TÉCNICO LISBOA

Trajectory generation for non-powered lift-enabled vehicles in planetary atmospheres

João Luis Pinto da Fonseca

Thesis to obtain the Master of Science Degree in

Technological Physics Engineering

Examination Committee

Chairperson: Prof. Carlos Renato de Almeida Matos Ferreira
Supervisor: Prof. Rui Manuel Agostinho Dilão
Co-supervisor: Prof. Ana Maria Ribeiro Ferreira Nunes
Members of the Committee: Prof. Luís Manuel Braga da Costa Campos
Prof. José Manuel Gutierrez Sá da Costa

December 2012

Acknowledgments

I have to give a special thanks to my supervisor, Rui Dilão, with whom the debates have always been incisive and kept me from trying to boil the ocean at all times. To Catarina, my wife, for her patience and support in this crazy endeavor of working in technical research at this stage of my life, and to my grandmother that, although passed away, is and will always be an incredible source of inspiration.

Abstract

Control and command of non-powered lift-enabled vehicles in planetary atmospheres.

We propose a new trajectory generation and dynamic control algorithm for directing a reentry vehicle coming from Space to a pre-assigned target point in the three-dimensional environment. The new algorithm runs iteratively by optimizing locally in time the instantaneous shortest distance to the target point, taking into account both structural and aerodynamic limitations of the glider. When the control and command algorithm takes over the flight control of the glider, the only condition needed to successfully arrive at the target point is to have enough energy, and a quick computation can determine if the target point is within the glider flight range.

Previous approaches have been based on pre-computed maneuvers with hoping across different trajectories. They may also rely on heavy computational methods based on shooting techniques to find admissible trajectories. Some of these approaches are limited to straight-line flight paths or assume an almost horizontal gliding approximation in three dimensional flights.

As a proof of concept, we have applied the new algorithm to the command and control of the atmospheric re-entry trajectory of the Space Shuttle during the Terminal Area Energy Management (TAEM) phase. Due to its simplicity, this algorithm is computationally fast and therefore adequate for onboard implementations in non-manned gliders.

Keywords: Atmospheric re-entry, TAEM, Space Shuttle, HAC, flight control of non-manned space vehicles.

Resumo

Controlo dinâmico de planadores em atmosferas planetárias

Nesta tese propomos um novo algoritmo dinâmico de controlo para guiar planadores até um alvo, em re-entradas atmosféricas vindas do Espaço num ambiente tridimensional. O novo algoritmo é iterativo, e a cada momento minimiza a distância ao alvo, tendo em conta tanto as limitações aerodinâmicas como as limitações estruturais do planador. A partir do momento em que o algoritmo toma o controlo do planador, a única condição necessária para que se possa atingir o ponto alvo é ter energia suficiente para tal; um calculo rápido permite determinar se o alvo pretendido está ou não dentro do alcance do aparelho.

As abordagens anteriores têm sido baseadas em manobras pré-programadas, com trocas entre diferentes trajectórias nominais. São baseadas num grande esforço computacional associado a métodos de "shooting" para encontrar trajectórias admissíveis. Algumas destas soluções estão limitadas a trajectórias em linha recta ou assumem voos praticamente horizontais.

Como prova de conceito, aplicámos o novo algoritmo à fase TAEM da rentrada atmosférica do Space Shuttle. Dada a simplicidade e rapidez computacional do algoritmo este pode ser implementado em sistemas automaticos de decisão a bordo de planadores sem piloto.

Palavras-chave: Reentrada atmosférica, TAEM, Space Shuttle, HAC, comando e controlo controlo de veículos não tripulados.

Table of Contents

List of Acronyms

HAC	Heading Alignment Circle
TAEM	Terminal Area Energy Management
RK	Runge-Kutta
CFD	Computational Fluid Dynamics

List of Figures

List of Tables

0 Overview

Returning from Space to Earth

Vehicles capable of performing atmospheric re-entries are typically organized into two main families:

Main Distinctive Charateristics	Ballistic Re-Entry	Lift-Enabled Re-Entry
Range	Tens of Kms	Hundreds of Kms
Flight Time	Minutes	Tens of Minutes
Accelerations	Up to 8-10 gjs	Up to 2-4 gjs
Flight Angle	Steep	Wide
Landing scheme	Parachutes, Rockets	Gliding (no fuel!)

Table 0: Main charateristicis of the two types of atmospheric entry.

In principle, lift-enabled vehicles are typically more versatile since they can perform missions both as a spacecraft outside the atmosphere and as an aircraft inside the atmosphere. They also allow for much more precise guidance in the re-entry phase and land gliding on their own, while the others need to be assisted with parachutes and rockets.

In particular, ballistic re-entries typically have the following limitations:

a) Landing with rockets is particularly demanding, since fuel during re-entry is a scarce good (the vast majority is used on the way up). Should the rockets be turned on too soon the vehicle will decelerate too soon and will again accelerate hitting the ground travelling too fast. Should the rockets be turned on too late the vehicle will not have enough time to decelerate and will again hit the ground travelling too fast;
b) Rocket-assisted landing typically needs to be complemented with parachutes. The downside is that this kind of landing is only adequate for areas with plane and smooth surfaces (such as oceans, tundra planes and deserts) located far away from populated areas and thus adding to the cost of the recovery mission.

Additional advantages of lift-enabled vehicles include extended range, lower decelerations and mild trajectory angles but, all these features come “at a price” in terms of structural limits:

a) During supersonic flights, a large heat inflow is caused by the air friction. Unlike ballistic vehicles that have blunt body shapes [18], lift-enabled vehicles need to have slender shapes to travel in the atmosphere and fuselage extremeties, such as the nose of the vehicle, are exposed to bigger heat fluxes forcing the usage of better heat insulation techniques. In addition, as flight times are longer than in ballistic entries the, lift.enabled vehicles are exposed to the heat flux for longer.
b) Lift-enabled vehicles have wings offering relatively large surfaces to the travelling air and that can only sustain a limited amount of pressure. Ballistic vehicles do not have wings and thus can sustain bigger amounts of mechanical pressure.

Among the lift-enabled vehicles, the most successful so far was the Space Shuttle and will be the base of our simulations. The Space Shuttle was originally designed as a multi-purpose ship capable of transporting passengers and cargo, and was widely used to deliver cargo to the MIR station, the Hubble telescope and to the International Space Station (ISS).

The re-entry phase of the Space Shuttle had three main sections faced in terms of descent heading:

a) Atmospheric re-entry: Typical altitudes of 120-40 km, far above 6 Mach.
b) Gliding to the landing site: Typical altitudes of 40-3 km, at .2 to 6 Mach.
c) Final approach and landing: Typical altitudes of 3-0 km, below .2 Mach.

The gliding phase is also refered to as TAEM (Terminal Area Energy Management). All simulations will be limited to the TAEM area since we did not have access to aerodynamic wind tunnel data (or CFD) above 5 Mach. The target point to reach will be the HAC (Heading Alignment Center) where the TAEM phase ends.

Key assumptions made when deriving the motion equations

We consider a flat non-moving earth in our simulations, which is a very good approximation in the TAEM area as the radius of the Earth (6.4 million kilometers) is much bigger than the altitude considered. Ignoring wind, the atmosphere can also be considered stationary in relation to the planet. The US 1976 standard atmosphere was used to model the atmospheric profiles, which cannot be assumed to be exponential since that approach would imply a constant atmospheric temperature (and thus constant speed of sound across the entire atmosphere).

Further assumptions include assuming the glider is a mass-point with Lift and Drag forces dependent on the angle of attack (angle between the velocity the airplane and the fuselage) and the Mach number (ratio of the relative speed to the atmosphere and the speed of sound). Other effects such as side-slip, gyroscopic factors are discarded in our approach [21]. Since the Space Shuttle, except for emergencies for which minimal fractions of the fuel available before launch is kept, travels with no thrust we have also assumed constant mass during the re-entry.

Using spherical coordinates for the velocity, the equations of motion are simpler and limited to a first differential order. Finally, in order to be able to perform simulations with the algorithm, we have applied it to the specific case of the Space Shuttle taking into account its aerodynamic capabilities and its structural limits.

Key dynamics of the equations of motion

The derived equations of motion have one fixed point, or one limit cycle, that depends on:

a) Atmospheric parameters: gravity acceleration, air density, speed of sound (air temperature);
b) Non in-flight controllable vehicle parameters: wing effective surface and total mass;
c) Flight controllable vehicle parameters: attack and bank angles.

The fixed point is always stable (negative real part of the eigenvalues of the matrix of the linear approximation at the fixed point) for all combinations of parameters, but allow for two different regimes of convergence for different combinations of speed and attack angles ("rolled convergence" to the fixed point or a straight-line convergence to the fixed point).

In order to control the vehicle, we will limit the decisions of the algorithm to two controls:

a) Attack angle that can be managed to increase or decrease the angle of descent;
b) Bank angle that can be managed to curve to the side.

Attack angle command and control

The equations of montion are affected by the attack angle have an inherent equilibrium dynamics between the lift and the drag forces, with different attack angles producing different descent angles. Leveraging on this feature, the algorithm developed here minimizes the distance to the target point (HAC where TAEM phase finishes and landing procedures are initiated) by finding the attack angle that will force the Space Shuttle to cover that distance in a straight-line while in equilibrium.

Given that the biggest structural constraint in the Shuttle descent is the heat flux, and the fact that equilibrium speeds are smaller on the right side of the L/D curve (ratio between drag and lift forces at a certain moment) we have built the algorithm to search for the correct L/D ratio only within the angle that delivers the maximum range (called "max glide angle" in our paper) and the angle at which lift peaks before starting to decay (called "stall angle" in our paper).

Three main situations are faced by the algorithm:

a) The needed descent angle is shallower than the Space Shuttle can deliver in equilibrium. In these cases the algorithm imposes the attack angle that delivers the highest range when travelling at equilibrium state, the maximum glide attack angle;
b) The needed descent angle is steeper than the Space Shuttle can deliver in equilibrium. Attack angles beyond stall are not stable and in these cases the algorithm imposes the stall angle.
c) The needed descent angle is within the window defined by the maximum glide and stall attack angles. In these situations, the algorithm numerically searches for the angle that delivers that descent angle when travelling at equilibrium state.

Given that drag grows faster with speed than lift does (lift itself induces drag), a certain L/D ratio may not be available for any attack angle at the current speed, but may be available later at lower speeds. To model the aerodynamic capabilities, we have used available wind tunnel data [9], covering angles of attack up to 45-50 degrees and Mach numbers up to 5.

The defined attack angle, arising from the heading or navigational decisions described above, is then validated not to infringe on any structural limits related to the fuselage of the Space Shuttle:

a) The heat flux at the nose, the surface with the smallest radius of curvature, cannot exceed a maximum value and in practice this may impose a minimum angle of attack overwriting the previous decision
b) The load factor at the wings, the biggest surface, cannot exceed a maximum value and in practice this may impose a maximum angle of attack overwriting the previous decision
c) The maximum acceleration recommended. Accelerations should be mild in order to ensure the safety and commodity of all occupants and cargo travelling inside the Space Shuttle. This variable cannot be controlled directly and is a direct consequence of the decisions made by our algorithm.

Bank angle command and control

The impact of the bank angle on the motion equations is direct, not an equilibrium dynamics such as the one generated by the attack angle. Leveraging on this behavior, the algorithm will not only manage the heading on the xy plane, but it will also have two additional features designed to "assist" the attack angle algorithm.

In order to determine the heading bank angle, the algorithm takes into account:

a) The magnitude of the current misalignment between the current velocity and the target measured by the dot product;
b) The direction of the current misalignment between the current velocity and the target measured by the exterior product.

In addition, two additional bank angle controls are computed in order to assist the bank angle decisions if needed:

a) Anti-stall bank angle control, computed when the stall angle attack is achieved, is the bank angle needed to ensure a steeper angle of descent asked by the bank angle heading;
b) The energy bank angle control, computed when travelling at speeds far above equilibrium, is the bank angle needed to ensure that the equilibrium speed matches the current speed avoiding the Space Shuttle from further gaining altitude.

The algorithm will then choose the highest of the three computed bank angles, thus giving priority to the most urgent request.

Simulations and results

We have used typical initial conditions for TAEM phase that we refer to as "reference conditions" [17]. These conditions were applied for controlled descents from 30,000 meters to 3,000 meters of altitude and four types of analysis were made, yielding the following conditions:

a) Range and error of the algorithm

Range of the Space Shuttle under the algorithm control
i) Hundreds of kilometers of range in any direction (highest for straight flights);
ii) Symmetric ranges for symmetric alignments with initial velocity;
iii) Different ranges for different alignments with initial velocity.

Error reaching specific targets at 3,000 meters
i) Typical error of the order of magnitude of 100 meters or below;
ii) Confirmation that any point inside MR is achievable (small error);
iii) Angular symmetry of the error distribution follows the angular symmetry of the range.

b) Typical trajectories generated by the algorithm

i) Long range trajectory, when the flight is made mostly in straight line (typical Shuttle strategy);
ii) Short range trajectory, when the target is "too close" to the xy origin the algorithm initiates a whirlpool approach while the altitude "slowly" decreases;
iii) Excessive energy trajectory, when the Space Shuttle is handled to the algorithm at speeds far above the equilibrium speed and that can only be controlled using the dynamic S-turns generated by the bank energy control feature.

c) Sensitivity analysis to initial conditions and control time interval

i) The simulations done for control time intervals between 0.1 and 30 seconds have proved that the algorithm is self-correcting by nature and that it can reach the target with small errors even for long control time intervals;
ii) The sensitivity analysis done on the initial energy conditions showed that in order to have range, it is crucial that the initial speed be high enough (typically not a problem in the TAEM are as the Space Shuttle is typically inherited above the equilibrium speed);
iii) The sensitivity analysis done on the orientation of the initial speed showed that the only sensitivity parameter is the descent angle as, in order to have range, the Space Shuttle needs to be inherited with a shallow negative descent angle (or even slightly positive).

d) Structural limits check on excessive speed entries

The algorithm was able to successfully deal with the heat flux and load structural limits while delivering smooth passes with moderate decelerations.

1 Trajectory Generation and Dynamic Control of Unpowered Vehicles Returning from Space

1.1 Introduction

While returning from space, spacecrafts travel at extreme conditions of speed and acceleration that typically do not allow for a "man-in-the loop" approach, forcing, at least partially, automation of flight controls. Thus, automated guidance and control systems are a critical component for any re-usable space flight vehicle.

For the Space Shuttle a typical return flight, from space to Earth, has three main phases:

1) The atmospheric re-entry phase, where the transition from spacecraft to aircraft flight mode occurs. Typical altitudes for this phase are in the range $120 - 40$ km.
2) The gliding phase, usually referred as the Terminal Area Energy Management (TAEM) phase, occurring in the altitude range $40 - 3$ km.
3) The final approach and landing phase, occurring in the altitude range $3 - 0$ km.

While in the atmosphere re-entry phase, the biggest priority is to ensure that the structural constraints of the vehicle are not exceeded; during the TAEM phase, the biggest priority is to ensure that the vehicle reaches the Heading Alignment Circle (HAC) where preparation for landing is initiated.

On a typical Space Shuttle mission, the TAEM phase begins at the altitude of $25,000 - 40,000$ m at a speed in the range $2 - 6$ M (Mach), and finishes at the HAC at the altitude of $1,500 - 3,000$ m, with a speed of the order of 0.20 M.

Previous approaches have been based on pre-computed manoeuvres [11] with hoping across different trajectories [13], [15], [6] and [1]. These approaches rely on heavy computational methods based on shooting techniques to find admissible trajectories [4]. Some are limited to straight-line flight paths or assume an almost horizontal gliding approximation for three dimensional flights [7].

In this paper, we propose a new dynamic control algorithm in order to redirect the trajectory of gliders to a pre-assigned target point. This algorithm runs iteratively enabling a self-correcting approach to the target and is applicable to any unpowered lift-enabled vehicle (glider) travelling in planetary atmospheres. As a proof of concept, we have applied the algorithm to the command and control of the Space Shuttle gliding during the TAEM phase on the Earth atmosphere, where the target point is the centre of the HAC.

This paper is organised as follows. In section 1.2, we present the equations of motion of a glider and we discuss the approximations we use to define the controllability conditions. In section 1.3, we briefly discuss the phenomenology of aircraft gliding motion, instrumental for the design of a dynamic control strategy. In section 1.4, we discuss the structural limits of the Space Shuttle. In section 1.5, we derive the dynamic control algorithm, and in section 1.6 we present realistic simulations for the Space Shuttle guidance and control during the TAEM phase. Finally, in section 1.7 we discuss the main conclusions of the paper.

1.2 Gliding motion

We consider that aircraft gliding motion in a planetary atmosphere is well described by a point mass vehicle model under the influence of a gravity field, [5], [8], [12] and [3]. In this case, the equations of motion of a gliding aircraft (no thrust forces) are (Appendix A),

$$\begin{cases} m\dot{V} = -mg(z)\sin\gamma - D(\alpha, Ma) \\ mV\dot{\gamma} = -mg(z)\cos\gamma + L(\alpha, Ma)\cos\mu \\ mV\dot{\chi}\cos\gamma = L\sin\mu \end{cases}, \quad \begin{cases} \dot{x} = V\cos\chi\cos\gamma \\ \dot{y} = V\sin\chi\cos\gamma \\ \dot{z} = V\sin\gamma \end{cases} \tag{1}$$

where m is the aircraft mass, $V = \sqrt{V_x^2 + V_y^2 + V_z^2}$ is the aircraft speed, γ is the flight path angle as defined in figures 1 and 2, μ is the bank angle as defined in figure 2c), $D(\alpha, Ma)$ and $L(\alpha, Ma)$ are the drag and lift forces induced by the atmosphere, α is the angle of attack and Ma is the Mach number. In general, the Mach number Ma is a function of V and z. The function $g(z) = g_0(R_E/(R_E + z))^2$ is the gravity acceleration, where $g_0 = 9.80665$ m/s^2 is the Earth standard gravitational acceleration constant and $R_E = 6.371 \times 10^6$ m is the Earth (or planetary) mean radius.

In the local reference frame of the centre of mass of the aircraft, figure 1, $V \in]0, \infty[$, $\gamma \in]-\pi/2, \pi/2[$, $\chi \in [0, 2\pi[$, $\mu \in]-\pi/2, \pi/2[$ and $\alpha \in]-\pi/2, \pi/2[$. As usual, $(x, y, z) \in \mathbb{R}^3$ and $(\dot{x}, \dot{y}, \dot{z}) \in \mathbb{R}^3$.

