Trajectory Optimization For Surveillance Using Small Unmanned Aerial Vehicle

A Project Report

submitted by

HARSHAL D. KAUSHIK

(AM13M025)

in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

in

APPLIED MECHANICS

FLUID MECHANICS GROUP

DEPARTMENT OF APPLIED MECHANICS

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

JUNE 2015

PROJECT CERTIFICATE

This is to certify that the project report titled **Trajectory Optimization For Surveillance Using Small Unmanned Aerial Vehicle**, submitted by **HARSHAL D. KAUSHIK (AM13M025)**, to the Indian Institute of Technology Madras, in partial fulfillment of the requirements for the award of the degree of **Master of Technology**, is a bonafide record of the work carried out by him in the Department of Applied Mechanics, IIT Madras. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. K. Arul Prakash
Project Guide
Associate Professor
Department of Applied Mechanics
IIT Madras, Chennai-600036

Dr. Ranjith Mohan
Project Co-Guide
Assistant Professor
Department of Aerospace Engineering
IIT Madras, Chennai-600036

Place: Chennai

Date: June 08, 2015

ACKNOWLEDGEMENTS

First, I want to thank my project guides, Dr. Arul Prakash and Dr. Ranjith Mohan for both the freedom they gave me to explore the research topics that interested me throughout the work, and for providing continuous support and guidance. I feel grateful for having both of them as my project adviser. Their constant encouragement and tireless support during each stage of the project has helped me to make this piece of work what it is.

I would like to thank all the other professors of Fluid Mechanics Section for sharing their valuable knowledge with us. I would also like to thank Prof. Ramasubba Reddy, Head of Applied Mechanics Department and Prof. Mahesh Panchangnula for providing the best of lab facilities to carry out the project smoothly. I am also thankful to Prof. Bhaskar Ramamurthi for giving me an opportunity to pursue my Masters in one of the finest institutes in the country.

I extend my warm regards to my lab mates Koushik, Sivasai with whom I had several interesting discussions, which refined my thought process during the course of my project. I am grateful to my batch mates Akhil, Manohar, Sivashankar, Nandulal, Rohit, Sonu to name quite a few. I want to thank my friends Nikhil, Ajinkya (π), Alok with whom I shared some joyful moments during the two years of span of my life at IIT.

Finally, I would like to thank my parents for all their love and moral support. Their belief in my abilities and the freedom they gave me to take my decisions in life has helped me to reach this stage. I dedicate this thesis to my beloved parents.

ABSTRACT

KEYWORDS: Dynamic soaring, Trajectory optimization, Stability analysis.

High endurance is the key factor for persistent surveying. For prolonged surveying, unmanned aerial vehicles (UAVs) need to be refueled continuously. Fuel consumption of UAVs can be reduced drastically, by decreasing the power requirement for propulsion. Dynamic soaring is a phenomenon which can be useful to power UAVs. Wandering albatrosses use dynamic soaring to fly thousands of kilometers with hardly flapping their wings. This phenomenon can be applied to power small UAVs in order to increase their endurance while surveying.

There is minimum requirement of wind shear for dynamic soaring to happen. In this work, trajectory optimization is performed for finding the variations in the required wind shear by changing the nature of wind profile. In the second case, trajectories are optimized for surveillance application. A set of '6 degrees of freedom point mass equations' governing the aircraft motion is used. Codes are developed in MATLAB for trajectory optimization by using an optimal control software GPOPS-II. First step is mathematical modeling, which includes generation of wind shear profile, framing the objective functions, and problem formulation for numerical optimization. Further, a code is developed to perform linear stability analysis of the optimized trajectories.

For performing dynamic soaring, minimum wind velocities are found out in the first case. Changes in the optimized trajectories are studied by varying the nature of wind shear. In the second case, trajectories are optimized for surveillance application. Variations in the optimized solutions are analyzed with the change in the view angles of camera (θ), wind shear coefficient (β_w), and the nature of wind shear profile. Also, stability of each trajectory is determined and validated.

Results from the analysis indicate that, the minimum requirement of wind velocity for performing dynamic soaring decreases as the wind shear profile goes towards the logarithmic variation. In the second case, the trajectory optimized for surveillance application spreads out with lowering the wind strength. Lastly, as the profile of wind shear changes from linear to logarithmic, optimized trajectories shrink in size.

For trajectory development, point mass model of glider is sufficient. But it lacks in giving any information about the rotation about axes. This work can be extended further by developing a high fidelity model of aircraft, which can be solved numerically to obtain the position vector, Euler angles, translational velocity, and rotational velocity of glider in the inertial space.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **i**

ABSTRACT **ii**

LIST OF TABLES **vi**

LIST OF FIGURES **viii**

ABBREVIATIONS **ix**

NOTATION **x**

1 Introduction **1**

1.1 Literature Survey 5

1.2 Outline of the Thesis 6

2 Mathematical Modelling **7**

2.1 Aircraft model for a Point Mass System 7

2.2 Wind Profile 9

2.3 Glider Selection 10

2.4 Summary 11

3 Problem Formulation and Solution Methodology **12**

3.1 Problem Formulation 12

3.1.1 Objective Function for Surveillance 14

3.2 Solution Methodology 15

3.3 Construction of an Optimal Control Problem in GPOPS-II 15

3.3.1 Syntax for Input Structure setup 15

3.3.2 Syntax of Endpoint Function 19

3.3.3 Syntax of Continuous Function 19
3.4 Validation of code 20

4 Stability Analysis **22**
4.1 Introduction to Floquet Theory 22
4.2 Perturbation Matrix 23

5 Trajectory Optimization Results **26**
5.1 Minimum Wind Shear Requirement for Dynamic Soaring 27
5.1.1 Linear Wind Shear Profile 27
5.1.2 Logarithmic Wind Shear Profile 28
5.2 Trajectory Optimization for Surveillance 31
5.3 Summary 39

6 Conclusions and Scope For Future Work **40**
6.1 Summary of Findings 40
6.2 Suggestions for Future Work 41

LIST OF TABLES

2.1 Glider's Model for point mass system simulations 11

3.1 List of trajectory optimization constraints, and whether they specify equality or inequality 13

3.2 Dimensions of the UAV considered by Gao *et al.* (2014) 21

5.1 Comparison of different wind profiles for minimum wind shear 30

5.2 Results for wind shear coefficient, $\beta_w = 0.25$ 36

5.3 Results for wind shear coefficient, $\beta_w = 0.2$ 36

5.4 Results for wind shear coefficient, $\beta_w = 0.15$ 37

LIST OF FIGURES

1.1 General illustration of dynamic soaring trajectory (Bower (2011)) 3

1.2 Shear layer formation by the edge of a sharp ridge (Bower (2011)) 4

2.1 Forces on a glider 7

2.2 Wind profile for $\beta_w = 0.2$ 10

3.1 Gray area 14

3.2 Area captured by camera 14

3.3 Variation of States and Controls with time for minimum wind shear . . . 21

5.1 Variation of States and Controls with time for linear wind shear 27

5.2 Variation of States and Controls with time for $A = 1.3$ 28

5.3 Variation of States and Controls with time for $A = 1.5$ 28

5.4 Variation of States and Controls with time for $A = 1.7$ 29

5.5 Variation of States and Controls with time for $A = 1.9$ 29

5.6 Area under surveillance for $\beta_w = 0.2, A = 1.0, \theta = 25$ deg. 31

5.7 States and controls variation for $\beta_w = 0.2, A = 1.0, \theta = 25$ deg. 31

5.8 First Eigen vector 32

5.9 Second Eigen vector 32

5.10 Third Eigen vector 32

5.11 Fourth Eigen vector 32

5.12 Fifth Eigen vector 32

5.13 Sixth Eigen vector 32

5.14 State variation with time 32

5.15 Trajectory for five cycles 32

5.16 Area under surveillance for $\beta_w = 0.2, A = 1.5, \theta = 25$ deg 33

5.17 States and controls variation for $\beta_w = 0.2, A = 1.5, \theta = 25$ deg. 33

5.18 First Eigen vector 34
5.19 Second Eigen vector 34
5.20 Third Eigen vector 34
5.21 Fourth Eigen vector 34
5.22 Fifth Eigen vector 34
5.23 Sixth Eigen vector 34
5.24 State variation with time 34
5.25 Trajectory for five cycles 34
5.26 Trajectory optimization: $\beta_w = 0.15$ 35
5.27 Trajectory optimization: $\beta_w = 0.12$ 35
5.28 Trajectory optimization: $\beta_w = 0.1$ 35
5.29 Trajectory optimization: $\beta_w = 0.08$ 35
5.30 Trajectory optimization: $A = 1.0$ 38
5.31 Trajectory optimization: $A = 1.1$ 38
5.32 Trajectory optimization: $A = 1.3$ 38
5.33 Trajectory optimization: $A = 1.5$ 38
5.34 Trajectory optimization: $A = 1.7$ 38
5.35 Trajectory optimization: $A = 1.9$ 38

ABBREVIATIONS

ODE	Ordinary Differential Equations
DS	Dynamic Soaring
DoF	Degrees of Freedom
GPOPS	Gauss Pseudospectral Optimization Software
UAV	Unmanned Aerial Vehicle

NOTATION

AR	=	Aspect Ratio (b^2/S)
b	=	wingspan of glider, m
c	=	chord, m
C_D	=	coefficient of drag
C_{D0}	=	parasite drag coefficient
C_L	=	coefficient of lift
$C_{L_{max}}$	=	maximum coefficient of lift
$C_{L_{min}}$	=	minimum coefficient of lift
D	=	drag force, N
g	=	acceleration due to gravity, m/s^2
h_{tr}	=	atmospheric boundary layer thickness, m
K	=	induced drag factor
L	=	lift force, N
m	=	mass of glider, kg
n	=	load factor (L/mg)
n_{max}	=	maximum load factor
S	=	area of wing, m^2
V_t	=	airspeed of glider, m/s
W_0	=	wind velocity at ground level, m/s
W_{max}	=	maximum wind velocity, m/s
W_z	=	wind velocity varying with altitude, m/s
x, y	=	horizontal coordinates of glider, m
z	=	altitude of glider, m
β_w	=	average slope of wind shear over (0, h_{tr}), s^{-1}
γ	=	flight path angle, deg

μ = bank angle, deg

μ_{max} = upper bank angle limit, deg

μ_{min} = lower bank angle limit, deg

ψ = heading angle, deg

ρ = air density, kg/m^3

θ = camera view angle, deg

CHAPTER 1

Introduction

Unmanned Aerial Vehicles (UAVs) are used in many applications such as search and rescue, surveillance, or the first responders in a natural disaster. UAVs are now very important for armed forces and rescuers. In particular, UAVs are useful in applications like coastal surveillance, hurricane watch, and traffic control. UAVs show great potential for many other applications like precision agriculture, package delivering, wind turbine maintenance etc.

Small UAVs can carry limited energy on board. Since having small size and low flying speeds, UAVs operate at low Reynolds number (Re: ratio of inertial forces to viscous forces). At low Re, the aerodynamic performance of a small UAV becomes much worse than its large counterpart, since viscous forces come into picture. Thus, the mission capabilities of UAVs are limited by incapability of carrying larger payloads and effect of reduced aerodynamic performance. Although improvements of battery technology can enhance their capabilities, immediate performance gains can be made by energy extraction from the atmosphere.

Albatrosses and other large seabirds extract the energy from wind gradients in the atmospheric boundary layer over the ocean by using a flight technique called dynamic soaring. By using this technique they can fly for long period of time, with hardly flapping their wings. In some recorded cases, they can even circumnavigate the globe by using the phenomenon of dynamic soaring. In this work, application of dynamic soaring for the propulsion of small UAV, specifically for surveillance application is examined.

There is plenty of energy available in the region of atmosphere, where UAVs fly. That energy can be used to increase the range and the endurance of UAVs. Energy extraction from the atmosphere is generally referred as soaring. Consumable energy is carried on board in the form of batteries or fuel which can last considerably longer if an aircraft obtains kinetic or potential energy from soaring fight.

Depending on the method of extraction of energy from atmosphere, soaring fight is typically categorized. Thermal soaring is the most common and well known category. Pockets of warm air that rise through the atmosphere generally referred as thermals. Uneven heating of the Earth's surface by solar radiation forms these thermals. Energy available in the vertically rising air is adequately used by birds and human glider pilots, which offsets the sink rate of the aircraft relative to the air. Thermal soaring over land is extensively used by raptors (Spaar and Bruderer (1996)). Pelicans, sea gulls, and frigate birds are well known to use thermal soaring near the sea shore or over water surface (Pennycuick (1982)). For the sports application, human pilots of both full size gliders and remote control gliders also make extensive use of thermals. Thermals are extended over a relatively small area, and to stay within them, circling flight is essential. There are a number of recent studies concentrated on autonomous thermal soaring to increase UAV endurance and range (Allen (2005), Allen (2006), Allen and Lin (2007), Edwards (2008))

Slope soaring is a different technique of soaring that takes advantage of vertically rising air. The vertical component of the wind offsets the sink rate of the bird or aircraft, which can be obtained easily from the wind blowing up a slope. Many bird species are observed to use this slope soaring in proximity to ridges and cliffs (Pennycuick (2008)). RC pilots also make use of this slope soaring frequently. Collectively, thermal soaring and slope soaring come under the category static soaring, as steady vertical component of the wind is used in obtaining the energy.

Energy extraction from high frequency turbulence in the atmosphere is known as gust soaring (Patel (2007)). In the gust soaring, energy is extracted from gusts by increasing lift in regions of updrafts and decreasing it in regions of downdrafts, while maintaining average lift similar to the vehicle weight. It is difficult to extract enough power from gusts, unless high levels of turbulence is present.

The final soaring technique is termed as dynamic soaring and this is the topic covered in this report. Dynamic soaring involves extraction of energy by using the wind shear present near the earth surface. In the lower atmosphere just above the ocean surface, due to the no slip condition, wind speed approaches to zero. A significant wind

gradient is established in that region, as the wind speed quickly increases with altitude. Many seabird species use this wind gradient to wander over the world's oceans. Wandering albatrosses are the most well known type of seabirds that use dynamic soaring for propulsion. They are able to travel throughout the Southern Ocean with minimal flapping of the wings. There are many documented researches of Gray-headed Albatrosses sailing all the way around the globe over the Southern Ocean in as little as 46 days (Pennycuick (2005)).

