Default* = None
        - aero -> OBJECT: Aerodynamic Solver Method, *Default* = None
        - flow -> OBJECT: Aerodynamics Flow Object
        - ref -> OBJECT: Aerodynamics Reference Object, *Default* = None
        - inertia -> OBJECT: Aircraft Inertia Model, *Default* = None
        - weight -> FLOAT: Aircraft Weight, *Default* = None
        - CL -> FLOAT: Lift Coefficient, *Default* = None

        Documentation last updated:  Nov 07, 2013 - Thomas R. Connerty
        '''

        # Input Checks
        if (aircraft != None):
            if not isinstance(aircraft,Aircraft):
                raise TypeError
                ("Dynamic: aircraft input is not a valid object instance\n")
            #end
        #end

        if (aero != None):
            if not isinstance(aero,dict):
                if not isinstance(aero,AeroSolver):
                    raise TypeError
                    ("Dynamic: aero input is not a valid object instance\n")
                #end
            #end
        #end

        if (flow != None):
            if not isinstance(flow,Flow):
                raise TypeError
```

```
                ("Dynamic: flow input is not a valid object instance\n")
            #end
        #end

        if (inertia != None):
            if not isinstance(inertia,InertiaModel):
                raise TypeError
                ("Dynamic: inertia input is not a valid object instance\n")
            #end
        #end

        if (ref != None):
            if not isinstance(ref,Reference):
                raise TypeError
                ("Dynamic: ref input is not a valid object instance\n")
            #end
        #end

        #
        name = 'Flight Dynamics Model'
        def_opts = {}

        # Initialize Abstract Class
        DynObject.__init__(self, name, def_opts, *args, **kwargs)

        #
        self.aircraft = aircraft
        self.aero = aero
        self.fuel = fuel
        self.flow = flow
        self.inertia = inertia
        self.payload = payload
        self.ref = ref
        self.weight = weight
        self.CL = CL

        if self.aircraft != None and self.aero != None and \
        self.inertia != None and self.ref != None and self.weight != None:
            self.__call__()
        #end


    def __call__(self, aircraft=None, aero=None, flow=None, fuel=None,
        inertia=None, payload=None, ref=None, weight=None, CL=None,
        *args, **kwargs):

        '''
        Calculate Flight Dynamics Model (Calling Routine)

        **Keyword Arguments:**

        - aircraft -> OBJECT: Aircraft Geometry Object, *Default* = None
        - aero -> OBJECT: Aerodynamic Solver Method, *Default* = None
        - flow -> OBJECT: Aerodynamics Flow Object
        - ref -> OBJECT: Aerodynamics Reference Object, *Default* = None
        - inertia -> OBJECT: Aircraft Inertia Model, *Default* = None
```

```
- weight -> FLOAT: Aircraft Weight, *Default* = None
- CL -> FLOAT: Lift Coefficient, *Default* = None

Documentation last updated:  Nov 07, 2013 - Thomas R. Connerty
'''

# Inputs Check
if (aircraft != None):
    if not isinstance(aircraft,Aircraft):
        raise TypeError
        ("Dynamic: aircraft input is not a valid object instance\n")
    #end
    self.aircraft = aircraft
else:
    aircraft = self.aircraft
#end

if (aero != None):
    if not isinstance(aero,dict):
        if not isinstance(aero,AeroSolver):
            raise TypeError ("Dynamic: aeroSolver input is not \
            a valid object instance\n")
        #end
    #end
    self.aero = aero
else:
    aero = self.aero
#end

if (inertia != None):
    if not isinstance(inertia,InertiaModel):
        raise TypeError
        ("Dynamic: inertia input is not a valid object instance\n")
    #end
    self.inertia = inertia
else:
    inertia = self.inertia
#end

if (ref != None):
    if not isinstance(ref,Reference):
        raise TypeError
        ("Dynamic: ref input is not a valid object instance\n")
    #end
    self.ref = ref
else:
    ref = self.ref
#end

if (CL != None):
    self.CL = CL
else:
    CL = self.CL
#end

# ================================================================
```

```
        # Define Constants
        # ====================================================================

        # General Constants
        g = 32.2
        self.g = g

        # Aircraft Geometric Parameters
        S = ref.sref
        b = ref.bref
        MAC_w = ref.cref

        # Aircraft Weight and Inertia Parameters

        W = weight
        m = W / g

        inertiaPara = inertia(aircraft, payload, fuel, W)

        Ixx = inertiaPara['Ixx']
        Iyy = inertiaPara['Iyy']
        Izz = inertiaPara['Izz']
        Ixy = inertiaPara['Ixy']
        Iyx = inertiaPara['Iyx']
        Ixz = inertiaPara['Ixz']
        Izx = inertiaPara['Izx']
        Iyz = inertiaPara['Iyz']
        Izy = inertiaPara['Izy']

        print inertiaPara

        # Aircraft Flight Condition
        V = flow.mach * flow.data.sound_speed
        self.V = V
        alpha = flow.alpha * math.pi / 180.0
        gamma = 0.0 * math.pi / 180.0
        theta = alpha + gamma
        rho = flow.data.density
        q = 0.5 * rho * (V ** 2)

        # ====================================================================
        # Non-Dimensional Stability Derivatives
        # ====================================================================

        if not isinstance(aero,dict):
            stabder = aero.getDerivatives(flow,'all',
                aircraft,ref,CL=self.CL)
        else:
            stabder = aero
        #end

        # Longitudinal Stability Coefficients
        CL = stabder['CL']
        CLa = stabder['CLa']
        CLu = stabder['CLu']
        CD = stabder['CD']
```

```
        CDa = stabder['CDa']
        CDu = stabder['CDu']
        CDq = stabder['CDq']
        Cma = stabder['Cma']
        Cmu = stabder['Cmu']
        Cma_dot = stabder['Cmad']
        Cmq = stabder['Cmq']
        Cmde = stabder['Cmde']
        Cxa = -(CDa - CL)
        Cxu = -(CDu + 2.0 * CD)
        Cxde = -stabder['CDde']
        Cza_dot = -stabder['CLad']
        Cza = -(CLa + CD)
        Czq = -stabder['CLq']
        Czu = -(CLu + 2.0 * CL)
        Czde = -stabder['CLde']

        # Lateral Directional Stability Coefficients
        Cyb = stabder['Cyb']
        Cyp = stabder['Cyp']
        Cyr = stabder['Cyr']
        Cyda = stabder['Cyda']
        Cydr = stabder['Cydr']
        Clb = stabder['Clb']
        Clp = stabder['Clp']
        Clr = stabder['Clr']
        Clda = stabder['Clda']
        Cldr = stabder['Cldr']
        Cnb = stabder['Cnb']
        Cnp = stabder['Cnp']
        Cnr = stabder['Cnr']
        Cnda = stabder['Cnda']
        Cndr = stabder['Cndr']

        # ======================================================================
        # Dimensional Stability Derivatives (Nelson)
        # ======================================================================

        # Longitudinal Derivatives
        self.Xu      = (q * S * Cxu) / (m * V)
        self.Xw      = (q * S * Cxa) / (m * V)
        self.Zu      = (q * S * Czu) / (m * V)
        self.Zw      = (q * S * Cza) / (m * V)
        self.Zw_dot  = (q * S * (MAC_w / (2.0 * V)) * Cza_dot) / (m * V)
        self.Za      = V * self.Zw_dot
        self.Za_dot  = V * self.Zw_dot
        self.Zq      = (q * S * (MAC_w / (2.0 * V)) * Czq) / m
        self.Zde     = (q * S * Czde) / m
        self.Mu      = (q * S * MAC_w * Cmu) / (V * Iyy)
        self.Mw      = (q * S * MAC_w * Cma) / (V * Iyy)
        self.Mw_dot  = (q * S * MAC_w * (MAC_w / (2.0 * V)) * Cma_dot) / \
        (V * Iyy)
        self.Ma      = V * self.Mw
        self.Ma_dot  = V * self.Mw_dot
        self.Mq      = (q * S * MAC_w * (MAC_w / (2.0 * V)) * Cmq) / Iyy
        self.Xde     = (q * S * Cxde) / m
```

```
        self.Mde     = (q * S * MAC_w * Cmde) / Iyy

        # Lateral Directional Derivatives
        self.Yb   = (q * S * Cyb) / m
        self.Yp   = (q * S * b * Cyp) / (2.0 * m * V)
        self.Yr   = (q * S * b * Cyr) / (2.0 * m * V)
        self.Yda  = (q * S * Cyda) / m
        self.Ydr  = (q * S * Cydr) / m
        self.Lb   = (q * S * b * Clb) / Ixx
        self.Lp   = (q * S * (b ** 2.0) * Clp) / (2.0 * Ixx * V)
        self.Lr   = (q * S * (b ** 2.0) * Clr) / (2.0 * Ixx * V)
        self.Lda  = (q * S * b * Clda) / Ixx
        self.Ldr  = (q * S * b * Cldr) / Ixx
        self.Nb   = (q * S * b * Cnb) / Izz
        self.Np   = (q * S * (b ** 2.0) * Cnp) / (2.0 * Izz * V)
        self.Nr   = (q * S * (b ** 2.0) * Cnr) / (2.0 * Izz * V)
        self.Nda  = (q * S * b * Cnda) / Izz
        self.Ndr  = (q * S * b * Cndr) / Izz

        # =====================================================================
        # State Space Representation
        # Dimensional Stability Derivatives (Nelson)
        # =====================================================================

        # Definition of the System Dynamics Matrix, A
        A = numpy.array([[self.Xu, self.Xw, 0, -self.g, 0, 0, 0, 0],
            [self.Zu, self.Zw, self.V, 0, 0, 0, 0, 0],
            [self.Mu + (self.Mw_dot * self.Zu), self.Mw +
            (self.Mw_dot * self.Zw), self.Mq + (self.Mw_dot * self.V),
            0, 0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0, 0, 0],[0, 0, 0, 0, (self.Yb / self.V),
            (self.Yp / self.V), -(1.0 - (self.Yr / self.V)),
            (g * math.cos(theta)) / self.V],[0, 0, 0, 0, self.Lb, self.Lp,
            self.Lr, 0],[0, 0, 0, 0, self.Nb, self.Np, self.Nr, 0],
            [0, 0, 0, 0, 0, 1, 0, 0]])

        self.A = A
        print 'A=', A

        # Definition of the Input Matrix, B
        B = numpy.array([[self.Xde, 0, 0],[self.Zde, 0 , 0],[self.Mde + \
            (self.Mw_dot * self.Zde), 0, 0], [0, 0, 0],[0, 0,
            (self.Ydr / self.V)],[0, self.Lda, self.Ldr],[0, self.Nda,
            self.Ndr],[0, 0, 0]])

        self.B = B
        print 'B=', B

        # Definition of the Output Matrix, C
        C = numpy.eye(8)
        self.C = C
        print 'C=', C

        # Definition of the Direct Transmission Matrix, D
        D = numpy.zeros((8,3))
        self.D = D
```

```
        print 'D=', D

        # Definition of States, Inputs and Outputs
        self.states = ['u', 'w', 'q', 'theta', 'beta', 'p', 'r', 'phi'];
        self.inputs = ['de', 'da', 'dr']
        self.outputs = ['u_dot', 'w_dot', 'q_dot', 'theta_dot', 'beta_dot',
            'p_dot', 'r_dot', 'phi_dot']


    def getEig(self, eigType=None):

        '''
        Computes the Eigenvalues of the Aircraft Dynamics

        **Arguments:**

        **Keyword Arguments:**

        - eigType -> STR: Eigenvalue Type (All, Long, LatDir),
            *Default* = None

        Documentation last updated:  Nov 06, 2013 - Thomas R. Connerty
        '''

        # Calculate Eigenvalues
        if eigType == None or eigType == 'All':
            self.eig = numpy.linalg.eigvals(self.A)
        elif eigType == 'Long':
            self.eig = numpy.linalg.eigvals(self.A[0:4,0:4])
        elif eigType == 'LatDir':
            self.eig = numpy.linalg.eigvals(self.A[4:,4:])
        #end

        print 'eig:', self.eig


    def getRoots(self, modeType=None, stabAxis=None):

        '''
        Computes the Damping Ratio and Natural Frequency

        **Arguments:**

        **Keyword Arguments:**

        - modType -> STR: Mode Type (All, sp, p, dr), *Default* = None
        - stabAxis -> STR: Stability Axis (All, Long, LatDir),
            *Default* = None

        Documentation last updated:  Nov 13, 2013 - Thomas R. Connerty
        '''

        # Calculate Damping Ratio and Natural Frequency for All Mode Types
        if modeType == None or modeType == 'All':
            if stabAxis == None or stabAxis == 'All':
```

```
            wn_p = math.sqrt((-self.Zu * self.g) / self.V)
            zeta_p = -self.Xu / (2.0 * wn_p)
            wn_sp = math.sqrt(((self.Za * self.Mq) / self.V) - self.Ma)
            zeta_sp = -(self.Mq + self.Ma_dot + \
                (self.Za / self.V)) / (2.0 * wn_sp)
            wn_dr = math.sqrt(((self.Yb * self.Nr) - \
                (self.Nb * self.Yr) + (self.V * self.Nb)) / self.V)
            zeta_dr = -((self.Yb + (self.V * self.Nr)) / self.V) / \
                (2.0 * wn_dr)

            print 'wn_p:', wn_p
            print 'zeta_p:', zeta_p
            print 'wn_sp:', wn_sp
            print 'zeta_sp:', zeta_sp
            print 'wn_dr:', wn_dr
            print 'zeta_dr:', zeta_dr

        elif stabAxis == 'Long':

            wn_p = math.sqrt((-self.Zu * self.g) / self.V)
            zeta_p = -self.Xu / (2.0 * wn_p)
            wn_sp = math.sqrt(((self.Za * self.Mq) / self.V) - self.Ma)
            zeta_sp = -(self.Mq + self.Ma_dot + (self.Za / self.V)) / \
                (2.0 * wn_sp)

            print 'wn_p:', wn_p
            print 'zeta_p:', zeta_p
            print 'wn_sp:', wn_sp
            print 'zeta_sp:', zeta_sp

        elif stabAxis == 'LatDir':

            wn_dr = math.sqrt(((self.Yb * self.Nr) - \
                (self.Nb * self.Yr) + (self.V * self.Nb)) / self.V)
            zeta_dr = -((self.Yb + (self.V * self.Nr)) / self.V) / \
                (2.0 * wn_dr)

            print 'wn_dr:', wn_dr
            print 'zeta_dr:', zeta_dr
        #end

    # Calculate the Short Period Approximations
    # Longitudinal Case

    elif modeType == 'sp':
        if stabAxis == None or stabAxis == 'All' or \
        stabAxis == 'LatDir':

            raise TypeError
            ("Dynamic: stabAxis input is not a valid axis for \
                modeType\n")

        elif stabAxis == 'Long':

            wn_sp = math.sqrt(((self.Za * self.Mq) / self.V) - \
                self.Ma)
```