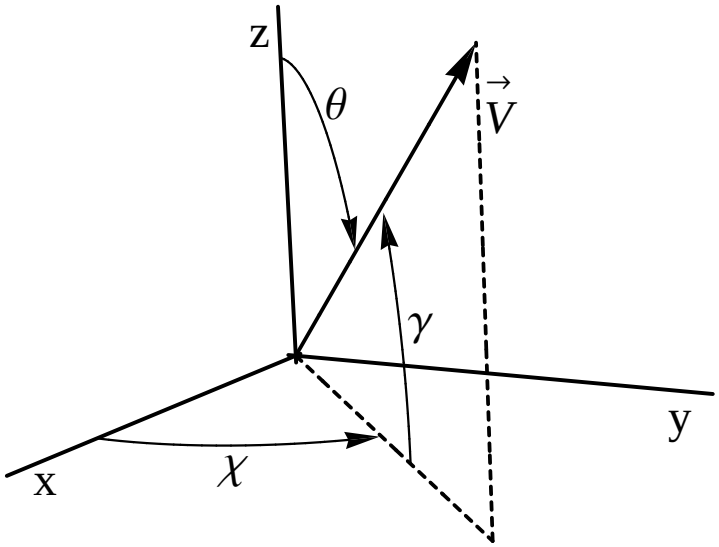


Fig. 1: Local coordinate system for the point mass glider model (1). The origin of coordinates is located at the centre of mass of the aircraft and **V** is the aircraft velocity, not necessarily aligned with the longitudinal reference line of the glider.

In figure 2a)-b), we show the angle of attack α defined as the angle between the longitudinal reference line of the aircraft and the vector velocity of the aircraft. While in most aircrafts attack angles are always smaller than 15^o, the Space Shuttle is capable of attack angles up to 45^o, [12] and [10]. The bank angle, as shown in figure 2c), defines the inclination of the aircraft relative to the (x, y) plane.

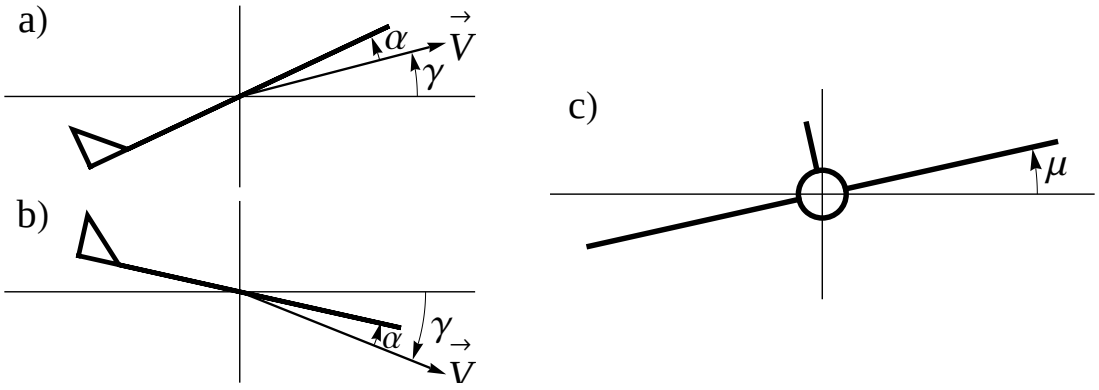


Fig. 2: In a) and b), we show the flight path angle γ and the angle of attack α of an aircraft. In c) we show the bank angle μ. The flight pass angle γ depends on the angle of attack, on the aerodynamic coefficients of the aircraft and on the Mach number. The control of a glider is done by the manipulation of the angle of attack and of the bank angle.

The drag and lift forces in the system of equations (1) are given by,

$$\begin{aligned} D(\alpha, Ma) &= \bar{q}SC_D(\alpha, Ma) = \tfrac{1}{2}\rho(z)V^2SC_D(\alpha, Ma) \\ L(\alpha, Ma) &= \bar{q}SC_L(\alpha, Ma) = \tfrac{1}{2}\rho(z)V^2SC_L(\alpha, Ma) \end{aligned} \tag{2}$$

where $\bar{q} = \rho(z)V^2/2$ is the dynamic pressure, S is the wing area of the aircraft, $\rho(z)$ is the atmosphere density as a function of altitude (Appendix B) and Ma is the Mach number. For each specific aircraft, the functions $C_D(\alpha, Ma)$ and $C_L(\alpha, Ma)$ are determined with wind tunnel experiments or CFD codes, which explicitly takes into account turbulent effects described by the Reynolds number [10].

The Mach number is defined by $Ma = V/V_{sound}$ where the sound speed is calculated with,

$$V_{sound} = \sqrt{\gamma_d R_s T(z)} \tag{3}$$

where $\gamma_d = 1.4$ is the diatomic non-ionised gas constant, $R_s = 287.04$ J/(K kg) is the ideal gas constant, and $T(z)$ is the temperature profile of the Earth atmosphere, given in table 2 of Appendix B.

Introducing expressions (2) into equations (1), we obtain the final form for the equations of motion of a glider,

$$\begin{cases} \dot{V} = -g(z)\sin\gamma - \left(\dfrac{1}{2m}\rho(z)SC_D(\alpha, Ma)\right)V^2 \\ \dot{\gamma} = -\dfrac{g(z)}{V}\cos\gamma + \left(\dfrac{1}{2m}\rho(z)SC_L(\alpha, Ma)\right)V\cos\mu \\ \dot{\chi} = \left(\dfrac{1}{2m}\rho SC_L(\alpha, Ma)\right)V\dfrac{\sin\mu}{\cos\gamma} \end{cases}, \begin{cases} \dot{x} = V\cos\chi\cos\gamma \\ \dot{y} = V\sin\chi\cos\gamma \\ \dot{z} = V\sin\gamma\,. \end{cases} \tag{4}$$

where $\rho(z)$ is given in the Appendix B.

The three dimensional phase space with coordinates (V, γ, χ) will be called the reduced phase space of the system of equation (4). We always consider that ρ, g and Ma are slowly varying functions of z, enabling the description of the aircraft local motion in a three dimensional phase space. Otherwise, the full description of equation (4) is in a six dimensional phase space.

When a glider is falling under a gravity field it converges to a steady state motion with a constant velocity and constant flight path angle given by,

$$\begin{aligned} V^* &= \sqrt{\frac{2mg}{\rho S}}\frac{1}{(C_D^2 + C_L^2\cos^2\mu)^{1/4}} \\ \gamma^* &= -\arctan\frac{C_D}{C_L\cos\mu} \\ \chi(t) &= \left(\sqrt{\frac{g\rho S}{2m}}(C_D^2 + C_L^2\cos^2\mu)^{1/4}\tan\mu\right)t + \chi(0) \mod(2\pi) \end{aligned} \tag{5}$$

where $\chi(0)$ is the initial longitudinal angle, $C_L \equiv C_L(\alpha, Ma)$, $C_D \equiv C_D(\alpha, Ma)$, $Ma \equiv Ma(V, V_{sound}(z))$, $g \equiv g(z)$ and $\rho \equiv \rho(z)$. If $\mu = 0$, the solution (5) is a fixed point in the three dimensional reduced phase space with coordinates (V, γ, χ). If $\mu \neq 0$, the solution (5) is a closed orbit or limit cycle in the three dimensional reduced phase space (V, γ, χ).

We analyse now the qualitative structure of the vector field of the three dimensional reduced phase space of equation (4). Linearizing the system of equations (4) in the neighbourhood of the steady solution (5), we obtain,

$$\begin{bmatrix} \dot{V} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = \begin{bmatrix} \frac{-C_D(\alpha,Ma)S\rho(z)}{m}V^* & \frac{-C_L(\alpha,Ma)S\rho(z)}{2m}V^{*2}\cos\mu & 0 \\ \frac{C_L(\alpha,Ma)S\rho(z)}{m}\cos\mu & \frac{-C_D(\alpha,Ma)S\rho(z)}{2m}V^* & 0 \\ \frac{g(z)}{V^{*2}}\tan\mu & \frac{-g(z)C_D(\alpha,Ma)\sin\mu}{V^*C_L(\alpha,Ma)\cos\mu^2} & 0 \end{bmatrix} \begin{bmatrix} V \\ \gamma \\ \chi \end{bmatrix} \tag{6}$$

and the matrix in (6) has eigenvalues,

$$\begin{aligned} \lambda_{1,2} &= \frac{S\rho(z)}{4m}V^*\left(-3C_D(\alpha, Ma) \pm \sqrt{C_D(\alpha, Ma)^2 - 4C_L(\alpha, Ma)^2(1 + \cos 2\mu)}\right) \\ \lambda_3 &= 0\,. \end{aligned} \tag{7}$$

The convergence of the glider trajectory to the steady motion is determined by the eigenvalues $\lambda_{1,2}$. If $\lambda_{1,2}$ are real, then $\lambda_{1,2} < 0$, and the convergence to the steady state motion is monotonic (in the (V, γ) subspace,

the fixed point is a stable node). If $\lambda_{1,2}$ are complex, then $\text{Real}(\lambda_{1,2}) < 0$, and the convergence to the steady state motion is oscillatory (in the (V, γ) subspace, the fixed point is a stable focus).

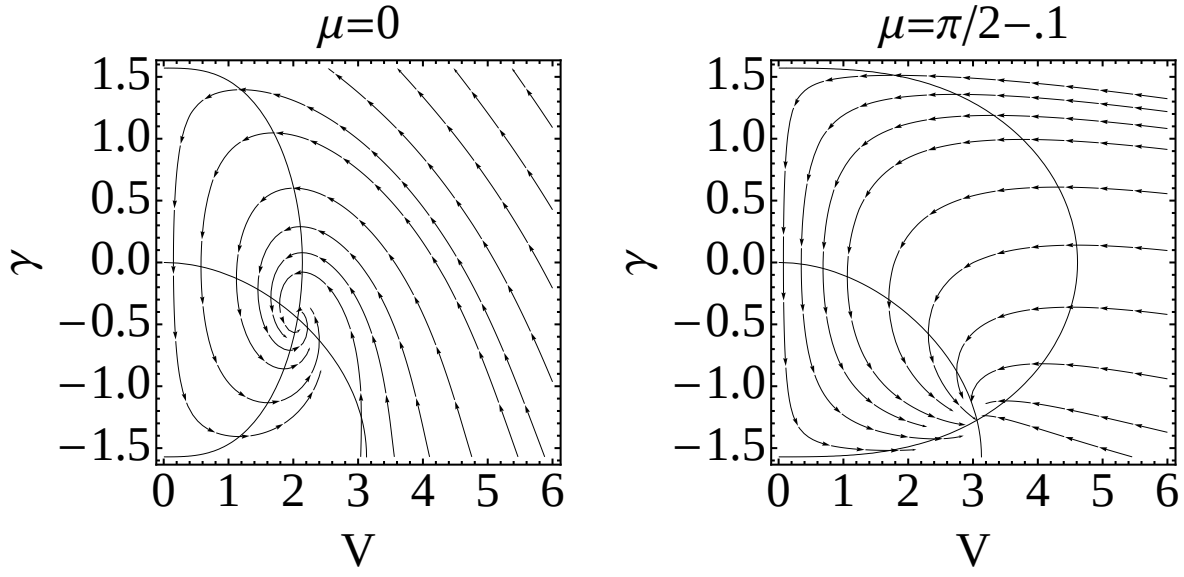


Fig. 3: Orbits in the reduced phase space of equation (4), for every χ = constant and for two values of the bank angle μ. We show also the V and γ nullclines of equation (4). The pictures have been obtained with the choice of parameters, $\rho(z)SC_D(\alpha, Ma)/(2m) = \rho(z)SC_L(\alpha, Ma)/(2m) = 1$, and $g = 9.8$. For $\mu = 0$, the eigenvalues $\lambda_{1,2}$ are complex and the fixed point is a stable focus. For $\mu = \pi/2 - 0.1$, the eigenvalues $\lambda_{1,2}$ are real and the fixed point is a stable node.

In figure 3, we show the reduced phase space orbits of equation (4) near the point (V^*, γ^*), for χ = constant. The bank angle μ, not only affects the position of the fixed point in phase space, but it also impacts the regime of convergence to the fixed point.

1.3 Phenomenology of Space Shuttle gliding motion

Using wind tunnel data, we have done the fits for the aerodynamic drag and lift coefficients C_D and C_L of the Space Shuttle, [9]. Our numerical tests have shown that C_D and C_L are well described by the parameterisions,

$$\begin{aligned} C_L(\alpha, Ma) &= (a_1 + a_2\alpha + a_3\alpha^2)K(Ma)^{b_1+\alpha b_2} \\ C_D(\alpha, Ma) &= (0.01 + f_1 Ma^{f_2} + d_3\alpha^2)K(Ma)^{e_1+\alpha e_2} \end{aligned} \tag{8}$$

where,

$$K(Ma) = \frac{1}{2}\left(1 + \sqrt{\left|1 - \left(\frac{Ma}{M_c}\right)^2\right|}\right) \tag{9}$$

is a simplification of the von Kármán function expanded to supersonic regimes, [12]. In table 1, we show the parameter estimation with wind tunnel data of expressions (8) and (9) for the Space Shuttle.

In figure 4a), we show the lift and drag coefficients C_L and C_D as a function of α, for several values of the Mach number. In figure 4b), we show the behaviour of the ratio L/D, as a function of the angle of attack α. As the Space Shuttle is a glider, it can only move across its $L/D = C_L/C_D$ curve. All the L/D curves intersect at the no-lift angle α_{nL}. The no-lift angle α_{nL} is the angle for which L/D is zero due the absence of

Parameter	Estimated	Standard error	t-statistics	p-value
a_1	-0.053	0.009	-6.15	9.8×10^{-8}
a_2	2.73	0.06	43.0	1.8×10^{-43}
a_3	-1.55	0.09	-18.0	2.0×10^{-24}
b_1	-1.01	0.09	-11.3	7.4×10^{-16}
b_2	1.1	0.1	8.7	7.6×10^{-12}
d_3	1.79	0.02	99.0	1.1×10^{-63}
e_1	-1.4	0.1	-12.6	1.2×10^{-17}
e_2	1.5	0.1	11.3	5.8×10^{-16}
f_1	0.028	0.004	6.46	2.9×10^{-8}
f_2	1.4	0.2	8.57	1.0×10^{-11}
M_c	1.25	0.03	49.6	1.0×10^{-46}

Table 1: Parameters of the aerodynamic drag and lift coefficients (8) for the Space Shuttle, estimated from wind tunnel data. The significance of the fits have been determined with a chi-squared test. The large values of the absolute value of the t-statistics measures the likelihood of the parameters in the fits. The low values of the p-values mean that the fits are highly significant and the probability of finding a value outside the fitted ones are in the range $10^{-8} - 10^{-63}$.

lift force, and is independent of the speed. The max-glide angle α_{maxgl} is the angle that maximises the ratio L/D, and is dependent on the Mach number. The stall angle α_{stall} is the angle at which lift peaks before beginning to decrease.

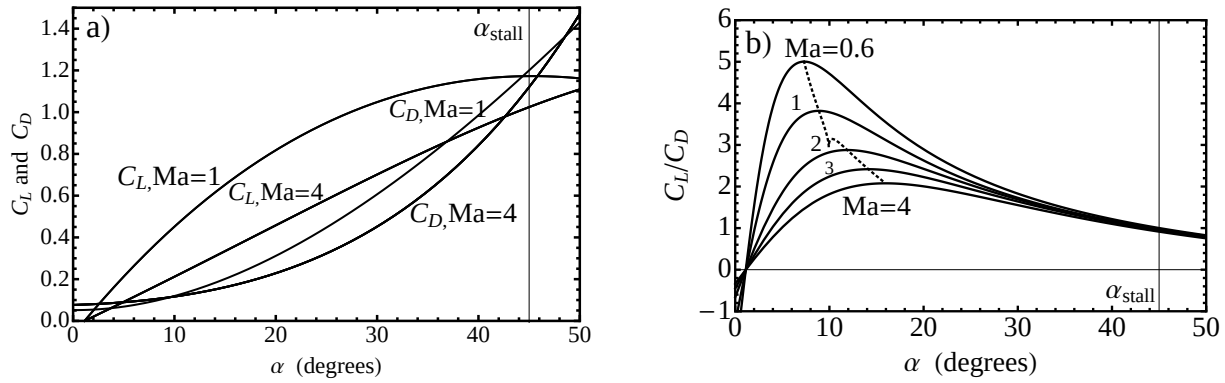


Fig. 4: a) Lift and drag coefficients C_L and C_D for the Space Shuttle as a function of α, for several values of the Mach number. These coefficients have been calculated from (8)-(9) and the parameters in table 1. b) Ratio L/D, as a function of the angle of attach α, for several values of the Mach number. The no-lift parameter $\alpha_{nL} = 1.5^o$ and the stall angle $\alpha_{stall} = 45^o$, are independent of the Mach number. The max-glide angle α_{maxgl} is given by (10) and is marked in b) by the dotted line. For higher speeds, the C_L/C_D curve will become increasingly flat and the max-glide angle α_{maxgl} will move further to the right reaching saturation.

With the functions (8)-(9), we have approximated the max-glide angle α_{maxgl} as a function of Mach number, obtaining,

$$\alpha_{maxgl} = \begin{cases} 0.0906 + 0.0573Ma + 0.0071Ma^2 & (Ma \leq 1.25) \\ 0.1070 + 0.0577Ma - 0.0037Ma^2 & (1.25 < Ma < 5) \end{cases} \tag{10}$$

determined with the correlation coefficient $r^2 = 0.999$.

Using the aerodynamic coefficients of the Space Shuttle it is possible to map the different regimes of convergence to the steady motion as a function of the attack angle and the speed of the glider, measured in Mach units. These regimes depend on the eigenvalues $\lambda_{1,2}$, (7). In figure 5, we show the different convergence

regimes for the Space Shuttle as a function of the attack angle and the Mach number. For large enough Mach numbers, the convergence to the steady motion is non oscillatory, independently of the attack angle α.

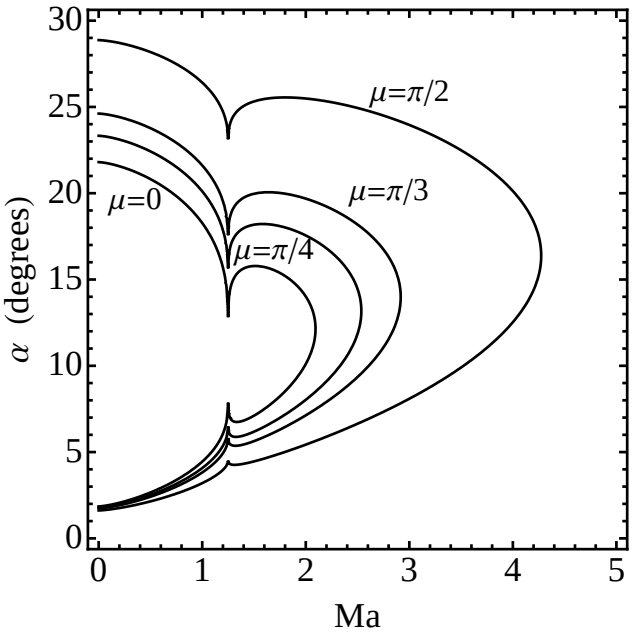


Fig. 5: Different convergence regimes for the Space Shuttle as a function of the attack angle α and of the Mach number Ma, for different values of μ. Trajectories with parameters in the interior of the corresponding μ-curve have oscillatory convergence to the steady state regime.