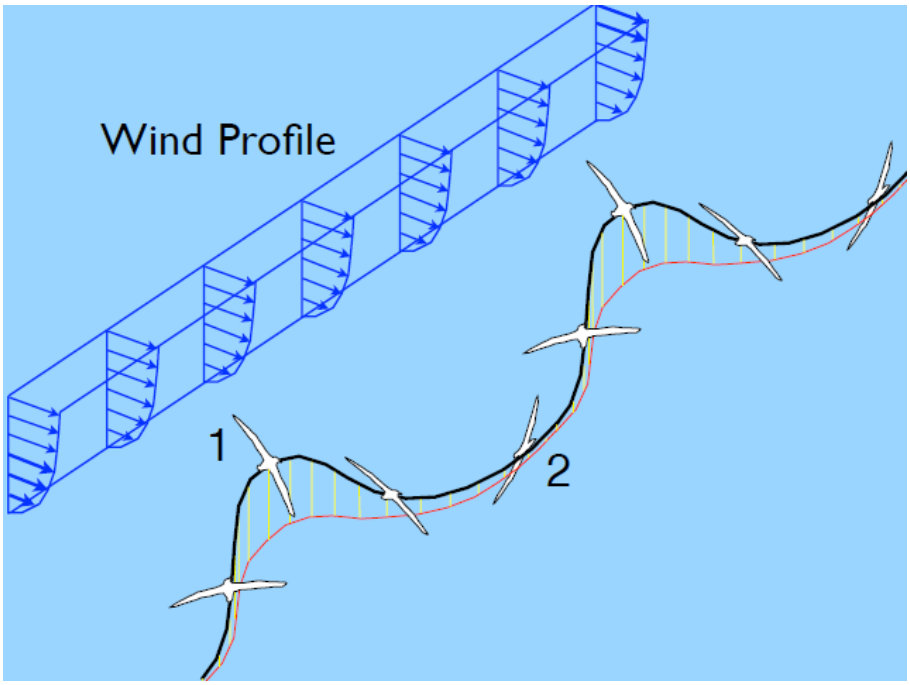

Figure 1.1: General illustration of dynamic soaring trajectory (Bower (2011))

A typical dynamic soaring trajectory is illustrated in Figure 1.1. When a component of the aerodynamic force on the vehicle is aligned with the local wind vector, the energy extraction from the atmosphere takes place. Now consider a section of the trajectory marked '1'. At this section, the lift vector has a component along the wind speed vector. At that high altitude, wind speed vector is sufficiently large in the magnitude, which indicates significant gain in energy. Along the section '2', the directions of lift vector and the wind speed vector are nearly opposite, indicating the loss of total energy due to the winds. However, this loss is smaller in amount than the gain at higher altitude, because of the wind gradient. If there is balance between the integrated energy gain from the

wind along the trajectory, and the energy loss due to drag, then a neutral energy cycle is possible without any energy expenditure for the propulsion of a bird or an aircraft.

In dynamic soaring, gain in the energy of aircraft is not possible at each and every point in the trajectory. Loss of energy at some points must be accepted to enable a greater gain at another point. Common aspect of almost all dynamic soaring trajectories is, instead of trying to extract energy at all the points along the trajectory, loss of energy at some points must be accepted in order to gain maximum energy at another points. This aspect of trading energy for the potential of extracting more energy later is an important consideration in the formulation of dynamic soaring control algorithms.

Remote controlled model airplanes use dynamic soaring to fly through the shear layers, that are formed when strong winds blow up a slope and separated by a sharp ridge at the top, as shown in Figure 1.2. Aircraft can extract significant energy from the winds, by flying orbits between the fast moving air above the shear layer, and the quiescent air below. This energy extraction can be understood using the loiter pattern of dynamic soaring. By using this phenomenon, gliders can fly at very high speeds. In fact, these are the fastest model airplanes, with the world record of 750 kph!

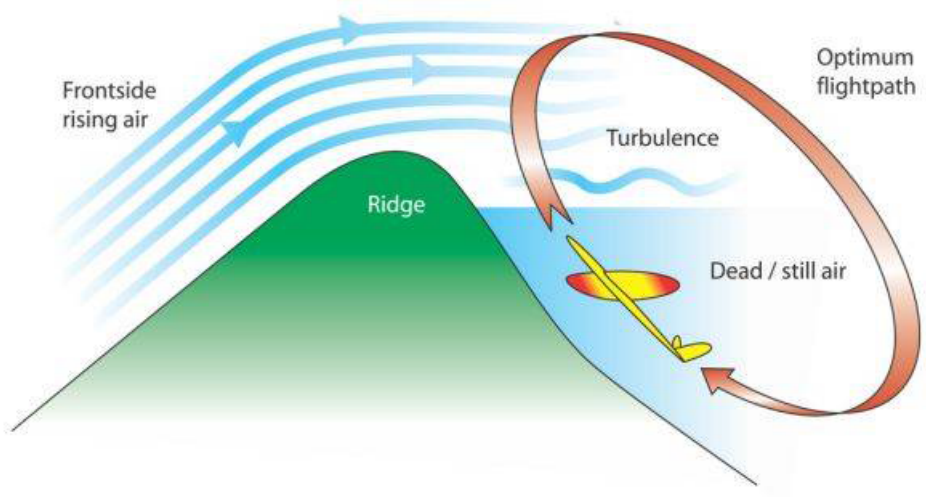

Figure 1.2: Shear layer formation by the edge of a sharp ridge (Bower (2011))

1.1 Literature Survey

Rayleigh (1883) is known to be the first person to propose that soaring is possible in a horizontal but non-uniform wind field. After him, this topic became continuous attraction for many scientists. In general, this research process can be divided into three periods. First period is, Rayleigh's time (1883) to the early 2000s. In this early stage, most researchers investigated on how birds such as albatrosses, jackdaw, falcon are able to cover huge distances with hardly flapping their wings. Many observations were taken and experiments were conducted on these birds during this period. Specifically, Turcker and Parrott (1970), Weimerskirch *et al.* (2000), Mikael and Anders (2001) worked on the subject. Gradually the theory of dynamic soaring was evolved in this era.

The second period can be stated as, early 2000s to 2010s. In this span of time, computational algorithms on optimal control were rapidly developed. Plenty of numerical software were emerged, which further used to solve the dynamic soaring problem aiming at different objectives. In particular, Sachs (2004) found the trajectory for minimum wind shear requirement to happen dynamic soaring. His work further extended by Zhao (2004) to solve the problem by using collocation method in combination with software NPSOL. Zhao (2004) focused on finding trajectories assisted with dynamic soaring. Other examples can be seen in Akhtar *et al.* (2009), Deittert *et al.* (2009), and Gordon (2006).

In the recent period after 2010, significant work has been done in making the flights of UAVs autonomous and close to perpetual. Langelaan and Roy (2009) provided the strategy for improved persistence by harvesting the energy from atmosphere. Some innovative works have been done, like a complete guidance and control strategy was designed for dynamic soaring by Lawrance and Sukkarieh (2009). Lawrance and Sukkarieh (2011), and Langelaan *et al.* (2012) have given the methods for estimating the wind field for autonomous dynamic soaring. However, in the sequence of development, optimization of trajectories for the surveillance and their analysis for stability were undetermined. Current work is devoted to accomplish the optimization of trajectories for surveillance purpose. First version of GPOPS first version Rao *et al.* (2010) but that software was

limited in total number of variables. So, GPOPS-II (Rao and Patterson (2013)) is the optimal control software used in this work.

1.2 Outline of the Thesis

- Chapter-2 starts with stating the governing equations of motion for a point mass aircraft model, flying using the loiter pattern of dynamic soaring. Equation for different profiles of wind shear is given. At the end, glider is are selected for trajectory optimization.

- Chapter-3 deals with the methodology behind the construction of codes for optimal control problem. General procedure to write a code in GPOPS-II software is discussed. First set of codes is developed for finding the minimum required wind shear. Second set of codes is developed for the surveillance application. In developing these codes, point mass model of glider is considered.

- Chapter-4 introduces the Floquet theory. Algorithm behind the construction of codes for stability analysis of optimized trajectory is given in this chapter. Perturbation matrix is obtained from the governing equations of dynamic soaring.

- In chapter-5 the results obtained from optimization are presented. Initially, trajectories with different natures of wind profiles are analysed. Results for surveillance application are shown. Results with varying different parameters such as view angle of camera, and different wind shear coefficients are presented.

- Chapter-6 deals with the conclusions from the obtained results. In the next part, scope for future work is explained.

CHAPTER 2

Mathematical Modelling

First part of this chapter deals with the equations governing the phenomenon dynamic soaring. This set of governing equations is used in the formulation of dynamic soaring problem for several applications. In the next part, a general equation is given to capture the variation of wind with altitude just by varying a single parameter. In the last part, glider selection procedure is discussed.

2.1 Aircraft model for a Point Mass System

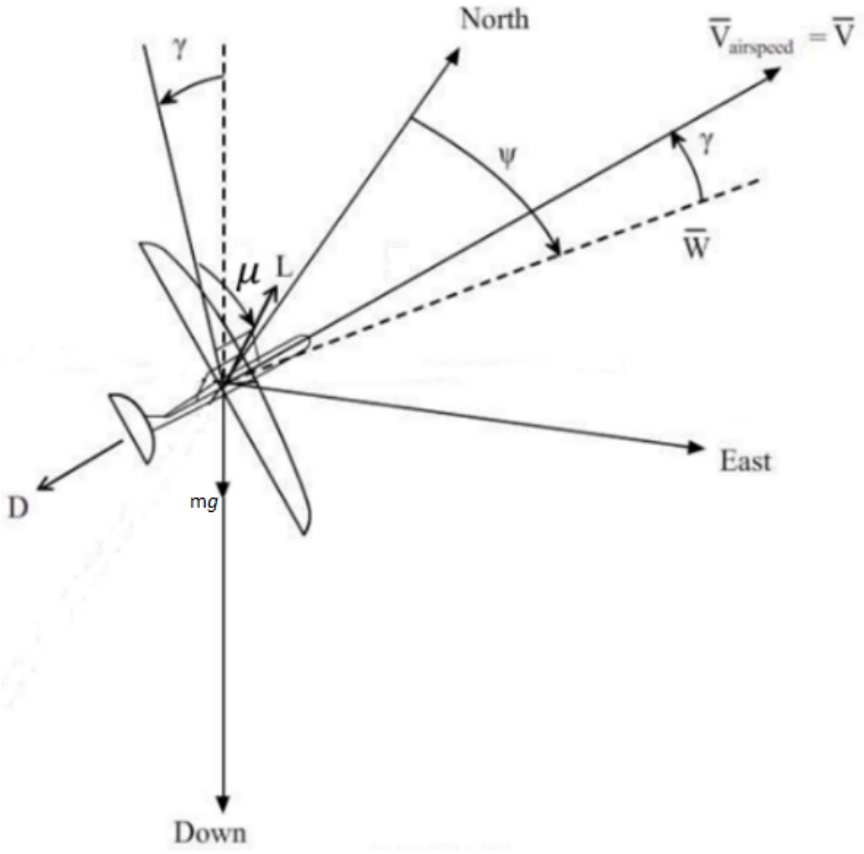

Figure 2.1: Forces on a glider

A point mass model is introduced here. Inertial North, East and Down (NED) reference frame is selected, where the x, y, and z axes are in the direction of North, East, and Down respectively. Wind field is assumed to be varying with space, and constant with time. To define the orientation of the aerodynamic forces on the aircraft, a set of three angular rotations is defined. First, the air relative heading angle (ψ) is defined as zero to the North and increases towards the East (clockwise rotation when viewed from top). Next, the air relative flight path angle (γ) is defined to be positive for nose up condition. The last rotation angle is the bank angle (μ). This is defined to be positive when the aircraft is rolling clockwise when seen from the backside.

Line of action of forces and the angles are as shown in the Figure 2.1. Thrust force is neglected in the governing equations, since the glider is not generating any propulsive force. It is completely powered by the phenomenon dynamic soaring.

The flight dynamics model includes six states: horizontal positions [x(t), y(t)], altitude z(t), airspeed V(t), flight path angle γ(t), and azimuth angle ψ(t). Control parameters are coefficient of lift C_L(t), and bank angle μ(t). Lift and Drag forces on the aircraft structure are expressed as,

$$L = \frac{1}{2}\rho S C_l V_t^2 \tag{2.1}$$

$$D = \frac{1}{2}\rho S C_D V_t^2 \tag{2.2}$$

Relation between the coefficient of lift and the coefficient of drag is,

$$C_D = C_{D0} + KC_L^2 \tag{2.3}$$

The governing equations are as following:

$$m\dot{V}_t = -D - mg\sin(\gamma) - m\dot{W}_z\cos(\gamma)\sin(\psi) \tag{2.4}$$

$$mV_t\cos(\gamma)\,\dot{\psi} = L\sin(\mu) - m\dot{W}_z\cos(\psi) \tag{2.5}$$

$$mV_t\,\dot{\gamma} = L\cos(\mu) - mg\cos(\gamma) + m\dot{W}_z\sin(\gamma)\sin(\psi) \tag{2.6}$$

$$\dot{z} = V_t\sin(\gamma) \tag{2.7}$$

$$\dot{x} = V_t\,cos(\gamma)sin(\psi) + W_z \tag{2.8}$$

$$\dot{y} = V_t\,cos(\gamma)cos(\psi) \tag{2.9}$$

2.2 Wind Profile

For simplicity, curvature effect of Earth is neglected. UAV is assumed to be flying over a flat surface. Variation of air density with altitude is neglected. Wind is assumed to be blowing in the North direction. Temporal variation of wind is neglected. To capture the wind variation with altitude, following equation is used:

$$W_z = \beta_w\left[A\,z + \frac{1-A}{h_{tr}}\right] + W_0 \tag{2.10}$$

where 'β_w' is the average slope of wind shear (0, h_{tr}). Boundary layer thickness over open sea 'h_{tr}' is taken as 213 m (Wai-Fah and Lui (1997)). Wind velocity at the surface of water is taken as zero, W_0 = 0.

$$\beta_w = \frac{W_{max}}{h_{tr}} \tag{2.11}$$

Depending on values of 'A', different wind profiles can be obtained as shown in Figure 2.2. It is required to have $0<A<2$ in order to ensure the wind component is between $(0,W_{max})$. For $0<A<1$, wind profile is similar to the exponential curve and for $1<A<2$ it

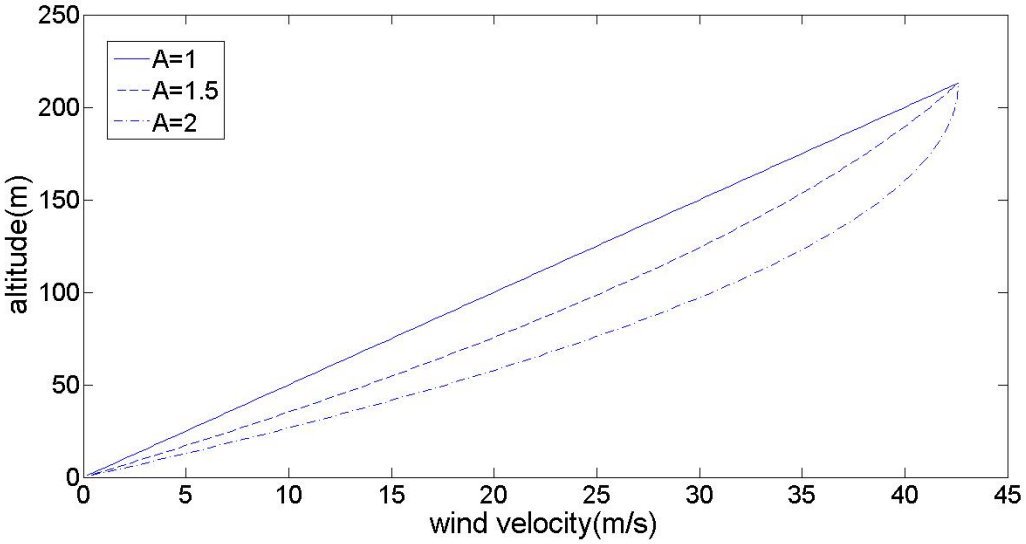

Figure 2.2: Wind profile for $\beta_w = 0.2$

looks like logarithmic profile.

$$\dot{W}_z = \frac{dW_z}{dz} \times \frac{dz}{dt} = \beta_w V \sin(\gamma)\left[A + \frac{(1-A)2z}{h_{tr}}\right] \tag{2.12}$$

Substituting expressions of 'W_z' and '$\dot{W}_z$' from Equation 2.10, and 2.12 into Equations 2.4 – 2.9 for the wind profile, governing equations are obtained.