```
                zeta_sp = -(self.Mq + self.Ma_dot + (self.Za / self.V)) / \
                    (2.0 * wn_sp)

                print 'wn_sp:', wn_sp
                print 'zeta_sp:', zeta_sp
            #end

        # Calculate the Long Period (Phugoid) Approximations
        # Longitudinal Case

        elif modeType == 'p':
            if stabAxis == None or stabAxis == 'All' or \
            stabAxis == 'LatDir':

                raise TypeError
                ("Dynamic: stabAxis input is not a valid axis for \
                    modeType\n")

            elif stabAxis == 'Long':
                wn_p = math.sqrt((-self.Zu * self.g) / self.V)
                zeta_p = -self.Xu / (2.0 * wn_p)

                print 'wn_p:', wn_p
                print 'zeta_p:', zeta_p
            #end

        # Calculate the Dutch-Roll Appoximation
        # Lateral Directional Case

        elif modeType == 'dr':
            if stabAxis == None or stabAxis == 'All' or \
            stabAxis == 'Long':

                raise TypeError
                ("Dynamic: stabAxis input is not a valid axis for \
                    modeType\n")

            elif stabAxis == 'LatDir':

                wn_dr = math.sqrt(((self.Yb * self.Nr) - \
                    (self.Nb * self.Yr) + (self.V * self.Nb)) / self.V)
                zeta_dr = -((self.Yb + (self.V * self.Nr)) / self.V) / \
                    (2.0 * wn_dr)

                print 'wn_dr:', wn_dr
                print 'zeta_dr:', zeta_dr
            #end
        #end


#============================================================================
#
#============================================================================
if __name__ == '__main__':

    print 'Testing ...'
```

```
#
fdm = FDM()
fdm.ListAttributes()
```

B Computer Source Code

B.1 Thermal Models

Listing B.1: Allen Chimney Thermal Model

```
#!/usr/local/bin/python
'''
pyThermal_Allen

Holds the NASA Thermal Model Class

Copyright (c) 2004-2014 by pyACDT Developers
All rights reserved.
Revision: 1.0  $Date: 06/01/2014 21:00$

References:
-----------
- M.J. Allen, "Updraft Model for Development of Autonomous Soaring
    Uninhabited Air Vehicles," 44th AIAA Aerospace Sciences Meeting and
    Exhibit, Reno, Nevada, January 2006.

Developers:
-----------
- Major Thomas R. Connerty (TC)

History:
--------
    v. 1.0 - Initial Creation in Python (TC, 2013)

'''

__version__ = '$Revision: $'

'''
To Do:
    -
'''

# ================================================================================
# Standard Python modules
# ================================================================================
import os, cmath, sys, time
```

```
import pdb # pdb.set_trace()

# ==========================================================================
# External Python modules
# ==========================================================================
import numpy
import random

# ==========================================================================
# Extension modules
# ==========================================================================

from pyThermal_object import thermalObject

# ==========================================================================
# Misc Definitions
# ==========================================================================

def cabs(x):

    '''
    Returns absolute value of a complex number
    '''

    if isinstance(x,complex):
        if (x.real < 0):
            cabsx = -x
        else:
            cabsx = x
        #end
    else:
        cabsx = abs(x)
    #end

    return complex(cabsx)


# ==========================================================================
# NASA Thermal Model Class
# ==========================================================================

class thermalAllen(thermalObject):

    '''
    NASA Thermal Model Class
    '''

    def __init__(self,seed=12345.6,*args, **kwargs):

        '''
        Thermal Model Class Initialization

        **Keyword Arguments:**

        - seed -> FLOAT: Define a seed for the Random Number Generator,
            *Default* = 12345.6
```

```
        *Outputs:*

        - W_h -> FLOAT: Updraft verticle velocity (m/s)

        Documentation last updated:  Jan 6, 2014 - Thomas R. Connerty
        '''

        #
        name = 'Allen Thermal Model'
        def_opts = {}

        # Initialize Abstract Class
        thermalObject.__init__(self, name, def_opts, *args, **kwargs)

        #
        self.seed = seed

        if self.seed == None:
            self.seed = time.time()
        #end

        random.seed(self.seed)

    def __call__(self,r=0.0,h_ac=0.0,theta=0.0,L=5280.0,W=5280.0,
        month_index=None,*args, **kwargs):

        '''
        Generate Thermal Model (Calling Routine)

        **Keyword Arguments:**

        - r -> FLOAT: Aircraft Radial Distance from Thermal Center (ft),
            *Default* = 0.0
        - h_ac -> FLOAT: Aircraft height above the ground (ft),
            *Default* = 0.0
        - theta -> FLOAT: Aircraft Angular position (rad),
            *Default* = 0.0
        - L -> FLOAT: Length of the Test Area (ft), *Default* = 5280.0
        - W -> FLOAT: Width of the Test Area (ft), *Default* = 5280.0
        - month_index -> INTEGER: Define the Month, *Default* = 0.0

        *Outputs:*

        - W_h -> FLOAT: Updraft verticle velocity (ft/s)

        Documentation last updated:  Jan 06, 2014 - Thomas R. Connerty
        '''

        # Convert Inputs to SI Units (ft to m)
        r = 0.3048 * r
        h_ac = 0.3048 * h_ac
        L = 0.3048 * L
        W = 0.3048 * W

        #
```

```
        self.r = r
        self.h_ac = h_ac
        self.theta = theta
        self.L = L
        self.W = W

        # ======================================================================
        # Define the Month of the Year
        # ======================================================================

        random.seed(self.seed)

        if month_index == None:
            self.month_index = random.randint(0,11)

        else:
            self.month_index = month_index
        #end

        # ======================================================================
        # Shape Constants for Bell-Shaped Vertical Velocity Distribution
        # ======================================================================

        r1r2shape = numpy.array([[0.14],[0.25],[0.36],[0.47],[0.58],
            [0.69],[0.80]])

        Kshape = numpy.array([[1.5352,2.5826,-0.0113,0.0008],[1.5265,
        3.6054,-0.0176,0.0005],[1.4866,4.8354,-0.0320,0.0001],[1.2042,
        7.7904,0.0848,0.0001],[0.8816,13.972,0.3404,0.0001],
        [0.7067,23.994,0.5689,0.0002],[0.6189,42.797,0.7157,0.0001]])

        # ======================================================================
        # Monthly Convective Scaling Points
        # ======================================================================

        convScale = numpy.array([[1.14,3.59,504.0,1800.0],[1.48,3.97,
        666.0,1970.0],[1.64,4.89,851.0,3900.0],[1.97,5.53,1213.0,2380.0],
        [2.53,5.49,1887.0,3833.0],[2.38,5.51,1728.0,4027.0],[2.69,6.30,
        1975.0,3962.0],[2.44,5.64,1755.0,4940.0],[2.25,5.97,1382.0,
        2460.0],[1.79,4.57,893.0,3285.0],[1.31,4.55,627.0,1783.0],[1.26,
        4.11,441.0,1680.0]])

        w_star = convScale[self.month_index,1]
        hi = convScale[self.month_index,3]

        # Calculates the Updraft Size
        r_bar = 0.102 * ((self.h_ac / hi) ** (1.0 / 3.0)) * (1.0 - \
        (0.25 * (self.h_ac / hi))) * hi

        # Calculates the Average Updraft Velocity (m/s)
        w_bar = w_star * ((self.h_ac / hi) ** (1.0 / 3.0)) * (1.0 - \
        (1.1 * (self.h_ac / hi)))

        # ======================================================================
        # Generate the Location of Updraft in the Test Area
        # ======================================================================
```

```
x_t = 0.0
y_t = 0.0

#
self.x_t = x_t
self.y_t = y_t

# ======================================================================
# Determine the Aircraft Position in Relation to the Thermal Centre
# ======================================================================

x_ac = (self.x_t + (self.r.real * cmath.sin(self.theta))).real
y_ac = (self.y_t + (self.r.real * cmath.cos(self.theta))).real

#
self.x_ac = x_ac
self.y_ac = y_ac

# ======================================================================
# Define the Perturbation Gains for each Updraft
# ======================================================================

wgain = random.random()

if wgain < 0.5:
    wgain = 0.5
#end

rgain = random.random()

if rgain < 0.5:
    rgain = 0.5
#end

# ======================================================================
# Calculate the Average Updraft Size
# ======================================================================

# Apply the Random Perturbation Gain to the Updraft
# Outer Radius, m
r2 = r_bar * rgain

# Apply the Random Perturbation Gain to the Average
# Updraft Velocity, m/s
wt = w_bar * wgain

# Limit Small Updrafts to 20m in Diameter
if complex(r2).real < 10.0:
    r2 = 10.0 # ft
#end

if complex(r2).real < 600.0:
    r1r2 = 0.0011 * r2 + 0.14
else:
    r1r2 = 0.8
```

```
        #end

        # Calculate the Updraft Core Radius (m)
        r1 = r1r2 * r2

        # ====================================================================
        # Calculate the Updraft Velocity
        # ====================================================================

        # Calculate the Peak Updraft Velocity (m/s)
        w_peak = 3.0 * wt * ((r2 ** 3.0) - ((r2 ** 2.0) * r1)) / \
        ((r2 ** 3.0) - (r1 ** 3.0))

        rr2 = self.r / r2

        # Determine the Shape of the Thermal
        if complex(self.h_ac).real < hi:
            if complex(r1r2).real < 0.5 * (r1r2shape[0] + r1r2shape[1]):
                k1 = Kshape[0,0]
                k2 = Kshape[0,1]
                k3 = Kshape[0,2]
                k4 = Kshape[0,3]
            elif complex(r1r2).real < 0.5 * (r1r2shape[1] + r1r2shape[2]):
                k1 = Kshape[1,0]
                k2 = Kshape[1,1]
                k3 = Kshape[1,2]
                k4 = Kshape[1,3]
            elif complex(r1r2).real < 0.5 * (r1r2shape[2] + r1r2shape[3]):
                k1 = Kshape[2,0]
                k2 = Kshape[2,1]
                k3 = Kshape[2,2]
                k4 = Kshape[2,3]
            elif complex(r1r2).real < 0.5 * (r1r2shape[3] + r1r2shape[4]):
                k1 = Kshape[3,0]
                k2 = Kshape[3,1]
                k3 = Kshape[3,2]
                k4 = Kshape[3,3]
            elif complex(r1r2).real < 0.5 * (r1r2shape[4] + r1r2shape[5]):
                k1 = Kshape[4,0]
                k2 = Kshape[4,1]
                k3 = Kshape[4,2]
                k4 = Kshape[4,3]
            elif complex(r1r2).real < 0.5 * (r1r2shape[5] + r1r2shape[6]):
                k1 = Kshape[5,0]
                k2 = Kshape[5,1]
                k3 = Kshape[5,2]
                k4 = Kshape[5,3]
            else:
                k1 = Kshape[6,0]
                k2 = Kshape[6,1]
                k3 = Kshape[6,2]
                k4 = Kshape[6,3]
            #end

            # Calculate the Smooth Vertical Velocity Distribution
            ws = (1.0 / (1 + (cabs((k1 * rr2) + k3) ** k2))) + (k4 * rr2)
```

```
        # Ensure no Negative Updraft Velocities
        if complex(ws).real < 0.0:
            ws = complex(0.0,0.0)
        #end

    else:
        ws = complex(0.0,0.0)
    #end

    # ================================================================
    # Calculate the Downdraft Velocity at the Edge of the Updraft
    # ================================================================

    # Calculate the Intermediate Variable in Downdraft Velocity (m/s)
    if (complex(self.r).real > complex(r1).real) and \
    ((complex(rr2).real) < 2.0):
        wl = -(cmath.pi / 6.0) * cmath.sin(cmath.pi * rr2)

    else:
        wl = complex(0.0,0.0)
    #end

    # Calculate the Downdraft Velocity (m/s)
    if (complex(self.h_ac / hi).real > 0.5) and \
    (complex(self.h_ac / hi).real < 0.9):
        wd = 2.5 * wl * ((self.h_ac / hi) - 0.5)

    else:
        wd = complex(0.0,0.0)
    #end

    # Downdraft Velocity Ratio
    if wl == 0.0:
        s_wd = 0.0
    else:
        s_wd = wd / wl
    #end

    # Scale the Updraft to the Actual Velocity (m/s)
    w2 = (ws * w_peak) + (wd * wt)

    # ================================================================
    # Calculate the Environmental Sink Velocity
    # ================================================================

    # Total Area taken up by Thermals (m**2.0)
    A_thermal = cmath.pi * (r2 ** 2.0)

    # Total Area of the Test Location (m**2.0)
    A = self.L * self.W

    if complex(A_thermal).real > A:
        raise ValueError, 'Area of Test Location is too Small'
    #end
```

```
        # Environmental Sink Velocity (m/s)
        we = -(w_bar * A_thermal * s_wd) / (A - A_thermal)