1.4 Structural limits of the Space Shuttle

The Space Shuttle, like any other machine, has structural limits that once exceeded may cause the destruction of the vehicle or seriously compromise its reliability for the future. In order to evaluate the effect of changing the α and μ commands, we have considered three main limitations: 1) heat flux limit at the Space Shuttle nose; 2) load factor limit at the Space Shuttle wings; and 3) acceleration limit. The heat flux limit imposes a minimum attack angle while the load factor limit imposes a maximum attack angle. The acceleration limit is related with the maximum acceleration that humans, cargo and electronics can sustain.

1.4.1 Heat flux limit

The Space Shuttle nose is the surface with the lowest curvature radius, being therefore the limiting factor for the heat flux. With the hypothesis that the heat flux on the nose of the Space Shuttle is the same as the convective heat flux Φ in a laminar flow at a stagnation point ($V = 0$), the heat flux at the Space Shuttle nose is given by, [3],

$$\Phi_{nose} = \Phi \cos\alpha = c_q \sqrt{\frac{\rho(z)}{R_N}} V^3 \cos\alpha \leq \Phi_{max} \tag{11}$$

where R_N is the radius of curvature of the Space Shuttle wall, $\rho(z)$ is the density of the atmosphere, V is the speed of the glider, α is the attack angle, and $c_q = 1.83 \times 10^{-4}$ Kg$^{1/2}$ m^{-1} is the heat conductivity for the Earth atmosphere. Typical values for the Space Shuttle are: $R_N = 1$ m and $\Phi_{max} = 5 \times 10^5$ J/(m^2 s). The

approximation (11) is realistic in the supersonic regime, where the heat flow to the glider fuselage is critical (typically above 2 Mach). In this critical region, the flow can be considered almost laminar. Heat radiation effects were not taken into account.

Also of interest is the air temperature near the Space Shuttle surface, also called stagnation temperature. For compressible flows, [12], we have $T_{stag} = T_{envi}(1 + Ma^2(\gamma_d(T_{stag}) - 1)/2)$, where T_{envi} is the ambient temperature, $\gamma_d(T_{stag}) = \gamma_{d0} + \gamma_{d1}T_{stag}$ is the temperature dependent diatomic constant, where $\gamma_{d0} = 1.42$ and $\gamma_{d1} = -7 \times 10^{-5}$ K^{-1}. Solving the first temperature equation in order to T_{stag}, we obtain,

$$T_{stag} = \frac{T_{envi}(2 + (\gamma_{d0} - 1)Ma^2)}{2 - T_{envi}\gamma_{d1}Ma^2} \tag{12}$$

where the dependency of the air adiabatic indexes γ_{d0} and γ_{d1} with extreme temperatures has been taken into account.

1.4.2 Load factor limit

In the Space Shuttle flight, the lift and the drag forces have a projection orthogonal to the wings, inducing a load on the wings of the glider. For convenience, the load factor on the wings (n_{load}) is measured as a non dimensional weight multiple,

$$n_{load} = \frac{1}{2}\rho(z)V^2S\frac{1}{mg(z)}\left(C_L(\alpha, Ma)\cos\alpha + C_D(\alpha, Ma)\sin\alpha\right) \leq n_{maxload}\,. \tag{13}$$

Typically, the maximum load factor for the Space Shuttle wings is around 4 to 5 times its weight. As we can see from figure 4, both the lift and drag coefficients increase with the attack angle and thus the angle with biggest wing load is the stall angle.

1.4.3 Acceleration limit

The acceleration limit is monitored during the flight and is measured as multiples of the gravity acceleration at sea level,

$$|N_g| = \left|\frac{\dot{V}}{g(z=0)}\right| \leq g_{lim}\,. \tag{14}$$

Typically, for lift enabled re-entry vehicles, $g_{lim} = 3$.

1.5 Dynamic trajectory control of gliders

On the high atmosphere where the air density is extremely low, a glider is not always in an equilibrium state, but, given enough time, it naturally converges to the equilibrium, [8]. As, by (7), $\lambda_{1,2} \simeq \rho(z)$ and since the air density increases as altitude decreases, the convergence time to the equilibrium state decreases as the glider approaches target points located at lower altitudes. As a result, the dynamic control algorithm presented here will take advantage of this behaviour by determining, at specific instants of time, the equilibrium conditions needed for the glider to reach selected target points at lower altitudes.

Target points will be delimited by a "safety sphere" or handover sphere, characterised by the radius r_{target}.

To define the command and control problem, we consider the initial condition,

$$(x_0, y_0, z_0, V_0, \gamma_0, \chi_0)$$

that, without loss of generality, corresponds to time $t = 0$. Let,

$$(x_f, y_f, z_f)$$

be the space coordinates of the target point. We consider that the target point is only defined by its spatial coordinates, and the direction of the velocity vector are arbitrary.

The intermediate coordinates of the glider path are,

$$(x_i, y_i, z_i, V_i, \gamma_i, \chi_i)$$

where $i = 0, 1, \ldots, f$. These intermediate coordinates are measured at times $t = iT_{con}$, where T_{con} is the control time interval. Commands and controls are checked at time intervals T_{con}.

In the configuration space (x, y, z), we define the direction vector from the actual position of the glider to the target point as,

$$\mathbf{P}_i = (x_f - x_i, y_f - y_i, z_f - z_i)\,. \tag{15}$$

In order to direct the aircraft to the target, we control the attack and bank angles separately. As we have seen in the previous sections 1.3 and 1.4, the attack angle is not only critical for navigation, but it is also closely tied with the physical limits of operation. In that sense, a trajectory alternating between sharp climbs and dives will typically push the Space Shuttle beyond its structural limits and it is therefore imperative to have trajectories as smooth as possible. Therefore, in our control algorithm, we first analyse the attack angle, subject to the structural limits if applicable. Only after, we compute the bank angle.

At step i of the dynamic control process, the initial conditions are $(x_i, y_i, z_i, V_i, \gamma_i, \chi_i)$, and the angle of attack and the bank angle are α_i and μ_i, respectively. Then, we calculate the new values of the glider control parameters α_{i+1} and μ_{i+1} by the procedures described below. With these new values for α and μ, the new trajectory of the aircraft is computed by equations (4) and the glider will follow the new trajectory during the control time interval T_{con}. From the numerical point of view, we use a fourth order Runge-Kutta integration method with integration time step T_{int}, where $nT_{int} = T_{con}$ and n is a positive integer.

The dynamic algorithm is divided into six stages:

1) **Attack angle heading control**: we analyse the instantaneous direction between the glider instantaneous position and the target point in the three-dimensional space (x, y, z), and we select the adequate angle of attack α in order to guide the glider vertically to the target point.
2) **Attack angle heat flux control**: we impose a minimum limit on the attack angle to meet the maximum heat flux at the Space Shuttle nose.
3) **Attack angle load factor control**: we impose a maximum limit on the attack angle to meet the maximum load limit on the Space Shuttle wings.
4) **Bank angle heading control**: we analyse the horizontal direction of flight, in the (x, y) plane, and we adjust the bank angle μ in order to guide the glider horizontally to the target point.
5) **Bank angle anti-stall control**: we compute the bank angle needed to move the glider out of stall in the approach to the target, eliminating the extra lift not need to reach the target in a straight line.
6) **Bank angle energy control**: we compute the bank angle needed to initiate a dynamic S-turn that will prevent the glider from gaining altitude.

With the command and control algorithm just enumerated, we iterate these procedures from the initial point to the final target point. The algorithm stops with success when the glider is inside the handover sphere and the altitude of the glider is below z_f. If the glider crosses the altitude z_f and is outside the handover region, then it fails the target.

We describe now each control individually.

Attack angle heading control

The attack angle heading control was designed so that the vehicle is always re-orienting vertically to the target through a straight line path. The tangent of the angle between the $x - y$ projection and the z component of the direction vector $\mathbf{P}_i$ to the target point is computed at each iteration, and we obtain,

$$G_{i+1} = \frac{(z_f - z_i)}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}}$$

where (x_i, y_i, z_i) is the position of the glider at time $t = iT_{con}$. At this position, the glider has flight path γ_i. Then, to direct the motion of the glider to the target with a steady flight path, by (5), we must have,

$$G_{i+1} = (\tan\gamma_{i+1} =) - \frac{C_D(\alpha_{i+1}, Ma_i)}{C_L(\alpha_{i+1}, Ma_i)\cos\mu_{i+1}}\,. \tag{16}$$

With the calculated value of G_{i+1}, we solve equation (16) in order to the ratio C_L/C_D, and we obtain the solution c_{i+1} ($= C_L/C_D$), assuming a null bank angle $\mu_{i+1} = 0$. This assumption, disentangles the bank angle from the attack angle. The C_L/C_D curve as a function of α and Mach number Ma is given by (8) and (9), and α_{maxgl} is calculated from (10) and (3). Then, we have:

a) If c_{i+1} is bigger than $C_L(\alpha_{maxgl}, Ma_i)/C_D(\alpha_{maxgl}, Ma_i)$, the max-glide attack angle will be selected, $\alpha_{i+1} = \alpha_{maxgl}$.
b) If c_{i+1} is smaller than $C_L(\alpha_{stall}, Ma_i)/C_D(\alpha_{stall}, Ma_i)$, the stall angle will be selected, $\alpha_{i+1} = \alpha_{stall}$.
c) Otherwise, the attack angle α_{i+1} is computed by numerically solving the resulting equation

$$C_L(\alpha_{i+1}, Ma_i)/C_D(\alpha_{i+1}, Ma_i) = c_{i+1}.$$

At this stage, a new attack angle α_{i+1} has been chosen.

Attack angle heat flux control

We check if the heat flux constraint (11) is violated at the Space Shuttle nose by verifying if the inequality holds,

$$c_q\sqrt{\frac{\rho(z_i)}{R_N}}V_i^3\cos(\alpha_{i+1}) \le \Phi_{max}\,. \tag{17}$$

If inequality (17) does not hold, a new angle of attack is calculated, regardless the output from the attack heading control. The new attack angle is,

$$\alpha_{i+1}^{heat} = \arccos\left(\Phi_{max}\frac{1}{c_q V_i^3}\sqrt{\frac{R_N}{\rho(z_i)}}\right)\,. \tag{18}$$

Attack angle load factor control

We check if the load constraint (13) is violated at the Space Shuttle wings,

$$n_{load_{i+1}} = \frac{\rho(z_i){V_i}^2 S}{2mg(z_i)}\left(C_L(\alpha_{i+1}, Ma_i)\cos\alpha_{i+1} + C_D(\alpha_{i+1}, Ma_i)\sin\alpha_{i+1}\right) \le n_{maxload}\,. \tag{19}$$

If this condition is violated, the angle of attack will be corrected by choosing the adequate α_{i+1} that makes the above equality true, regardless the output from the attack heading control. The new values of the attack angle is denoted by α_{i+1}^{load}.

Bank angle heading control

The bank angle heading control was constructed in such a way that, in the (x, y) plan, the aircraft is always horizontally re-orienting itself to the target.

The angular misalignment between the direction vector to the target point (15) and the speed in the (x, y) plane is measured using the dot product. The direction is measured by the z component of the exterior product between the direction vector to the target point $\mathbf{P}_i$ and the aircraft speed V_i. Therefore, in order to align the aircraft to the target point in the (x, y) plane, the new bank angle is,

$$\mu_{i+1}^{heading} = -T_{har} \arccos\left(\frac{P_{i_x}V_{i_x} + P_{i_y}V_{i_y}}{\sqrt{(P_{i_x}^2 + P_{i_y}^2)(V_{i_x}^2 + V_{i_y}^2)}}\right) \text{Sign}\left(P_{i_x}V_{i_y} - P_{i_y}V_{i_x}\right) \tag{20}$$

where we have introduced the constant $T_{hard} \in [0,1]$, $\mu_{i+1}^{heading} \in [-\pi, \pi]$ and $\text{Sign}(x) = 1$ if $x \geq 0$ and $\text{Sign}(x) = -1$, otherwise. The higher the values of T_{hard}, the faster the vehicle will turn for the same angular deviation from the target.

Bank angle anti-stall control

If the attack angle α_{i+1} is different from the stall angle α_{stall}, then $\mu_{i+1}^{stall} = 0$. Otherwise ($\alpha_{i+1} = \alpha_{stall}$), the anti-stall bank angle is,

$$\mu_{i+1}^{stall} = \arccos\left(-\frac{1}{G_{i+1}}\frac{C_D(\alpha_{stall}, Ma_i)}{C_L(\alpha_{stall}, Ma_i)}\right) \tag{21}$$

where we are assuming that $G_{i+1} < 0$ and $\mu_{i+1}^{stall} \in [0, \pi/2]$. Furthermore, in order to avoid oscillatory behaviours during the anti-stall manoeuvre, if $\alpha_{i+1} = \alpha_{stall}$ and $\text{Sign}(\mu_{i+1}^{heading}.\mu_i^{heading}) = -1$, then we change the sign of $\mu_{i+1}^{heading}$.

Bank angle energy control

This control is introduced in order to prevent transients of sharp climbs and dives, by performing dynamic S-turns when needed. It modifies the trajectory of the glider for speeds far above the equilibrium speed.

If this command has been not initiated previously, we define the new constants, $\phi_0 = 0$ and $t_{j0} = 0$.

At each control time t_i, this command checks if following conditions take place simultaneously:

a) The speed is above the critical supersonic threshold: $Ma_i > M_c$ (table 1).

b) The speed is above the equilibrium speed (5) by a certain threshold $\Delta V_{Threshold}$, with $\mu = 0$: $V_i > {V^*}_{\mu=0} + \Delta V_{Threshold}$.

c) The trajectory angle γ is above the equilibrium angle, with $\mu = 0$: $\gamma_{i+1} > {\gamma^*}_{\mu=0}$.

If the three conditions a), b) and c) apply, from the condition $\dot{\gamma} = 0$, we compute the auxiliary bank angle,

$$\mu' = \arccos\left(\frac{2mg(z_i)\cos\gamma_i}{S\rho(z_i)C_L(\alpha_{i+1}, Ma_i)}\right)$$

otherwise, $\mu' = 0$.

We then check the following additional conditions:

i) The three conditions a), b) and c) are verified in the current control time t_i but were not verified in the previous control time t_{i-1}.

ii) In the previous control time t_{i-1}, $\max\{|\mu_i^{heading}|, |\mu_i^{stall}|\} > |\mu_i^{energy}|$.

If at least one of the conditions i) or ii) is verified, we make $t_{j0} = t_i$ and we choose,

$$\phi_0 = \begin{cases} 0 & \text{if} \quad \text{Sign}(\mu_i).\text{Sign}\left(\sin(2\pi f_{energy}(t_i - t_{j0} + T_{con}))\right) \geq 0 \\ \pi & \text{if} \quad \text{Sign}(\mu_i).\text{Sign}\left(\sin(2\pi f_{energy}(t_i - t_{j0} + T_{con}))\right) = -1 \end{cases}$$

where f_{energy} is a low frequency parameter defining the rate of change of the S-turn direction. Then we define,

$$\mu_{i+1}^{energy} = \mu'.\text{Sign}\left(\sin(2\pi f_{energy}(t_i - t_{j0} + T_{con}) + \phi_0)\right) \tag{22}$$

where $\mu_{i+1}^{energy} \in [-\pi, \pi]$.

1.5.1 Combining attack angle and bank angle controls

Attack angle

The attack angle control α_{i+1} is the result of the choices made in the control 1), subject to restrictions 2) and 3) from section 1.5. If the load condition (19) is violated, we make the choice $\alpha_{i+1} = \alpha_{i+1}^{load}$. Then, if the heat flux condition (17) is violated, we make the choice $\alpha_{i+1} = \alpha_{i+1}^{heat}$.

In very extreme situations, it can happens that the structural limit conditions are incompatible ($\alpha_{i+1}^{heat} > \alpha_{i+1}^{load}$, and both are real). In this case, we impose the condition $\alpha_{i+1} = \alpha_{i+1}^{heat}$.

Inside the handover sphere and in order to prevent abrupt changes the angle of attack, we impose the additional condition $\alpha_{i+1} = \alpha_{stall}$, whenever $\alpha_i = \alpha_{stall}$.

Bank angle

For the bank angle control, we first define,

$$\mu_{i+1}^{value} = \max(|\mu_{i+1}^{heading}|, |\mu_{i+1}^{stall}|, |\mu_{i+1}^{energy}|)$$

and,

$$\mu_{i+1}^{sign} = \begin{cases} \text{Sign}(\mu_{i+1}^{energy}) & \text{if} \quad |\mu_{i+1}^{energy}| > \max(|\mu_{i+1}^{heading}|, |\mu_{i+1}^{stall}|) \\ \text{Sign}(\mu_{i+1}^{heading}) & \text{otherwise} \end{cases} .$$

The magnitude of the bank angle is determined by finding the maximum bank angle computed from the three bank angle algorithms, followed by the computation of the adequate bank angle signal. In the computation of the bank angle signal, the heading signal will always prevail except if the Space Shuttle is performing an S-turn.

As $\mu \in]-\pi/2, \pi/2[$, by (20), it can happen that $|\mu_{i+1}^{heading}| > \pi/2$. In this case, we can not point the glider to the target in one single manoeuvre. On the other hand, as bank angle choices near $\pi/2$ are aerodynamically unstable, we define $\mu_{max} < \pi/2$ as the maximum bank angle that the glider can realistically support.

Therefore, the new control bank angle is,

$$\mu_{i+1} = \min(\mu_{i+1}^{value}, \mu_{max}).\mu_{i+1}^{sign} . \tag{23}$$

In order to prevent abrupt manoeuvres inside the handover sphere, if $|\mu_{i+1}| = \mu_{max}$, then the bank heading control will be set to zero ($\mu_{i+1} = 0$).