2.3 Glider Selection

The glider used in the simulation is similar in shape and size as albatross. Glider is equipped with a camera, which always remains horizontal irrespective of rolling or pitching motions of aircraft. This glider is used in the simulations of point mass system.

Parameter	Value	Unit	Explanation
AR	16		Aspect Ratio
b	4	m	Wing-Span
C_{D0}	0.00873		Parasite drag coefficient
$C_{L_{max}}$	1.5		Maximum coefficient of lift
$C_{L_{min}}$	−1		Minimum coefficient of lift
K	0.045		Induced drag factor
m	10	kg	Mass of glider
S	1	m^2	Wing Area

Table 2.1: Glider's Model for point mass system simulations

2.4 Summary

In this chapter, initially the methodology behind the mathematical modeling of dynamic soaring problem is explained. There are several ways through which dynamic soaring can be explained, but in this work, methodology given by Zhao (2004) is considered and used for getting the governing equations. These equations can be solved numerically by optimal control software GPOPS-II, this is the topic of next chapter. Also an equation is derived to obtain the wind variation. Finally, the dimensions of a glider are resolved to perform dynamic soaring optimizations.

CHAPTER 3

Problem Formulation and Solution Methodology

First part of this chapter deals with the formulation of an optimal control problem by using the governing equations, stated in Chapter-2. In the later part, a numerical method to solve the optimal control problem by using the software GPOPS-II is discussed. Next part of the chapter deals with the validation of the code written in GPOPS-II by comparing it with the results obtained previously.

3.1 Problem Formulation

Trajectory optimization problem is formed with the goal of finding the state and control time histories, which will minimize the selected cost function while satisfying the 6 DoF point mass equations governing the motion of an aircraft. In general, a typical optimal control problem can be defined as following. 'J' is the objective function, $\overline{x}$ is the set of state variation with time histories, $\overline{u}$ is a set of controls varying with time, 'g' is a set of inequality constraints, and 'h' is a set of equality constraints.

$$\text{Minimize: } J(\overline{x}, \overline{u})$$
$$\text{with respect to, } \overline{x}, \overline{u}$$
$$\text{such that, } g(\overline{x}, \overline{u}) \leq 0$$
$$h(\overline{x}, \overline{u}) = 0.$$

A common constraint for dynamic soaring trajectories is the requirement of periodicity, which means the optimal trajectories are periodic in some or all of the state variables. For the loiter pattern of dynamic soaring, periodic boundary condition should be imposed on all the state variables.

Equations governing the phenomenon dynamic soaring (2.4 – 2.9) are given in Chapter-2. In that the state variables include the x, y, and z positions of glider, airspeed (V), flight path angle (γ), and heading angle (ψ). The control variables include the coefficient of lift (C_l), and bank angle (μ).

There are many interesting objectives that could be investigated, but we have restricted ourselves to the following two objectives in this work:

1. Minimum wind shear requirement for dynamic soaring to accomplish
2. Surveillance application

By knowing the minimum wind shear requirement for dynamic soaring, it can be easily predicted at any place, whether it is possible to perform dynamic soaring, in the present wind condition.

Other than the above two objectives, previous researchers have found the states and controls for several other objectives such as maximum distance traveled, minimum cycle time, altitude gain (Zhao (2004)), minimum average power or thrust.

In addition to satisfying the governing equations of motion, solution has to satisfy some constraints of equalities and inequalities. List of trajectory optimization constraints, state variables, and whether they are satisfying equality or non-equality is given in the Table 3.1.

Constraint	State Variables	= or ≤	Explanation
Equations of motion	All	=	equation 2.4 – 2.9
State bounds	x, y, z, V, γ, ψ	≤	
Control bounds	C_l, μ	≤	
Load Factor	V, C_l	≤	n = $\frac{\frac{1}{2}\rho S C_l V^2}{mg}$
Initial condition	All	=	
Periodicity condition	All	=	

Table 3.1: List of trajectory optimization constraints, and whether they specify equality or inequality

There are mainly three patterns of dynamic soaring flights: basic, traveling, and loiter pattern. Different patterns are chosen for various missions. Each pattern can be

achieved by varying the constraints on the states. Here, Loiter pattern is used for the surveillance application.

3.1.1 Objective Function for Surveillance

While surveying, glider traces a periodic orbit (which satisfies the phenomenon dynamic soaring), and uses a camera to capture the image on the ground. In the process some area on ground may not be captured by the camera. This area is termed here as *gray* area, shown in Figure 3.1. Our objective is to minimize this *gray* area. In order to achieve this, appropriate objective function is defined. As stated earlier, the camera mounted on a glider always remains horizontal, and it captures images in the plane parallel to the Earth surface, irrespective of pitching or rolling motions of an aircraft. Assuming the images captured are circular in cross section. In Figure 3.2, 'z' is the altitude of the glider and 'θ' is the view angle of camera.

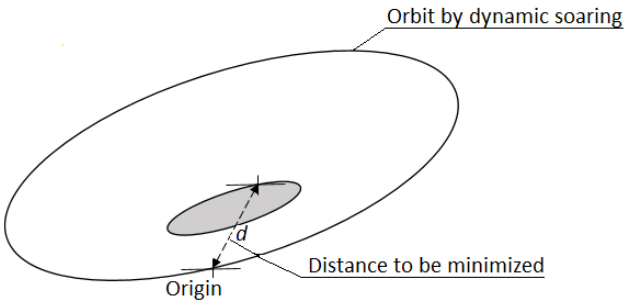

Figure 3.1: Gray area

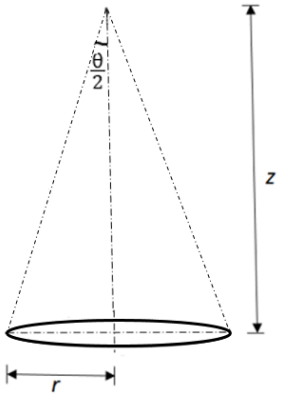

Figure 3.2: Area captured by camera

Radius (r) of the image captured in Figure 3.2 is given as,

$$r = z \times \tan(\theta/2)$$

To minimize the gray area, distance 'd' in the Figure 3.1 has to be minimized. Objective function for this purpose can be formulated as:

$$d = \sqrt{x^2 + y^2} - z \times \tan(\theta/2) \tag{3.1}$$

3.2 Solution Methodology

Equations governing the phenomenon dynamic soaring (Equation 2.4 – 2.9) are used in formulating the problem. It includes six states and two control parameters. Appropriate constraints on the state variables and control parameters are put as explained in Table 3.1. Objective function is given by Equation 3.1 for trajectory optimization. GPOPS-II (Rao and Patterson (2013)) is a software used to optimize the optimal control problem. Appropriate initial guess is provided to converge the solution fast. A stepwise procedure for construction of problem in GPOPS-II is given in the following section.

3.3 Construction of an Optimal Control Problem in GPOPS-II

In this section details are provided for constructing an optimal control problem in GPOPS-II. Key MATLAB programming element used in formulating the code in GPOPS-II are structure and arrays of structures. A call to GPOPS-II can be given by the syntax,

output = gpops2(input)

Where, input is the structure defined by the user which includes all the information about optimal control problems. Output is the structure obtained by solving an optimal control problem.

In following sections, stepwise procedure for building input structure is given.

3.3.1 Syntax for Input Structure setup

3.3.1.1 Auxiliary Data for Problem

This is a structure in which auxiliary data is provided. This data can be used by different functions in the problem. Any need of global variable in the problem can be eliminated

by providing *auxdata*. Syntax for giving auxiliary data input is *setup.auxdata*.

3.3.1.2 Syntax for bounds Structure

Bound is an array of structures of length K (where K is the number of phases. In this problem $K = 1$) which specifies the bounds on the time, state, control, path constraints, and integrals in each phase of the problem. There are 'H' number of variables in the state, 'I' number of control parameters in the problem, and 'J' number of integrals in an optimal control problem.

- *bounds.phase.initialtime.lower* and *bounds.phase.initialtime.upper*: These are the scalars which contain information about the lower and upper bounds on the initial time in each phase.
- *bounds.phase.finaltime.lower* and *bounds.phase.finaltime.upper*: These are the scalars which contain information about the lower and upper bounds on the final time in each phase.
- *bounds.phase.initialstate.lower* and *bounds.phase.initialstate.upper*: These are the row vectors (of length = H) which contain the lower and upper bounds on the initial state in a phase.
- *bounds.phase.state.lower* and *bounds.phase.state.upper*: These are the row vectors (of length = H) which contains lower and upper bounds on the states during the phase.
- *bounds.phase.finalstate.lower* and *bounds.phase.finalstate.upper*: These are the row vectors (of length = H) which contains lower and upper bounds on the final states during the phase.
- *bounds.phase.control.lower* and *bounds.phase.control.upper*: Row vectors (of length = I) that contain the lower and upper bounds on the controls during the phase.
- *bounds.phase.path.lower* and *bounds.phase.path.upper*: Row vectors that contain the lower and upper bounds on the path during the phase.

- *bounds.phase.integral.lower* and *bounds.phase.integral.upper*: Row vectors (of length = J) that contain the lower and upper bounds on the integrals during the phase.
- *bounds.phase.eventgroup.lower* and *bounds.phase.eventgroup.upper*: Row vectors that contain the lower and upper bounds on the groups of event constraints. This is of length 'G', where 'G' is number of event groups in the problem.

3.3.1.3 Initial Guess

This is a structure that contains the guesses of time, states, controls, integrals in the problem. Assuming R is the number of values used in the guess for time, state, and control in each state.

- *setup.guess.phase.time*: This is a column vector of length R.
- *setup.guess.phase.state*: This is a matrix of size $R \times H$.
- *setup.guess.phase.control*: This is a matrix for initial guess of size $R \times I$.
- *setup.guess.phase.integral*: This is a row vector of length J.

The following set of initial conditions is used for the current problem, which are finalized after huge trial and error procedures.

- t_guess = linspace(0, 24, 100)'
- x_guess = 500 × cos(2 × π × t_guess / 24) – 500
- y_guess = 300 × sin(2 × π × t_guess / 24)
- z_guess = – 400 × cos(2 × π × t_guess / 24) + 400
- v_guess = 0.8 × v_0 × [1.5 + cos(2 × π × t_guess / 24)]
- γ_guess =π / 6 × sin(2 × π × t_guess / 24)
- ψ_guess = – 1 – t_guess / 4

- C_L_guess = C_{L_0} × ones(100,1) / 3
- μ_guess = – ones(100,1)

3.3.1.4 Configuring final set up of the problem

- *mesh.maxiteration*: This tells the maximum number of allowed mesh iterations. Default value is 10.
- *mesh.method*: This specifies the particular mesh refinement method to be used. There are two options *RPM-Differentiation* and *RPM-Integration*. In this case *RPM-Differentiation* is used.
- *name*: A string should be provided without any blank space.
- *function*: This is a structure containing the name of the continuous function and endpoint function.
- *setup.nlp.solver*: This structure specifies which non linear programming (*nlp*) solver is used. In this case *ipopt* is the *nlp* solver used.
- *setup.nlp.ipoptoptions.linear_solver*: For this there are two options *mumps* and *ma57*, *mumps* is used in the current case.
- *setup.nlp.ipoptoptions.tolerance*: This is having a default value 10^{-6}. In this case 10^{-7} is considered.
- *setup.nlp.ipoptoptions.maxiterations*: This has default value 2000. In current problem, we have used 2×10^{8}.
- *setup.derivatives.supplier*: There are four possible options for this, *sparseFD*, *sparseCD*, *sparseBD*, and adigator. We have used *sparseCD*.
- *setup.derivatives.derivativelevel*: For this two options *first*, and *second*. We have used *second*.

- *setup.method*: This defines the version of collocation to be used while solving the problem. There are two options *RPM-Differentiation*, and *RPM-Integration*. *RPM-Differentiation* is used in the current problem.

3.3.2 Syntax of Endpoint Function

- *input.phase.initialstate*: This is a row vector of length H containing the initial state in the phase.
- *input.phase.finalstate*: This is a row vector of length H containing the final state in the phase.
- *input.phase.integral*: A row vector of length J that contains the integral in phase.
- *output.objective*: A scalar that contains the result of computing the objective function.
- *output.eventgroup*: This is an array of structure of length l. In *output.eventgroup*, l^{th} element is the row vector of length H, which contains the result of evaluating l^{th} group of event constraints at the value given in the call to the function *input.functions.endpoint*.

3.3.3 Syntax of Continuous Function

- *input.phase.time*: A column vector of length O, where O is the number of collocation points.
- *input.phase.state*: A matrix of size $O \times H$, where H is number of states in a phase.
- *input.phase.control*: A matrix of size $O \times I$, where I is number of controls in a phase.
- *phaseout.dynamics*: A matrix of size $O \times H$, where O is a number of collocation points, and H is a number of controls in a phase.
- *phaseout.path*: A matrix of size $O \times D$, where D is a number of path constraints in a phase.

- *phaseout.integrand*: A matrix of size $O \times J$, where J is a number of integrals in a phase.

There is similarity between Pseudospectral methods and Collocation method. In these two methods, equations of motion are discretized in time. However, to minimize the error in the constraints equations, the control points are intelligently chosen. In addition, more recently, knot points are implemented which allow the switching conditions in the optimal control problem to be captured (Gong *et al.* (2008)). The second version of General Pseudospectral Optimal Control Software (GPOPS) (Rao and Patterson (2013)) is used to solve the optimal control problem. Both Gauss pseudospectral method and the Radu pseudospectral method are implemented in GPOPS-II software.

There are different cameras available with different ranges of view angles. To capture the maximum area, larger value of view angle is used, similarly smaller view angles are used to concentrate at a point. In this work, attention is given on surveying the large portion rather than at a point. Accordingly, the larger values of view angle is used.

3.4 Validation of code

Gao *et al.* (2014) generated the results for the minimum wind speed for the model UAV by using the first version of GPOPS. The dimension of the glider used by him are shown in Table 3.2. The results obtained by him are used for comparison. For this problem coefficient of lift is bounded between –0.2 and 1.5, limits on the bank angle is fixed to –80 to +80 degrees, and the maximum limit on the load factor (n) is 5.

All of the following results are generated on a Lenovo Z580 laptop with a 2.4 GHz Intel i5 processor and 4 GB of RAM. Solutions for the collocation method are performed using GPOPS-II. Codes are developed in MATLAB for the specified objective and the respective constraint. The pseudospectral method is used to generate the results using the GPOPS library running through MATLAB.

The minimum value of wind shear required to perform dynamic soaring is 0.0629 m/s,

Parameter	Value	Unit	Explanation
m	81.7	kg	Mass of glider
S	4.19	m^2	Wing Area
C_{D0}	0.00873		Parasite drag coefficient
n_{max}	5		Maximum load factor
$C_{l_{max}}$	1.0		Maximum coefficient of lift
K	0.045		Induced drag factor

Table 3.2: Dimensions of the UAV considered by Gao *et al.* (2014)

which agrees with the results obtained by Gao *et al.* (2014) using GPOPS. Time taken to solve the code by GPOPS-II is 4.4151 seconds.