        # Ensure no Positive Sink Velocities
        if complex(we).real > 0:
            we = complex(0.0,0.0)
        #end

        # Stretch the Updraft to Blend with Sink at Edges (m/s)
        if complex(self.r).real > complex(r1).real:
            W_h = w2 * (1 - (we / w_peak)) + we

        else:
            W_h = w2
        #end

        #
        self.W_h = W_h

        # Convert Outputs to English Units (m/s to ft/s, m to ft)

        self.W_h = 3.28084 * self.W_h
        self.x_t = 3.28084 * self.x_t
        self.y_t = 3.28084 * self.y_t
        self.x_ac = 3.28084 * self.x_ac
        self.y_ac = 3.28084 * self.y_ac

        data = {'Updraft_Velocity': self.W_h, 'Thermal_Center_x': self.x_t,
        'Thermal_Center_y': self.y_t, 'AC_Location_x': self.x_ac,
        'AC_Location_y': self.y_ac}

        return data

    def getDerivatives(self, r=0.0, h_ac=0.0, month_index=None,
        h=1e-20, *args, **kwargs):

        '''
        Computes the Wind Derivatives of the Thermal Model

        **Keyword Arguments:**

        - r -> FLOAT: Aircraft Radial Distance from Thermal Center (ft),
            *Default* = 0.0
        - h_ac -> FLOAT: Aircraft height above the ground (ft),
            *Default* = 0.0
        - month_index -> INTEGER: Define the Month, *Default* = 0.0
        - h -> FLOAT: Step Size for Complex Step Method, *Default* = 1e-20

        Documentation last updated:  Jan 17, 2014 - Thomas R. Connerty
        '''

        # Convert Inputs to SI Units (ft to m)
        r = 0.3048 * r
        h_ac = 0.3048 * h_ac

        # ================================================================
```

```
        # Gradient Calculation - Change in Updraft Velocity with Radius
        # ====================================================================

        drt = self.__call__(complex(r,h),h_ac,
            self.month_index)['Updraft_Velocity']
        dWh_dr = drt.imag / h

        #
        self.dWh_dr = dWh_dr

        # ====================================================================
        # Gradient Calculation -
        # Change in Updraft Velocity with Angular Position
        # ====================================================================

        dWh_dtheta = 0.0

        #
        self.dWh_dtheta = dWh_dtheta

        # ====================================================================
        # Gradient Calculation - Change in Updraft Velocity with Height
        # ====================================================================

        dht = self.__call__(r,complex(h_ac,h),
            self.month_index)['Updraft_Velocity']
        dWh_dh = dht.imag / h

        #
        self.dWh_dh = dWh_dh

        # Convert Outputs to English Units (m/s**2.0 to ft/s**2.0)
        self.dWh_dr = 3.28084 * self.dWh_dr
        self.dWh_dtheta = 3.28084 * self.dWh_dtheta
        self.dWh_dh = 3.28084 * self.dWh_dh

        gradients = {'dWh_dr': self.dWh_dr, 'dWh_dtheta': self.dWh_dtheta,
        'dWh_dh': self.dWh_dh}

        return gradients

#============================================================================
#
#============================================================================
if __name__ == '__main__':

    print 'Testing ...'

    #
    allen = thermalAllen()
    allen.ListAttributes()
```

Listing B.2: Lawrance Bubble Model

```
#!/usr/local/bin/python
'''
```

```
pyThermal_lawrance

Holds the Lawrance Bubble Thermal Model Class

Copyright (c) 2004-2014 by pyACDT Developers
All rights reserved.
Revision: 1.0  $Date: 22/01/2014 21:00$

References:
-----------
- N.R. Lawrance and S. Sukkarieh, "Wind Energy Based Path Planning for
    a Small Gliding Unmanned Aerial Vehicle," AIAA Guidance, Navigation,
    and Control Conference, Chicago, Illinois, 2009.

Developers:
-----------
- Major Thomas R. Connerty (TC)

History:
--------
    v. 1.0 - Initial Creation in Python (TC, 2013)

'''

__version__ = '$Revision: $'

'''
To Do:
    -
'''

# ============================================================================
# Standard Python modules
# ============================================================================
import os, math, sys, time
import pdb # pdb.set_trace()

# ============================================================================
# External Python modules
# ============================================================================
import numpy
import random

# ============================================================================
# Extension modules
# ============================================================================

from pyThermal_object import thermalObject

# ============================================================================
# Misc Definitions
# ============================================================================


# ============================================================================
# Lawrance Bubble Thermal Model Class
```

```
# ==============================================================================

class thermalLawrance(thermalObject):

    '''
    Lawrance Bubble Thermal Model Class
    '''

    def __init__(self,seed=12345.6,*args, **kwargs):

        '''
        Thermal Model Class Initialization

        **Keyword Arguments:**

        - seed -> FLOAT: Define a seed for the Random Number Generator,
            *Default* = 12345.6

        *Outputs:*

        - wx -> FLOAT: x component of the flow field velocity (ft/s)
        - wy -> FLOAT: y component of the flow field velocity (ft/s)
        - wz -> FLOAT: z component of the flow field velocity (ft/s)

        Documentation last updated:  Jan 22, 2014 - Thomas R. Connerty
        '''

        #
        name = "Lawrance Bubble Thermal Model"
        def_opts = {}

        # Initialize Abstract Class
        thermalObject.__init__(self, name, def_opts, *args, **kwargs)

        #
        self.seed = seed

        if self.seed == None:
            self.seed = time.time()
        #end

        random.seed(self.seed)

    def __call__(self,r=0.0,z_ac=0.0,R=0.0,w_core=0.0,z_t=0.0,theta=0.0,
        L=5280.0,W=5280.0,*args, **kwargs):

        '''
        Generate Thermal Model (Calling Routine)

        **Keyword Arguments:**

        - r -> FLOAT: Aircraft Radial Distance from Thermal Center (ft),
            *Default* = 0.0
        - z_ac -> FLOAT: Aircraft height above the ground (ft),
            *Default* = 0.0
        - R -> FLOAT: Distance that Limits the Updraft Area (ft),
```

```
        *Default* = 0.0
    - w_core -> FLOAT: Updraft velocity at center of thermal (ft/s),
        *Default* = 0.0
    - z_t -> FLOAT: Height of the thermal bubble (ft), *Default* = 0.0
    - theta -> FLOAT: Aircraft Angular position (rad), *Default* = 0.0
    - L -> FLOAT: Length of the Test Area (ft), *Default* = 1000.0
    - W -> FLOAT: Width of the Test Area (ft), *Default* = 1000.0

    *Outputs:*

    - wx -> FLOAT: x component of the flow field velocity (ft/s)
    - wy -> FLOAT: y component of the flow field velocity (ft/s)
    - wz -> FLOAT: z component of the flow field velocity (ft/s)

    Documentation last updated:  Jan 22, 2014 - Thomas R. Connerty
    '''

    #
    self.r = r
    self.z_ac = z_ac
    self.R = R
    self.w_core = w_core
    self.z_t = z_t
    self.theta = theta
    self.L = L
    self.W = W

    # ================================================================
    # Generate the Location of Updraft in the Test Area
    # ================================================================

    x_t = 0.0
    y_t = 0.0

    #
    self.x_t = x_t
    self.y_t = y_t

    # Compute the Area of the Test Area, (ft**2)
    A = self.L * self.W

    # ================================================================
    # Determine the Aircraft Position in Relation to the Thermal Centre
    # ================================================================

    x_ac = (self.x_t + (self.r.real * math.sin(self.theta))).real
    y_ac = (self.y_t + (self.r.real * math.cos(self.theta))).real

    #
    self.x_ac = x_ac
    self.y_ac = y_ac

    # ================================================================
    # Calculate the Distance of the Aircraft to the Updraft, ft
    # ================================================================
```

```
        xd = abs(self.x_ac - self.x_t)
        yd = abs(self.y_ac - self.y_t)
        zd = abs(self.z_ac - (self.z_t / 2.0))

        # Distance of the Aircraft to the Thermal Center, ft
        dh = self.r

        # ====================================================================
        # Calculate the Thermal Flow Field, ft/s
        # ====================================================================

        # Calculate the Bubble Eccentricity Factor
        k = self.z_t / (2.0 * self.R)

        # Calculate the z Component of the Flow Field Velocity, (ft/s)
        if dh == 0.0:
            wz = self.w_core

        elif dh > 0.0 and dh < (2.0 * self.R):
            wz = ((math.cos(1.0 + ((math.pi * zd) / (k * self.R))) * \
            (self.R * w_core)) / (2.0 * math.pi * dh)) * \
            math.sin((math.pi * dh) / self.R)

        else:
            wz = 0.0
        #end

        # Calculate the x Component of the Flow Field Velocity, (ft/s)
        wx = -wz * (zd / ((dh - self.R) * (k ** 2.0))) * (xd / dh)

        # Calculate the y Component of the Flow Field Velocity, (ft/s)
        wy = -wz * (zd / ((dh - self.R) * (k ** 2.0))) * (yd / dh)

        # Calculate the r Component of the Flow Field Velocity, (ft/s)
        wr = math.sqrt((wx ** 2.0) + (wy ** 2.0))

        #
        self.W_h = wz
        self.W_r = wr

        data = {'Updraft_Velocity': self.W_h, 'x_Velocity_Component': wy,
        'x_Velocity_Component': wy, 'Updraft_Radial_Velocity': self.W_r,
        'Thermal_Center_x': self.x_t, 'Thermal_Center_y': self.y_t,
        'AC_Location_x': self.x_ac, 'AC_Location_y': self.y_ac}

        return data

    def getDerivatives(self, r=0.0,z_ac=0.0,R=0.0,w_core=0.0,z_t=0.0,
        theta=0.0,h=1e-06, *args, **kwargs):

        '''
        Computes the Wind Derivatives of the Thermal Model

        **Keyword Arguments:**

        - r -> FLOAT: Aircraft Radial Distance from Thermal Center (ft),
```

```
        *Default* = 0.0
    - z_ac -> FLOAT: Aircraft height above the ground (ft),
        *Default* = 0.0
    - R -> FLOAT: Distance that Limits the Updraft Area (ft),
        *Default* = 0.0
    - w_core -> FLOAT: Updraft velocity at center of thermal (ft/s),
        *Default* = 0.0
    - z_t -> FLOAT: Height of the thermal bubble (ft), *Default* = 0.0
    - theta -> FLOAT: Aircraft Angular position (rad), *Default* = 0.0
    - h -> FLOAT: Step Size for Complex Step Method, *Default* = 1e-20

    Documentation last updated:  Jan 22, 2014 - Thomas R. Connerty
    '''

    # ==========================================================
    # Gradient Calculation - Change in Updraft Velocity with Radius
    # ==========================================================

    fx = self.__call__(r,z_ac,R,w_core,z_t,theta)['Updraft_Velocity']
    fxh = self.__call__((r+h),z_ac,R,w_core,z_t,
        theta)['Updraft_Velocity']
    dWh_dr = (fxh - fx) / h

    #
    self.dWh_dr = dWh_dr

    # ==========================================================
    # Gradient Calculation -
    # Change in Updraft Velocity with Angular Position
    # ==========================================================

    dWh_dtheta = 0.0

    #
    self.dWh_dtheta = dWh_dtheta

    # ==========================================================
    # Gradient Calculation - Change in Updraft Velocity with Height
    # ==========================================================

    fx = self.__call__(r,z_ac,R,w_core,z_t,theta)['Updraft_Velocity']
    fxh = self.__call__(r,(z_ac+h),R,w_core,z_t,
        theta)['Updraft_Velocity']
    dWh_dh = (fxh - fx) / h

    #
    self.dWh_dh = dWh_dh

    # ==========================================================
    # Gradient Calculation -
    # Change in Updraft Radial Velocity with Radius
    # ==========================================================

    fx = self.__call__(r,z_ac,R,w_core,z_t,
        theta)['Updraft_Radial_Velocity']
    fxh = self.__call__((r+h),z_ac,R,w_core,z_t,
```

```
            theta)['Updraft_Radial_Velocity']
        dWr_dr = (fxh - fx) / h

        #
        self.dWr_dr = dWr_dr

        # ================================================================
        # Gradient Calculation -
        # Change in Updraft Radial Velocity with Angular Position
        # ================================================================

        dWr_dtheta = 0.0

        #
        self.dWr_dtheta = dWr_dtheta

        # ================================================================
        # Gradient Calculation -
        # Change in Updraft Radial Velocity with Height
        # ================================================================

        fx = self.__call__(r,z_ac,R,w_core,z_t,
            theta)['Updraft_Radial_Velocity']
        fxh = self.__call__(r,(z_ac+h),R,w_core,z_t,
            theta)['Updraft_Radial_Velocity']
        dWr_dh = (fxh - fx) / h

        #
        self.dWr_dh = dWr_dh

        gradients = {'dWh_dr': self.dWh_dr, 'dWh_dtheta': self.dWh_dtheta,
        'dWh_dh': self.dWh_dh, 'dWr_dr': self.dWr_dr,
        'dWr_dtheta': self.dWr_dtheta, 'dWr_dh': self.dWr_dh}

        return gradients

#============================================================================
#
#============================================================================
if __name__ == '__main__':

    print 'Testing ...'