1.6 Simulations

Using numerical simulations, we test the control and command algorithm developed above. We start by defining the following variables, in line with typical TAEM phase conditions for the Space Shuttle, [2]:

i) Initial conditions:

$$\begin{array}{l} (x_0, y_0, z_0) = (0, 0, 30\,000 \text{ m}),\ V_0 = 1\,100 \text{ m/s } (= 3.65 \text{ M}), \\ \mu_0 = 0^o,\ \alpha_0 = 30^o,\ \gamma_0 = -3^o, \chi_0 = 0^o . \end{array} \tag{24}$$

ii) Physical constraints of the glider:

$$\begin{array}{l} S = 249.9 \text{ m}^2,\ m = 82\,500 \text{ kg},\ \alpha_{min} = \ \alpha_{nl} = 1.5^o,\ \alpha_{max} = \ \alpha_{stall} = 45^o, \\ \mu_{max} = 70^o,\ \Phi_{max} = 5 \times 10^5 \text{ W}/m^2,\ n_{maxload} = 5,\ g_{lim} = 3 . \end{array} \tag{25}$$

iii) Control and command algorithm parameters:

$$T_{con} = 0.1 \text{ s},\ T_{int} = 0.1 \text{ s},\ r_{target} = 100 \text{ m}, \alpha_{precision} = 0.0573^o,$$
$$T_{har} = 1.0, f_{energy} = 1/120 \text{ Hz},\ \Delta V_{Threshold} = 200 \text{ m/s}\,. \tag{26}$$

iv) Target coordinates (HAC center):

$$(x_f, y_f, z_f) = (x_f, y_f, 3\,000 \text{ m})\,. \tag{27}$$

The goal of the control algorithm is to reach the target point considered here as the centre of the HAC. We first analyse the maximum range of the glider and its accessibility region at the target altitude. Secondly, we analyse the distance error when the glider approaches the target, chosen inside the accessibility region. After these analysis, we calculate the typical trajectory generated by the control algorithm, and we perform the analysis of the sensitivity of the trajectories upon variations of the initial conditions and of the control time interval.

The trajectory of the glider is simulated with a fourth order Runge-Kutta integration method with integration time $T_{int} = 0.1$ s, and the reference algorithm control time interval is $T_{con} = 0.1$ s.

1.6.1 Glider range at the target altitude

For straight flights ($y(t) = 0$), the maximum range of a glider is achieved with attack angle chosen at max-glide angle, $\alpha = \alpha_{maxgl}$, figure 6a). The minimum range in straight flight is obtained with the choice $\alpha = \alpha_{stall}$, figure 6a). From the simulations in figure 6a), for straight flights, the smaller range is $x_{sr} = 80\,657$ m, and the maximum range is $x_{mr} = 268\,140$ m.

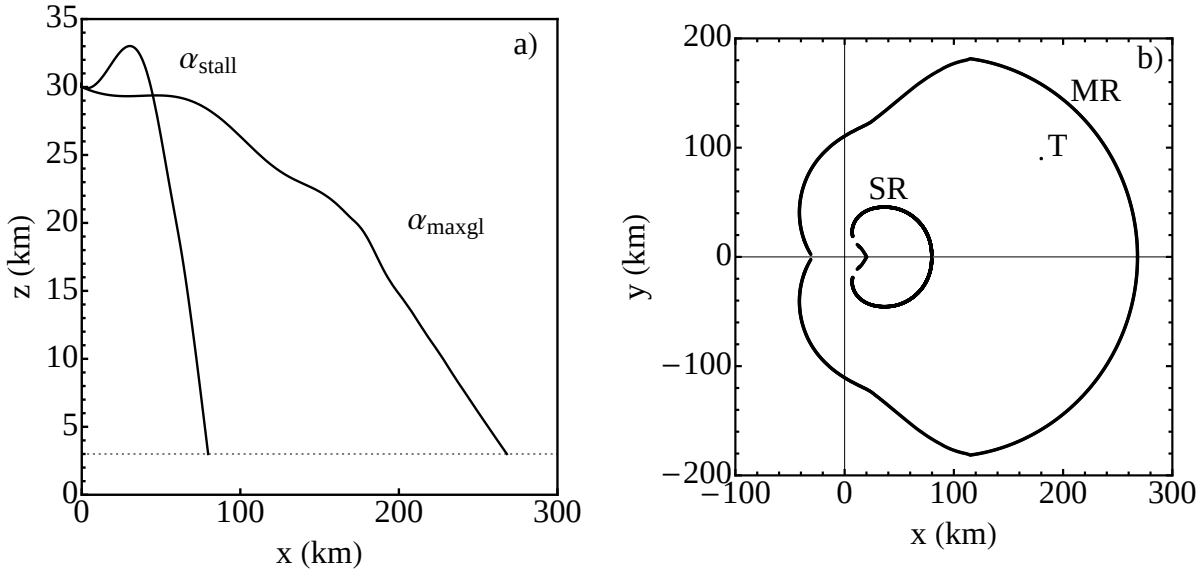


Fig. 6: a) Maximal and minimal range trajectories for a Space Shuttle flight with $y(t) = 0$ and the reference conditions (24)-(27). We show the trajectories of the glider with angles of attack at stall and at max-glide. The minimum and maximum ranges are $x_{sr} = 80\,657$ m and $x_{mr} = 268\,140$ m, for flight times 378.2 s and 675.1 s, respectively. In b), we show the maximum range curve (MR) and the stall range curve (SR) in the (x, y) plane at the altitude $z_f = 3\,000$ m, for the reference conditions (24)-(27). For the initial condition (24), the target point can be any point inside the region bounded by the MR curve. The letter T refers to the target coordinate (28).

For flights not restricted to the (x, z) vertical plane, we have set target coordinates on a sufficiently large two-dimensional circular domain at the altitude level $z_f = 3000$ m, outside the accessible region, and we have applied the control algorithm under analysis. In figure 6b), we show the two dimensional maximum range curve (MR). We have also calculated the range achieved when travelling at stall angle $\alpha = \alpha_{stall}$, marked SR in figure 6b).

1.6.2 Error reaching the target

The arrival to the target point (HAC) occurs when the vertical coordinate is below z_f. We call distance error, the distance from the arrival position to the centre of the HAC, and we denote it by e_d. In order to measure the performance of the algorithm, we have calculated the distance error for target points aligned with the direction of the initial velocity, figure 7a), and for target points inside the MR curve, figure 7b). From these simulations, we conclude that the algorithm achieves any point inside the MR curve with errors of the order of magnitude of 100 m or below.

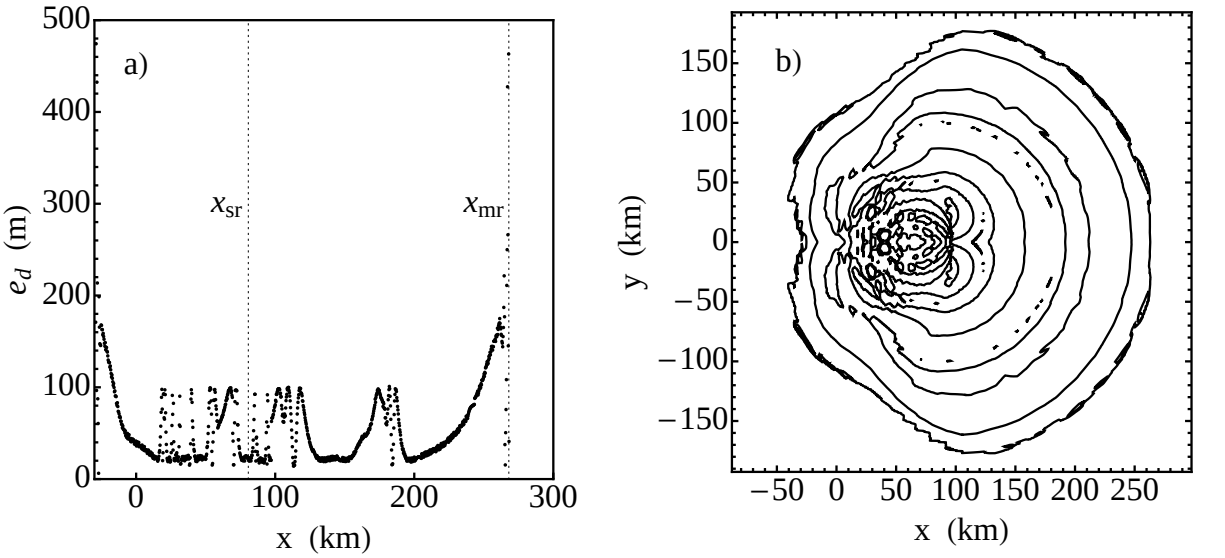


Fig. 7: a) Distance error for target points aligned with the direction of the initial velocity (24), at the altitude $z_f = 3\,000$ m. b) Distance error level sets inside the accessibility curve MR of figure 6b). We have marked the level sets corresponding to the distance error $e_d = 30, 100, 180$ m. Typically, the distance error in reaching the HAC point is of the order of 100 m or below.

1.6.3 Dynamically controlled trajectories

We have chosen target coordinates,

$$(x_f, y_f, z_f) = (180\,000 \text{ m}, 90\,000 \text{ m}, 3\,000 \text{ m}) . \tag{28}$$

as indicated by T in figure 6b). The three dimensional dynamically controlled trajectory obtained with the algorithm of section 1.5 is shown in figure 8a). In figures 8b) and c) we show projected trajectories.

In figure 8d), we show that the speed of the glider is very close to the steady state speed (5), justifying the assertions made in the derivation of the control algorithm. This occurs because convergence times to steady

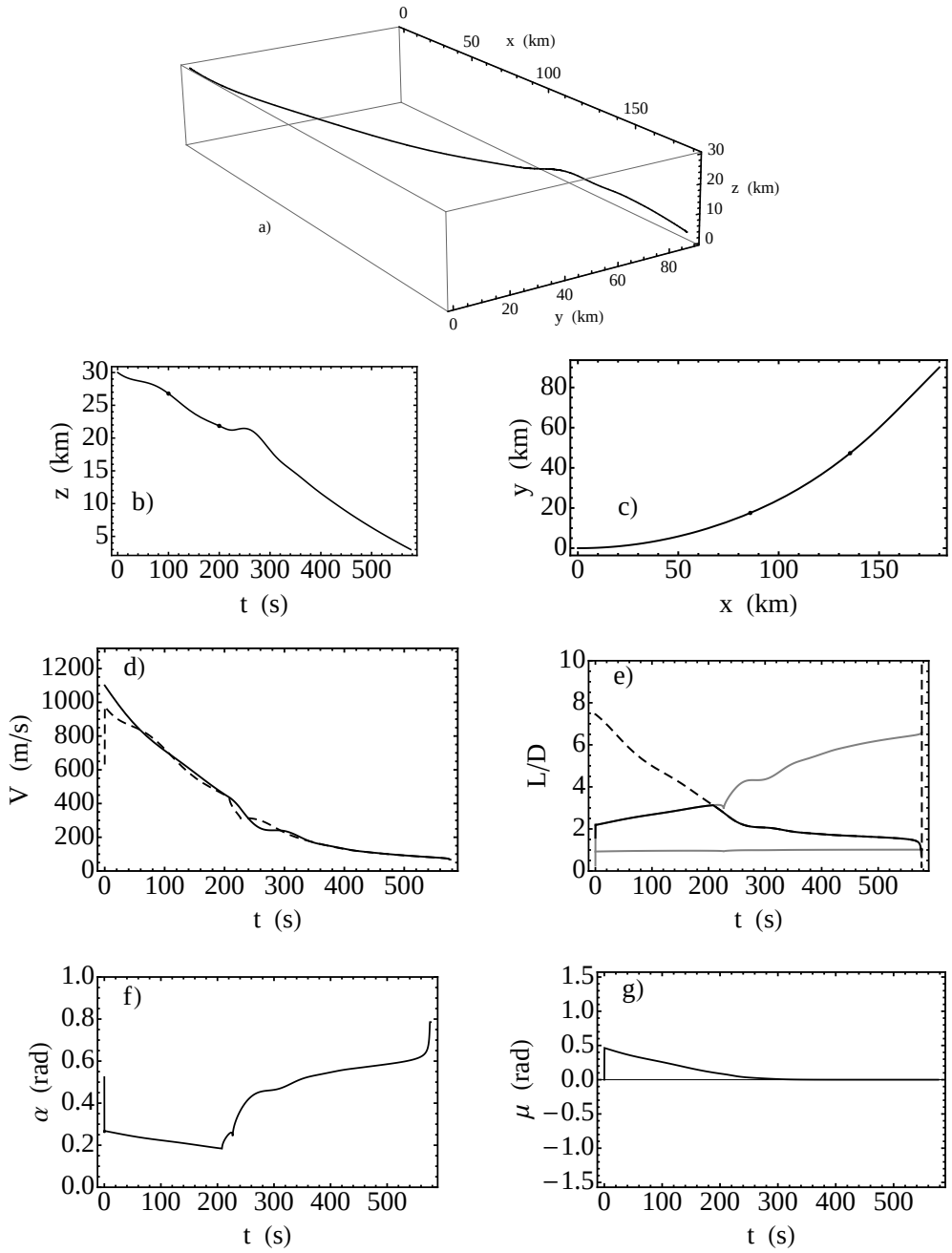


Fig. 8: a) Three dimensional dynamically controlled trajectory of the glider in the physical space, calculated with the control and command algorithm of section 1.5. The reference conditions are given in (24)-(26) and the coordinates of the target are indicated in (28). b) Time evolution of the altitude of the glider. c) Projection of the dynamically controlled trajectory in the (x, y) horizontal plane. The time of arrival at the target is $t = 577.3$ s, the final speed is $V_f = 0.205$ M and the distance error is $e_d = 35.1$ m. The dots indicate the position of the glider after 100 s, 200 s of flight and the target position (HAC centre). In d), it is shown the speed and the equilibrium speed (dashed) as a function of time. In e) it is shown the time evolution of the lift over drag ratio during the TAEM flight. The grey lines represent the aerodynamic limits of flying either with attack angle at max-range or flying with attack angle at stall. The dashed line is the L/D ratio needed to achieve the target in a straight line path at each control time, and the full line shows the decisions made by the algorithm during the flight. In f) and g) we show the time sequence of controls made by the algorithm and responsible for determining the overall trajectory shown in a).

state are proportional to the atmospheric density and, within the TAEM area, this convergence time can be as low as a few seconds. As the control history imposed by the algorithm varies slowly across the entire flight, the dynamic trajectories are smooth and the glider travels most of the time in the equilibrium state defined by the attack angle and bank angle commands. At the altitude of $z_f = 3\,000$ m, the glider will typically be at speeds of the order of magnitude of $\sqrt{2mg(z)/(\rho(z)S)} \simeq 84$ m/s (304 km/h or 0.26Ma) and thus in good conditions to initiate the landing procedures.

When the algorithm takes control of the flight, the glider is travelling at a supersonic speed and as a result the L/D ratio is quite small and the HAC seems to be "non-achievable", figure 8e). In this region, the algorithm responds by imposing the angle of attack that delivers the biggest range at that speed. As the glider looses speed, the aerodynamic balance of the L/D ratio improves, and the L/D ratio enters the region delimited by the aerodynamic limits defined by attack angles at max-glide and stall angles.

For target points inside the stall range curve (SR) of figure 6b), the glider trajectory will be shaped mostly by the anti-stall bank angle command and it will force the glider trajectory to whirlpool around the target as it descends. This effect can be seen in figure 9.

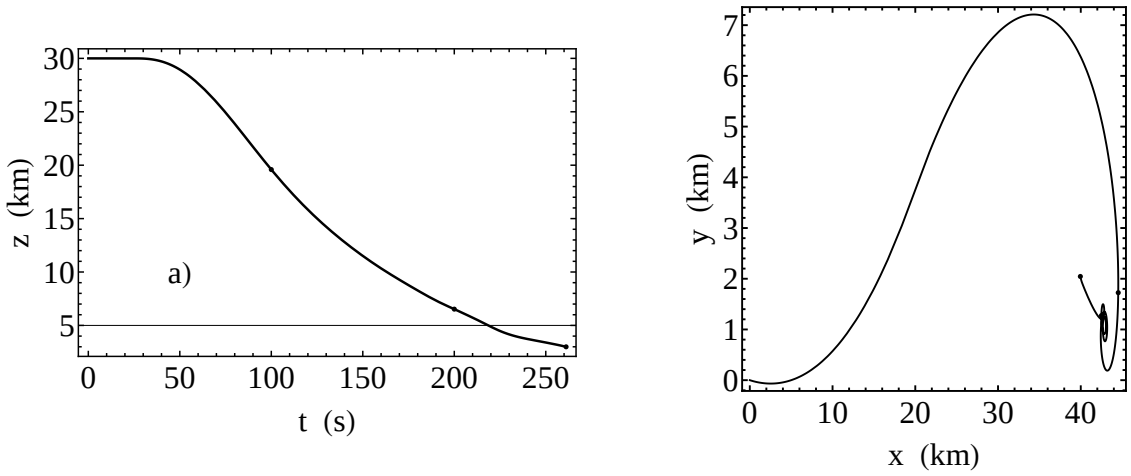


Fig. 9: Dynamically controlled trajectory of a glider in the physical space, calculated with the control and command algorithm of section 1.5. The reference conditions are given in (24)-(26) and target coordinates are $(x_f, y_f, z_f) = (40\,000 \text{ m}, 2\,000 \text{ m}, 3\,000 \text{ m})$, inside the stall range curve (SR) of figure 6b). In a) we show the altitude of the glider as a function of time and, in b), we show the projected trajectory in the (x, y) plane. When the target point is located close to the origin of the target plane, the glider cannot descend fast enough to achieve the target in a straight line. The bank angle anti-stall command imposes an helix based trajectory in order to prevent the glider from reaching the target (x, y) coordinates before the glider can achieve the target altitude.

For the typical initial conditions of the TAEM zone, the glider initial speed is not much larger than the steady state speed, and thus normally the bank angle energy control is not a key factor in shaping the glider trajectory. Nonetheless, should the initial speed be much larger than the equilibrium speed, the bank angle energy control will actively kick-in, avoiding trajectories with abrupt climbs and dives. In figure 10, we show two dynamically computed trajectories with the parameters (24)-(26) and (28), with initial speed $V_0 = 3\,300$ m/s, with and without the bank angle energy control command.

1.6.4 Sensitivity to initial conditions and to control time intervals

We have analysed the changes in the dynamically computed trajectory of the glider with different control time interval T_{con} and for the reference conditions (24)-(26) and (28). We have calculated dynamically

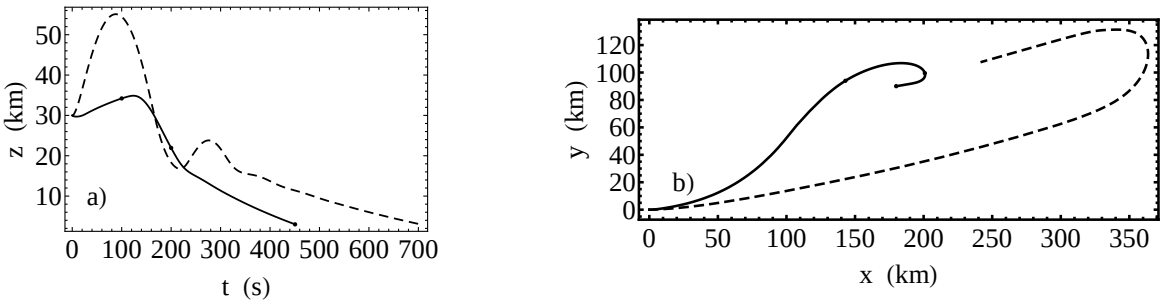


Fig. 10: Dynamically controlled trajectories of a glider in the physical space, calculated with the control and command algorithm of section 1.5 with (full line) and without (dashed line) the bank angle energy control command. The reference parameters are the ones in (24)-(26) and (28), but the initial speed has been changed to $V_0 = 3\,300$ m/s. From these dynamically computed trajectories, we conclude that, for excessive initial speeds, the target is attained only when the bank angle energy control command is switched on.

controlled trajectories for $T_{con} = 0.1, 1, 10, 20, 40$ s. In figure 11, we show the trajectories computed with the different control time intervals (a) and the distance error to the target as a function of T_{con} (b). The distance error e_d increases linearly and slowly as a function of T_{con}. For the reference conditions (24)-(26) and (28), $30 \leq e_d \leq 100$ m, where $0.1 \leq T_{con} \leq 30$ s.