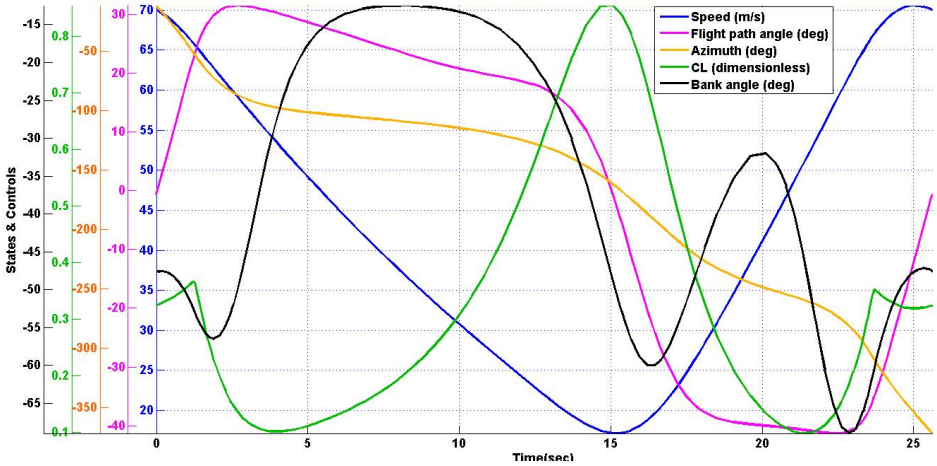

Figure 3.3: Variation of States and Controls with time for minimum wind shear

The variation of states and controls is as shown in Figure 3.3. This Figure is individually alike with the one obtained by Gao *et al.* (2014).

CHAPTER 4

Stability Analysis

In this chapter a methodology is explained for checking the stability of the optimized trajectories. Initially a brief introduction about a Floquet theory is given. In the later part, method behind the stability calculation of optimized trajectory is explained. A perturbation matrix is displayed, which is obtained by perturbation of the equations governing dynamic soaring (Equation 2.4 – 2.9).

4.1 Introduction to Floquet Theory

Floquet theory is used to find the solutions of periodic linear differential equations of the form,

$$\dot{X} = M(\mathrm{t})\, X \tag{4.1}$$

where X is n-dimensional column vector and $M(\mathrm{t})$ is a square matrix with order n having time period T. Although parameters of $M(\mathrm{t})$ vary periodically, the solutions of Equation 4.1 are typically not periodic. Despite of its linearity, closed form solutions of Equation 4.1 are very difficult to find. General solution of Equation 4.1 can be given in the form:

$$X(\mathrm{t}) = \sum_{i}^{n} c_i \; e^{\mu_i \mathrm{t}} \; \mathrm{P}_i(\mathrm{t}) \tag{4.2}$$

Where c_i are the constants, which depend upon the initial conditions. $\mathrm{P}_i(\mathrm{t})$ are the vector-valued functions having period T, and μ_i are complex numbers, which are also called characteristic or Floquet exponents. The relationship between characteristic or

Floquet multipliers and the Floquet exponents is $\rho_i = e^{\mu_i T}$. From Equation 4.2 it can be seen that, the solution to Equation 4.1 is the summation of n periodic functions multiplied by exponentially growing or shrinking terms. The longterm behavior of the complete system can be predicted by the Floquet exponents.

If all the Floquet exponents have negative real parts, the system is stable. The real parts of all Floquet multipliers lie between –1 and 1. If there is any Floquet exponent having a positive real part, the system is unstable. Thus, Floquet exponents can be treated in the same way as Eigenvalues are in models with constant coefficients in continuous / discrete time. Floquet exponent represent the average growth rate of various perturbations over a cycle.

4.2 Perturbation Matrix

Floquet theory (Chicone (2006)) is used to perform linear stability analysis of a set periodic ordinary differential equations of the form:

$$\dot{X} = M(\mathrm{t})\, X$$

M(t) is a periodic continuous matrix-valued function, which is obtained by first order perturbations of the governing Equations 2.4 – 2.9. Perturbation is introduced in the state variables of the equations by keeping the control variables unperturbed.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{V}_t \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} = M(t) \begin{bmatrix} x \\ y \\ z \\ V_t \\ \gamma \\ \psi \end{bmatrix} \tag{4.3}$$

$$M(t) = \begin{bmatrix} 0 & 0 & E_{13} & E_{14} & E_{15} & E_{16} \\ 0 & 0 & 0 & E_{24} & E_{25} & E_{26} \\ 0 & 0 & 0 & E_{34} & E_{35} & 0 \\ 0 & 0 & E_{43} & E_{44} & E_{45} & E_{46} \\ 0 & 0 & E_{53} & E_{54} & E_{55} & E_{56} \\ 0 & 0 & E_{63} & E_{64} & E_{65} & E_{66} \end{bmatrix}$$

Elements of the matrix M(t) are displayed as follows:

$$\mathrm{H} = \frac{\rho S}{2m}$$

$E_{13} = \beta_w(A+\frac{2}{213}(1-A)Z(\mathrm{t}))$

$E_{14} = \cos(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))$

$E_{15} = (-1)\sin(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))V_t(\mathrm{t})$

$E_{16} = \cos(\gamma(\mathrm{t}))\cos(\psi(\mathrm{t}))V_t(\mathrm{t})$

$E_{24} = \cos(\gamma(\mathrm{t}))\cos(\psi(\mathrm{t}))$

$E_{25} = (-1)\cos(\psi(\mathrm{t}))\sin(\gamma(\mathrm{t}))V_t(\mathrm{t})$

$E_{26} = (-1)\cos(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))V_t(\mathrm{t})$

$E_{34} = \sin(\gamma(\mathrm{t}))$

$E_{35} = \cos(\gamma(\mathrm{t}))V_t(\mathrm{t})$

$E_{43} = \frac{-2}{213}\ (1-A)\beta_w\cos(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))V_t(\mathrm{t})\sin(\gamma(\mathrm{t}))$

$E_{44} = -2(\mathrm{H})(C_D)V_t(\mathrm{t}) - \beta_w\sin(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))\cos(\gamma(\mathrm{t}))(A + \frac{2}{213}(1-A)z(\mathrm{t}))$

$E_{45} = -g\cos(\gamma(\mathrm{t}))+\beta_w\sin(\psi(\mathrm{t}))(A+\frac{2}{213}(1-A)z(\mathrm{t}))V_t(\mathrm{t})(\sin^2(\gamma(\mathrm{t})) - \cos^2(\gamma(\mathrm{t})))$

$E_{46} = -\beta_w V_t(\mathrm{t})\sin(\gamma(\mathrm{t}))\cos(\gamma(\mathrm{t}))\cos(\psi(\mathrm{t}))(A + \frac{2}{213}(1-A)z(\mathrm{t}))$

$E_{53} = \frac{2}{213}(1-A)\beta_w\sin(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))\sin(\gamma(\mathrm{t}))$

$E_{54} = C_L(\mathrm{t})(\mathrm{H})\cos(\mu(\mathrm{t}))+\frac{g\cos(\gamma(\mathrm{t}))}{(V_t(\mathrm{t}))^2}$

$E_{55} = \frac{g\sin(\gamma(\mathrm{t}))}{V_t(\mathrm{t})} + \cos(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))\sin(\gamma(\mathrm{t}))2\beta_w(A + \frac{2}{213}(1-A)z(\mathrm{t}))$

$E_{56} = \cos(\psi(\mathrm{t}))\sin^2(\gamma(\mathrm{t}))\beta_w(A + \frac{2}{213}(1-A)z(\mathrm{t}))$

$E_{63} = \frac{-2}{213}(1-A)\beta_w\cos(\psi(\mathrm{t}))\tan(\gamma(\mathrm{t}))$

$E_{64} = C_L(\mathrm{t})(\mathrm{H})\sec(\gamma(\mathrm{t}))\sin(\mu(\mathrm{t}))$

$E_{65} = C_L(\mathrm{t})(\mathrm{H})\sin(\mu(\mathrm{t}))V_t(\mathrm{t})\tan(\gamma(\mathrm{t}))\sec(\gamma(\mathrm{t})) - \beta_w\cos(\psi(\mathrm{t}))(A + \frac{2}{213}(1-A)z(\mathrm{t}))(\sec(\gamma(\mathrm{t})))^2$

$E_{66} = \beta_w\tan(\gamma(\mathrm{t}))\sin(\psi(\mathrm{t}))(A + \frac{2}{213}(1-A)z(\mathrm{t}))$

This matrix is solved over a time period of the system. Further, the Floquet exponents are obtained. For this a code in MATLAB is written. Stability of the system can be determined by the Floquet exponents. If the real part of exponents is negative, the system is stable, otherwise it is unstable.

CHAPTER 5

Trajectory Optimization Results

From the equations governing dynamic soaring (Equations 2.4 – 2.9) the primary characteristics of unpowered flight by using a phenomenon dynamic soaring have become clear. In general, while performing motion by using dynamic soaring, there should be turning of aircraft or bird from upwind to downwind, in the portion of maximum wind speed, whereas at the lowest wind speed region, it should turn from downwind to upwind. Additionally, there should be an increasing headwind while climbing, and while diving it should go through a decreasing tailwind. Along with satisfying the constraints imposed for a particular problem, the optimal trajectories are expected to have these general characteristics.

In this chapter, trajectory optimization results for dynamic soaring problems are presented. First, trajectories are found for minimum required wind speed of a flight through a linear boundary layer. Then, for logarithmic wind shear, trajectories are found out for minimum wind speed calculation. In the following section, flight trajectories for the reduction of gray area in the surveillance application are given, considering both linear and logarithmic wind shear profiles. All the results are obtained by using a MATLAB based optimal control software GPOPS-II.

5.1 Minimum Wind Shear Requirement for Dynamic Soaring

5.1.1 Linear Wind Shear Profile

The dimensions of the glider model used for simulation are given in Table 2.1. Linear wind shear is obtained by the equation 2.12. Various wind shear profiles can be obtained just by varying the value of a parameter 'A'. Those are shown in the Figure 2.2. For linear wind shear profile (A = 1.0) requirement of minimum wind shear coefficient (β_w) is 0.0879. Minimum required wind speed is 10.81 m/s. For this trajectory optimization, variation of states and controls is shown in Figure 5.1. Stability of the trajectory is checked by Floquet theory. The optimized trajectory for linear wind shear is stable.

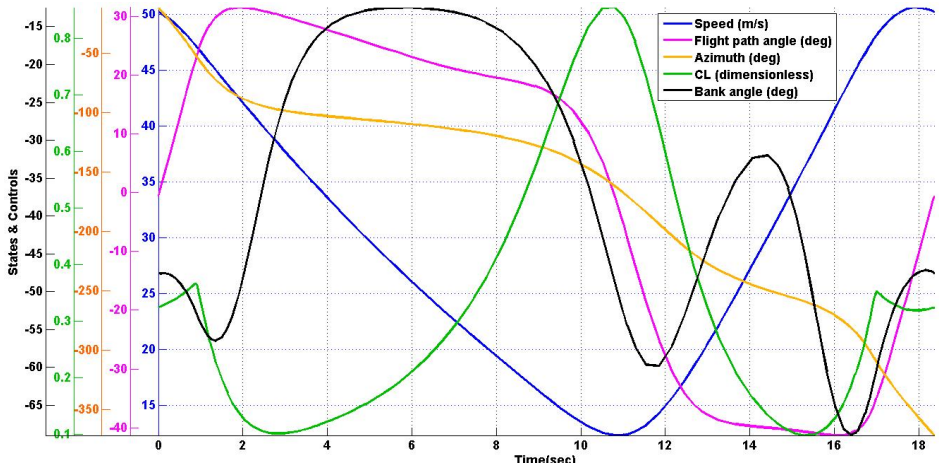

Figure 5.1: Variation of States and Controls with time for linear wind shear

5.1.2 Logarithmic Wind Shear Profile

A wind profile similar to Logarithmic profile can be obtained by the substitution, $1 < A < 2$ in the equation 2.12. Trajectories for minimum requirement of wind shear are found out for different values of a parameter 'A'. Variation of states and controls for those trajectories is shown in the Figures 5.2 – 5.5. In Table 5.1 concisely all the information is fitted.

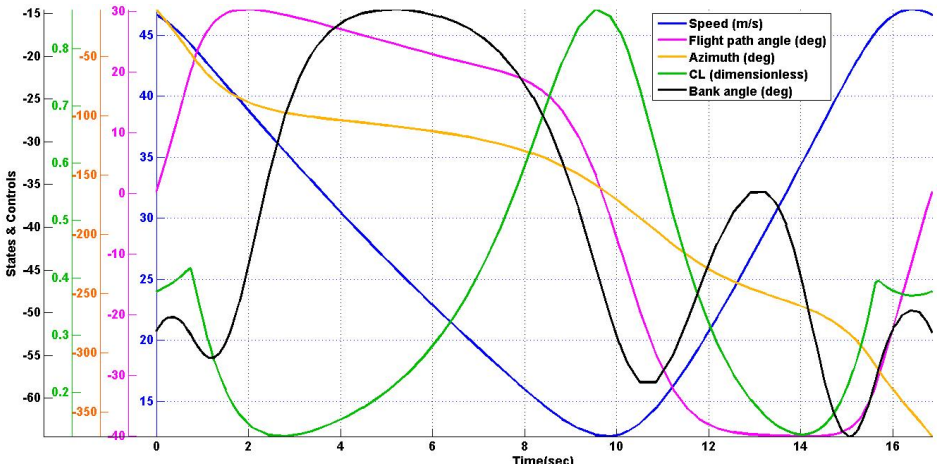

Figure 5.2: Variation of States and Controls with time for A = 1.3

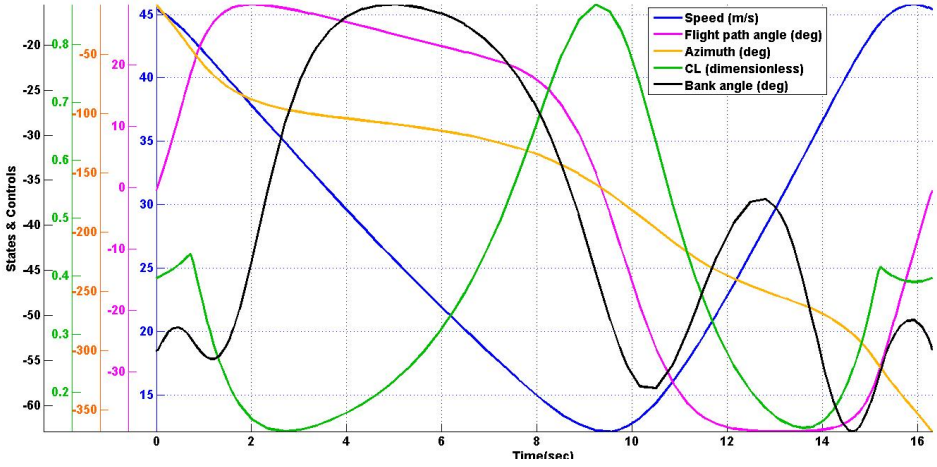

Figure 5.3: Variation of States and Controls with time for A = 1.5

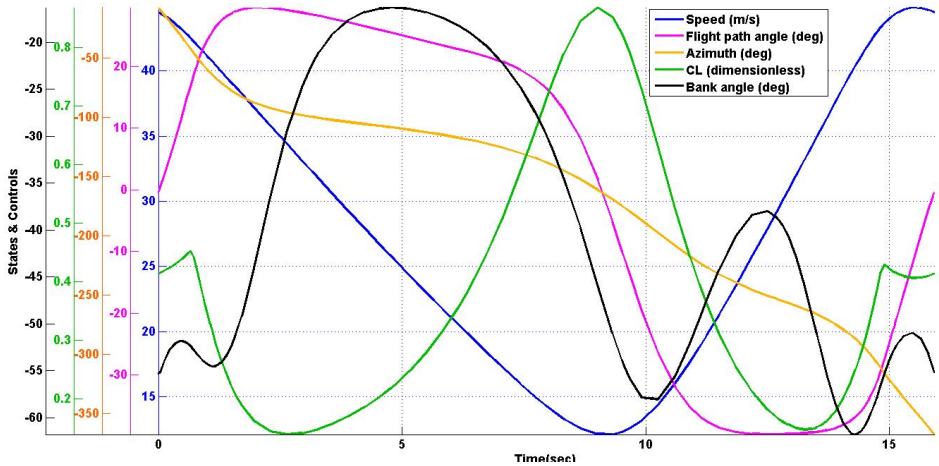

Figure 5.4: Variation of States and Controls with time for A = 1.7

.