    #
    lawrance = thermalLawrance()
    lawrance.ListAttributes()
```

B.2 Trajectory Optimization

Listing B.3: Trajectory Optimization Problem Formulation - Chimney Model

```
#!/usr/local/bin/python
'''
Cularis_D_5223_thermalClimb

Trajectory Optimization Problem (in Cylindrical Coordinates)

Copyright (c) 2004-2014 by pyACDT Developers
All rights reserved.
Revision: 1.0  $Date: 26/02/2014 21:00$

References:
-----------



Developers:
-----------
- Major Thomas R. Connerty (TC)

History:
--------
    v. 1.0 - Initial Creation in Python (TC, 2014)

'''

__version__ = '$Revision: $'

'''
To Do:
    -
'''

# ============================================================================
# Standard Python modules
# ============================================================================
import os, math, sys, time
import pdb

# ============================================================================
# External Python modules
# ============================================================================
import mpi4py
from mpi4py import MPI
import numpy
import random

comm=MPI.COMM_WORLD

# ============================================================================
# Extension modules
# ============================================================================
```

```
sys.path.append(os.path.abspath('..'))
sys.path.append(os.path.abspath('../../../..'))
sys.path.append(os.path.abspath('../pyThermal'))
sys.path.append(os.path.abspath('../../../../pyACDT/Optimization/pyDTO'))
sys.path.append(os.path.abspath('../../../../pyACDT/Optimization/pyOpt'))
sys.path.append(os.path.abspath
    ('../../../../pyACDT/Optimization/pyOpt/pySNOPT'))

from pyAtmosphere_US1976 import atmosphere
from pyDCNLP import DCNLP
from pyDTO_controls import Controls
from pyDTO_parameters import Parameters
from pyDTO_knots import Knots
from pyDTO_problem import Phase,Trajectory
from pyDTO_states import States
from pySNOPT import SNOPT
from pyThermal_allen import thermalAllen

# ============================================================================
# Misc Definitions
# ============================================================================
inf = 1e20  # define a value for infinity


# ============================================================================
# Define the  Aircraft Characteristics
# ============================================================================

# Misc Constants and Definitions
acID = 'D-5223'
costID = 'ratio_Err_Eh'
g = 32.174              # Acceleration due to Gravity       ft/s ** 2.0
rho0 = 0.0023769        # Density at Sea Level Conditions   slug/ft ** 3.0

# Cularis D-5223
if acID == 'D-5223':
    b = 8.536           # Wing Span                         ft
    S = 4.574           # Wing Reference Area               ft ** 2.0
    AR = (b ** 2.0) / S # Wing Aspect Ratio                 Dimensionless
    W = 4.807           # Aircraft Weight                   lbf
    m = W / g           # Aircraft Mass                     lbm
    CLo = 0.261         # Coefficient Lift at Zero Alpha    Dimensionless
    CLa = 5.865         # Lift Curve Slope                  /rad
    CLmax = 1.674       # Maximum Coefficient of Lift       Dimensionless
    LDmax = 23.09       # Maximum L/D Ratio                 Dimensionless
    CDo = 0.0223        # Parasitic Drag Coefficient        Dimensionless
    n_max = 2.0         # Maximum g Limit                   Dimensionless
    n_min = -1.5        # Minimum g Limit                   Dimensionless
    Vmo = 73.0          # Maximum Operating Speed           ft/s
    # Stall Speed (Sea Level)                             ft/s
    Vstall = math.sqrt((2.0 * W) / (rho0 * S * CLmax))
    # Induced Drag Coefficient                            Dimensionless
    K = 1.0 / (4.0 * CDo * (LDmax ** 2.0))
    # Best Glide Speed                                    ft/s
    Vmd = math.sqrt((2.0 * W) / (rho0 * S) * math.sqrt(K / CDo))
    # Best Glide Angle                                    rad
```

```
    theta_min = -math.atan(1.0 / LDmax)
    # Maximum Pitch Rate
    gamma_2_dot_max = math.pi / 12.0
    # Maximum Roll Rate
    gamma_1_dot_max = math.pi / 6.0

# ASW-20
elif acID == 'ASW-20':
    b = 49.3            # Wing Span                         ft
    S   = 112.6         # Wing Reference Area               ft ** 2.0
    AR = (b ** 2.0) / S # Wing Aspect Ratio                 Dimensionless
    W   = 759.0         # Aircraft Weight                   lbf
    m = W / g           # Aircraft Mass                     lbm
    CLo = 0.69          # Coefficient Lift at Zero Alpha    Dimensionless
    CLa = 4.90          # Lift Curve Slope                  /rad
    CLmax = 1.35        # Maximum Coefficient of Lift       Dimensionless
    LDmax = 38.77       # Maximum L/D Ratio                 Dimensionless
    CDo = 0.00924       # Parasitic Drag Coefficient        Dimensionless
    n_max = 4.0         # Maximum g Limit                   Dimensionless
    n_min = -1.5        # Minimum g Limit                   Dimensionless
    Vmo = 164.0         # Maximum Operating Speed           ft/s
    # Stall Speed (Sea Level)                               ft/s
    Vstall = math.sqrt((2.0 * W) / (rho0 * S * CLmax))
    # Induced Drag Coefficient                              Dimensionless
    K = 1.0 / (4.0 * CDo * (LDmax ** 2.0))
    # Best Glide Speed                                      ft/s
    Vmd = math.sqrt((2.0 * W) / (rho0 * S) * math.sqrt(K / CDo))
    # Best Glide Angle                                      rad
    theta_min = -math.atan(1.0 / LDmax)
    # Maximum Pitch Rate
    gamma_2_dot_max = math.pi / 12.0
    # Maximum Roll Rate
    gamma_1_dot_max = math.pi / 6.0

if costID == 'intEh_Err' or costID == 'intErr' or costID == 'ratio_Err_Eh':
    # Define Target Location
    x_tgt = 300.0
    y_tgt = 300.0

    # Camera Focal Length (mm)
    focal = 4.2
    # Convert Focal Length from mm to m
    focal = focal * 1.0e-03
    # Define the CCD Element Size (m)
    CCD_unit = 5.0e-06
    # Define the Desired Pixel Resolution
    pixel_res = 0.5
    # Define the Maximum Line of Site Distance for Desired Resolution
    d_los_max = (focal * pixel_res) / CCD_unit

    # Define the Camera Location
    xcam = 0.0
    ycam = 0.0
    zcam = 0.0
#end
```

```
if costID == 'seeability' or costID == 'seeability_constraint':
    # Define Target Location
    x_tgt = 0.0
    y_tgt = 0.0
    z_tgt = 0.0

    # Optimal Viewing Angle (rad)
    theta_opt = math.pi / 4.0
    # Camera Focal Length (mm)
    focal = 12.5
    # Convert Focal Length from mm to ft
    focal = focal * 0.00328084

    # Define a Scaling Factor
    mu = 1.0e04
    # Define the CCD Element Size (m)
    CCD_unit = 5.0e-06
    # Define the Desired Pixel Resolution (m)
    pixel_res = 0.25
    # Define Minimum Seeability Ratio
    S_min = mu * (CCD_unit / pixel_res)

# Instanciate Thermal Class
allen = thermalAllen(seed=12345.6)

# ============================================================================
# Define the Cost Function
# ============================================================================

def topt_cst(t,x,u,p,*args,**kwargs):

    '''
    Definition of the Cost Function

    Inputs
    t -> current time
    x -> current state vector
    u -> current control vector
    p -> parameter vector

    Output
    f -> cost of current trajectory
    '''

    # State Vector Definition
    V       = x[:,0]
    gamma_2 = x[:,1]
    r       = x[:,2]
    theta   = x[:,3]
    z       = x[:,4]
    epsilon = x[:,5]

    # Control Vector Definition
    alpha   = u[:,0]
    gamma_1 = u[:,1]
```

```
    if costID == 'intEh_Err' or costID == 'intErr' \
    or costID == 'ratio_Err_Eh':
        kappa   = u[:,2]
        tau     = u[:,3]
    #end

    if costID == 'Eh':

        # Maximize Specific Energy Height
        f = -(-z[-1]) - ((V[-1] ** 2.0) / (2.0 * g))

    elif costID == 'intEh':

        # Maximize Instantaneous Specific Energy Height
        Eh = numpy.zeros(len(t))

        for i in xrange(len(t)):
            Eh[i] = -(-z[i]) - ((V[i] ** 2.0) / (2.0 * g))
        #end

        f = numpy.trapz(Eh,t)

    elif costID == 'intEh_Err':

        mu = 10.0
        Eh = numpy.zeros(len(t))
        Err = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*
```

```
            (numpy.cos(kappa[i])),-numpy.sin(tau[i]),(numpy.cos(tau[i]))*
            (numpy.sin(kappa[i])),xcam],[(numpy.sin(tau[i]))*
            (numpy.cos(kappa[i])),numpy.cos(tau[i]),(numpy.sin(tau[i]))*
            (numpy.sin(kappa[i])),ycam],
            [-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            # Maximize Instantaneous Specific Energy Height
            Eh[i] = -(-z[i]) - ((V[i] ** 2.0) / (2.0 * g))

            # Minimize the Distance Between Camera Pointer and
            # Target Location
            Err[i] = mu * (((x_tgt - xlos) ** 2.0) +
            ((y_tgt - ylos) ** 2.0))
        #end

        f = numpy.trapz(Eh,t) + numpy.trapz(Err,t)

    elif costID == 'intErr':

        mu = 10.0
        Err = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
```

```
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0,0,0,1]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*
            (numpy.cos(kappa[i])),-numpy.sin(tau[i]),(numpy.cos(tau[i]))*
            (numpy.sin(kappa[i])),xcam],[(numpy.sin(tau[i]))*
            (numpy.cos(kappa[i])),numpy.cos(tau[i]),(numpy.sin(tau[i]))*
            (numpy.sin(kappa[i])),ycam],
            [-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0,0,0,1]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            # Minimize the Distance Between Camera Pointer and
            # Target Location
            Err[i] = mu * (((x_tgt - xlos) ** 2.0) +
            ((y_tgt - ylos) ** 2.0))
        #end

        f = numpy.trapz(Err,t)

    elif costID == 'ratio_Err_Eh':

        Eh = numpy.zeros(len(t))
        Err = numpy.zeros(len(t))
        r_Err_Eh = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
```

```
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*
            (numpy.cos(kappa[i])),-numpy.sin(tau[i]),(numpy.cos(tau[i]))*
            (numpy.sin(kappa[i])),xcam],[(numpy.sin(tau[i]))*
            (numpy.cos(kappa[i])),numpy.cos(tau[i]),(numpy.sin(tau[i]))*
            (numpy.sin(kappa[i])),ycam],
            [-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            # Define Instantaneous Specific Energy Height
            Eh[i] = -z[i] + ((V[i] ** 2.0) / (2.0 * g))

            # Define the Distance Between Camera Pointer and
            # Target Location
            Err[i] = ((x_tgt - xlos) ** 2.0) + \
            ((y_tgt - ylos) ** 2.0)

            # Minimize the Distance Between Camera Pointer and
            # Target Location While Maximizing the Specific Energy Height
            r_Err_Eh[i] = Err[i] / Eh[i]
        #end

        f = numpy.trapz(r_Err_Eh,t)

    elif costID == 'seeability':

        K_Sa = 0.5
        K_Eh = 0.5

        Eh = numpy.zeros(len(t))
        S = numpy.zeros(len(t))
        # Define the Scaling Factor
        mu1 = 1.0e03
        mu2 = 1.0e-03

        for i in xrange(len(t)):
```

```
            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])

            # Define the Viewing Angle and Distance
            dlos = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 +
            (-z[i] - z_tgt)**2.0)
            theta_v = numpy.arccos(-z[i] / dlos)

            # Maximize Seeability
            S[i] = mu1 * (numpy.cos(theta_v - theta_opt) /
            (1.0 + (dlos / focal)))

            # Maximize Instantaneous Specific Energy Height
            Eh[i] = mu2 * (-(-z[i]) - ((V[i] ** 2.0) / (2.0 * g)))

        f = (K_Sa * numpy.trapz(S,t)) + (K_Eh * numpy.trapz(Eh,t))
        #end

    elif costID == 'seeability_constraint':

        # Maximize Instantaneous Specific Energy Height
        Eh = numpy.zeros(len(t))

        for i in xrange(len(t)):
            Eh[i] = -(-z[i]) - ((V[i] ** 2.0) / (2.0 * g))
        #end

        f = numpy.trapz(Eh,t)

    fail = 0

    return f, fail

# ============================================================================
# Define the System Dynamics
# ============================================================================

def topt_dyn(t,x,u,p,*args,**kwargs):

    '''
    Definition of the System Dynamics

    Inputs
    t -> current time
    x -> current state vector
    u -> current control vector
    p -> parameter vector

    Output
    xdot -> equations of motion
    '''

    # State Vector Definition
    V       = x[:,0]
    gamma_2 = x[:,1]
```

```
    r       = x[:,2]
    theta   = x[:,3]
    z       = x[:,4]
    epsilon = x[:,5]

    # Control Vector Definition
    alpha   = u[:,0]
    gamma_1 = u[:,1]

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        kappa   = u[:,2]
        tau     = u[:,3]
    #end

    xdot = numpy.zeros((len(t),6))

    for i in xrange(len(t)):

        # Atmospheric Conditions
        conditions = atmosphere(-z[i],'ENG')
        rho = conditions['Density']

        # Aerodynamic Characteristics
        CL = CLo + (CLa * alpha[i])
        L = 0.5 * rho * (V[i] ** 2.0) * S * CL

        CD = CDo + K * CL ** 2.0
        D = 0.5 * rho * (V[i] ** 2.0) * S * CD

        # Updraft Strength and Derivatives
        data = allen(r=r[i],h_ac=-z[i],theta=theta[i],L=5280.0,W=5280.0,
            month_index=6)
        gradient = allen.getDerivatives(r=r[i], h_ac=-z[i],
            month_index=6, h=1e-20)

        # Updraft Strength
        W_z = -data['Updraft_Velocity'].real
        W_r = 0.0

        # Updraft Partial Derivatives (Cylindrical Coordinates)
        dWz_dr = -gradient['dWh_dr']
        dWz_dtheta = -gradient['dWh_dtheta']
        dWz_dz = -gradient['dWh_dh']

        r_dot = (V[i] * math.cos(gamma_2[i]) *
            math.cos(epsilon[i])) + W_r
        theta_dot = (V[i] * math.cos(gamma_2[i]) *
            math.sin(epsilon[i])) / r[i]
        z_dot = -V[i] * math.sin(gamma_2[i]) + W_z