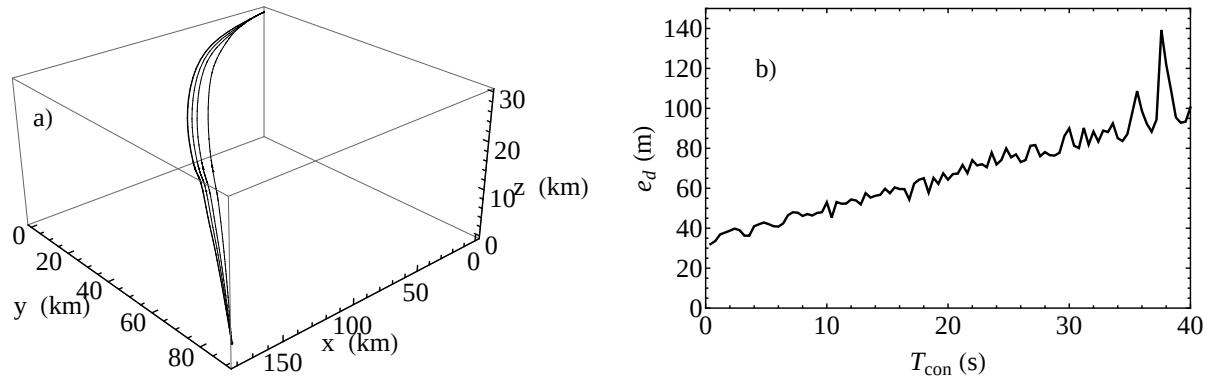


Fig. 11: a) Dynamically controled trajectories for $T_{con} = 0.1, 1, 10, 20, 40$ s, and reference conditions (24)-(26) and (28). In the first part of the trajectories the L/D ratio changes fast and the trajectories deviate slightly for different control times. In the lower atmosphere, different trajectories approach each other, as the L/D ratio changes slowly. b) Error distance to the target as a function of the control time interval T_{con}.

To measure the sensitivity of the trajectories of the glider to changes in the initial conditions, we have changed independently the four initial condition parameters γ_0, χ_0, z_0 and V_0 and then we have calculated the distance errors at the target point. The variations of the error distances are shown in figure 12. We conclude that the acceptable angular misalignments for γ_0 and χ_0 are very different. While in the horizontal plane there is a wide window of acceptable misalignments with the direction of the target (χ_T), in the vertical plane there is a clear threshold (γ_T) on the value that γ_0 can have. Since the glider enters the TAEM phase almost at equilibrium, in order to have enough range, it is crucial that the initial alignment γ_0 is close to zero. Normally this is the case since the glider enters the TAEM phase at excessive speed with γ close to

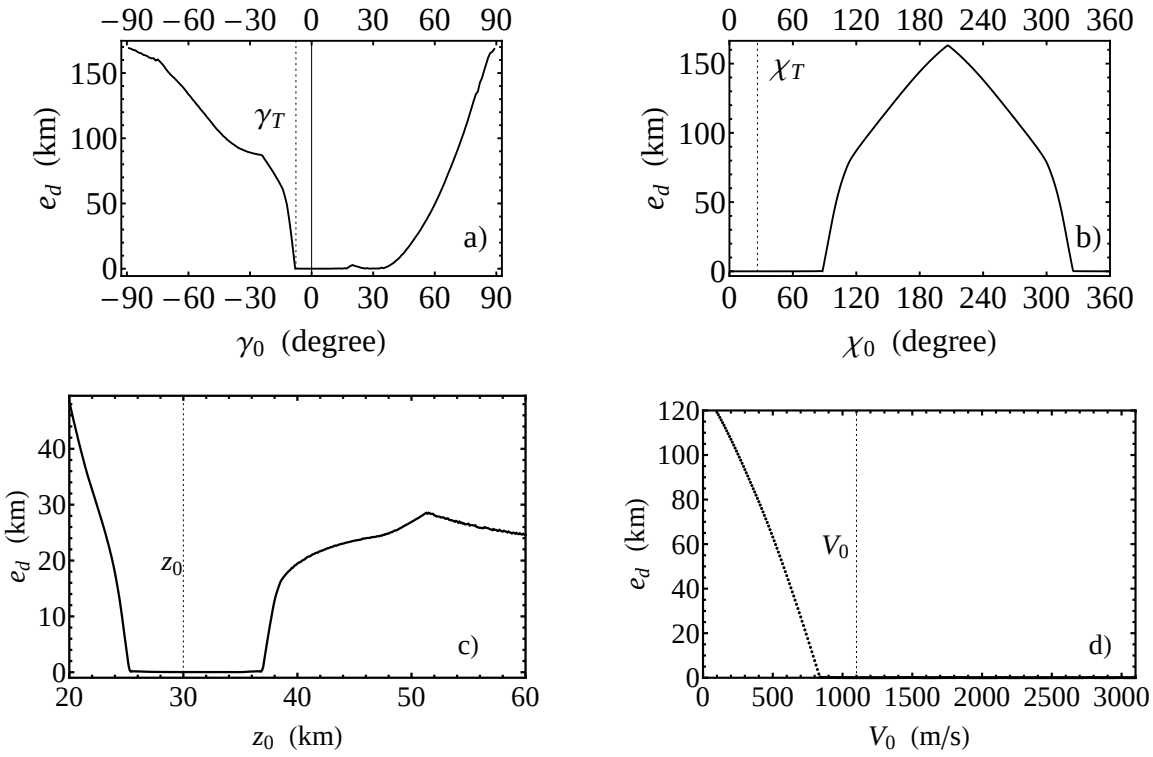


Fig. 12: Distance errors at the target point (28) for different values of γ_0 (a), χ_0 (b), z_0 (c) and V_0 (d). From a) and b), we conclude that the acceptable angular misalignments for γ_0 and χ_0 are very different. While in the horizontal plane there is a wide window of acceptable misalignments with the direction of the target (χ_T), in the vertical plane there is a clear threshold (γ_T) on the value that γ_0 can have. From c) and d), we conclude that the algorithm is resilient to small changes in z_0 and V_0.

zero. From c) and d), we conclude that the algorithm has a wide range of possible initial conditions reaching the HAC point, provided the initial speed is large enough.

Furthermore, we have analysed the impact of different initial speeds on the structural limits of the glider. For the reference normal initial speed ($V_0 = 1\,100$m/s), no structural limits are infringed in the TAEM phase. For extreme initial speeds, there are constraints in the range of possible values for the attack angle and these are successfully imposed by the algorithm. In figure 13, we show the variation of the structural limits along on the glider for different initial speeds.

1.7 Conclusions

In this paper, we have derived a new algorithm for the command and control of re-usable space gliders. The new algorithm determines locally the shortest path to the target point, ensuring that it is compatible with the aerodynamic characteristics and structural limits of the operated glider. With this algorithm, we successfully guided a glider to a target point with minimum errors by making only local guidance decisions. Due to its simplicity, the algorithm is computationally fast and thus adequate for fully automated onboard dynamic controls guiding non-manned gliders.

In this proof of concept, we have tested the ability of the algorithm to guide the Space Shuttle during the TAEM phase, successfully delivering the glider to the HAC centre with distance errors of the order of magnitude of 100 m or below, even with control time intervals of up to 30 s. Furthermore, the algorithm has done the trajectory control without violating the structural limits of the Space Shuttle both in terms of the

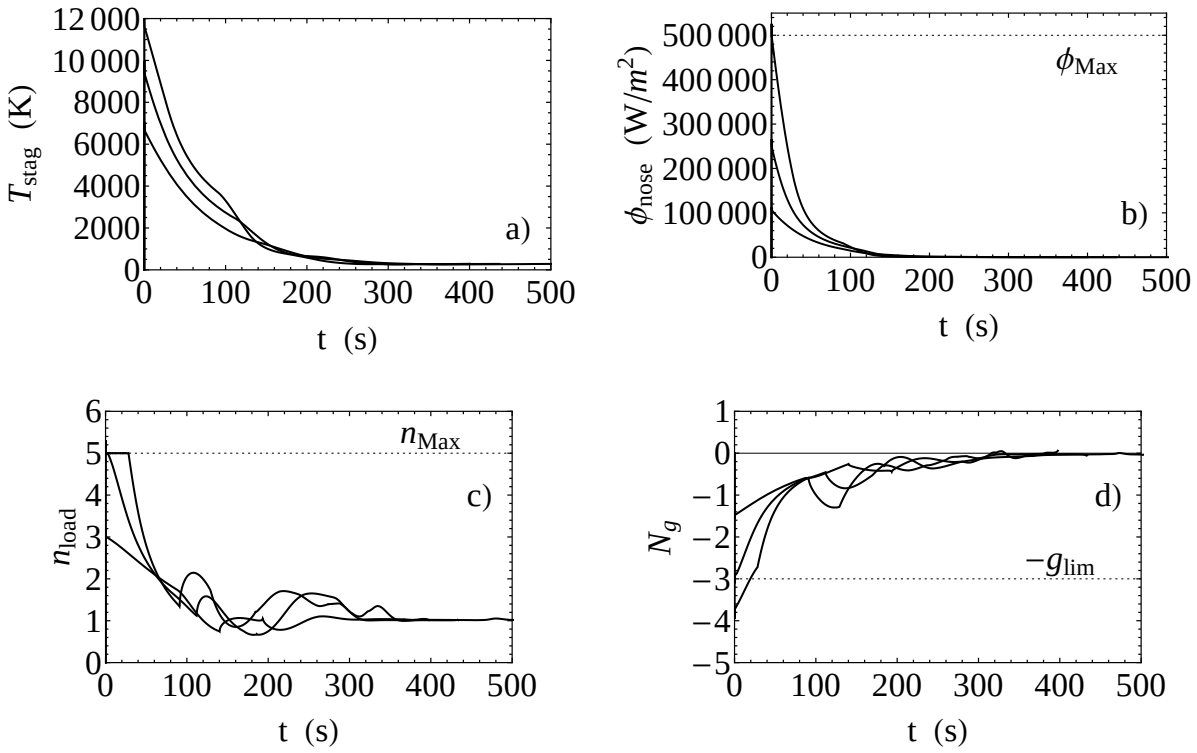


Fig. 13: Sensitivity analysis of structural limits for different initial speeds. We have chosen $V_0 = 1\,650, 2\,200$ and $2\,750$ m/s, far above typical initial speeds. In a), we show the stagnation temperature, with the highest temperature achieved for the highest initial speed. In b), we show the corresponding heat flux at the Space Shuttle nose. For the highest initial speed the attack angle heat limit control intervened in the algorithm in order to maintain the structural limit Φ_{Max}. In c), we show the load at the Space Shuttle wings, where, for the two highest initial speeds, the attack angle load limit control intervened to maintain the maximum wing load, according to existing structural limit n_{Max}. In d), we show the acceleration inside the Space Shuttle, calculated by (14). The most negative accelerations are obtained for the highest initial speeds and the typical maximum admissible acceleration ($g_{lim} = 3$) is violated during a short period for the highest initial speed. In this case, we have no control mechanism.

maximum heat flux at the nose and the maximum load at the wings. Applying the algorithm to any other glider is straightforward and can easily be done by adjusting the physical constraints of the glider (25) and the fits made for the aerodynamic coefficients (8)-(10).

This algorithm can be further extended in order to ensure that the velocity at the target is aligned with a specific direction, as for example the direction of the runway measured from the HAC point. It can also be extended to deal with thrust and variable mass, making it applicable to powered vehicles.

The content of this chapter will appear as a scientific paper [20].

Acknowledgements We would like to acknowledge João Graciano from AEVO GmbH (Munich) for suggestions and critical reading of this paper.

Appendix A

We consider a point mass vehicle or airplane falling in the gravitational field. We start with the usual Newton equations in cartesian coordinates for a falling body in the gravity field of a flat Earth,

$$\begin{cases} \dot{x} = V_x \, , \dot{y} = V_y \, , \dot{z} = V_z \\ m\dot{V}_x = 0 \\ m\dot{V}_y = 0 \\ m\dot{V}_z = -mg(z) \end{cases} \tag{29}$$

where $g(z) = g_0(R_E/(R_E + z))^2$ is the gravity acceleration, g_0 is the standard gravitational acceleration and R_E is the Earth mean radius.

To describe the orientation of the falling airplane, we introduce polar coordinates in the velocity. In the reference frame represented in figure 1, we define,

$$\begin{cases} V_x = V\cos\chi\sin\theta = V\cos\chi\cos\gamma \\ V_y = V\sin\chi\sin\theta = V\sin\chi\cos\gamma \\ V_z = V\cos\theta = V\sin\gamma \end{cases} \tag{30}$$

where $\theta + \gamma = \pi/2$ and $V^2 = V_x^2 + V_y^2 + V_z^2$. Calculating the time derivatives of (30) and introducing it into (29), for the three last equations in (29), we obtain,

$$\begin{cases} m\dot{V}\cos\chi\cos\gamma - mV\dot{\chi}\sin\chi\cos\gamma - mV\dot{\gamma}\cos\chi\sin\gamma = 0 \\ m\dot{V}\sin\chi\cos\gamma + mV\dot{\chi}\cos\chi\cos\gamma - mV\dot{\gamma}\sin\chi\sin\gamma = 0 \\ m\dot{V}\sin\gamma + mV\dot{\gamma}\cos\gamma = -mg(z)\,. \end{cases} \tag{31}$$

Solving equations (31) in order to $\dot{V}$, $\dot{\gamma}$ and $\dot{\chi}$, we obtain,

$$\begin{cases} m\dot{V} = -mg(z)\sin\gamma \\ mV\dot{\gamma} = -mg(z)\cos\gamma \\ mV\dot{\chi}\cos\gamma = 0\,. \end{cases} \tag{32}$$

Equations (32) together with the first three equations in (29) describe the motion of a mass point falling body in a flat Earth.

Due to the spatial dimensions and form of aircrafts and airplanes, they are subjects to drag, lift and thrust forces.

The drag force D acting on an aircraft is parallel to the velocity vector but with opposite direction. The lift force L is perpendicular to the velocity vector of the spacecraft and perpendicular to the plane of the wings of the aircraft. Introducing these forces into equation (32), the equations of motion of the falling aircraft in the Earth atmosphere is,

$$\begin{cases} m\dot{V} = -mg(z)\sin\gamma - D(\alpha, Ma) \\ mV\dot{\gamma} = -mg(z)\cos\gamma + L(\alpha, Ma)\cos\mu \\ mV\dot{\chi}\cos\gamma = L\sin\mu \end{cases} \quad \begin{cases} \dot{x} = V\cos\chi\cos\gamma \\ \dot{y} = V\sin\chi\cos\gamma \\ \dot{z} = V\sin\gamma \end{cases} \tag{33}$$

where γ is the flight path angle as defined in figure 2a)-b), μ is the bank angle, α is the angle of attack, figure 2c), and Ma is the Mach number. In this reference frame, $V \in]0, \infty[$, $\gamma \in]-\pi/2, \pi/2[$, $\chi \in [0, 2\pi[$, $\mu \in]-\pi/2, \pi/2[$ and $\alpha \in]-\pi/2, \pi/2[$.

Appendix B

The Earth atmosphere parameters are based on the 1976 US Standard Atmosphere Model. For the first seven layers we have used the formulas described in [14]. In table 2, we show the parameterisation of the thermodynamic quantities for the Earth atmosphere.

Layer	z_0 (m)	T_0 (K)	λ_0(K/m)	P_0 (Pa)
1	0	288.15	-0.0065	101325.00
2	11019	216.65	$-$	22632.10
3	20063	216.65	0.0010	5474.89
4	32162	228.65	0.0028	868.02
5	47359	270.65	$-$	110.91
6	51412	270.65	-0.0028	66.94
7	71802	214.65	-0.0020	3.96

Layer	T (K)	P (Pa)	ρ (kg/m^3)
1	$T_0 + \lambda_0(z - z_0)$	$P_0(\frac{T_0}{T})^{g(z)M_{air}/(R\lambda_0)}$	$\frac{P}{TR_s}$
2	T_0	$P_0 e^{-g(z)M_{air}(z-z_0)/(RT)}$	$\frac{P}{TR_s}$
3	$T_0 + \lambda_0(z - z_0)$	$P_0(\frac{T_0}{T})^{g(z)M_{air}/(R\lambda_0)}$	$\frac{P}{TR_s}$
4	$T_0 + \lambda_0(z - z_0)$	$P_0(\frac{T_0}{T})^{g(z)M_{air}/(R\lambda_0)}$	$\frac{P}{TR_s}$
5	T_0	$P_0 e^{-gM_{air}(z-z_0)/(RT)}$	$\frac{P}{TR_s}$
6	$T_0 + \lambda_0(z - z_0)$	$P_0(\frac{T_0}{T})^{g(z)M_{air}/(R\lambda_0)}$	$\frac{P}{TR_s}$
7	$T_0 + \lambda_0(z - z_0)$	$P_0(\frac{T_0}{T})^{g(z)M_{air}/(R\lambda_0)}$	$\frac{P}{TR_s}$

Table 2: Characteristic parameters for the lower layers of the atmosphere. The parameter z_0 is the lower altitude of the layer, $R = 8.31432$ J/(K mol) and $R_s = 287.04$ J/(K kg) are ideal gas constants, $M_{air} = 0.0289644$ kg/mol, $g(z) = g_0(R_E/(R_E + z))^2$ is the gravity acceleration, $g_0 = 9.80665$ m/s^2 is the standard gravitational acceleration constant and $R_E = 6.371 \times 10^6$ m is the Earth mean radius. All formulas have been taken from [14].