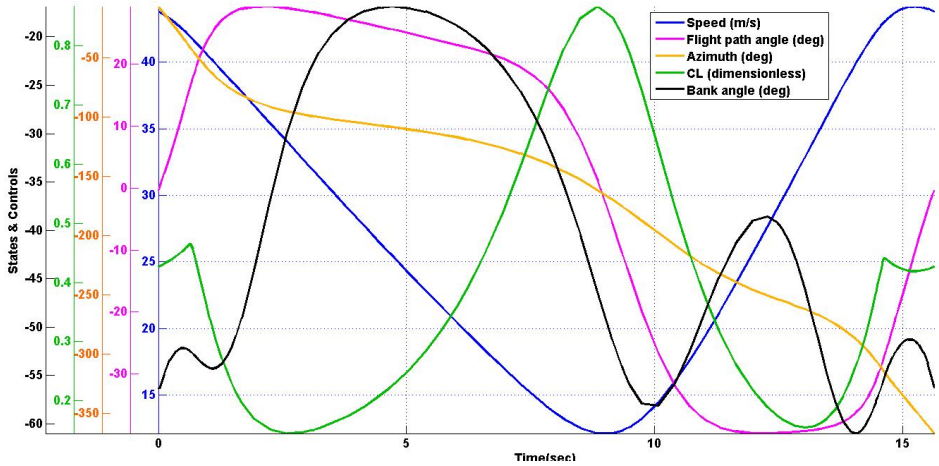

Figure 5.5: Variation of States and Controls with time for A = 1.9

Information about the stability, minimum requirement of wind speed, run time are provided in Table 5.1

Nature of wind shear profile: Governed by 'A'	Minimum wind shear coefficient 'β_w'	Minimum required wind speed (m/s)	Stability	Run time (sec)
1.0	0.08786	10.81015	stable	6.49072
1.3	0.07571	9.17826	stable	7.35974
1.5	0.06885	8.6375	stable	6.76173
1.7	0.06297	8.27792	stable	10.11729
1.9	0.05794	8.02221	stable	9.51732

Table 5.1: Comparison of different wind profiles for minimum wind shear

From Table 5.1, we can say that, as the profile tends towards the logarithmic variation, the requirement of minimum wind shear decreases. For first entry in the Table, the profile is linear and as we go down the profile gradually tends towards the logarithmic profile. Correspondingly, the values of minimum required wind shear coefficient 'β_w' and 'minimum required wind speed' decreases. Optimized trajectories are further checked for linear stability analysis by using Floquet theory, all the curves are stable as mentioned in the Table.

In general, it can be concluded that the value of minimum required wind velocity for dynamic soaring to happen, decreases, as the profile governing the wind shear tends towards the logarithmic profile.

5.2 Trajectory Optimization for Surveillance

Trajectories are obtained for the reduction of gray area. Objective function defined in the section 3.2, is used for optimization. Simulations are run in a software GPOPS-II, for the selected glider's model (Table: 2.1). In the first case, wind shear is assumed to be varying linearly with altitude, with wind shear coefficient β = 0.2. The view angle of camera (θ) is taken as 25 deg. The optimized trajectory is shown in Figure 5.6 and corresponding states and controls are displayed in the Figure 5.7.

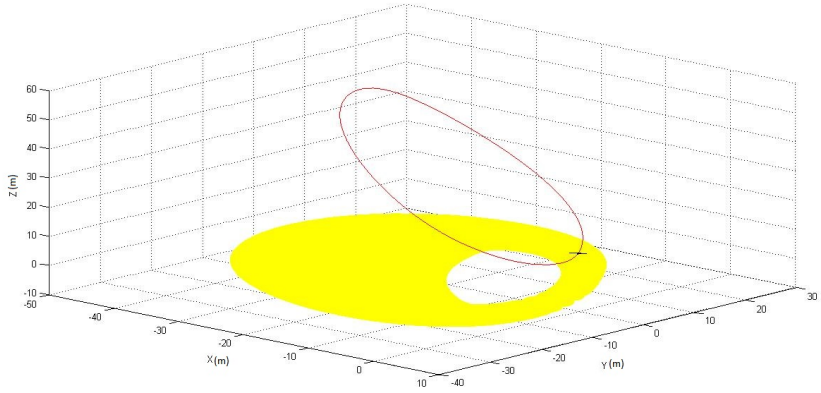

Figure 5.6: Area under surveillance for β_w = 0.2, A = 1.0, θ = 25 deg.

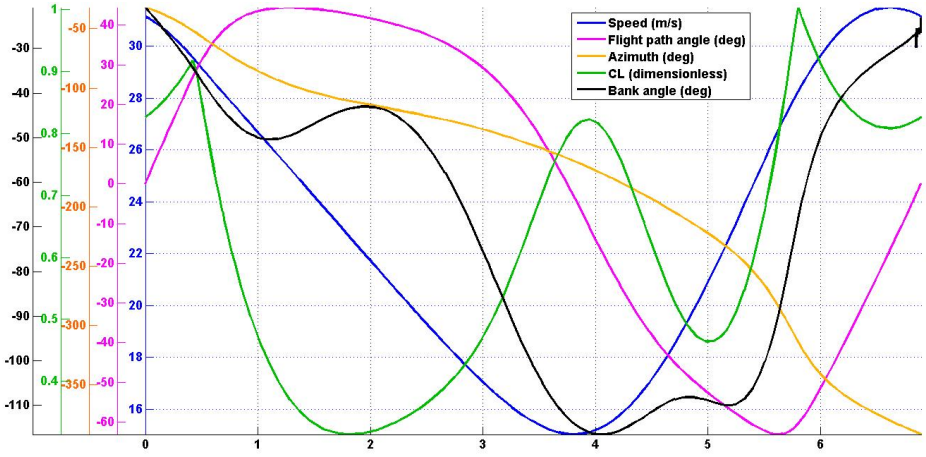

Figure 5.7: States and controls variation for β_w = 0.2, A = 1.0, θ = 25 deg.

Linear stability analysis of the trajectory is performed by using Floquet theory. For the linear variation of wind shear with β_w = 0.2, trajectory is unstable. Corresponding Eigen vectors for the states are displayed in Figures 5.8 – 5.13.

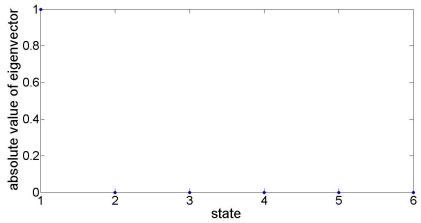

Figure 5.8: First Eigen vector

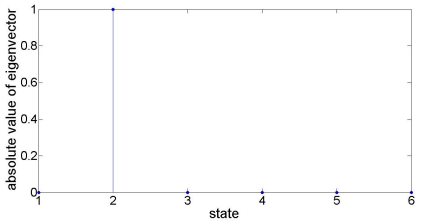

Figure 5.9: Second Eigen vector

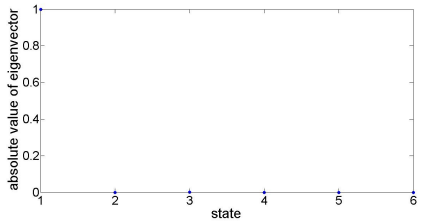

Figure 5.10: Third Eigen vector

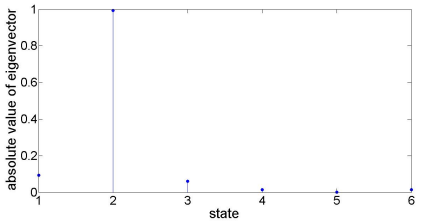

Figure 5.11: Fourth Eigen vector

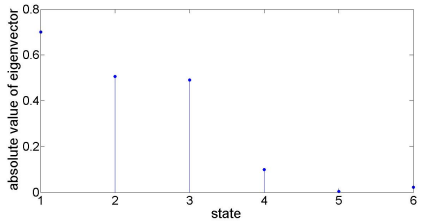

Figure 5.12: Fifth Eigen vector

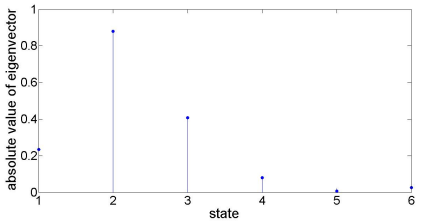

Figure 5.13: Sixth Eigen vector

Results of stability analysis are cross checked by performing the perturbation analysis of the set of governing equations. Perturbations are introduced the states of governing Equations 2.4 – 2.9. System of coupled ordinary differential equations is solved numerically in MATLAB. Results show that trajectory doesn't come back to its original orbit after perturbation. It can be seen in Figures 5.14 and 5.15

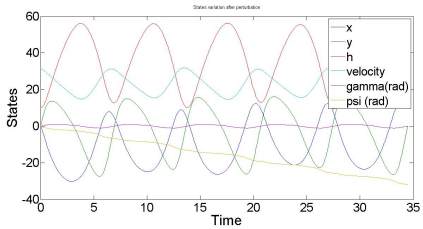

Figure 5.14: State variation with time

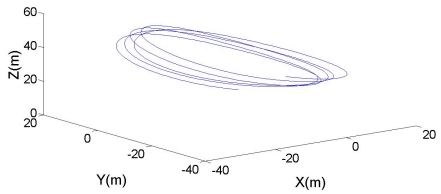

Figure 5.15: Trajectory for five cycles

In the second case, logarithmic variation of wind shear is used. Wind shear (β_w) is 0.2 and view angle of camera (θ) is 25 deg.

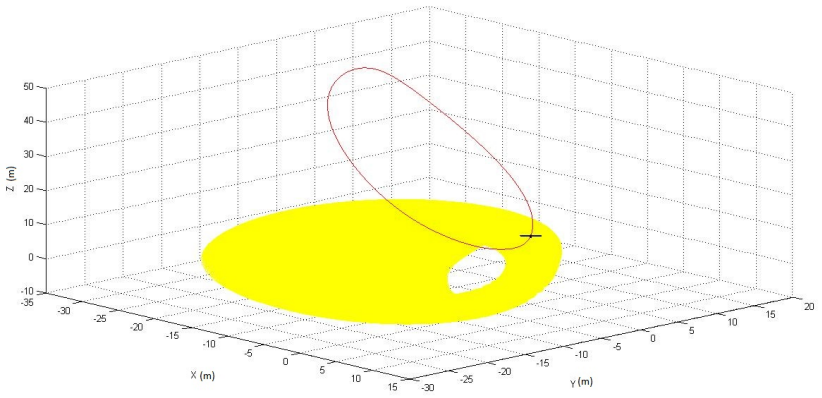

Figure 5.16: Area under surveillance for β_w = 0.2, A = 1.5, θ = 25 deg

Corresponding variation of states and controls is shown in Figure 5.17

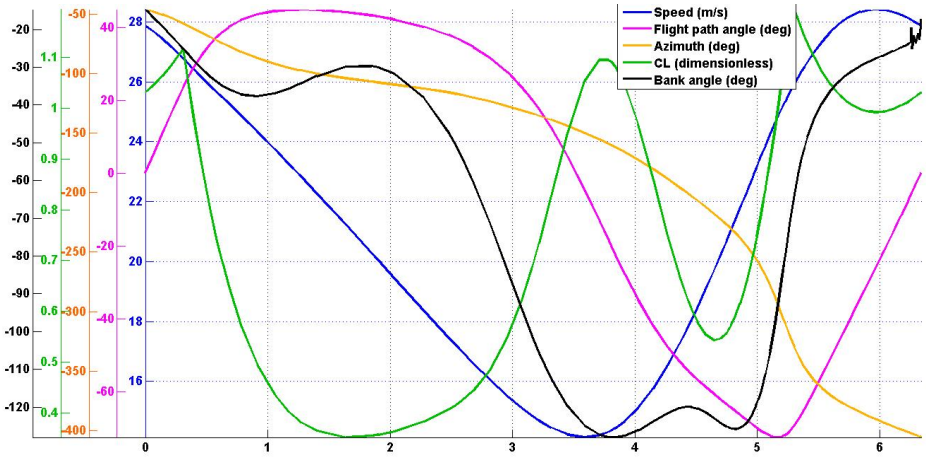

Figure 5.17: States and controls variation for β_w = 0.2, A = 1.5, θ = 25 deg.

Linear stability analysis is performed by using Floquet theory. For the logarithmic variation of wind shear with β_w = 0.2, the optimized trajectory is stable. Corresponding Eigen vectors are displayed in Figures 5.18 – 5.23.

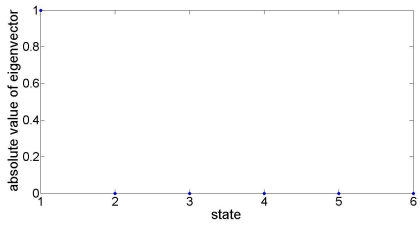

Figure 5.18: First Eigen vector

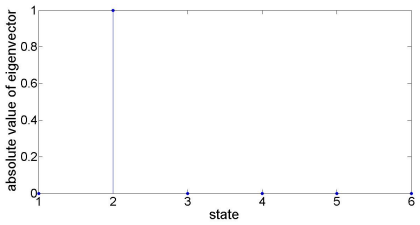

Figure 5.19: Second Eigen vector

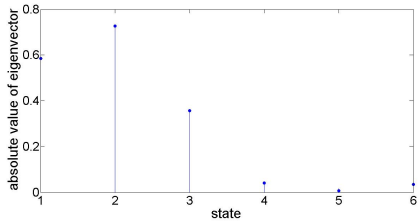

Figure 5.20: Third Eigen vector

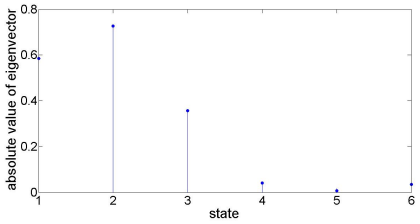

Figure 5.21: Fourth Eigen vector

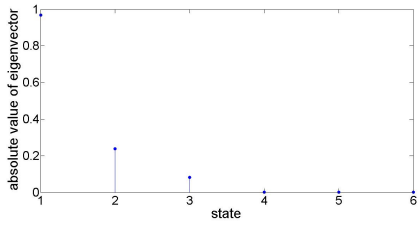

Figure 5.22: Fifth Eigen vector

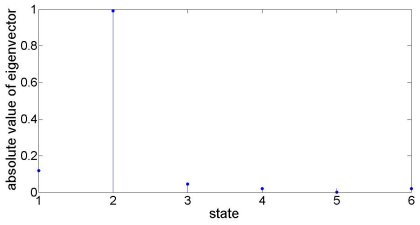

Figure 5.23: Sixth Eigen vector

Results of perturbation analysis for logarithmic wind shear profile, θ= 25 deg, and β_w = 0.2. are displayed in Figures 5.24, 5.25. This shows that trajectory comes back to its original orbit after perturbation. This is results given by Floquet theory.