        # Total Wind Derivatives
        W_z_dot = (dWz_dr * r_dot) + (dWz_dtheta * theta_dot) + \
            (dWz_dz * z_dot)

        # Total Wind Accelerations (Cartesian Coordinates)
```

```
        Wx_v_dot = -W_z_dot * math.sin(gamma_2[i])
        Wy_v_dot = 0.0
        Wz_v_dot = W_z_dot * math.cos(gamma_2[i])

        # Define the Equations of Motion
        xdot[i,0] = (-D - (m * g * math.sin(gamma_2[i])) +
            (m * Wx_v_dot)) / m
        xdot[i,1] = ((L * math.cos(gamma_1[i])) - (m * g *
            math.cos(gamma_2[i])) + (m * Wz_v_dot)) / (m * V[i])
        xdot[i,2] = r_dot
        xdot[i,3] = theta_dot
        xdot[i,4] = z_dot
        xdot[i,5] = (((L * math.sin(gamma_1[i])) + (m * Wy_v_dot)) / \
            (m * V[i] * math.cos(gamma_2[i]))) - theta_dot

        # Temporary Check for NaN and inf value
        for j in xrange(6):
            if abs(xdot[i,j]) > inf:
                #pdb.set_trace()
                fail = 1
                break
            elif isinstance(xdot[i],complex):
                #pdb.set_trace()
                fail = 1
                break
            elif int(xdot[i,j]+10000) == 0:
                #pdb.set_trace()
                fail = 1
                break
            #end
        #end
    #end

    fail = 0

    return xdot, fail

# ==============================================================================
# Define the Contraints
# ==============================================================================

def topt_con(t,x,u,p,*args,**kwargs):

    '''
    Definition of the System Constraints

    Inputs
    t -> current time
    x -> current state vector
    u -> current control vector
    p -> parameter vector

    Output
    ge -> equality constraints
    gi -> inequality constraints
    '''
```

```
# State Vector Definition
V       = x[:,0]
gamma_2 = x[:,1]
r       = x[:,2]
theta   = x[:,3]
z       = x[:,4]
epsilon = x[:,5]

# Control Vector Definition
alpha   = u[:,0]
gamma_1 = u[:,1]

if costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':
    kappa   = u[:,2]
    tau     = u[:,3]
#end

# Define the Equality Constraints (Initial Conditions)
ge = []

ge.append(V[0] - V_0)
ge.append(gamma_2[0] - gamma_2_0)
ge.append(r[0] - r_0)
ge.append(theta[0] - theta_0)
ge.append(z_0 - z[0])
ge.append(epsilon[0] - epsilon_0)
ge.append(alpha[0] - alpha_0)
ge.append(gamma_1[0] - gamma_1_0)

if costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':
    ge.append(kappa[0] - kappa_0)
    ge.append(tau[0] - tau_0)
#end

# Define the Inequality Constraints
gi = numpy.array([])

# Atmospheric Conditions
sigma = numpy.zeros(len(t))

for i in xrange(len(t)):
    conditions = atmosphere(-z[i],'ENG')
    sigma[i] = conditions['Relative_Dens']
#end

# Calculate the Load Factor
n = 1.0 / numpy.cos(gamma_1)

# Minimum Control Speed
Vmc = (1.1 * Vstall * (n ** 0.5)) / (sigma ** 0.5)
gi = numpy.concatenate((gi, Vmc - V))

# Maximum Operating Speed
```

```
Vne = Vmo / (sigma ** 0.5)
gi = numpy.concatenate((gi, V - Vne))

# Maximum Load Factor
gi = numpy.concatenate((gi, n - n_max))

# Minimum Load Factor
gi = numpy.concatenate((gi, n_min - n))

# Maximum Lift Coefficient
CL = CLo + (CLa * alpha)
gi = numpy.concatenate((gi, CL - CLmax))

# Maximum Pitch Rate
gamma_2_dot = (gamma_2[1:len(t)] - gamma_2[0:(len(t)-1)]) / \
    (t[1:len(t)] - t[0:(len(t)-1)])
gi = numpy.concatenate((gi, abs(gamma_2_dot) - gamma_2_dot_max))

# Maximum Roll Rate
gamma_1_dot = (gamma_1[1:len(t)] - gamma_1[0:(len(t)-1)]) / \
    (t[1:len(t)] - t[0:(len(t)-1)])
gi = numpy.concatenate((gi, abs(gamma_1_dot) - gamma_1_dot_max))

# Maximum LOS Vector
if costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':

    dlos = numpy.zeros(len(t))

    for i in xrange(len(t)):

        # Define the Aircraft Location
        xac = r[i] * numpy.cos(theta[i])
        yac = r[i] * numpy.sin(theta[i])
        zac = z[i]

        # Define the Camera Pointer Location
        H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
        (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
        epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
        epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
        (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
        (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
        (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
        [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
        gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
        (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
        (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
        -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
        (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
        gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
        [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
        gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
        gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
        [0.,0.,0.,1.]])
```

```
            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*
            (numpy.sin(kappa[i])),xcam],
            [(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),numpy.cos(tau[i]),
            (numpy.sin(tau[i]))*(numpy.sin(kappa[i])),ycam],
            [-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            dlos[i] = (deltaX**2.0 + deltaY**2.0 + (-H_ce[2,3])**2.0)**0.5
        #end

        gi = numpy.concatenate((gi, dlos - d_los_max))
    #end

    # Seeability Defined as a Constraint (Minimum Resolution)

    if costID == 'seeability_constraint':

        # Define a Scaling Factor
        mu = 1.0e04

        S = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])

            # Define the Viewing Angle and Distance
            dlos = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
                (-z[i] - z_tgt)**2.0)
            theta_v = numpy.arccos(-z[i] / dlos)

            # Maximize Seeability
            S[i] = mu * (numpy.cos(theta_v - theta_opt) / (1.0 +
                (dlos / focal)))
        #end

        gi = numpy.concatenate((gi, S_min - S))
    #end

    fail = 0

    return ge, gi, fail
```

```
# ====================================================================
# Trajectory Optimization Problem Definition
# ====================================================================

# Define the Number of Collocation Points
N = 120

traj = Trajectory('Thermal Climb Problem - Cularis')
traj.addPhase('Phase 1',N,topt_cst,topt_dyn,topt_con,colType='Trapezoidal')

traj[0].setKnots(start=[0.0,120.0])

V_0 = Vmd
gamma_2_0 = -math.pi / 100.0
r_0 = 150.0
theta_0 = 0.0
z_0 = -300.0
epsilon_0 = math.pi / 2.0

traj[0].setStates(6,\
    names=['V', 'gamma_2', 'r', 'theta', 'z', 'epsilon'],\
    start=[V_0, gamma_2_0, r_0, theta_0, z_0, epsilon_0],\
    final=[V_0, gamma_2_0, r_0, theta_0, z_0, epsilon_0],\
    lower=[Vstall, -math.pi/2.0, 25.0, 0.0, -9000.0, 0.5*math.pi/2.0],\
    upper=[Vmo, math.pi/2.0, 1000.0, inf, -50.0, 1.5*math.pi/2.0])

alpha_0 = theta_min
gamma_1_0 = 0.0

if costID == 'intEh_Err' or costID == 'intErr' or costID == 'ratio_Err_Eh':
    kappa_0 = -math.pi /3.0
    tau_0 = 0.0
#end

if costID == 'Eh' or costID =='intEh' or costID == 'seeability' or \
costID == 'seeability_constraint':

    traj[0].setControls(2,\
        names=['alpha',       'gamma_1'],\
        start=[alpha_0,       gamma_1_0],\
        final=[alpha_0,       gamma_1_0],\
        lower=[-math.pi/10.0, -math.pi/4.0],\
        upper=[math.pi/10.0,  math.pi/4.0])

elif costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':

    traj[0].setControls(4,\
        names=['alpha', 'gamma_1', 'kappa', 'tau'],\
        start=[alpha_0, gamma_1_0, kappa_0, tau_0],\
        final=[alpha_0, gamma_1_0, kappa_0, tau_0],\
        lower=[-math.pi/10.0, -math.pi/6.0, -math.pi*1.0/2.0, -math.pi],\
        upper=[math.pi/10.0, math.pi/6.0, -math.pi*1.0/5.0, math.pi])
#end
```

```
# ============================================================================
# Solve the Trajectory Optimization Problem
# ============================================================================

# Instanciate Optimizers
dcnlp = DCNLP()
snopt = SNOPT()

prntname = 'run_%s_%i_r%.1f_z%.1f_xtgt%.1f_ytgt%.1f_print.out' \
    %(acID,N,r_0,z_0,x_tgt,y_tgt)
snopt.setOption('Print file',prntname)
summname = 'run_%s_%i_r%.1f_z%.1f_xtgt%.1f_ytgt%.1f_summary.out' \
    %(acID,N,r_0,z_0,x_tgt,y_tgt)
snopt.setOption('Summary file',summname)
snopt.setOption('Major iterations limit',100000)
snopt.setOption('Iterations limit',1000000)
snopt.setOption('Major feasibility tolerance',1.0e-5)
snopt.setOption('Major optimality tolerance',1.0e-5)

t,x,u,p,f = dcnlp(traj,snopt,opt_solve_opts='sens_mode=''pgc''')

if(comm.Get_rank()==0):

    # States
    V       = x[0][:,0]
    gamma_2 = x[0][:,1]
    r       = x[0][:,2]
    theta   = x[0][:,3]
    z       = x[0][:,4]
    epsilon = x[0][:,5]

    # Controls
    alpha   = u[0][:,0]
    gamma_1 = u[0][:,1]

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        kappa   = u[0][:,2]
        tau     = u[0][:,3]
    #end

    # Cartesian Locations
    x = r * numpy.cos(theta)
    y = r * numpy.sin(theta)
    theta_fd = gamma_2 + alpha
    psi = epsilon + theta

    # Aerodynamics
    tmp = numpy.shape(t)
    SZ  = tmp[1]
    rho = numpy.zeros(SZ)
    CL  = numpy.zeros(SZ)

    for i in xrange(SZ):
        conditions = atmosphere(-z[i],'ENG')
        rho[i] = conditions['Density']
```

```
        CL[i]  = CLo + CLa*alpha[i]
    #end

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':

        xlos = numpy.zeros(SZ)
        ylos = numpy.zeros(SZ)
        dlos = numpy.zeros(SZ)

        for i in xrange(SZ):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*(numpy.sin(kappa[i])),
            xcam],[(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),
            numpy.cos(tau[i]),(numpy.sin(tau[i]))*(numpy.sin(kappa[i])),
            ycam],[-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos[i] = H_ce[0,3] + deltaX
            ylos[i] = H_ce[1,3] + deltaY
            dlos[i] = (deltaX**2.0 + deltaY**2.0 + (-H_ce[2,3])**2.0)**0.5
        #end
```

```
if costID == 'seeability':

    dlos = numpy.zeros(SZ)
    theta_v = numpy.zeros(SZ)
    S = numpy.zeros(SZ)
    Eh = numpy.zeros(SZ)

    # Define the Scaling Factor
    mu1 = 1.0e03
    mu2 = 1.0e-03

    for i in xrange(SZ):

        # Define the Aircraft Location
        xac = r[i] * numpy.cos(theta[i])
        yac = r[i] * numpy.sin(theta[i])

        # Define the Viewing Angle and Distance
        dlos[i] = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
            (-z[i] - z_tgt)**2.0)
        theta_v[i] = numpy.arccos(-z[i] / dlos[i])

        # Calculate the Seeability
        S[i] = mu1 * (numpy.cos(theta_v[i] - theta_opt) / \
            (1.0 + (dlos[i] / focal)))
        f1 = numpy.trapz(S,t)
        Eh[i] = mu2 * ((-z[i]) + ((V[i] ** 2.0) / (2.0 * g)))
        f2 = numpy.trapz(Eh,t)
    #end
#end

if costID == 'seeability_constraint':

    dlos = numpy.zeros(SZ)
    theta_v = numpy.zeros(SZ)
    S = numpy.zeros(SZ)

    for i in xrange(SZ):

        # Define the Aircraft Location
        xac = r[i] * numpy.cos(theta[i])
        yac = r[i] * numpy.sin(theta[i])

        # Define the Viewing Angle and Distance
        dlos[i] = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
            (-z[i] - z_tgt)**2.0)
        theta_v[i] = numpy.arccos(-z[i] / dlos[i])

        # Calculate the Seeability
        S[i] = (numpy.cos(theta_v[i] - theta_opt) / (1.0 +
            (dlos[i] / focal)))
    #end
#end

# Print Results
```

```
textf = 'f = %f\n' %(f)

textt = 't = [%f' %(t[0][0])
for i in xrange(1,len(t[0])):
    textt+= ',%f'%(t[0][i])
#end
textt += ']\n'

textx = 'x = [%f' %(x[0])
for i in xrange(1,len(x)):
    textx+= ',%f'%(x[i])
#end
textx += ']\n'

texty = 'y = [%f' %(y[0])
for i in xrange(1,len(y)):
    texty+= ',%f'%(y[i])
#end
texty += ']\n'

texth = 'h = [%f' %(-z[0])
for i in xrange(1,len(z)):
    texth+= ',%f'%(-z[i])
#end
texth += ']\n'

textr = 'r = [%f' %(r[0])
for i in xrange(1,len(r)):
    textr+= ',%f'%(r[i])
#end
textr += ']\n'

textv = 'v = [%f' %(V[0])
for i in xrange(1,len(V)):
    textv+= ',%f'%(V[i])
#end
textv += ']\n'

textswp = 'theta = [%f' %(theta[0])
for i in xrange(1,len(theta)):
    textswp+= ',%f'%(theta[i])
#end
textswp += ']\n'

texttht = 'theta_fd = [%f' %(theta_fd[0])
for i in xrange(1,len(theta_fd)):
    texttht+= ',%f'%(theta_fd[i])
#end
texttht += ']\n'

textpsi = 'psi = [%f' %(psi[0])
for i in xrange(1,len(psi)):
    textpsi+= ',%f'%(psi[i])
#end
textpsi += ']\n'
```