References

1. Costa, R. R.: Studies for terminal area GNC of reusable launch vehicles, AIAA, paper 2003-5438, 2003.
2. Findlay, J. T., Kelly, G. M., Heck, M. L.: Reconstruction of the 1st Space Shuttle (STS-1) Entry Trajectory, NASA Contractor Report 3561, 1982.
3. Gallais, P.: Atmospheric Re-Entry Vehicle Mechanics, Springer, 2007.
4. Horneman, K., Automated Trajectory Generation and Guidance for a New Launch Vehicle Flight Phases, Faculty of the Graduate School - University of Missouri-Columbia, 2010.
5. Hull, D.G.: Fundamentals of Airplane Flight Mechanics, Springer, 2007.
6. Jiang, Z., Ordonez, R.: Trajectory Generation on Approach and Landing for RLVs Using Motion Primitives and Neighboring Optimal Control, Proceedings of the 2007 American Control Conference, 2007.
7. Mease, K. D., Teufel, P., Schonenberger, H., Chen, D. T., Bharadwaj, S., Re-entry trajectory planning for a reusable launch vehicle, AIAA paper 99-4160, 1999.
8. Miele, A.: Flight Mechanics, Vol. I, Theory of Flight Paths, Addison-Wesley, Reading MA, 1962.
9. Ramsey, P.E.: Space Shuttle Aerodynamic Stability, Control Effectiveness and Drag Characteristics of a Shuttle Orbiter at Mac Numbers from 0.6 to 4.96, NASA/MSFC, 1972.
10. Raymer, D.P.: Aircraft Design: A Conceptual Approach, Fourth Edition, AIAA Education Series, 2006.
11. Rehder, J. J., Holloway, P. F., Orbiter Entry Trajectory Considerations, NASA Langley Research Center, 1972.
12. Shevell, R.S.: Fundamentals of Flight, 2nd Edition, Prentice Hall, 1988.
13. Trelat, E.: Optimal Control of a Space Shuttle and Numerical Simulations, Proceedings of the Fourth International Conference on Dynamical Systems and Differential Equations, Wilmington NC USA, 2002.
14. US Standard Atmosphere, NASA-TM-X-74335, NASA, 1976.
15. Vernis, P., Ferreira, E.: On-Board Trajectory Planner for the TAEM Guidance of a Winged-Body, EADS Space Transportation, 2004.

2 Additional simulations

2.1 Detailed analysis of the different trajectory types

In order to present the full comparison between the typical types of trajectory, we have included this section that was removed from the main paper, Chapter 1, due to space limitations. As such, we present additional graphs describing the typical trajectories designed by the algorithm.

The three main types of trajectories designed by the algorithm are:

a) Anti-stall trajectory (short range trajectories);
b) Excess Energy trajectory with energy control;
c) Excess Energy trajectory without energy control.

Anti-stall trajectory

In this type of trajectory, the target point is located very near the origin of the xy plane (where the algorithm is activated). In order to compensate for that, the glider will "roll around" the target until it can lose enough altitude.

This trajectory is typically done in high angle of attack (at the stall limit) and poses high structural stress in terms of pressure on the wings.

i) Trajectory

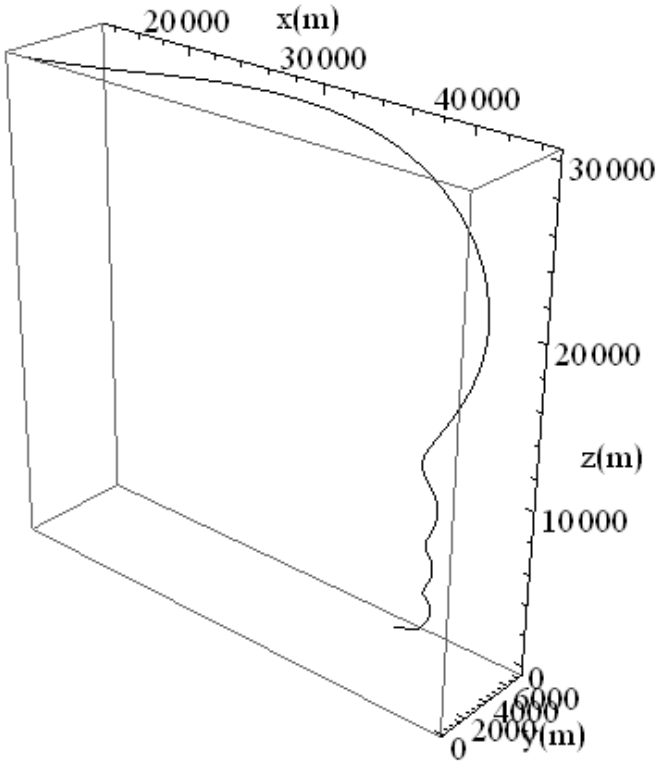


Figure 14

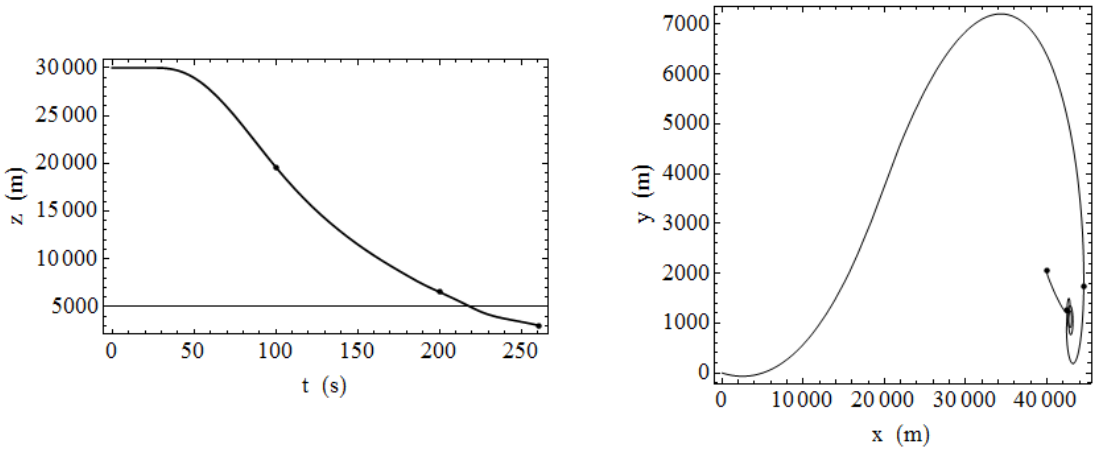


Figure 15

ii) Speed and Aerodynamic balance

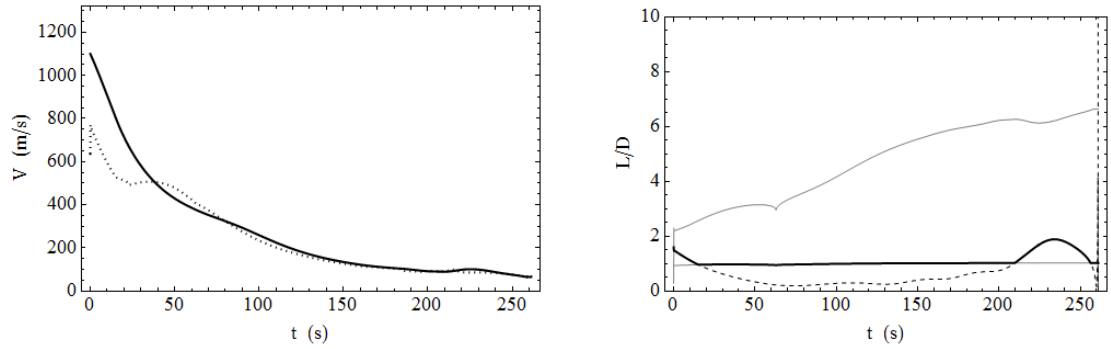


Figure 16

iii) Control history

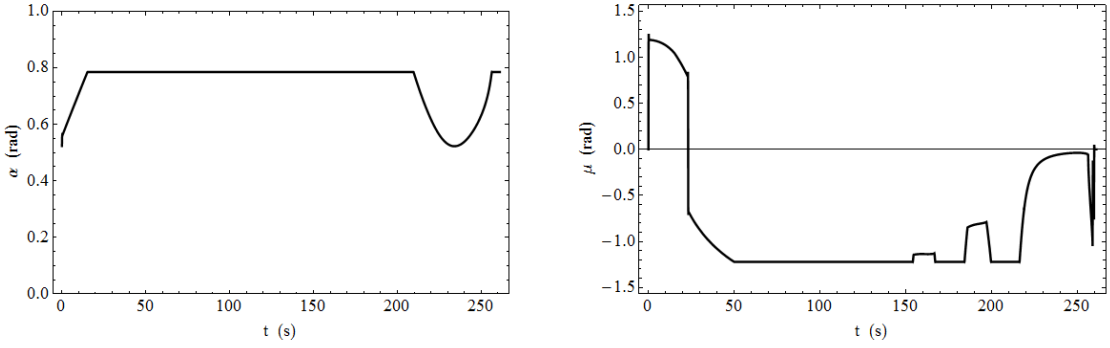


Figure 17

iv) Structural limits

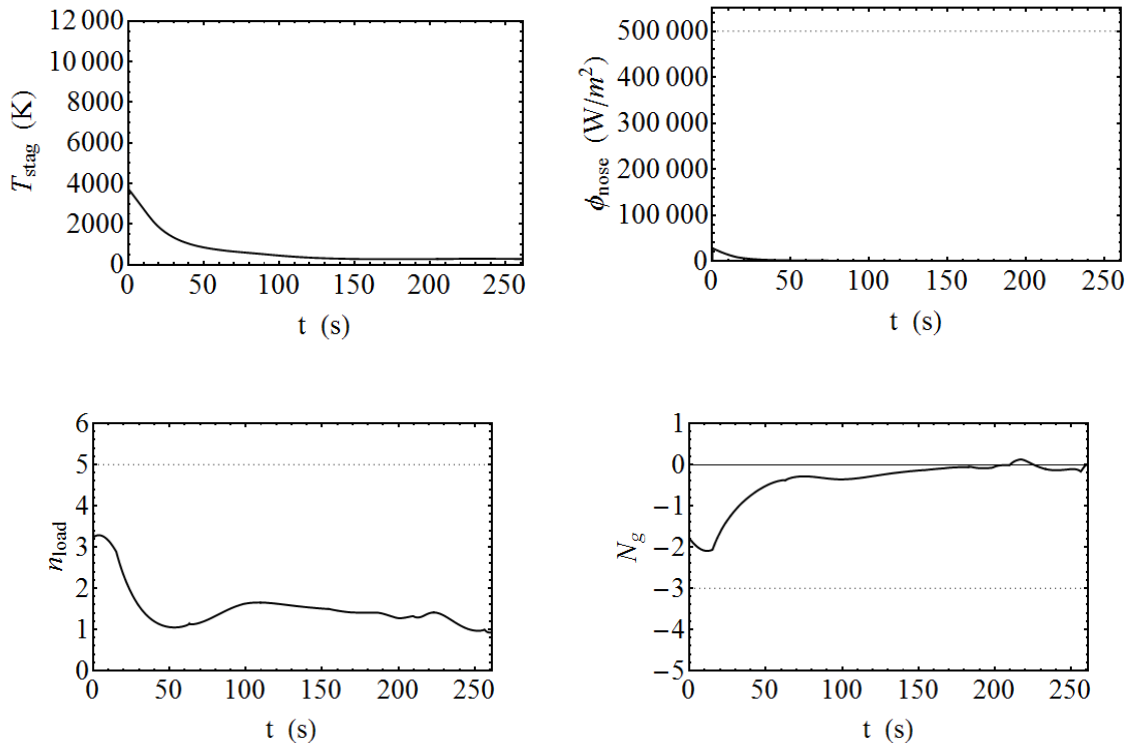


Figure 18

Excess Energy trajectory with energy control

In this type of trajectory, the glider enters the TAEM phase with clear excess speed. When the algorithm is activated and the bank energy control feature is enabled, the glider initiates a series of dynamic S-turns that will prevent the glider from increasing further the trajectory angle (γ) and thus stop it from gaining further altitude.

When the bank energy control feature is enabled, the trajectory is smooth and all structural limits are preserved (except maybe in the initial stages of the flight should the algorithm inherent the control with an initial condition that clearly violates those structural limits).

Typically, when the bank energy control feature is enabled the glider is quickly brought back to equilibrium and reaches the desired target.

i) Trajectory

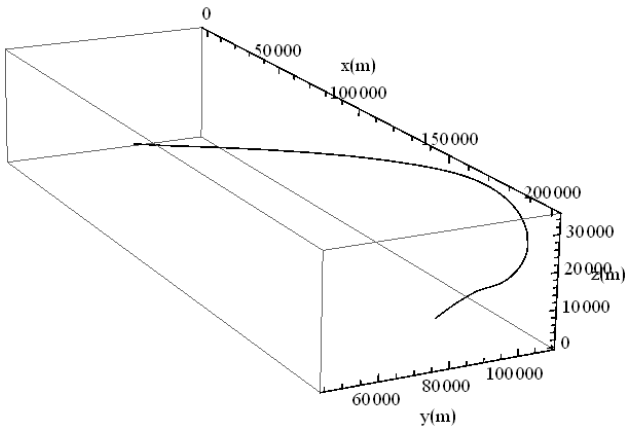


Figure 19

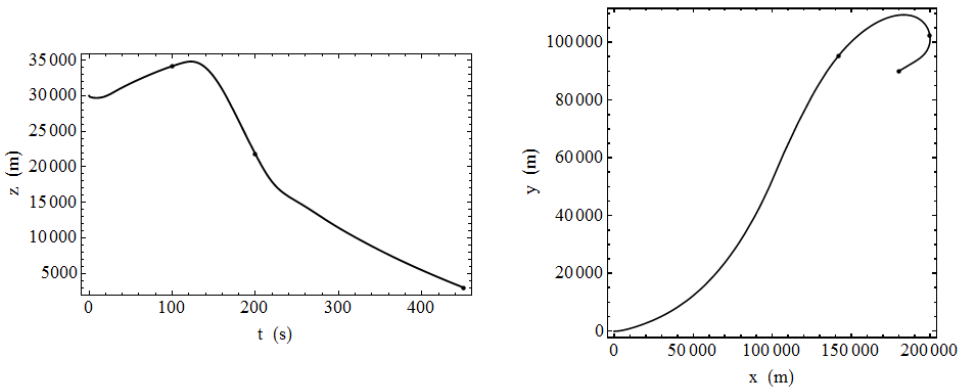


Figure 20

ii) Speed and Aerodynamic Forces balance

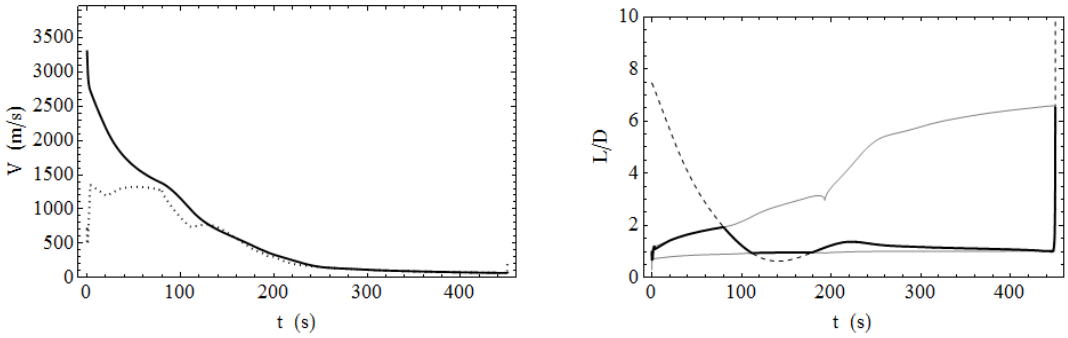


Figure 21

iii) Control history

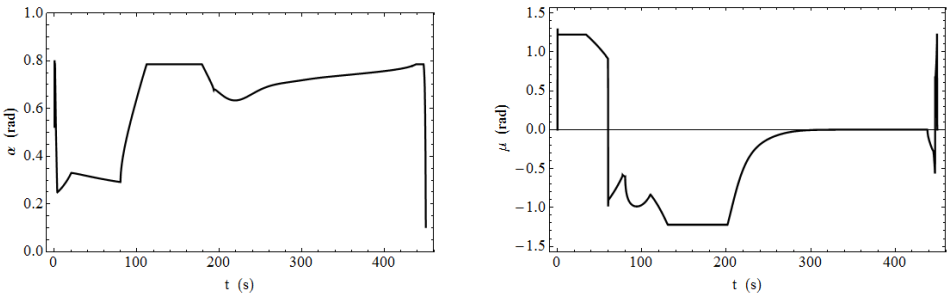
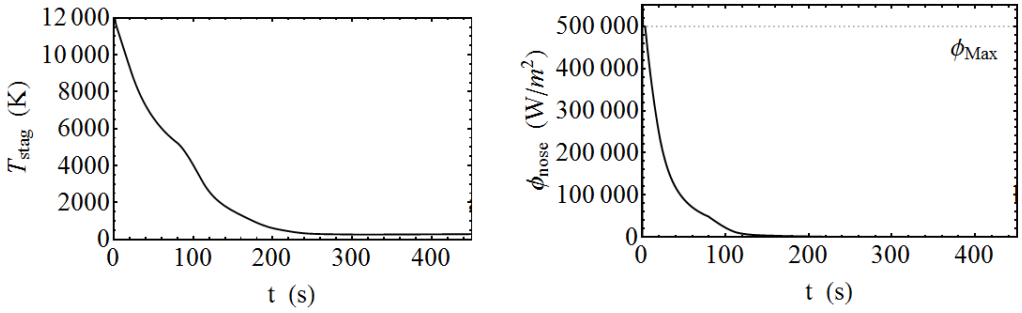


Figure 22

iv) Structural limits



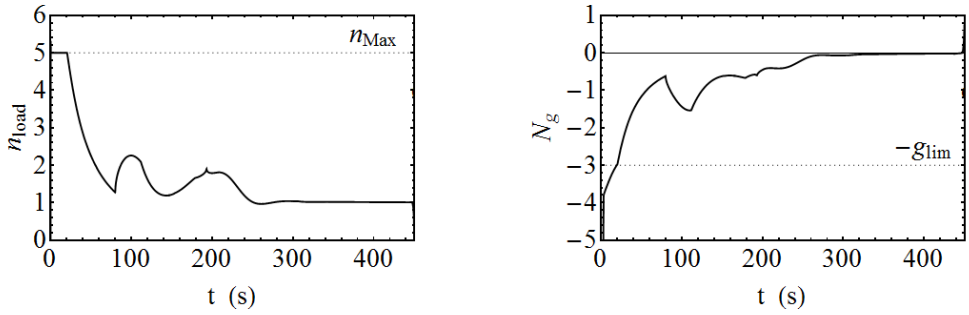


Figure 23

Excess Energy trajectory without energy control

In this type of trajectory, the glider enters the TAEM with clear excess speed. When the algorithm is activated and the bank energy control feature is enabled, the glider initiates a series of dynamic S-turns that will prevent the glider from increasing further the trajectory angle (γ) and thus stop it from gaining further altitude.