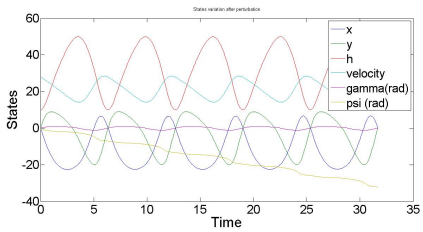

Figure 5.24: State variation with time

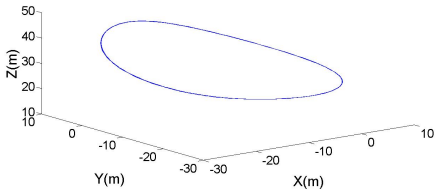

Figure 5.25: Trajectory for five cycles

Different trajectories are obtained by varying the value of β_w. In Figures 5.26 – 5.29, variation of surveillance area with the change in wind shear coefficient is displayed. For those trajectories θ = 25 degree and wind shear profile is logarithmic in nature.

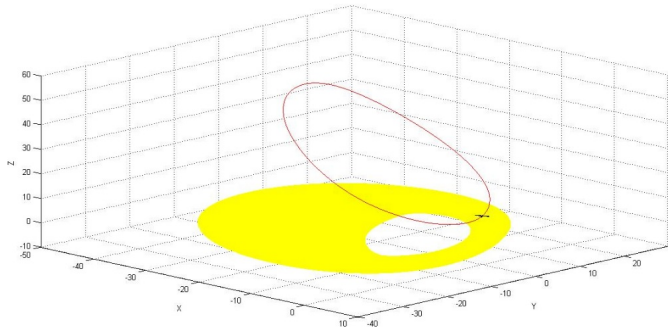

Figure 5.26: Trajectory optimization: β_w = 0.15

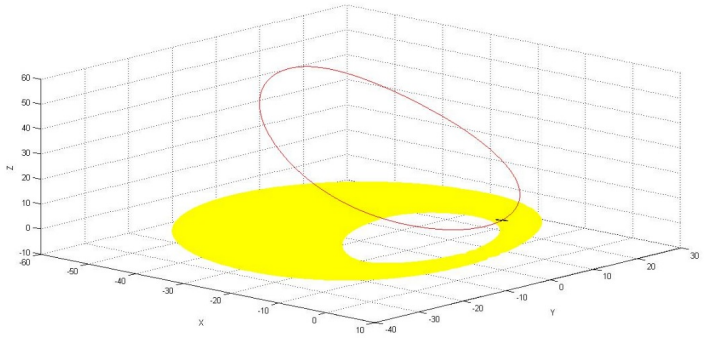

Figure 5.27: Trajectory optimization: β_w = 0.12

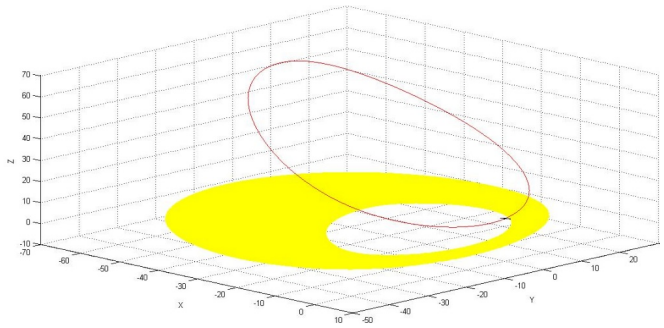

Figure 5.28: Trajectory optimization: β_w = 0.1

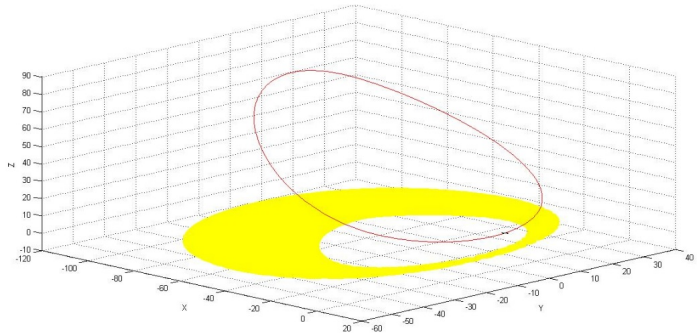

Figure 5.29: Trajectory optimization: β_w = 0.08

Numerous results are obtained by varying the different parameters. Mainly, variation of three parameters is useful in coming towards some conclusions. These three parameters are:

1 wind shear coefficient 'β_w'

2 Nature of wind profile governed by the parameter 'A'

3 View angle of camera 'θ'

For the other cases, instead of displaying all the trajectories and corresponding states-controls variations, final results are given in the tabular form and the final conclusions are written from the analysis of tabular data.

View angle of camera (θ) in degrees	Nature of wind profile: Governed by 'A'	Stability check
0	1.0	unstable
11	1.0	unstable
17	1.0	unstable
20	1.0	unstable
0	1.5	unstable
11	1.5	unstable
17	1.5	unstable
20	1.5	unstable

Table 5.2: Results for wind shear coefficient, β_w = 0.25

View angle of camera (θ) in degrees	Nature of wind profile: Governed by 'A'	Stability check
0	1.0	unstable
11	1.0	unstable
17	1.0	unstable
20	1.0	unstable
23	1.0	unstable
25	1.0	unstable
0	1.5	stable
11	1.5	stable
17	1.5	stable
20	1.5	stable
23	1.5	stable
25	1.5	stable

Table 5.3: Results for wind shear coefficient, β_w = 0.2

View angle of camera (θ) in degrees	Nature of wind profile: Governed by 'A'	Stability check
0	1.0	unstable
11	1.0	unstable
17	1.0	unstable
20	1.0	unstable
23	1.0	unstable
25	1.0	unstable
28	1.0	unstable
31	1.0	unstable
33	1.0	unstable
37	1.0	unstable
39	1.0	unstable
0	1.5	unstable
11	1.5	unstable
17	1.5	unstable
20	1.5	unstable
23	1.5	unstable
25	1.5	unstable
28	1.5	unstable
31	1.5	unstable
33	1.5	unstable
37	1.5	unstable
39	1.5	unstable

Table 5.4: Results for wind shear coefficient, β_w = 0.15

As shown in the Table 5.3, all the optimized trajectories for the linear variation of wind profile (A = 1), are unstable, whereas for logarithmic profile, trajectories are stable, when checked for linear stability using Floquet theory.

From the above Tables it can be concluded that, as the wind shear coefficient 'β_w' increases, the simulations can not be run for more view angle of camera 'θ'. So, in general as the value of β_w increases, the trajectories shrink and which decreases the gray portion. This effect can be observed in the Figures 5.26 – 5.29.

Figures 5.30 – 5.35 are obtained with a constant value of wind shear coefficient (β_w) and the nature of the wind shear is varied gradually from linear to logarithmic, by changing the value of a parameter 'A'. As the wind shear changes from linear to logarithmic, the gray portion decreases. So, in general, it can be predicted that, as the curvature

of wind shear increases (towards the logarithmic), the optimized trajectories contract inside.

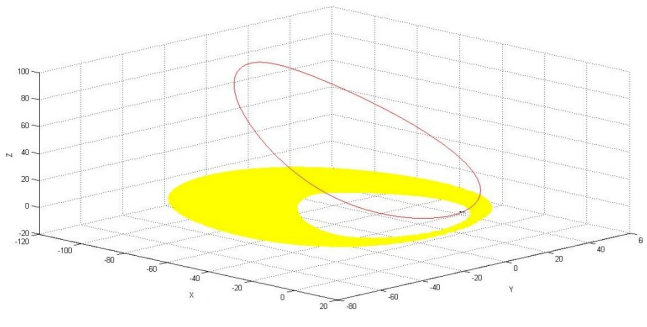

Figure 5.30: Trajectory optimization: A = 1.0

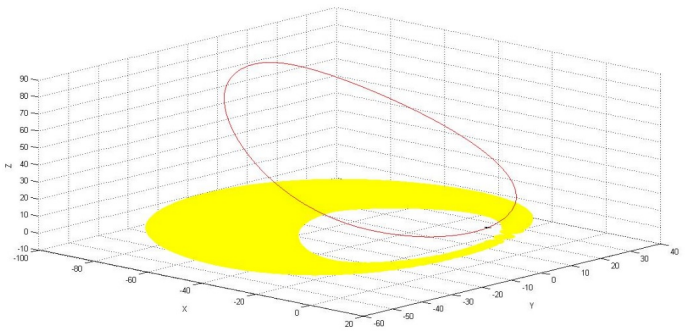

Figure 5.31: Trajectory optimization: A = 1.1

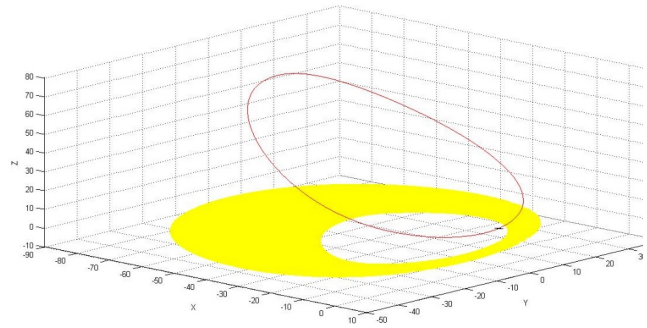

Figure 5.32: Trajectory optimization: A = 1.3

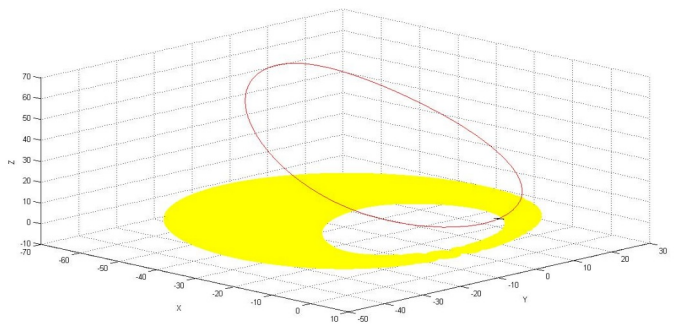

Figure 5.33: Trajectory optimization: A = 1.5

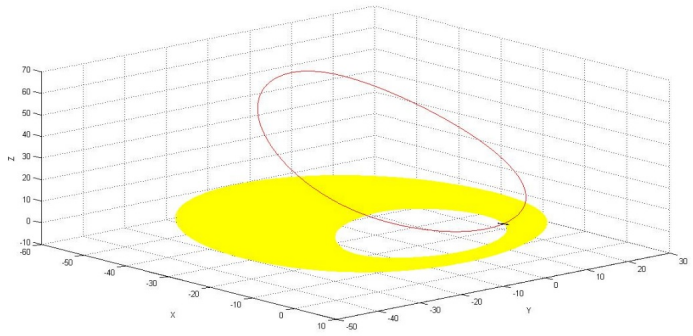

Figure 5.34: Trajectory optimization: A = 1.7

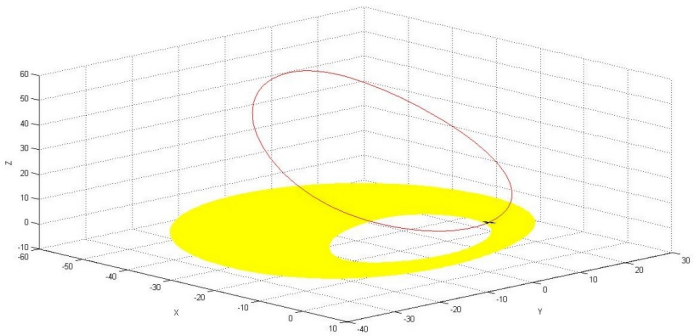

Figure 5.35: Trajectory optimization: A = 1.9

5.3 Summary

In the first part of the chapter, results for the optimized trajectory for minimum requirement of wind shear are discussed. Linear stability of optimized trajectory is checked using Floquet theory, also variations of results with respect to a parameters '*A*', (with the nature of wind shear) are studied.

In the second part, trajectories are optimized with the objective of reduction of gray area which comes into picture while surveying. Numerous results are obtained by varying the parameters β_w, θ, and A. Variation of the results is studied by keeping the one parameter constant at a time. Also linear stability analysis of the optimized trajectories is performed.

CHAPTER 6

Conclusions and Scope For Future Work

In the past, study of dynamic soaring was mainly focused on understanding the performance of albatrosses and other large seabirds who use dynamic soaring for the propulsion. Recently, some researchers have started to check the possibility of dynamic soaring as an alternative energy source for small UAVs. This work extends those studies by considering dynamic soaring as the only energy source for propulsion of small UAV, for surveillance application.

6.1 Summary of Findings

- It can be concluded that, the value of minimum required wind velocity to perform dynamic soaring decreases, as the profile governing the wind shear variation goes towards the logarithmic profile.
- Linear stability analysis of optimized trajectories is performed by using Floquet theory. Perturbations are introduced in the states variables of the optimized trajectories and it is concluded that, the trajectories for the minimum wind shear calculations are stable, provided the control variables are unperturbed.
- There is minimum requirement of wind shear coefficient ($\beta_{w_{min}}$) for dynamic soaring to happen. In simulations, the condition $\beta_w \geq \beta_{w_{min}}$ is always satisfied in the trajectory optimization for surveillance application. It is observed that, with increase of wind shear coefficient (β_w), trajectories shrink, which is good for surveillance application as the gray area reduces. Similarly, with decrease of wind shear coefficient (β_w), trajectories spread out.
- Trajectories for the surveillance application are studied for the variation of nature of wind shear. It is concluded that, as the wind shear changes from linear

to logarithmic, trajectories contract inside and ultimately the inner gray portion decreases.

- Gray area can be minimized further by increasing the view angle of camera (θ). However, the value of 'θ' depends on the application for which surveillance is happening. For patrolling purpose, value of 'θ' can be higher, but for some cases such as focusing on small objects or at a point, the view angle has to be smaller. In the second case, actually there is no need to observe the gray area, since the attention is given on the particular object or at a specific point. In this work, trajectories are found particularly for the first application, patrolling purpose, with the higher values of 'θ'.

- Simplest way to optimize trajectory for dynamic soaring is by using point mass equations of motion. In this approach, number of state and control variables are less. But it's main disadvantages are: unable to capture rotation angles and angular velocities about the axes.

6.2 Suggestions for Future Work

In many ways the work presented in this thesis can be extended. Ultimately the goal would be to design an autonomous dynamic soaring UAV that is capable of surveying without any power input. This vehicle could also be useful for delivering the science data of the ocean surface, or in monitoring the shipping lanes and commercial fisheries. In the shorter term, some of the tasks that can present interesting research opportunities are following:

- Perform trajectory optimization for a higher–fidelity model by including the forces and moments on the parts of aeroplane. It would be really interesting to model the wind speed variation across the span and its effect on the aircraft parts. These considerations will be important when flying in stronger wind gradients or with huge vehicles.

- Build an autonomous UAV flying using dynamic soaring as the propulsive force, which will track the optimized trajectory. This is possible in practice by coupling the flight dynamics of UAV with the controls defined for optimized trajectoried.

APPENDIX

In appendix codes developed for the trajectory optimization and for checking the stability are given. All the codes are developed in MATLAB based software GPOPS-II.

- First set of codes is used in finding the trajectory for minimum wind shear require ment.