```
texteps = 'eps = [%f' %(epsilon[0])
for i in xrange(1,len(epsilon)):
    texteps+= ',%f'%(epsilon[i])
#end
texteps += ']\n'

textgam = 'gamma_2 = [%f' %(gamma_2[0])
for i in xrange(1,len(gamma_2)):
    textgam+= ',%f'%(gamma_2[i])
#end
textgam += ']\n'

textalp = 'alp = [%f' %(alpha[0])
for i in xrange(1,len(alpha)):
    textalp+= ',%f'%(alpha[i])
#end
textalp += ']\n'

textphi = 'gamma_1 = [%f' %(gamma_1[0])
for i in xrange(1,len(gamma_1)):
    textphi+= ',%f'%(gamma_1[i])
#end
textphi += ']\n'

textrho = 'rho = [%f' %(rho[0])
for i in xrange(1,len(rho)):
    textrho+= ',%f'%(rho[i])
#end
textrho += ']\n'

textlft = 'lft = [%f' %(CL[0])
for i in xrange(1,len(CL)):
    textlft+= ',%f'%(CL[i])
#end
textlft += ']\n'

if costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':
    textxlos = 'xlos = [%f' %(xlos[0])
    for i in xrange(1,len(xlos)):
        textxlos+= ',%f'%(xlos[i])
    #end
    textxlos += ']\n'

    textylos = 'ylos = [%f' %(ylos[0])
    for i in xrange(1,len(ylos)):
        textylos+= ',%f'%(ylos[i])
    #end
    textylos += ']\n'

    textdlos = 'dlos = [%f' %(dlos[0])
    for i in xrange(1,len(dlos)):
        textdlos+= ',%f'%(dlos[i])
    #end
    textdlos += ']\n'
```

```
        textkap = 'kappa = [%f' %(kappa[0])
        for i in xrange(1,len(kappa)):
            textkap+= ',%f'%(kappa[i])
        #end
        textkap += ']\n'

        texttau = 'tau = [%f' %(tau[0])
        for i in xrange(1,len(tau)):
            texttau+= ',%f'%(tau[i])
        #end
        texttau += ']\n'
    #end

    if costID == 'seeability':
        textdlos = 'dlos = [%f' %(dlos[0])
        for i in xrange(1,len(dlos)):
            textdlos+= ',%f'%(dlos[i])
        #end
        textdlos += ']\n'

        texttheta_v = 'theta_v = [%f' %(theta_v[0])
        for i in xrange(1,len(theta_v)):
            texttheta_v+= ',%f'%(theta_v[i])
        #end
        texttheta_v += ']\n'

        textS = 'S = [%f' %(S[0])
        for i in xrange(1,len(S)):
            textS+= ',%f'%(S[i])
        #end
        textS += ']\n'

        textEh = 'Eh = [%f' %(Eh[0])
        for i in xrange(1,len(Eh)):
            textEh+= ',%f'%(Eh[i])
        #end
        textEh += ']\n'

        textf1 = 'f1 = %f\n' %(f1)
        textf2 = 'f2 = %f\n' %(f2)
    #end

    if costID == 'seeability_constraint':
        textdlos = 'dlos = [%f' %(dlos[0])
        for i in xrange(1,len(dlos)):
            textdlos+= ',%f'%(dlos[i])
        #end
        textdlos += ']\n'

        texttheta_v = 'theta_v = [%f' %(theta_v[0])
        for i in xrange(1,len(theta_v)):
            texttheta_v+= ',%f'%(theta_v[i])
        #end
        texttheta_v += ']\n'

        textS = 'S = [%f' %(S[0])
```

```
        for i in xrange(1,len(S)):
            textS+= ',%f'%(S[i])
        #end
        textS += ']\n'
    #end

    filenam = 'run_%s_%i_r%.1f_z%.1f_xtgt%.1f_ytgt%.1f_post.out' \
        %(acID,N,r_0,z_0,x_tgt,y_tgt)
    fileout = open(filenam,'w')
    fileout.write(textf)
    fileout.write(textt)
    fileout.write(textx)
    fileout.write(texty)
    fileout.write(texth)
    fileout.write(textr)
    fileout.write(textv)
    fileout.write(textswp)
    fileout.write(texttht)
    fileout.write(textpsi)
    fileout.write(texteps)
    fileout.write(textgam)
    fileout.write(textalp)
    fileout.write(textphi)
    fileout.write(textrho)
    fileout.write(textlft)

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        fileout.write(textxlos)
        fileout.write(textylos)
        fileout.write(textdlos)
        fileout.write(textkap)
        fileout.write(texttau)
    #end

    if costID == 'seeability':
        fileout.write(textdlos)
        fileout.write(texttheta_v)
        fileout.write(textS)
        fileout.write(textEh)
        fileout.write(textf1)
        fileout.write(textf2)
    #end

    if costID == 'seeability_constraint':
        fileout.write(textdlos)
        fileout.write(texttheta_v)
        fileout.write(textS)
    #end

    fileout.close()
```

Listing B.4: Trajectory Optimization Problem Formulation - Bubble Model

```
#!/usr/local/bin/python
'''
```

```
Cularis_D_5223_thermalClimb

Trajectory Optimization Problem (in Cylindrical Coordinates)

Copyright (c) 2004-2014 by pyACDT Developers
All rights reserved.
Revision: 1.0  $Date: 22/01/2014 21:00$

References:
-----------



Developers:
-----------
- Major Thomas R. Connerty (TC)

History:
--------
    v. 1.0 - Initial Creation in Python (TC, 2014)

'''

__version__ = '$Revision: $'

'''
To Do:
    -
'''

# ==============================================================================
# Standard Python modules
# ==============================================================================
import os, math, sys, time
import pdb # pdb.set_trace()

# ==============================================================================
# External Python modules
# ==============================================================================
import mpi4py
from mpi4py import MPI
import numpy
import random

comm=MPI.COMM_WORLD

# ==============================================================================
# Extension modules
# ==============================================================================

sys.path.append(os.path.abspath('..'))
sys.path.append(os.path.abspath('../../../..'))
sys.path.append(os.path.abspath('../pyThermal'))
sys.path.append(os.path.abspath('../../../../pyACDT/Optimization/pyDTO'))
sys.path.append(os.path.abspath('../../../../pyACDT/Optimization/pyOpt'))
sys.path.append(os.path.abspath
```

```
    ('../../../../pyACDT/Optimization/pyOpt/pySNOPT'))

from pyAtmosphere_US1976 import atmosphere
from pyDCNLP import DCNLP
from pyDTO_controls import Controls
from pyDTO_parameters import Parameters
from pyDTO_knots import Knots
from pyDTO_problem import Phase,Trajectory
from pyDTO_states import States
from pySNOPT import SNOPT
from pyThermal_lawrance import thermalLawrance

# ================================================================================
# Misc Definitions
# ================================================================================
inf = 1e20  # define a value for infinity


# ================================================================================
# Define the  Aircraft Characteristics
# ================================================================================

# Misc Constants and Definitions
acID = 'D-5223'
costID = 'ratio_Err_Eh'
g = 32.174              # Acceleration due to Gravity       ft/s ** 2.0
rho0 = 0.0023769        # Density at Sea Level Conditions   slug/ft ** 3.0

# Cularis D-5223
if acID == 'D-5223':
    b = 8.536           # Wing Span                         ft
    S = 4.574           # Wing Reference Area               ft ** 2.0
    AR = (b ** 2.0) / S # Wing Aspect Ratio                 Dimensionless
    W = 4.807           # Aircraft Weight                   lbf
    m = W / g           # Aircraft Mass                     lbm
    CLo = 0.261         # Coefficient Lift at Zero Alpha    Dimensionless
    CLa = 5.865         # Lift Curve Slope                  /rad
    CLmax = 1.674       # Maximum Coefficient of Lift       Dimensionless
    LDmax = 23.09       # Maximum L/D Ratio                 Dimensionless
    CDo = 0.0223        # Parasitic Drag Coefficient        Dimensionless
    n_max = 2.0         # Maximum g Limit                   Dimensionless
    n_min = -1.5        # Minimum g Limit                   Dimensionless
    Vmo = 73.0          # Maximum Operating Speed           ft/s
    # Stall Speed (Sea Level)                             ft/s
    Vstall = math.sqrt((2.0 * W) / (rho0 * S * CLmax))
    # Induced Drag Coefficient                            Dimensionless
    K = 1.0 / (4.0 * CDo * (LDmax ** 2.0))
    # Best Glide Speed                                    ft/s
    Vmd = math.sqrt((2.0 * W) / (rho0 * S) * math.sqrt(K / CDo))
    # Best Glide Angle                                    rad
    theta_min = -math.atan(1.0 / LDmax)
    # Maximum Pitch Rate
    gamma_2_dot_max = math.pi / 12.0
    # Maximum Roll Rate
    gamma_1_dot_max = math.pi / 6.0
```

```
# ASW-20
elif acID == 'ASW-20':
    b = 49.3            # Wing Span                          ft
    S   = 112.6         # Wing Reference Area                ft ** 2.0
    AR = (b ** 2.0) / S # Wing Aspect Ratio                  Dimensionless
    W   = 759.0         # Aircraft Weight                    lbf
    m = W / g           # Aircraft Mass                      lbm
    CLo = 0.69          # Coefficient Lift at Zero Alpha     Dimensionless
    CLa = 4.90          # Lift Curve Slope                   /rad
    CLmax = 1.35        # Maximum Coefficient of Lift        Dimensionless
    LDmax = 36.27       # Maximum L/D Ratio                  Dimensionless
    CDo = 0.00924       # Parasitic Drag Coefficient         Dimensionless
    n_max = 4.0         # Maximum g Limit                    Dimensionless
    n_min = -1.5        # Minimum g Limit                    Dimensionless
    Vmo = 164.0         # Maximum Operating Speed            ft/s
    # Stall Speed (Sea Level)                                ft/s
    Vstall = math.sqrt((2.0 * W) / (rho0 * S * CLmax))
    # Induced Drag Coefficient                               Dimensionless
    K = 1.0 / (4.0 * CDo * (LDmax ** 2.0))
    # Best Glide Speed                                       ft/s
    Vmd = math.sqrt((2.0 * W) / (rho0 * S) * math.sqrt(K / CDo))
    # Best Glide Angle                                       rad
    theta_min = -math.atan(1.0 / LDmax)
    # Maximum Pitch Rate
    gamma_2_dot_max = math.pi / 12.0
    # Maximum Roll Rate
    gamma_1_dot_max = math.pi / 6.0

if costID == 'intEh_Err' or costID == 'intErr' or costID == 'ratio_Err_Eh':
    # Define Target Location
    x_tgt = 300.0
    y_tgt = 300.0

    # Camera Focal Length (mm)
    focal = 42.0
    # Convert Focal Length from mm to m
    focal = focal * 1.0e-03
    # Define the CCD Element Size (m)
    CCD_unit = 5.0e-06
    # Define the Desired Pixel Resolution
    pixel_res = 0.5
    # Define the Maximum Line of Site Distance for Desired Resolution
    d_los_max = (focal * pixel_res) / CCD_unit

    # Define the Camera Location
    xcam = 0.0
    ycam = 0.0
    zcam = 0.0
#end

if costID == 'seeability' or costID == 'seeability_constraint':
    # Define Target Location
    x_tgt = 0.0
    y_tgt = 0.0
    z_tgt = 0.0
```

```
    # Optimal Viewing Angle (rad)
    theta_opt = math.pi / 4.0
    # Camera Focal Length (mm)
    focal = 12.5
    # Convert Focal Length from mm to ft
    focal = focal * 0.00328084

    # Define a Scaling Factor
    mu = 1.0e04
    # Define the CCD Element Size (m)
    CCD_unit = 5.0e-06
    # Define the Desired Pixel Resolution (m)
    pixel_res = 0.25
    # Define Minimum Seeability Ratio
    S_min = mu * (CCD_unit / pixel_res)

# Instanciate Thermal Class
lawrance = thermalLawrance(seed=12345.6)

# Define Thermal Characteristics
R = 500.0               # Distance that Limits the Updraft Area (ft)
w_core = 10.0           # Updraft velocity at center of thermal (ft/s)
z_t = 1500.0            # Height of the thermal bubble (ft)

# ========================================================================
# Define the Cost Function
# ========================================================================

def topt_cst(t,x,u,p,*args,**kwargs):

    '''
    Definition of the Cost Function

    Inputs
    t -> current time
    x -> current state vector
    u -> current control vector
    p -> parameter vector

    Output
    f -> cost of current trajectory
    '''

    # State Vector Definition
    V       = x[:,0]
    gamma_2 = x[:,1]
    r       = x[:,2]
    theta   = x[:,3]
    z       = x[:,4]
    epsilon = x[:,5]

    # Control Vector Definition
    alpha   = u[:,0]
    gamma_1 = u[:,1]

    if costID == 'intEh_Err' or costID == 'intErr' or \
```

```
    costID == 'ratio_Err_Eh':
        kappa   = u[:,2]
        tau     = u[:,3]
    #end

    if costID == 'Eh':

        # Maximize Specific Energy Height
        f = -(-z[-1]) - ((V[-1] ** 2.0) / (2.0 * g))

    elif costID == 'intEh':

        # Maximize Instantaneous Specific Energy Height
        Eh = numpy.zeros(len(t))

        for i in xrange(len(t)):
            Eh[i] = -(-z[i]) - ((V[i] ** 2.0) / (2.0 * g))
        #end

        f = numpy.trapz(Eh,t)

    elif costID == 'intEh_Err':

        mu = 10.0
        Eh = numpy.zeros(len(t))
        Err = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*(numpy.sin(kappa[i])),
```