When the bank energy control feature is not enabled, the is described by a sharp succession of climbs and dives, that not only test the structural limits of the glider, but in addition prolongs the flight thereby subjecting the glider to those limits for a longer period of time.

Typically, when the bank angle energy control feature is not enabled the glider overshoots the target and then runs out of energy to return to the desired target.

i) Trajectory

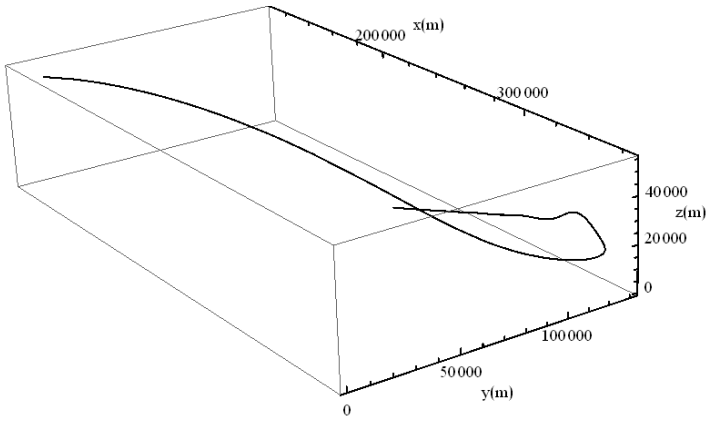


Figure 24

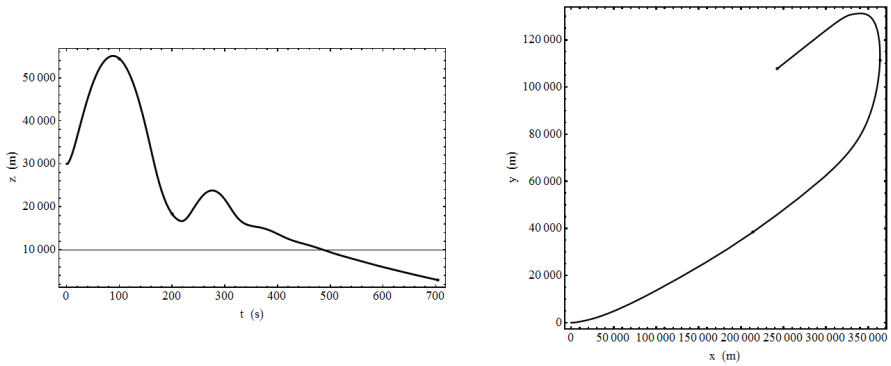


Figure 25

ii) Speed and Aerodynamic Forces balance

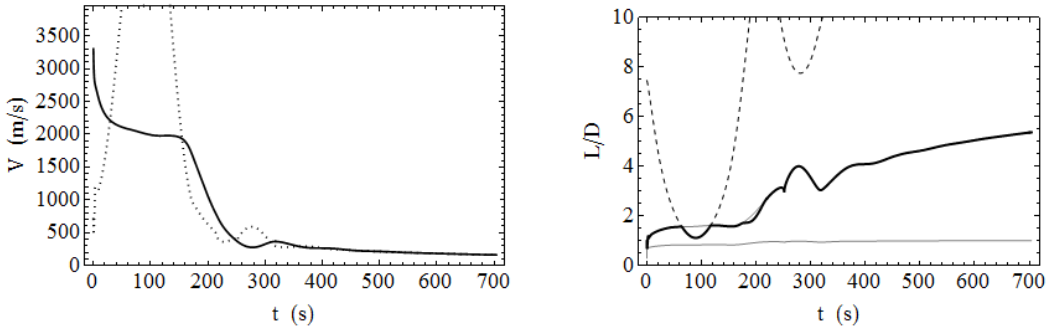


Figure 26

iii) Control history

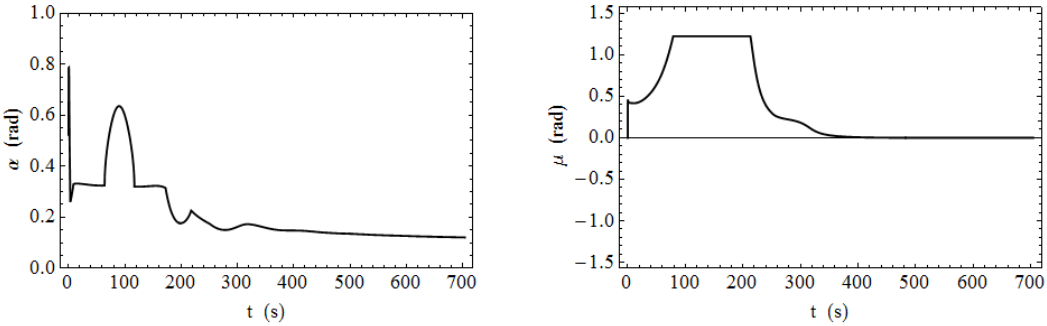


Figure 27

iv) Structural limits

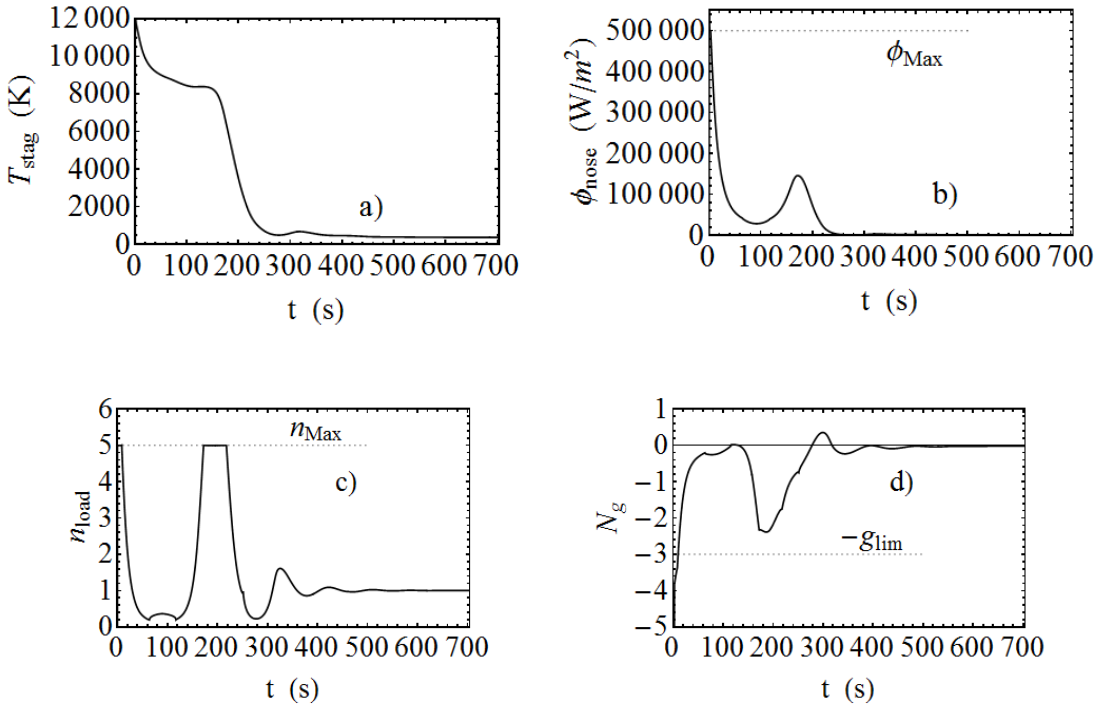


Figure 28

2.2 Controlling the error of the Runge-Kutta numerical integration

In order to control the error on the numerical integration done on the equations of motion used in our simulations, we have derived an invariant that we use as a benchmark for the quality of the numerical integration.

Through the manipulation of equations (3) in the main paper, in Chapter 1, we arrive to an invariant relating, at any point, the aerodynamic forces, the inertial forces and the only external source of energy in the system the gravity force. This invariant is given by:

$$(\dot{V} + \frac{C_D(\alpha, Ma)\,\rho(z)\,S}{2\,m} V^2)^2 + (V\dot{\gamma} - \frac{C_L(\alpha, Ma)\,\rho(z) * S}{2\,m} V^2 Cos\ \mu)^2 = g^2$$

As a particular case, when we make the linear and angular accelerations equal to zero, this equation yields the equilibrium speed and equilibrium angle.

For our simulations, we have derived the following indicator of quality of the numerical integration:

$$\%_{Numerical\ Error} = \left(\left|\frac{Inv}{g^2(z_i)}\right| - 1\right) * 100\%$$

Where we have defined,

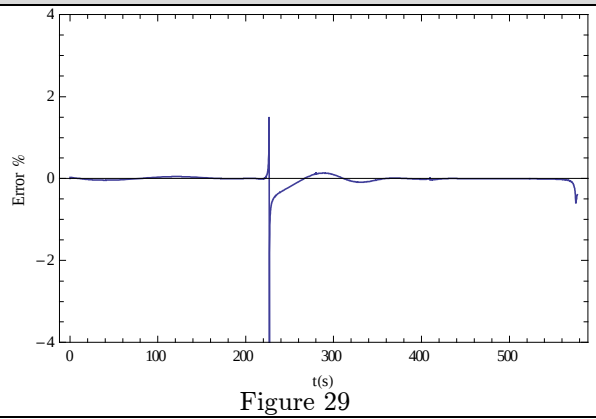
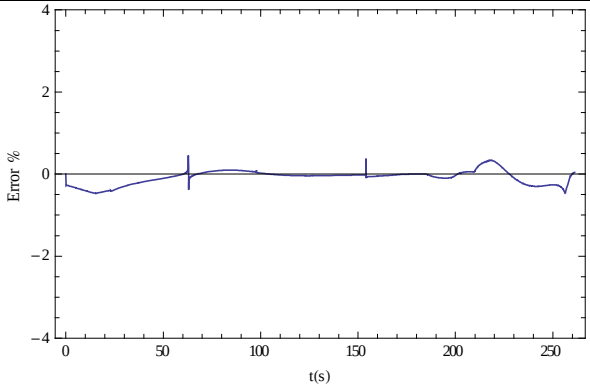
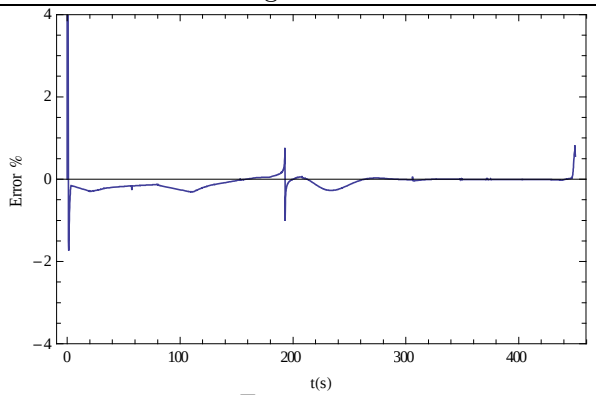
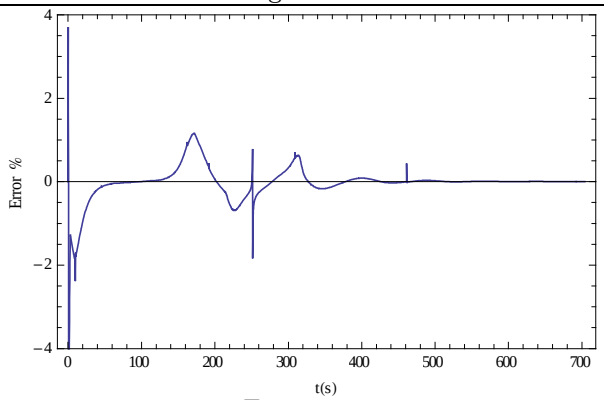
$$Inv = (\dot{v}_i + \frac{\rho(z_i)\,S\,C_D(\alpha_i, Ma_i)}{2\,m} {v_i}^2)^2 + (V_i\dot{\gamma}_i - \frac{\rho(z_i)\,S\,C_D(\alpha_i, Ma_i)}{2\,m} {v_i}^2 Cos\ \varphi_i)^2$$

In the simulations, we used a 4th order Runge-Kutta method that delivers a good integration on all points, except on two situations that are sporadic in TAEM trajectories:

a) The supersonic to sub-sonic transition (also known as "sonic-boom");
b) Abrupt changes in the command history. The only situations where the algorithm may introduce abrupt changes are:
    I. Near the target when travelling in trajectories close to the maxmimum glide limite
    II. Eventually, on the moment it takes over the control of the glider should the initial conditions of attack and bank angles substantially different from the first decision made by the algorithm.

A further improvement can be done in the numerical integration, not related to the algorithm itself, which consists in using a variable integration step strategy in the areas where the integration deviates further from the equation invariant.

Application to the three types of trajectory:

Type of trajectory	Invariant evolution across time	% Numerical Error
Midpoint trajectory	 Figure 29	0.05 %
Anti-stall trajectory	 Figure 30	0.13 %
Excess Energy trajectory (with energy control)	 Figure 31	0.12 %
Excess Energy trajectory (without energy control)	 Figure 32	0.19%

3 Conclusions and future work

3.1 Recent evolutions in the space industry

Space exploration is "back in vogue", with the entry of pure-private players (SpaceX) for the first time in the aerospace picture. Previous contracts were on "cost-plus" basis, ideal to start an industry but not adequate for massification. New emerging countries, such as Brazil, China and to some extent India are also entering the picture for the first time. Europe is again considering launching its own space program for reusable vehicles.

Despite the crash of the "Beagle 2", many probes have been landed succesfully in Mars. "Curiosity" landed in Mars a few weeks ago performing a guided-landing maneuver, sometimes described as "7 minutes of terror" [11], assisted by both rockets and parachutes involving complex maneuvers managed dynamically by in-built algorithms. The maneuvers were too complex not to be managed dynamically and had to be automatic since the distance from Earth (14 light minutes) did not allow for remote monitoring and control.

The previous probe sent to Mars, Spirit, had non-dynamic automated maneuvers that generated a much bumpier landing on Mars [16]. A smooth landing is critical since, not only it allows for more fragile high-precision material to be onboard, but it also frees space that would other wise be occupied by air-bags or other anti-crash system.

Manned missions to Mars are being planned for 2020-2040 by both private and public players [12] as technology evolution since the early days of the Space Shuttle (in the late 1960's). Several key breakthroughts are not available:

a) When the Space Shuttle was developed, the aerodynamic testing relied so heavily on wind tunnel data that the world's record hours of usage of wind tunnel was established during the program [9]. Today CFD codes allow for far more return on invested money and faster development.
b) The thermal insulation of the Space Shuttle was achieved through a very complex system of individual tiles glued together on the body of the vehicle, which added considerable mass to the Space Shuttle [19]. Today, new materials allow for Kevlar-based inflatable heat shields [13].
c) Apart from Saturn V that was not widely used, the Space Shuttle established the record for the maximum payload delivered to orbit (24,400 kg) while today "Falcon Heavy" from SpaceX can deliver the double (53,000 kg) routinely in orbit [14].
d) Even on other fronts such as astronaut space suits used to weight more than 140 kg, making them usable only on environments with no gravity (e.g. orbit or taxing through planets) or very low orbits (e.g. moon), but completely inadequate for planetary environment (e.g. Mars or Venus). New suits can now weight as little as a few kilograms [15].

Below, we should the detail of the Space Shuttle tile structure, gently shared by Professor Rui Dilao. The photos were taken in a field visit to Titus Ville.

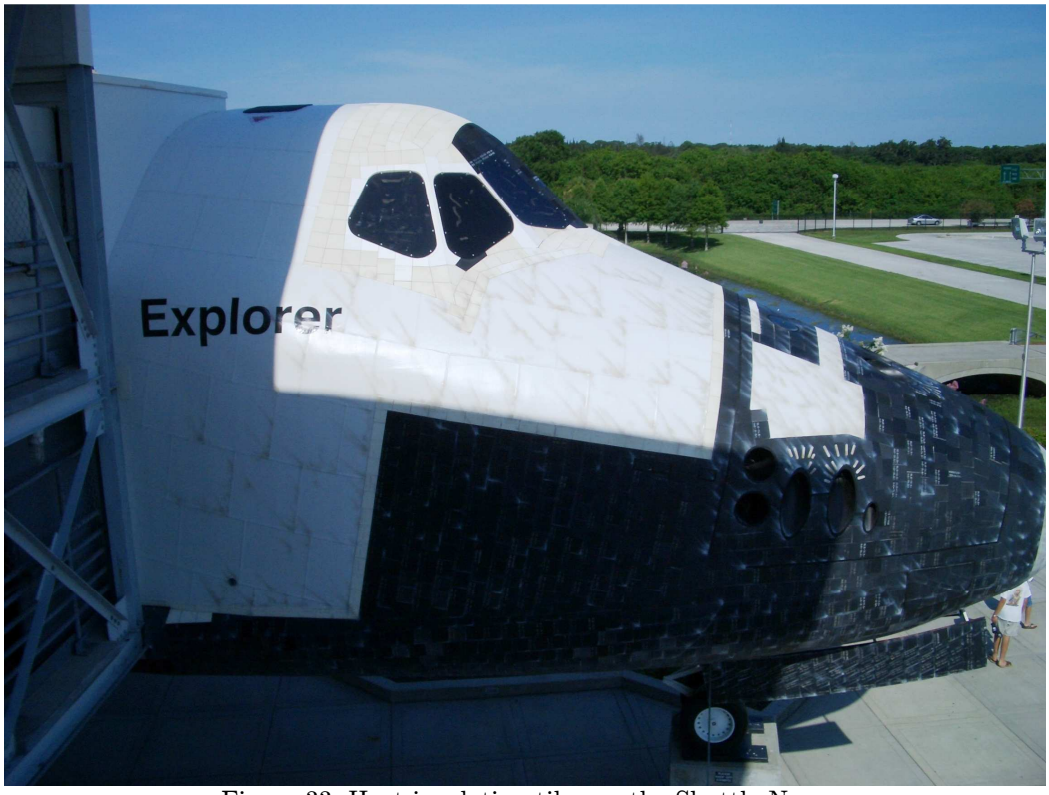


Figure 33: Heat insulation tiles on the Shuttle Nose


Figure 34: Detailed zoom in of the heat insulation tiles on the Shuttle Nose

3.2 Potential evolutions for the new algorithm

In line with new technological development, control and command algorithms are crucial to evolve in line with the new technological developments in order to ensure the best return is taken out of those developments. In that sense, we have tried to develop an algorithm for gliders re-entering on Earth atmosphere that would be capable of delivering:

a) Maneuverability: ability to reach targets in any direction.
b) Range and error: ability to reach distant targets with limited error.
c) Resilience: ability to sustain a wide range of initial conditions.
d) Safety: ability to manage the structural limits of the vehicle.
e) Flexibility: ability to work with a wide range of control time intervals.
f) "Local" optimization: ability to work on onboard computers due to being computationally fast.
g) Smooth controls: Ability to deal vehicles that do not respond instantaneously and thus successive commands ask similar angles avoiding disruptions should commands have to wait in a queue to be executed.