6/5/15 8:57 PM E:\SOFTWARE\gpops-ii-version-2...\DS_MAIN.m 1 of 3

```
clear all
clc
tic
%-------------------------------------------------------------------------%
%------------------ Provide Auxiliary Data for Problem -------------------%
%-------------------------------------------------------------------------%
auxdata.rho  = 1.2;
auxdata.CD0  = 0.00873;
auxdata.K    = 0.045;
auxdata.g    = 9.81;
auxdata.m    = 10;
auxdata.S    = 1;
auxdata.W0   = 0;
auxdata.A    = 1.3;

%-------------------------------------------------------------------------%
%---------------------- Boundary Conditions ------------------------------%
%-------------------------------------------------------------------------%
t0 = 0; x0 = 0; y0 = 0; z0 = 0; v0 = 100;

%-------------------------------------------------------------------------%
%---------------------- Limits on Variables ------------------------------%
%-------------------------------------------------------------------------%
c = pi/180;
tf_min    = 1;          tf_max    = 100;
x_min     = -1000;      x_max     = +1000;
y_min     = -1000;      y_max     = +1000;
z_min     =  0;         z_max     = +1000;
v_min     = +10;        v_max     = +350;
gamma_min = -75*c;      gamma_max = 75*c;
psi_min   = -540*c;     psi_max   = 90*c;
beta_min  = 0.005;      beta_max  = 0.15;
CL_min    = -0.5;       CL_max    = 1.5;
mu_min   = -75*c;       mu_max    = 75*c;

%-------------------------------------------------------------------------%
%--------------- Set Up Problem Using Data Provided Above ----------%
%-------------------------------------------------------------------------%
bounds.phase.initialtime.lower  = t0;
bounds.phase.initialtime.upper  = t0;
bounds.phase.finaltime.lower    = tf_min;
bounds.phase.finaltime.upper    = tf_max;
bounds.phase.initialstate.lower = [x0, y0, z0, v_min, gamma_min, psi_min];
bounds.phase.initialstate.upper = [x0, y0, z0, v_max, gamma_max, psi_max];
bounds.phase.state.lower        = [x_min, y_min, z_min, v_min, gamma_min, psi_min];
bounds.phase.state.upper        = [x_max, y_max, z_max, v_max, gamma_max, psi_max];
bounds.phase.finalstate.lower   = [x0, y0, z0, v_min, gamma_min, psi_min];
bounds.phase.finalstate.upper   = [x0, y0, z0, v_max, gamma_max, psi_max];
bounds.phase.control.lower      = [CL_min, mu_min];
bounds.phase.control.upper      = [CL_max, mu_max];
bounds.phase.path.lower         = -2;
```

```
bounds.phase.path.upper         = 5;
bounds.eventgroup(1).lower      = [0, 0, -2*pi];
bounds.eventgroup(1).upper      = [0, 0, -2*pi];
bounds.parameter.lower          = beta_min;
bounds.parameter.upper          = beta_max;

%------------------------------------------------------------------------%
%---------------------- Provide Guess of Solution -----------------------%
%------------------------------------------------------------------------%
N                               = 100;
CL0                             = CL_max;
tGuess                          = linspace(0,24,N).';
xguess                          = 500*cos(2*pi*tGuess/24)-500;
yguess                          = 300*sin(2*pi*tGuess/24);
zguess                          = -400*cos(2*pi*tGuess/24)+400;
vguess                          = 0.8*v0*(1.5+cos(2*pi*tGuess/24));
gammaguess                      = pi/6*sin(2*pi*tGuess/24);
psiguess                        = -1-tGuess/4;
CLguess                         = CL0*ones(N,1)/3;
muguess                         = -ones(N,1);
betaguess                       = 0.08;
guess.phase.time                = tGuess;
guess.phase.state               = [xguess, yguess, zguess, vguess, gammaguess, psiguess];
guess.phase.control             = [CLguess, muguess];
guess.parameter                 = betaguess;

%------------------------------------------------------------------------%
%----------Provide Mesh Refinement Method and Initial Mesh ---------------%
%------------------------------------------------------------------------%
mesh.maxiteration               = 10;
mesh.method                     = 'hp-LiuRao';
mesh.tolerance                  = 1e-6;

%------------------------------------------------------------------------%
%---------- Configure Setup Using the information provided --------------%
%------------------------------------------------------------------------%
setup.name                              = 'DS_MAIN';
setup.functions.continuous              = @DSContinuous;
setup.functions.endpoint                = @DSEndpoint;
setup.nlp.solver                        = 'ipopt';
setup.nlp.ipoptoptions.linear_solver    = 'ma57';
setup.displaylevel                      = 2;
setup.auxdata                           = auxdata;
setup.bounds                            = bounds;
setup.guess                             = guess;
setup.mesh                              = mesh;
setup.derivatives.supplier              = 'sparseCD';
setup.derivatives.derivativelevel       = 'second';
setup.scales.method                     = 'automatic-bounds';
setup.method                            = 'RPM-Differentiation';
```

```
%-------------------------------------------------------------------------%
%------------------ Solve Problem Using GPOPS-II--------------------%
%-------------------------------------------------------------------------%
output = gpops2(setup);
solution = output.result.solution;
multi_axes;
minimum_beta = solution.parameter
z = max(solution.phase.state(:,3));
A = auxdata.A ;
Min_required_windspeed = minimum_beta.*(A.*z + (1 - A)./213.*z.^2)
toc
DS_Stability_logarithmic;
```

```
function phaseout = DSContinuous(input)
s             = input.phase(1).state;
u             = input.phase(1).control;
p             = input.phase(1).parameter;

z             = s(:,3);
v             = s(:,4);
gamma         = s(:,5);
psi           = s(:,6);

CL            = u(:,1);
mu            = u(:,2);

beta          = p(:,1);

rho           = input.auxdata.rho;
S             = input.auxdata.S;
CD0           = input.auxdata.CD0;
K             = input.auxdata.K;
g             = input.auxdata.g;
m             = input.auxdata.m;
W0            = input.auxdata.W0;

% % % w_h           = beta.*z + W0;
% % % DWxDt         = beta.*v.*sin(gamma);

A             = input.auxdata.A;
w_h           = beta.*(A.*z + (1 - A)./213.*z.^2) + W0;
DWxDt         = beta.*v.*sin(gamma).*(A + (1 - A)./213.*2.*z);


xdot          = v.*cos(gamma).*sin(psi) + w_h;
ydot          = v.*cos(gamma).*cos(psi);
zdot          = v.*sin(gamma);
vdot          = -(rho*S)/(2*m)*(CD0+K*CL.^2).*v.^2 - g*sin(gamma) - DWxDt.*sin(psi).*cos↙
(gamma);
gammadot      = (rho*S)/(2*m)*CL.*v.*cos(mu) - g*cos(gamma)./v + DWxDt.*sin(psi).*sin↙
(gamma)./v;
psidot        = ((rho*S)/(2*m)*CL.*v.*sin(mu)- DWxDt.*cos(psi)./v)./cos(gamma);

ngconstant    = (0.5*rho*S/m/g);
ng            = ngconstant.*CL.*v.^2;

phaseout.dynamics  = [xdot, ydot, zdot, vdot, gammadot, psidot];
phaseout.path = ng;
```

```
function output = DSEndpoint(input)
x0    = input.phase(1).initialstate;
xf    = input.phase(1).finalstate;
beta  = input.parameter;
output.eventgroup(1).event = [xf(4)-x0(4), xf(5)-x0(5), xf(6)-x0(6)];
output.objective = beta;
```

- This set of codes is used in trajectory optimization for surveillance application.

6/5/15 8:48 PM E:\SOFTWARE\gpops-i...\dynamicSoaringMain.m 1 of 3

```
clear all
clc

%-------------------------------------------------------------------------%
%------------------ Provide Auxiliary Data for Problem -------------------%
%-------------------------------------------------------------------------%
auxdata.rho  = 1.225;
auxdata.CD0  = 0.00873;
auxdata.K    = 0.045;
auxdata.g    = 9.81;
auxdata.m    = 10;
auxdata.S    = 1;
% % % auxdata.beta = 0.063726299384995;
auxdata.beta = 0.1;
% % % auxdata.beta = 0.08;
auxdata.W0   = 0;
auxdata.lmin = -2;
auxdata.lmax = 5;
auxdata.A    = 1.1;

%-------------------------------------------------------------------------%
%---------------------- Boundary Conditions ------------------------------%
%-------------------------------------------------------------------------%
t0 = 0; x0 = 0; y0 = 0; z0 = 10; v0 = 100; A0 = 0;

%-------------------------------------------------------------------------%
%---------------------- Limits on Variables ------------------------------%
%-------------------------------------------------------------------------%
tfmin    = 1;          tfmax    = 1000;
xmin     = -1000;      xmax     = +1000;
ymin     = -1000;      ymax     = +1000;
zmin     =  10;        zmax     = +1000;
vmin     = +10;        vmax     = +350;
gammamin = -75*pi/180; gammamax = 75*pi/180;
psimin   = -3*pi;      psimax   = +pi/2;
% % % Amin     = 0;          Amax     = 1e10;
CLmin    = -2;         CLmax    = 1.5;
mumin    = -180/180*pi;mumax    = 75/180*pi;
% % % beta_min = 0.005;      beta_max = 0.15;

k_max = (0.5*auxdata.rho.*auxdata.S.*CLmax) / auxdata.m;
k_min = (0.5*auxdata.rho.*auxdata.S.*CLmin) / auxdata.m;

%-------------------------------------------------------------------------%
%--------------- Set Up Problem Using Data Provided Above ----------%
%-------------------------------------------------------------------------%
bounds.phase.initialtime.lower  = t0;
bounds.phase.initialtime.upper  = t0;
bounds.phase.finaltime.lower    = tfmin;
bounds.phase.finaltime.upper    = tfmax;
bounds.phase.initialstate.lower = [x0, y0, z0, vmin, gammamin, psimin];
```

```
bounds.phase.initialstate.upper = [x0, y0, z0, vmax, gammamax, psimax];
bounds.phase.state.lower        = [xmin, ymin, zmin, vmin, gammamin, psimin];
bounds.phase.state.upper        = [xmax, ymax, zmax, vmax, gammamax, psimax];
bounds.phase.finalstate.lower   = [x0, y0, z0, vmin, gammamin, psimin];
bounds.phase.finalstate.upper   = [x0, y0, z0, vmax, gammamax, psimax];
bounds.phase.control.lower      = [CLmin, mumin];
bounds.phase.control.upper      = [CLmax, mumax];
bounds.phase.path.lower         = [auxdata.lmin , k_min];
bounds.phase.path.upper         = [auxdata.lmax , k_max];
% % % bounds.phase.integral.lower     = -1e15;
bounds.phase.integral.lower     = 0;
bounds.phase.integral.upper     = 1e15;
bounds.eventgroup.lower         = [0, 0, -2*pi];
bounds.eventgroup.upper         = [0, 0, -2*pi];
% % % bounds.parameter.lower          = beta_min;
% % % bounds.parameter.upper          = beta_max;

%---------------------------------------------------------------------------%
%---------------------- Provide Guess of Solution --------------------------%
%---------------------------------------------------------------------------%
N                               = 100;
CL0                             = CLmax;
tGuess                          = linspace(0,24,N).';
xguess                          = 500*cos(2*pi*tGuess/24)-500;
yguess                          = 300*sin(2*pi*tGuess/24);
zguess                          = -400*cos(2*pi*tGuess/24)+400;
vguess                          = 0.8*v0*(1.5+cos(2*pi*tGuess/24));
gammaguess                      = pi/6*sin(2*pi*tGuess/24);
psiguess                        = -1-tGuess/4;
CLguess                         = CL0*ones(N,1)/3;
muguess                         = -ones(N,1);
% % % beta_guess                     = 0.08;
guess.phase.time                = tGuess;
guess.phase.state               = [xguess, yguess, zguess, vguess, gammaguess, psiguess];
guess.phase.control             = [CLguess, muguess];
% % % guess.parameter                = beta_guess;
guess.phase.integral            = 1e+02;

%---------------------------------------------------------------------------%
%----------Provide Mesh Refinement Method and Initial Mesh ----------------%
%---------------------------------------------------------------------------%
mesh.maxiteration               = 10;
mesh.method                     = 'hp-LiuRao';
mesh.tolerance                  = 1e-5;
% % % mesh.tolerance                 = 1e-6;
% % % mesh.tolerance                 = 1e0;

%---------------------------------------------------------------------%
%---------- Configure Setup Using the information provided ---------%
%---------------------------------------------------------------------%
setup.name                              = 'Dynamic-Soaring-Problem';
```

```
setup.functions.continuous              = @dynamicSoaringContinuous;
setup.functions.endpoint                = @dynamicSoaringEndpoint;
setup.nlp.solver                        = 'ipopt';
setup.nlp.ipoptoptions.linear_solver    = 'mumps';
setup.nlp.ipoptoptions.tolerance        = 1e-7;
setup.nlp.ipoptoptions.maxiterations    = 2e8;
setup.nlp.ipoptoptions.linear_solver    = 'ma57';
setup.displaylevel                      = 2;
setup.auxdata                           = auxdata;
setup.bounds                            = bounds;
setup.guess                             = guess;
setup.mesh                              = mesh;
setup.derivatives.supplier              = 'sparseCD';
setup.derivatives.derivativelevel       = 'second';
setup.scales.method                     = 'none';
setup.method                            = 'RPM-Differentiation';

%---------------------------------------------------------------------%
%------------------ Solve Problem Using GPOPS-II---------------------%
%---------------------------------------------------------------------%
output = gpops2(setup);
solution = output.result.solution;

% % % dynamicSoaringPlot;
multi_axes;
t_f = max(solution.phase.time)
avg_Minimum_Difference_radii = solution.phase.integral/ max(solution.phase.time)
```

```
function phaseout = dynamicSoaringContinuous(input)

s             = input.phase.state;
u             = input.phase.control;
% % % p             = input.phase.parameter;

x             = s(:,1);
y             = s(:,2);
z             = s(:,3);
v             = s(:,4);
gamma         = s(:,5);
psi           = s(:,6);

CL            = u(:,1);
mu            = u(:,2);

% % % beta          = p(:,1);

S             = input.auxdata.S;
beta          = input.auxdata.beta;
rho           = input.auxdata.rho;
CD0           = input.auxdata.CD0;
K             = input.auxdata.K;
g             = input.auxdata.g;
m             = input.auxdata.m;
W0            = input.auxdata.W0;
% %
% % w_h           = beta.*z + W0;
% % DWxDt         = beta.*v.*sin(gamma);

A             = input.auxdata.A;
w_h           = beta.*(A.*z + (1 - A)./213.*z.^2) + W0;
DWxDt         = beta.*v.*sin(gamma).*(A + (1 - A)./213.*2.*z);

xdot          = v.*cos(gamma).*sin(psi) + w_h;
ydot          = v.*cos(gamma).*cos(psi);
zdot          = v.*sin(gamma);
vdot          = -(rho*S)/(2*m)*(CD0+K*CL.^2).*v.^2 - g*sin(gamma) - DWxDt.*sin(psi).*cos↙
(gamma);
gammadot      = (rho*S)/(2*m)*CL.*v.*cos(mu) - g*cos(gamma)./v + DWxDt.*sin(psi).*sin↙
(gamma)./v;
psidot        = ((rho*S)/(2*m)*CL.*v.*sin(mu) - DWxDt.*cos(psi)./v)./cos(gamma);

ngconstant    = (0.5*rho*S/m/g);
ng            = ngconstant.*CL.*v.^2;
ngv           = ng./v.^2;
phaseout.path = [ng , ngv];

phaseout.dynamics  = [xdot, ydot, zdot, vdot, gammadot, psidot];

% % % phaseout.integrand = 0.5*(sqrt(x.^2 + y.^2) - 0.14.*z).^2.*abs(psidot);
```

```
% % % phaseout.integrand = sqrt(x.^2 + y.^2) - 0.24.*z;
% % % phaseout.integrand = sqrt(x.^2 + y.^2) - 0.25.*z;
phaseout.integrand = sqrt(x.^2 + y.^2) - 0.22.*z;
```

```
function output = dynamicSoaringEndpoint(input)

x0               = input.phase.initialstate;
xf               = input.phase.finalstate;

% % % beta             = input.parameter;
% % % output.objective = beta;

q                = input.phase.integral;
output.objective = q;

output.eventgroup.event = [xf(4)-x0(4), xf(5)-x0(5), xf(6)-x0(6)];
```

- This code is developed to find out the stability of optimized trajectory using Floquet theory.