```
            xcam],[(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),
            numpy.cos(tau[i]),(numpy.sin(tau[i]))*(numpy.sin(kappa[i])),
            ycam],[-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            # Maximize Instantaneous Specific Energy Height
            Eh[i] = -(-z[i]) - ((V[i] ** 2.0) / (2.0 * g))

            # Minimize the Distance Between Camera Pointer and
            # Target Location
            Err[i] = mu * (((x_tgt - xlos) ** 2.0) +
                ((y_tgt - ylos) ** 2.0))
        #end

        f = numpy.trapz(Eh,t) + numpy.trapz(Err,t)

    elif costID == 'intErr':

        mu = 10.0
        Err = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
```

```
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0,0,0,1]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*(numpy.sin(kappa[i])),
            xcam],[(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),
            numpy.cos(tau[i]),(numpy.sin(tau[i]))*(numpy.sin(kappa[i])),
            ycam],[-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0,0,0,1]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            # Minimize the Distance Between Camera Pointer and
            # Target Location
            Err[i] = mu * (((x_tgt - xlos) ** 2.0) +
                ((y_tgt - ylos) ** 2.0))
        #end

        f = numpy.trapz(Err,t)

    elif costID == 'ratio_Err_Eh':

        Eh = numpy.zeros(len(t))
        Err = numpy.zeros(len(t))
        r_Err_Eh = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
```

```
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*(numpy.sin(kappa[i])),
            xcam],[(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),
            numpy.cos(tau[i]),(numpy.sin(tau[i]))*(numpy.sin(kappa[i])),
            ycam],[-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            # Define Instantaneous Specific Energy Height
            Eh[i] = -z[i] + ((V[i] ** 2.0) / (2.0 * g))

            # Define the Distance Between Camera Pointer and
            # Target Location
            Err[i] = ((x_tgt - xlos) ** 2.0) + \
                ((y_tgt - ylos) ** 2.0)

            # Minimize the Distance Between Camera Pointer and
            # Target Location While Maximizing the Specific Energy Height
            r_Err_Eh[i] = Err[i] / Eh[i]
        #end

        f = numpy.trapz(r_Err_Eh,t)

    elif costID == 'seeability':

        K_Sa = 0.5
        K_Eh = 0.5

        Eh = numpy.zeros(len(t))
        S = numpy.zeros(len(t))
        # Define the Scaling Factor
        mu1 = 1.0e03
        mu2 = 1.0e-03

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
```

```
            # Define the Viewing Angle and Distance
            dlos = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
                (-z[i] - z_tgt)**2.0)
            theta_v = numpy.arccos(-z[i] / dlos)

            # Maximize Seeability
            S[i] = mu1 * (numpy.cos(theta_v - theta_opt) / (1.0 +
                (dlos / focal)))

            # Maximize Instantaneous Specific Energy Height
            Eh[i] = mu2 * (-(-z[i]) - ((V[i] ** 2.0) / (2.0 * g)))

        f = (K_Sa * numpy.trapz(S,t)) + (K_Eh * numpy.trapz(Eh,t))
        #end

    elif costID == 'seeability_constraint':

        # Maximize Instantaneous Specific Energy Height
        Eh = numpy.zeros(len(t))

        for i in xrange(len(t)):
            Eh[i] = -(-z[i]) - ((V[i] ** 2.0) / (2.0 * g))
        #end

        f = numpy.trapz(Eh,t)

    fail = 0

    return f, fail

# ======================================================================
# Define the System Dynamics
# ======================================================================

def topt_dyn(t,x,u,p,*args,**kwargs):

    '''
    Definition of the System Dynamics

    Inputs
    t -> current time
    x -> current state vector
    u -> current control vector
    p -> parameter vector

    Output
    xdot -> equations of motion
    '''

    # State Vector Definition
    V       = x[:,0]
    gamma_2 = x[:,1]
    r       = x[:,2]
    theta   = x[:,3]
    z       = x[:,4]
    epsilon = x[:,5]
```

```
    # Control Vector Definition
    alpha   = u[:,0]
    gamma_1 = u[:,1]

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        kappa   = u[:,2]
        tau     = u[:,3]
    #end

    xdot = numpy.zeros((len(t),6))

    for i in xrange(len(t)):

        # Atmospheric Conditions
        conditions = atmosphere(-z[i],'ENG')
        rho = conditions['Density']

        # Aerodynamic Characteristics
        CL = CLo + (CLa * alpha[i])
        L = 0.5 * rho * (V[i] ** 2.0) * S * CL

        CD = CDo + K * CL ** 2.0
        D = 0.5 * rho * (V[i] ** 2.0) * S * CD

        # Updraft Strength and Derivatives
        data = lawrance(r=r[i],z_ac=-z[i],R=R,w_core=w_core,z_t=z_t,
            theta=theta[i],L=5280.0,W=5280.0)
        gradient = lawrance.getDerivatives(r=r[i],z_ac=-z[i],R=R,
            w_core=w_core,z_t=z_t,theta=theta[i],h=1e-20)

        # Updraft Strength
        W_z = data['Updraft_Velocity']
        W_r = data['Updraft_Radial_Velocity']

        # Updraft Partial Derivatives (Cylindrical Coordinates)
        dWz_dr = gradient['dWh_dr']
        dWz_dtheta = gradient['dWh_dtheta']
        dWz_dz = gradient['dWh_dh']
        dWr_dr = gradient['dWr_dr']
        dWr_dtheta = gradient['dWr_dtheta']
        dWr_dh = gradient['dWr_dh']

        r_dot = (V[i] * math.cos(gamma_2[i]) * \
            math.cos(epsilon[i])) + W_r
        theta_dot = (V[i] * math.cos(gamma_2[i]) * \
            math.sin(epsilon[i])) / r[i]
        z_dot = -V[i] * math.sin(gamma_2[i]) + W_z

        # Total Wind Derivatives
        W_z_dot = (dWz_dr * r_dot) + (dWz_dtheta * theta_dot) + \
            (dWz_dz * z_dot)
        W_r_dot = (dWr_dr * r_dot) + (dWr_dtheta * theta_dot) + \
            (dWr_dh * z_dot)
        W_x_dot = (W_r_dot * math.cos(theta[i])) - (W_r *
```

```
            math.sin(theta[i]) * theta_dot)
        W_y_dot = (W_r_dot * math.sin(theta[i])) + (W_r *
            math.cos(theta[i]) * theta_dot)

        # Total Wind Accelerations (Cartesian Coordinates)
        Wx_v_dot = (W_x_dot * math.cos(gamma_2[i]) * math.cos(epsilon[i] +
            theta[i])) + (W_y_dot * math.cos(gamma_2[i]) *
            math.sin(epsilon[i] + theta[i])) - (W_z_dot *
            math.sin(gamma_2[i]))
        Wy_v_dot = -(W_x_dot * math.sin(epsilon[i] + theta[i])) + \
            (W_y_dot * math.cos(epsilon[i] + theta[i]))
        Wz_v_dot = (W_x_dot * math.sin(gamma_2[i]) * math.cos(epsilon[i] +
            theta[i])) + (W_y_dot * math.sin(gamma_2[i]) *
            math.sin(epsilon[i] + theta[i])) + (W_z_dot *
            math.cos(gamma_2[i]))

        # Define the Equations of Motion
        xdot[i,0] = (-D - (m * g * math.sin(gamma_2[i])) +
            (m * Wx_v_dot)) / m
        xdot[i,1] = ((L * math.cos(gamma_1[i])) - (m * g *
            math.cos(gamma_2[i])) + (m * Wz_v_dot)) / (m * V[i])
        xdot[i,2] = r_dot
        xdot[i,3] = theta_dot
        xdot[i,4] = z_dot
        xdot[i,5] = (((L * math.sin(gamma_1[i])) + (m * Wy_v_dot)) /
            (m * V[i] * math.cos(gamma_2[i]))) - theta_dot

        # Temporary Check for NaN and inf value
        for j in xrange(6):
            if abs(xdot[i,j]) > inf:
                #pdb.set_trace()
                fail = 1
                break
            elif isinstance(xdot[i],complex):
                #pdb.set_trace()
                fail = 1
                break
            elif int(xdot[i,j]+10000) == 0:
                #pdb.set_trace()
                fail = 1
                break
            #end
        #end
    #end

    fail = 0

    return xdot, fail

# ============================================================================
# Define the Contraints
# ============================================================================

def topt_con(t,x,u,p,*args,**kwargs):

    '''
```

```
    Definition of the System Constraints

    Inputs
    t -> current time
    x -> current state vector
    u -> current control vector
    p -> parameter vector

    Output
    ge -> equality constraints
    gi -> inequality constraints
    '''

    # State Vector Definition
    V       = x[:,0]
    gamma_2 = x[:,1]
    r       = x[:,2]
    theta   = x[:,3]
    z       = x[:,4]
    epsilon = x[:,5]

    # Control Vector Definition
    alpha   = u[:,0]
    gamma_1 = u[:,1]

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        kappa   = u[:,2]
        tau     = u[:,3]
    #end

    # Define the Equality Constraints (Initial Conditions)
    ge = []

    ge.append(V[0] - V_0)
    ge.append(gamma_2[0] - gamma_2_0)
    ge.append(r[0] - r_0)
    ge.append(theta[0] - theta_0)
    ge.append(z_0 - z[0])
    ge.append(epsilon[0] - epsilon_0)
    ge.append(alpha[0] - alpha_0)
    ge.append(gamma_1[0] - gamma_1_0)

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        ge.append(kappa[0] - kappa_0)
        ge.append(tau[0] - tau_0)
    #end

    # Define the Inequality Constraints
    gi = numpy.array([])

    # Atmospheric Conditions
    sigma = numpy.zeros(len(t))

    for i in xrange(len(t)):
```

```
        conditions = atmosphere(-z[i],'ENG')
        sigma[i] = conditions['Relative_Dens']
    #end

    # Calculate the Load Factor
    n = 1.0 / numpy.cos(gamma_1)

    # Minimum Control Speed
    Vmc = (1.1 * Vstall * (n ** 0.5)) / (sigma ** 0.5)
    gi = numpy.concatenate((gi, Vmc - V))

    # Maximum Operating Speed
    Vne = Vmo / (sigma ** 0.5)
    gi = numpy.concatenate((gi, V - Vne))

    # Maximum Load Factor
    gi = numpy.concatenate((gi, n - n_max))

    # Minimum Load Factor
    gi = numpy.concatenate((gi, n_min - n))

    # Maximum Lift Coefficient
    CL = CLo + (CLa * alpha)
    gi = numpy.concatenate((gi, CL - CLmax))

    # Maximum Pitch Rate
    gamma_2_dot = (gamma_2[1:len(t)] - gamma_2[0:(len(t)-1)]) / \
        (t[1:len(t)] - t[0:(len(t)-1)])
    gi = numpy.concatenate((gi, abs(gamma_2_dot) - gamma_2_dot_max))

    # Maximum Roll Rate
    gamma_1_dot = (gamma_1[1:len(t)] - gamma_1[0:(len(t)-1)]) / \
        (t[1:len(t)] - t[0:(len(t)-1)])
    gi = numpy.concatenate((gi, abs(gamma_1_dot) - gamma_1_dot_max))

    # Maximum LOS Vector
    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':

        dlos = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
```

```
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*(numpy.sin(kappa[i])),
            xcam],[(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),
            numpy.cos(tau[i]),(numpy.sin(tau[i]))*(numpy.sin(kappa[i])),
            ycam],[-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)

            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos = H_ce[0,3] + deltaX
            ylos = H_ce[1,3] + deltaY

            dlos[i] = (deltaX**2.0 + deltaY**2.0 + (-H_ce[2,3])**2.0)**0.5
        #end

        gi = numpy.concatenate((gi, dlos - d_los_max))
    #end

    # Seeability Defined as a Constraint (Minimum Resolution)

    if costID == 'seeability_constraint':

        # Define a Scaling Factor
        mu = 1.0e04

        S = numpy.zeros(len(t))

        for i in xrange(len(t)):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])

            # Define the Viewing Angle and Distance
            dlos = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
                (-z[i] - z_tgt)**2.0)
            theta_v = numpy.arccos(-z[i] / dlos)
```

```
            # Maximize Seeability
            S[i] = mu * (numpy.cos(theta_v - theta_opt) / (1.0 +
                (dlos / focal)))
        #end

        gi = numpy.concatenate((gi, S_min - S))
    #end

    fail = 0

    return ge, gi, fail

# ======================================================================
# Trajectory Optimization Problem Definition
# ======================================================================

# Define the Number of Collocation Points
N = 120

traj = Trajectory('Thermal Climb Problem - Cularis')
traj.addPhase('Phase 1',N,topt_cst,topt_dyn,topt_con,colType='Trapezoidal')

traj[0].setKnots(start=[0.0,120.0])

V_0 = Vmd
gamma_2_0 = -math.pi / 100.0
r_0 = 150.0
theta_0 = 0.0
z_0 = -300.0
epsilon_0 = math.pi / 2.0

traj[0].setStates(6,\
    names=['V', 'gamma_2', 'r', 'theta', 'z', 'epsilon'],\
    start=[V_0, gamma_2_0, r_0, theta_0, z_0, epsilon_0],\
    final=[V_0, gamma_2_0, r_0, theta_0, z_0, epsilon_0],\
    lower=[Vstall, -math.pi/2.0, 25.0, 0.0, -9000.0, 0.5*math.pi/2.0],\
    upper=[Vmo, math.pi/2, 400.0, inf, -50.0, 1.5*math.pi/2.0])

alpha_0 = theta_min
gamma_1_0 = 0.0

if costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':
    kappa_0 = -math.pi*1.0/3.0
    tau_0 = 0.0
#end

if costID == 'Eh' or costID =='intEh':

    traj[0].setControls(2,\
        names=['alpha',     'gamma_1'],\
        start=[alpha_0,     gamma_1_0],\
        final=[alpha_0,     gamma_1_0],\
        lower=[-math.pi/10, -math.pi/4],\
        upper=[math.pi/10,  math.pi/4])
```