6/5/15 9:09 PM E:\SOFTWARE\g...\DS Stability logarithmic.m 1 of 3

```
t = solution.phase.time;
EQsol = solution.phase.state;
control = solution.phase.control;

% beta = solution.parameter;
beta = auxdata.beta; A = auxdata.A;
options = odeset('RelTol',1e-10,'AbsTol',1e-10);

m = length(t);
period = t(m);                                    % selected by GPOPS for given nodes

N = 6;
K = auxdata.K;
H = auxdata.rho*auxdata.S/(2*auxdata.m);

CLpp = spline(t,control(:,1));
mupp = spline(t,control(:,2));
xpp = spline(t,EQsol(:,1));
ypp = spline(t,EQsol(:,2));
zpp = spline(t,EQsol(:,3));
vpp = spline(t,EQsol(:,4));
gamapp = spline(t,EQsol(:,5));
psipp = spline(t,EQsol(:,6));

CL = @(t)(ppval(CLpp,t));
mu = @(t)(ppval(mupp,t));
x = @(t)(ppval(xpp,t));
y = @(t)(ppval(ypp,t));
z = @(t)(ppval(zpp,t));
v = @(t)(ppval(vpp,t));
gamma = @(t)(ppval(gamapp,t));
psi = @(t)(ppval(psipp,t));

g = 9.81;
Cd0 = 0.00873;

Pertder = @(t)[0,0,beta.*(A+(2/213).*(1+(-1).*A).*z(t)),cos(gamma(t)).*sin(psi(t)),...
    (-1).*sin(gamma(t)).*sin(psi(t)).*v(t),cos(gamma(t)).*cos(psi(t)).*v(t);

    0,0,0,cos(gamma(t)).*cos(psi(t)),(-1).*cos(psi(t)).*sin(gamma(t)).*v(t),(-1).*...
    cos(gamma(t)).*sin(psi(t)).*v(t);

    0,0,0,sin(gamma(t)),cos(gamma(t)).*v(t),0;

    0,0,(-2/213).*(1+(-1).*A).*beta.*cos(gamma(t)).*sin(psi(t)).*v(t).*...
    sin(gamma(t)), (-2).*H.*(Cd0+CL(t).^2.*K).*v(t) - beta.*sin(gamma(t)).*...
    sin(psi(t)).*cos(gamma(t)).*(A + (1 - A)./213.*2.*z(t)), (-1).*g.*...
    cos(gamma(t)) - beta.*(cos(gamma(t))).^2.*sin(psi(t)).*(A + (1 - A)./213....
    *2.*z(t)).*v(t) + beta.*v(t).*(sin(gamma(t))).^2.*sin(psi(t)).*(A + (1 - A)./213....
    *2.*z(t)), -beta.*v(t).*sin(gamma(t)).*cos(gamma(t)).*
    cos(psi(t)).*(A + (1 - A)./213.*2.*z(t));
```

```
    0,0,(2/213).*(1+(-1).*A).*beta.*sin(gamma(t)).*sin(psi(t)).*sin(gamma(t)),...
    CL(t).*H.*cos(mu(t))+g.*cos(gamma(t)).*v(t).^(-2), g.*sin(gamma(t)).*...
    v(t).^(-1) + cos(gamma(t)).*sin(psi(t)).*sin(gamma(t)).*2.*beta.*...
    (A + (1 - A)./213.*2.*z(t)), cos(psi(t)).*sin(gamma(t)).*...
    sin(gamma(t)).*beta.*(A + (1 - A)./213.*2.*z(t));

    0,0,(-2/213).*(1+(-1).*A).*beta.*cos(psi(t)).*tan(gamma(t)), CL(t).*H.*...
    sec(gamma(t)).*sin(mu(t)), CL(t).*H.*sin(mu(t)).*v(t).*tan(gamma(t)).*...
    sec(gamma(t)) + (-1).*beta.*cos(psi(t)).*(A + (1 - A)./213.*2.*z(t))*...
    (sec(gamma(t))).^2 , beta.*tan(gamma(t)).*sin(psi(t)).*(A + (1 - A)./213.*2.*z(t))];


odeLinDS = @(tau,xstate)(Pertder(tau)*(xstate));

initial = eye(N);
FTM = zeros(N);

for i = 1:N
[T,X] = ode45(odeLinDS,0:period/10:period,initial(:,i),options);
FTM(:,i) = X(end,:);
end

[V,D] = eig(FTM);
freq = (1/period).*angle(diag(D))
damp = (1/period).*log(abs(diag(D)))

V1 = V(:,1);
V2 = V(:,2);
V3 = V(:,3);
V4 = V(:,4);
V5 = V(:,5);
V6 = V(:,6);

figure(1);
stem(abs(V1),'filled');
xl = xlabel('state');
yl = ylabel('absolute value of eigenvector');
set(xl,'FontSize',30);
set(yl,'FontSize',30);
set(gca,'FontSize',25);

figure(2);
stem(abs(V2),'filled');
xl = xlabel('state');
yl = ylabel('absolute value of eigenvector');
set(xl,'FontSize',30);
set(yl,'FontSize',30);
set(gca,'FontSize',25);
```

```
figure(3);
stem(abs(V3),'filled');
xl = xlabel('state');
yl = ylabel('absolute value of eigenvector');
set(xl,'FontSize',30);
set(yl,'FontSize',30);
set(gca,'FontSize',25);

figure(4);
stem(abs(V4),'filled');
xl = xlabel('state');
yl = ylabel('absolute value of eigenvector');
set(xl,'FontSize',30);
set(yl,'FontSize',30);
set(gca,'FontSize',25);

figure(5);
stem(abs(V5),'filled');
xl = xlabel('state');
yl = ylabel('absolute value of eigenvector');
set(xl,'FontSize',30);
set(yl,'FontSize',30);
set(gca,'FontSize',25);

figure(6);
stem(abs(V6),'filled');
xl = xlabel('state');
yl = ylabel('absolute value of eigenvector');
set(xl,'FontSize',30);
set(yl,'FontSize',30);
set(gca,'FontSize',25);
```

- This code is used to plot the variations of states and controls w.r.t. time.


```
x = solution.phase.time;
N = numel(x);
y1 = solution.phase.state(:,4);
y2 = 180./pi.*solution.phase.state(:,5);
y3 = 180./pi.*solution.phase.state(:,6);
y4 = solution.phase.control(:,1);
y5 = 180./pi.*solution.phase.control(:,2);


%# Some initial computations:
axesPosition = [220 40 1100 605];  %# Axes position, in pixels
yWidth = 40;                       %# y axes spacing, in pixels
xLimit = [min(x) max(x)];          %# Range of x values
xOffset = -yWidth*diff(xLimit)/axesPosition(3);

%# Create the figure and axes:
figure('Units','pixels','Position',[200 200 330 260]);
h1 = axes('Units','pixels','Position',axesPosition,...
          'Color','w','XColor','k','YColor', 'b',...
          'XLim',xLimit,'YLim',[min(y1) max(y1)],'NextPlot','add');

h2 = axes('Units','pixels','Position',axesPosition+yWidth.*[-1 0 1 0],...
          'Color','none','XColor','k','YColor','m',...
          'XLim',xLimit+[xOffset 0],'YLim',[min(y2) max(y2)],...
          'XTick',[],'XTickLabel',[],'NextPlot','add');

h3 = axes('Units','pixels','Position',axesPosition+yWidth.*[-2 0 2 0],...
          'Color','none','XColor','k','YColor',[1.0, 0.7, 0.0],...
          'XLim',xLimit+[2*xOffset 0],'YLim',[min(y3) max(y3)],...
          'XTick',[],'XTickLabel',[],'NextPlot','add');

h4 = axes('Units','pixels','Position',axesPosition+yWidth.*[-3 0 3 0],...
          'Color','none','XColor','k','YColor',[0.0, 0.75, 0.0],...
          'XLim',xLimit+[3*xOffset 0],'YLim',[min(y4) max(y4)],...
          'XTick',[],'XTickLabel',[],'NextPlot','add');

h5 = axes('Units','pixels','Position',axesPosition+yWidth.*[-4 0 4 0],...
          'Color','none','XColor','k','YColor', 'k',...
          'XLim',xLimit+[4*xOffset 0],'YLim',[min(y5) max(y5)],...
          'XTick',[],'XTickLabel',[],'NextPlot','add');

xlabel(h1,'Time(sec)');
ylabel(h5,'States & Controls');

%# Plot the data:
a = plot(h1,x,y1,'b', 'linewidth', 2);
b = plot(h2,x,y2,'m', 'linewidth', 2);
c = plot(h3,x,y3,'Color', [1.0, 0.7, 0.0], 'linewidth', 2);
d = plot(h4,x,y4,'Color', [0.0, 0.75, 0.0], 'linewidth', 2);
%plot(h5,x,y5,'Color', [1.0, 0.6, 0.0]);
e = plot(h5,x,y5,'k', 'linewidth', 2);
```

```
grid (h1,'on');
% % % grid (h2,'on');
% % % grid (h3,'on');
% % % grid (h4,'on');
% % % grid (h5,'on');

legend([a; b; c; d; e], {'Speed (m/s)','Flight path angle (deg)','Azimuth (deg)','CL↙
(dimensionless)','Bank angle (deg)'});
% % % Pos = get(Leg, 'Position');
% % % set(Leg, 'Position', [1 - Pos(3), Pos(2:4)]);

%legend('Speed (m/s)','Flight path angle (deg)','Azimuth (deg)','CL↙
(dimensionless)','Bank angle (deg)')
```

REFERENCES

Akhtar, N., **J. F. Whidborne**, and **A. K. Cooke**, Wind shear energy extraction using dynamic soaring techniques. *In 47 th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*. 2009.

Allen, M., Autonomous soaring for improved endurance of a small uninhabited air vehicle. *In 43 rd AIAA Aerospace science Meeting and Exhibit, Reno, NV,*. AIAA 2005, 2005.

Allen, M., Updraft model for development of autonomous soaring uninhabited air vehicles. *In 44 rd AIAA Aerospace science Meeting and Exhibit, Reno, NV,*. AIAA 2006–1510, 2006.

Allen, M. J. and **V. Lin**, Guidance and control of an autonomous soaring vehicle with flight test results. *In In 45 th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV*. AIAA 2007–867, 2007.

Bower, G. C. (2011). *BOUNDARY LAYER DYNAMIC SOARING FOR AUTONOMOUS AIRCRAFT: DESIGN AND VALIDATION*. Ph.D. thesis, Stanford University.

Chicone, C., *Ordinary Differential Equations with Applications*. Springer, 2006.

Deittert, M., **A. Richards**, **C. A. Toomer**, and **A. Pipe** (2009). Engineless unmanned aerial vehicle propulsion by dynamic soaring. *Journal of Guidance Control and Dynamics*, **32**.

Edwards, D. J., Implimentation details and flight test results of an autonomous soaring controller. *In AIAA Guidance, Navigation and Control Conference*. AIAA 2008–7244, 2008.

Gao, X.-Z., **Z.-X. Hou**, **Z. Guo**, **R.-F. Fan**, and **X.-Q. Chen** (2014). Analysis and design of guidance-strategy for dynamic soaring with uavs. *Elsevier–Control Engineering Practice*, **32**(1), 218–226.

Gong, Q., **F. Fahroo**, and **I. Ross** (2008). Spectral algorithm for pseudospectral methods in optimal control. *Journal of guidance, control, and dynamics*, **31**(3).

Gordon, R. J. (2006). *Optimal dynamic soaring for full size sailplanes*. Ph.D. thesis, Ohio Air University, Ohio.

Langelaan, J. W. and **N. Roy** (2009). Enabling new missions for robotic aircraft. *Science*, **326**.

Langelaan, J. W., **J. Spletzer**, **C. Montella**, and **J. Grenestedt**, Wind field estimation forautonomous dynamic soaring. *In IEEE international conference on robotics and automation*. 2012.

Lawrance, R. J. and **S. Sukkarieh**, Wind shear energy extraction using dynamic soaring techniques. *In A guidance and control strategy for dynamic soaring with a gliding UAV*. 2009.

Lawrance, R. J. and **S. Sukkarieh** (2011). Autonomous exploration of a wind field with a gliding aircraft. *Journal of Guidance, Control, and Dynamics*, **34**(3), 719–723.

Mikael, R. and **H. Anders** (2001). Gliding flight in ajackdaw: A wind tunnel study. *Journal of Experimental Biology*, **204**(2), 1153–1166.

Patel, C. K. (2007). *Energy extraction from atmospheric turbulence to improve fight vehicle performance*. Ph.D. thesis, Stanford University.

Pennycuick, C. J. (1982). Thermal soaring compared in three dissimilar tropical bird species, fregata magnificens, pelecanus occidentals and coragyps atratus. *Journal of Experimental Biology*, **102**(1), 307–325.

Pennycuick, C. J. (2005). Global circumnavigations: Tracking year-round ranges of nonbreeding albatrosses. *Science*, **307**(5707), 249–250.

Pennycuick, C. J. (2008). Soaring behavior and performance of some east african birds, observed from a motor-glider. *IBIS, International Journal of Avian Science*, **114**.

Rao, A., **D. Benson**, and **C. Darby** (2010). Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudo spectral method. *ACM Transaction on Mathematical Software*, **1**(2).

Rao, A. and **M. Patterson** (2013). 1gpops-ii: A matlab software for solving multiple-phase optimalcontrol problems usinghpâĂŞadaptive gaussian quadraturecollocation methods and sparse nonlinear programming. *ACM Transaction on Mathematical Software*, **39**(3).

Rayleigh, L. (1883). The soaring of birds. *Nature (London)*, **27**(1), 534–535.

Sachs, G. (2004). Minimum shear wind strength required for dynamic soaring of albatrosses. *IBIS*.

Spaar, R. and **B. Bruderer** (1996). Soaring migration of steppe eagles aquila nipalensis in southern israel: Flight behaviour under various wind and thermal conditions. *Journal of Avian Biology*, **27**(4), 289–301.

Turcker, V. A. and **G. Parrott** (1970). Aerodynamics of gliding flight in a falcon and other birds. *Journal ofExperimentalBiology*, **52**(2), 345–367.

Wai-Fah, C. and **E. Lui**, *Handbook of Structural Engineering*. Boca Raton: CRC Press, 1997.

Weimerskirch, H., **T. Guionnet**, and **J. Martin**, Fast and fuel efficient, optimal use of wind by flying albatrosses. *In Royal Society of London B*. 2000.

Zhao, Y. J. (2004). Optimal patterns of glider dynamic soaring. *Optimal control application and methods*, **25**(2), 67–89.