```
elif costID == 'intEh_Err' or costID == 'intErr' or \
costID == 'ratio_Err_Eh':

    traj[0].setControls(4,\
        names=['alpha', 'gamma_1', 'kappa', 'tau'],\
        start=[alpha_0, gamma_1_0, kappa_0, tau_0],\
        final=[alpha_0, gamma_1_0, kappa_0, tau_0],\
        lower=[-math.pi/10.0, -math.pi/4.0, -math.pi*1.0/2.0, -math.pi],\
        upper=[math.pi/10.0, math.pi/4.0, -math.pi*1.0/3.0, math.pi])
#end

# ======================================================================
# Solve the Trajectory Optimization Problem
# ======================================================================

# Instanciate Optimizers
dcnlp = DCNLP()
snopt = SNOPT()

prntname = 'run_%s_%i_r%.1f_z%.1f_xtgt%.1f_ytgt%.1f_print.out' \
    %(acID,N,r_0,z_0,x_tgt,y_tgt)
snopt.setOption('Print file',prntname)
summname = 'run_%s_%i_r%.1f_z%.1f_xtgt%.1f_ytgt%.1f_summary.out' \
    %(acID,N,r_0,z_0,x_tgt,y_tgt)
snopt.setOption('Summary file',summname)
snopt.setOption('Major iterations limit',100000)
snopt.setOption('Iterations limit',1000000)
snopt.setOption('Major feasibility tolerance',1.0e-5)
snopt.setOption('Major optimality tolerance',1.0e-5)

t,x,u,p,f = dcnlp(traj,snopt,opt_solve_opts='sens_mode=''pgc''')

if(comm.Get_rank()==0):

    # States
    V       = x[0][:,0]
    gamma_2 = x[0][:,1]
    r       = x[0][:,2]
    theta   = x[0][:,3]
    z       = x[0][:,4]
    epsilon = x[0][:,5]

    # Controls
    alpha   = u[0][:,0]
    gamma_1 = u[0][:,1]

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        kappa   = u[0][:,2]
        tau     = u[0][:,3]
    #end

    # Cartesian Locations
    x = r * numpy.cos(theta)
    y = r * numpy.sin(theta)
    theta_fd = gamma_2 + alpha
```

```
    psi = epsilon + theta

    # Aerodynamics
    tmp = numpy.shape(t)
    SZ  = tmp[1]
    rho = numpy.zeros(SZ)
    CL  = numpy.zeros(SZ)

    for i in xrange(SZ):
        conditions = atmosphere(-z[i],'ENG')
        rho[i] = conditions['Density']
        CL[i]  = CLo + CLa*alpha[i]
    #end

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':

        xlos = numpy.zeros(SZ)
        ylos = numpy.zeros(SZ)
        dlos = numpy.zeros(SZ)

        for i in xrange(SZ):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])
            zac = z[i]

            # Define the Camera Pointer Location
            H_be = numpy.array([[(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(alpha[i]+gamma_2[i])),-(numpy.sin(theta[i]+
            epsilon[i]))*(numpy.cos(gamma_1[i])) + (numpy.cos(theta[i]+
            epsilon[i]))*(numpy.sin(alpha[i]+gamma_2[i]))*
            (numpy.sin(gamma_1[i])),(numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(gamma_1[i])) + (numpy.cos(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.cos(gamma_1[i])),xac],
            [(numpy.sin(theta[i]+epsilon[i]))*(numpy.cos(alpha[i]+
            gamma_2[i])),(numpy.cos(theta[i]+epsilon[i]))*
            (numpy.cos(gamma_1[i])) + (numpy.sin(theta[i]+epsilon[i]))*
            (numpy.sin(alpha[i]+gamma_2[i]))*(numpy.sin(gamma_1[i])),
            -(numpy.cos(theta[i]+epsilon[i]))*(numpy.sin(gamma_1[i])) +
            (numpy.sin(theta[i]+epsilon[i]))*(numpy.sin(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),yac],
            [-numpy.sin(alpha[i]+gamma_2[i]),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.sin(gamma_1[i])),(numpy.cos(alpha[i]+
            gamma_2[i]))*(numpy.cos(gamma_1[i])),zac],
            [0.,0.,0.,1.]])

            H_bc = numpy.array([[(numpy.cos(tau[i]))*(numpy.cos(kappa[i])),
            -numpy.sin(tau[i]),(numpy.cos(tau[i]))*(numpy.sin(kappa[i])),
            xcam],[(numpy.sin(tau[i]))*(numpy.cos(kappa[i])),
            numpy.cos(tau[i]),(numpy.sin(tau[i]))*(numpy.sin(kappa[i])),
            ycam],[-numpy.sin(kappa[i]),0,numpy.cos(kappa[i]),zcam],
            [0.,0.,0.,1.]])

            H_ce = numpy.dot(H_be,H_bc)
```

```
            zeta = numpy.arctan(H_ce[0,0]/H_ce[2,0])
            eta = numpy.arctan(H_ce[1,0]/H_ce[2,0])

            deltaX = -H_ce[2,3] * numpy.tan(zeta)
            deltaY = -H_ce[2,3] * numpy.tan(eta)

            xlos[i] = H_ce[0,3] + deltaX
            ylos[i] = H_ce[1,3] + deltaY
            dlos[i] = (deltaX**2.0 + deltaY**2.0 + (-H_ce[2,3])**2.0)**0.5
        #end

    if costID == 'seeability':

        dlos = numpy.zeros(SZ)
        theta_v = numpy.zeros(SZ)
        S = numpy.zeros(SZ)
        Eh = numpy.zeros(SZ)

        # Define the Scaling Factor
        mu1 = 1.0e03
        mu2 = 1.0e-03

        for i in xrange(SZ):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])

            # Define the Viewing Angle and Distance
            dlos[i] = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
                (-z[i] - z_tgt)**2.0)
            theta_v[i] = numpy.arccos(-z[i] / dlos[i])

            # Calculate the Seeability
            S[i] = mu1 * (numpy.cos(theta_v[i] - theta_opt) / (1.0 +
                (dlos[i] / focal)))
            f1 = numpy.trapz(S,t)
            Eh[i] = mu2 * ((-z[i]) + ((V[i] ** 2.0) / (2.0 * g)))
            f2 = numpy.trapz(Eh,t)
        #end
    #end

    if costID == 'seeability_constraint':

        dlos = numpy.zeros(SZ)
        theta_v = numpy.zeros(SZ)
        S = numpy.zeros(SZ)

        for i in xrange(SZ):

            # Define the Aircraft Location
            xac = r[i] * numpy.cos(theta[i])
            yac = r[i] * numpy.sin(theta[i])

            # Define the Viewing Angle and Distance
```

```
            dlos[i] = math.sqrt((xac - x_tgt)**2.0 + (yac - y_tgt)**2.0 + \
                (-z[i] - z_tgt)**2.0)
            theta_v[i] = numpy.arccos(-z[i] / dlos[i])

            # Calculate the Seeability
            S[i] = (numpy.cos(theta_v[i] - theta_opt) / (1.0 +
                (dlos[i] / focal)))
        #end
    #end

    # Print Results
    textf = 'f = %f\n' %(f)

    textt = 't = [%f' %(t[0][0])
    for i in xrange(1,len(t[0])):
        textt+= ',%f'%(t[0][i])
    #end
    textt += ']\n'

    textx = 'x = [%f' %(x[0])
    for i in xrange(1,len(x)):
        textx+= ',%f'%(x[i])
    #end
    textx += ']\n'

    texty = 'y = [%f' %(y[0])
    for i in xrange(1,len(y)):
        texty+= ',%f'%(y[i])
    #end
    texty += ']\n'

    texth = 'h = [%f' %(-z[0])
    for i in xrange(1,len(z)):
        texth+= ',%f'%(-z[i])
    #end
    texth += ']\n'

    textr = 'r = [%f' %(r[0])
    for i in xrange(1,len(r)):
        textr+= ',%f'%(r[i])
    #end
    textr += ']\n'

    textv = 'v = [%f' %(V[0])
    for i in xrange(1,len(V)):
        textv+= ',%f'%(V[i])
    #end
    textv += ']\n'

    textswp = 'theta = [%f' %(theta[0])
    for i in xrange(1,len(theta)):
        textswp+= ',%f'%(theta[i])
    #end
    textswp += ']\n'

    texttht = 'theta_fd = [%f' %(theta_fd[0])
```

```
    for i in xrange(1,len(theta_fd)):
        texttht+= ',%f'%(theta_fd[i])
    #end
    texttht += ']\n'

    textpsi = 'psi = [%f' %(psi[0])
    for i in xrange(1,len(psi)):
        textpsi+= ',%f'%(psi[i])
    #end
    textpsi += ']\n'

    texteps = 'eps = [%f' %(epsilon[0])
    for i in xrange(1,len(epsilon)):
        texteps+= ',%f'%(epsilon[i])
    #end
    texteps += ']\n'

    textgam = 'gamma_2 = [%f' %(gamma_2[0])
    for i in xrange(1,len(gamma_2)):
        textgam+= ',%f'%(gamma_2[i])
    #end
    textgam += ']\n'

    textalp = 'alp = [%f' %(alpha[0])
    for i in xrange(1,len(alpha)):
        textalp+= ',%f'%(alpha[i])
    #end
    textalp += ']\n'

    textphi = 'gamma_1 = [%f' %(gamma_1[0])
    for i in xrange(1,len(gamma_1)):
        textphi+= ',%f'%(gamma_1[i])
    #end
    textphi += ']\n'

    textrho = 'rho = [%f' %(rho[0])
    for i in xrange(1,len(rho)):
        textrho+= ',%f'%(rho[i])
    #end
    textrho += ']\n'

    textlft = 'lft = [%f' %(CL[0])
    for i in xrange(1,len(CL)):
        textlft+= ',%f'%(CL[i])
    #end
    textlft += ']\n'

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        textxlos = 'xlos = [%f' %(xlos[0])
        for i in xrange(1,len(xlos)):
            textxlos+= ',%f'%(xlos[i])
        #end
        textxlos += ']\n'

        textylos = 'ylos = [%f' %(ylos[0])
```

```
        for i in xrange(1,len(ylos)):
            textylos+= ',%f'%(ylos[i])
        #end
        textylos += ']\n'

        textdlos = 'dlos = [%f' %(dlos[0])
        for i in xrange(1,len(dlos)):
            textdlos+= ',%f'%(dlos[i])
        #end
        textdlos += ']\n'

        textkap = 'kappa = [%f' %(kappa[0])
        for i in xrange(1,len(kappa)):
            textkap+= ',%f'%(kappa[i])
        #end
        textkap += ']\n'

        texttau = 'tau = [%f' %(tau[0])
        for i in xrange(1,len(tau)):
            texttau+= ',%f'%(tau[i])
        #end
        texttau += ']\n'
    #end

    if costID == 'seeability':
        textdlos = 'dlos = [%f' %(dlos[0])
        for i in xrange(1,len(dlos)):
            textdlos+= ',%f'%(dlos[i])
        #end
        textdlos += ']\n'

        texttheta_v = 'theta_v = [%f' %(theta_v[0])
        for i in xrange(1,len(theta_v)):
            texttheta_v+= ',%f'%(theta_v[i])
        #end
        texttheta_v += ']\n'

        textS = 'S = [%f' %(S[0])
        for i in xrange(1,len(S)):
            textS+= ',%f'%(S[i])
        #end
        textS += ']\n'

        textEh = 'Eh = [%f' %(Eh[0])
        for i in xrange(1,len(Eh)):
            textEh+= ',%f'%(Eh[i])
        #end
        textEh += ']\n'

        textf1 = 'f1 = %f\n' %(f1)
        textf2 = 'f2 = %f\n' %(f2)
    #end

    if costID == 'seeability_constraint':
        textdlos = 'dlos = [%f' %(dlos[0])
        for i in xrange(1,len(dlos)):
```

```
            textdlos+= ',%f'%(dlos[i])
        #end
        textdlos += ']\n'

        texttheta_v = 'theta_v = [%f' %(theta_v[0])
        for i in xrange(1,len(theta_v)):
            texttheta_v+= ',%f'%(theta_v[i])
        #end
        texttheta_v += ']\n'

        textS = 'S = [%f' %(S[0])
        for i in xrange(1,len(S)):
            textS+= ',%f'%(S[i])
        #end
        textS += ']\n'
    #end

    filenam = 'run_%s_%i_r%.1f_z%.1f_xtgt%.1f_ytgt%.1f_post.out' \
        %(acID,N,r_0,z_0,x_tgt,y_tgt)
    fileout = open(filenam,'w')
    fileout.write(textf)
    fileout.write(textt)
    fileout.write(textx)
    fileout.write(texty)
    fileout.write(texth)
    fileout.write(textr)
    fileout.write(textv)
    fileout.write(textswp)
    fileout.write(texttht)
    fileout.write(textpsi)
    fileout.write(texteps)
    fileout.write(textgam)
    fileout.write(textalp)
    fileout.write(textphi)
    fileout.write(textrho)
    fileout.write(textlft)

    if costID == 'intEh_Err' or costID == 'intErr' or \
    costID == 'ratio_Err_Eh':
        fileout.write(textxlos)
        fileout.write(textylos)
        fileout.write(textdlos)
        fileout.write(textkap)
        fileout.write(texttau)
    #end

    if costID == 'seeability':
        fileout.write(textdlos)
        fileout.write(texttheta_v)
        fileout.write(textS)
        fileout.write(textEh)
        fileout.write(textf1)
        fileout.write(textf2)
    #end

    if costID == 'seeability_constraint':
```

```
        fileout.write(textdlos)
        fileout.write(texttheta_v)
        fileout.write(textS)
    #end

    fileout.close()
```