The results achieved in the simulation were quite satisfactory with distance errors of the order of magnitude of 100 meters or below. Previous algorithms were able to deliver target error distances of 2 nautical miles (3,704 meters) [6]. Moreover, the ability to deliver low distance errors with control times in the order of magnitude of seconds, allow for great flexibility and is quite astonishing when one considers that this is attained while controlling a vehicle that is travelling above the speed of sound. The base case control time interval is 0.1s, in line with typically reaction times measured in jet pilots (for moderate to large lateral airplane disturbances average reaction times are 0.23 second and for moderate longitudinal airplane disturbances average reactions times are 0.33 second) [8].

The algorithm was applied to the Space Shuttle re-entry, but can be applied to any vehicle (either ballistic or lift-enabled) and could easily be applied to the re-entry of the Solid Rocket Boosters used to launch the Space Shuttle and other vehicles into orbit. Current practices involve sending those vehicles to far unpopulated regions increasing the recovery cost somewhat. The ability to increase the precision on their navigation would allow them to be recovered closer to the launching site without risking the surrounding populations.

Future work on this control algorithm should focus on:

1. Upgrade the scope of the algorithm to include new factors and better describe reality
    a) Include stability and attitude analysis;
    b) Include heat radiation into the heat flux model;
    c) Include a moving non-flat Earth with wind.

2. Develop additional functionalities to enchance the range of applications of the algorithm
    a) Control the direction of the velocity at the target;
    b) Include the ability to perform final approach and landing;
    c) Include thrust capabilities (and thus variable mass).

3. Improving the core of the algorithm
    a) Lower the error reaching the target;
    b) Confirm the applicability to higher speeds (straight forward exercise should wind tunnel be readily available).

Regarding points 3a) (lowering error) and 2a) (defining velocity direction), we have initiated preliminary analysis using a "virtual target" and initial developments are presented in the section "Future evolutions of the algorithm". So far, in additional simulation not yet included on this thesis, the algorithm proved able to reaching both moving and stationary virtual targets.

The final approach and landing preliminary tests have also been done successfully. The path was modeled in such a way that the descent angle and speed was in the desired region, and a few meters above the ground, the stall angle was forced to enable a smooth transient before the landing gear hit the ground. The algorithm can be upgraded to perform this kind of maneuvers

through the usage of a series of target points (not just one target as modeled for the TAEM phase).

Finally, the algorithm can be extended to a non-flat moving Earth, for which new equations of motion will be needed. On this approach special attention is required when computing the lift and drag coefficients, as these will depend on the relative speed to the atmosphere and not on the overall speed (like we have done in the approach of the flat non-moving Earth).

References

[1] Costa, R. R.: Studies for terminal area GNC of reusable launch vehicles, AIAA, paper 2003-5438, 2003.

[2] Rehder, J. J., Holloway, P. F., Orbiter Entry Trajectory Considerations, NASA Langley Research Center, 1972.

[3] Vernis, P., Ferreira, E.: On-Board Trajectory Planner for the TAEM Guidance of a Winged-Body, EADS Space Transportation, 2004.

[4] Jiang, Z., Ordonez, R.: Trajectory Generation on Approach and Landing for RLVs Using Motion Primitives and Neighboring Optimal Control, Proceedings of the 2007 American Control Conference, 2007.

[5] Horneman, K., Automated Trajectory Generation and Guidance for a New Launch Vehicle Flight Phases, Faculty of the Graduate School - University of Missouri-Columbia, 2010.

[6] Mease, K. D., Teufel, P., Schonenberger, H., Chen, D. T., Bharadwaj, S., Re-entry trajectory planning for a reusable launch vehicle, AIAA paper 99-4160, 1999.

[7] Trelat, E.: Optimal Control of a Space Shuttle and Numerical Simulations, Proceedings of the Fourth International Conference on Dynamical Systems and Differential Equations, Wilmington NC USA, 2002.

[8] Helmut A. Kuehnel, In-flight measurement of the time required for a pilot to respond to an aircraft disturbance, NASA Langley Research Center, 1960.

[9] Ramsey, P.E.: Space Shuttle Aerodynamic Stability, Control Effectiveness and Drag Characteristics of a Shuttle Orbiter at Mac Numbers from 0.6 to 4.96, NASA/MSFC, 1972.

[10] Findlay, J. T., Kelly, G. M., Heck, M. L.: Reconstruction of the 1st Space Shuttle (STS-1) Entry Trajectory, NASA Contractor Report 3561, 1982.

[11] NASA video of the landing of Curiosity on Mars, 7 minutes of terror.
http://www.youtube.com/watch?v=h2I8AoB1xgU

[12] Musk E, SpaceX founder, What is the business model for Mars?
http://www.youtube.com/watch?v=4fS1FxBq64A

[13] NASA Marketing, Inflatable heat shield made of Kevlar,
http://www.nasa.gov/offices/oct/stp/game˙changing˙development/HIAD/irve3-prelaunch.html

[14] SpaceX, Falcon Heavy technical characteristics,
http://www.spacex.com/falcon˙heavy.php

[15] Newman D, MIT, A better built space suit,
http://www.youtube.com/watch?v=XfsmEYPSTtk

[16] NASA video of the landing of Spirit on Mars, How to get to Mars?
http://www.youtube.com/watch?v=XRCIzZHpFtY

[17] Findlay, J. T., Kelly, G. M., Heck, M. L.: Reconstruction of the 1st Space Shuttle (STS-1) Entry Trajectory, NASA Contractor Report 3561, 1982.

[18] Allen H J and Eggers A J, NACA, A study of the motion and aerodynamic heating of ballistic missiles entering the Earth's atmosphere at hypersonic speeds, 1957.

[19] Iliff K W and Shafer M F, NASA, Space Shuttle Hypersonic Aerodynamic and Aerothermic Flight Research and the Comparison to Ground Test Results, 1993.

[20] Dilao, Rui and Fonseca J, Trajectory Generation and Dynamic Control of Unpowered Vehicles returning from Space, Pre-Print, IST, 2012.

[21] Miele, A.: Flight Mechanics, Vol. I, Theory of Flight Paths, Addison-Wesley, Reading MA, 1962.

A Appendix

"Mathematica" Control Program

A.1 Mathematica Control Program: Structure and Logic

The "Mathematica" program is organized in a modular way, using global variables for the physics variables, and local variables for specific functions that return the result of their analysis at the end.

The "Main Area" contains the master code that will do a specific analysis calling the modular "Functions and Procedures" when needed. The "Procedures and Functions" area is to be run **before** any "Main Area" module is run.

"Main Area" modules

- A - 1D Range in stall and max glide
- B1 - Major part 2D Range in stall and max glide)
- B2 - Minor part 2D Range in stall and max glide)
- C1 - Positive x: Error reaching fixed target along x axis)
- C2 - Negative x: Error reaching fixed target along x axis)
- D - 2D Error reaching fixed target along xy plane)
- E1 - Typical Trajectory: Midpoint HAC)
- E2 - Typical Trajectory: Anti stall contribution)
- E3 - Typical Trajectory: Energy contribution)
- F1 - Sensitivity: Control Time Trajectories)
- F2 - Sensitivity: Control Time Error)
- G1 - Sensitivity: Error with $\gamma 0$)
- G2 - Sensitivity: Error with $\chi 0$)
- G3 - Sensitivity: Error with z0)
- G4 - Sensitivity: Error with V0)
- G5 - Sensitivity: Impact on Structural Limits)

"Functions and Procedures" Module

- 1 - Initiate variables
- 2 - Atmosphere: Density, Pressure and Temperature
- 3 - Aerodynamics: Max Glide, Lift, Drag, Bissetrix Method
- 4 - Control Algorithm : Attack and Bank Angles
- 5 - Motion Equation: Numerical Integration with RK 4th Order
- 6 - Motion Equation: Store motion data for analysis
- 7 - Motion Equation: Interpolation data for graphs
- 8 - Output: Diagnosis pack per target point

A.2 Full Mathematica Control Program

Main Programs

- **A - Main program (1D Range in stall and max glide)**

```
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Initial conditions *)
μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100; V0 = Ventry;
χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
(* Integration *)
dt = 0.1; dt2 = dt / 2; t = 0.0;
(*Target coordinates *)
xf = 300 000; yf = 0; zf = 3000.; error = 100.0;
dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;

(*First trajectory in Stall*)
InitiateVariables[];
While[ z0 ≥ zf,
      ControlAlgorithm[];
      α1 = alfastall; (*Overwrite decision*)
      For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
           RKT[];
           If[ z0 ≥ zf, StoreData[]];
           x0 = x1; y0 = y1; z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
InterpolateData[];
figu1 = ParametricPlot[{fx[t] / 1000, fz[t] / 1000}, {t, 0, tmax},
   AspectRatio → 1, Frame → True, FrameLabel → {"x (km)", "z (km)"},
   PlotStyle → Directive[Thickness[.005], Black],
   FrameStyle → Directive[Thickness[.005], 28], PlotRange → {{0, 300}, {0, 35}},
   FrameTicks → {{0, 100, 200, 300}, Automatic, None, None}];

(* Initial conditions *)
μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100; V0 = Ventry;
χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];

(*Second trajectory in Max Glide*)
InitiateVariables[];
While[ z0 ≥ zf,
      ControlAlgorithm[];
      α1 = alfamaxg; (*Overwrite decision*)
```

```
      For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
           RKT[];
           If[ z0 ≥ zf, StoreData[]];
           x0 = x1; y0 = y1; z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
InterpolateData[];
figu2 = ParametricPlot[{fx[t] / 1000, fz[t] / 1000}, {t, 0, tmax},
   AspectRatio → 1, Frame → True, FrameLabel → {"x (km)", "z (km)"},
   PlotStyle → Directive[Thickness[.005], Black],
   FrameStyle → Directive[Thickness[.005], 28], PlotRange → {{0, 300}, {0, 35}},
   FrameTicks → {{0, 100, 200, 300}, Automatic, None, None}];

tot2 = Show[{figu2, figu1},
  Epilog → {Text[Style["α_stall", 28], {80, 32}], Text[Style["a)", 28], {280, 33}],
    Text[Style["α_maxgl", 28], {230, 20}], {Dotted, Line[{{0, 3}, {300, 3}}]}}]
```

■ B1 - Main program (major part 2D Range in stall and max glide)

```
Rangemax = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
(* Points per sweep (will be double this number due to simmetry) *)
Npoints = 500;

For[j = 0, j ≤ Npoints, j++,
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*First trajectory in MaxGlide*)
  xf = 105 000. + 185 000. * Cos[2 j π / (2 * Npoints)] ;
  yf = 185 000 * Sin[2 j π / (2 * Npoints)] ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  memDateList = DateList[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       α1 = alfamaxg; (*Overwrite decision*)
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            RKT[];
            If[ z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  InterpolateData[];
  Rangemax = Append[Rangemax, {x1 / 1000, y1 / 1000}];
  Rangemax = Append[Rangemax, {x1 / 1000, -y1 / 1000}];
```

```
  (*Symetric in y (about x axis)*)
  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , j + 1, " of ",
   Npoints + 1, " cycles in MaxGlide. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (Npoints - j) / 60,
   " minutes or ", Elapsedsec * (Npoints - j) / 60 / 60, " hours."];
 ];

For[j = 0, j ≤ Npoints, j++,
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*First trajectory in Stall*)
  xf = 105000. + 185000. * Cos[2 j π / (2 * Npoints)] ;
  yf = 185000 * Sin[2 j π / (2 * Npoints)] ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  memDateList = DateList[];
  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       α1 = alfastall; (*Overwrite decision*)
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            RKT[];
            If[ z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  InterpolateData[];
  Rangemax = Append[Rangemax, {x1 / 1000, y1 / 1000}];
  Rangemax = Append[Rangemax, {x1 / 1000, -y1 / 1000}];
  (*Symetric in y (about x axis)*)
  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , j + 1, " of ",
   Npoints + 1, " cycles in Stall. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (Npoints - j) / 60,
   " minutes or ", Elapsedsec * (Npoints - j) / 60 / 60, " hours."];
 ];

g1 = ListPlot[Rangemax, PlotRange → {{-100, 300}, {-200, 200}},
   Epilog → {Point[{180, 90}],
     Text[Style["T", 28], {195, 100}], Text[Style["MR", 28], {230, 150}],
     Text[Style["SR", 28], {40, 60}], Text[Style["b)", 28], {280, 180}]},
   PlotStyle → Black, Frame → True, FrameLabel → {"x (km)", "y (km)"},
   FrameStyle → Directive[Thickness[.005], 28]];
Show[{g1}] // Print;
```

■ B2 - Main program (minor part 2D Range in stall and max glide)

```
Rangemax2 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
(* Points per sweep (will be double this number due to simmetry) *)
Npoints = 20;

For[j = 0, j ≤ Npoints, j++,
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*First trajectory in MaxGlide*)
  xf = -80 000. + 45 000. * j / Npoints ; yf = 0 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  memDateList = DateList[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       α1 = alfamaxg; (*Overwrite decision*)
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            RKT[];
            If[ z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];
  Rangemax2 = Append[Rangemax2, {x1 / 1000, y1 / 1000}];
  Rangemax2 = Append[Rangemax2, {x1 / 1000, -y1 / 1000}];
  (*Symetric in y (about x axis)*)
  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , j + 1, " of ",
   Npoints + 1, " cycles in MaxGlide. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (Npoints - j) / 60,
   " minutes or ", Elapsedsec * (Npoints - j) / 60 / 60, " hours."];
 ];

For[j = 0, j ≤ Npoints, j++,
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*First trajectory in Stall*)
  xf = -1. + 20 000. * j / Npoints ; yf = 0 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
```

```
  InitiateVariables[];
  memDateList = DateList[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       α1 = alfastall; (*Overwrite decision*)
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            RKT[];
            If[ z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];
  Rangemax2 = Append[Rangemax2, {x1 / 1000, y1 / 1000}];
  Rangemax2 = Append[Rangemax2, {x1 / 1000, -y1 / 1000}];
  (*Symetric in y (about x axis)*)
  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , j + 1, " of ",
   Npoints + 1, " cycles in Stall. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (Npoints - j) / 60,
   " minutes or ", Elapsedsec * (Npoints - j) / 60 / 60, " hours."];
 ];

g1A = ListPlot[Rangemax2, PlotRange → {{-100, 300}, {-200, 200}},
   Epilog → {Text[Style["MR", 28], {230, 150}], Text[Style["SR", 28], {40, 55}]},
   PlotStyle → Black, Frame → True, FrameLabel → {"x (km)", "y (km)"},
   FrameStyle → Directive[Thickness[.005], 28]];
Show[
 {g1,
  g1A}]

g2A = ListPlot[Rangemax2, PlotRange → {{-100, 300}, {-200, 200}},
   Epilog → {Text[Style["b)", 28], {280, 180}]},
   PlotStyle → Black, Frame → True, FrameLabel → {"x (km)", "y (km)"},
   FrameStyle → Directive[Thickness[.005], 28]];
Show[
 {g1,
  g2A}]

GraphicsRow[{tot2, Show[{g1, g2A}]}] // Print;
```

C1 - Main program (Positive x: Error reaching fixed target along x axis)

```
Erroaux = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

LinearRange = 268140;
Npoints = 1000;

For[j = 0, j ≤ Npoints, j++,
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Error sweep*)
  xf = 1. + LinearRange / Npoints * j ; yf = 0. ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  memDateList = DateList[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            RKT[];
            If[ z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];
  Erroaux = Append[Erroaux, {xf, error2}];
  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: ", j + 1, " of ",
   Npoints + 1, " cycles in sweep. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (Npoints - j) / 60,
   " minutes or ", Elapsedsec * (Npoints - j) / 60 / 60, " hours."];
 ];

Erroauxa = {};
For[i = 1, i ≤ Length[Erroaux], i++,

  Erroauxa = Append[Erroauxa, {Erroaux[[i, 1]] / 1000, Erroaux[[i, 2]]}];
 ];
```

```
g2 = ListPlot[Erroauxa, PlotRange → {{0, 300}, {0, 500}},
   AspectRatio → 1, Frame → True, FrameLabel → {"x  (km)", "e_d  (m)"},
   PlotStyle → Directive[Thickness[.005], Black],
   FrameStyle → Directive[Thickness[.005], 28],
   Epilog → {Text[Style["x_sr", 28], {70, 450}],
     Text[Style["x_mr", 28], {250, 450}], {Dotted,
      Line[{{268, 0}, {268, 1000}}], Line[{{80.657, 0}, {80.657, 1000}}]}},
   FrameTicks → {{0, 100, 200, 300}, Automatic, None, None}, Joined → False];
Show[{g2}]
```

■ C2 - Main program (Negative x: Error reaching fixed target along x axis)

```
Erroaux2 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

LinearRange = 30 000; (* Backwards direction *)
Npoints = 1000 / 10;

For[j = -Npoints, j ≤ 0, j++,
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Error sweep*)
  xf = 1. + LinearRange / Npoints * j ; yf = 0. ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  memDateList = DateList[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            RKT[];
            If[ z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];
  Erroaux2 = Append[Erroaux2, {xf, error2}];
  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , j, ". Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (-j) / 60,
   " minutes or ", Elapsedsec * (-j) / 60 / 60, " hours."];
 ];
```

```
Erroaux2a = {};
For[i = 1, i ≤ Length[Erroaux2], i++,

  Erroaux2a = Append[Erroaux2a, {Erroaux2[[i, 1]] / 1000, Erroaux2[[i, 2]]}];
 ];

g2A = ListPlot[Erroaux2a, PlotRange → {{-30, 300}, {0, 500}},
    AspectRatio → 1, Frame → True, FrameLabel → {"x  (km)", "e_d  (m)"},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.005], 28],
    Epilog → {Text[Style["x_sr", 28], {65, 300}],
      Text[Style["x_mr", 28], {250, 300}], Text[Style["a)", 28], {0, 450}], {Dotted,
       Line[{{268, 0}, {268, 1000}}], Line[{{80.657, 0}, {80.657, 1000}}]}},
    FrameTicks → {{0, 100, 200, 300}, Automatic, None, None}, Joined → False];
g2final = Show[{g2A, g2}]

SumError = 0; CntError = 0;
For[i = 1, i < Length[Erroaux], i++,
  SumError = SumError + Part[Erroaux, i][[2]];
  CntError = CntError + 1;];
For[i = 1, i < Length[Erroaux2], i++,
  SumError = SumError + Part[Erroaux2, i][[2]];
  CntError = CntError + 1;];
Print[SumError, " ", CntError, " ", SumError / CntError];
g2B = ListPlot[Erroaux2, PlotRange → {{-30 000, 300 000}, {0, 500}},
    AspectRatio → 1, Frame → True, FrameLabel → {"x (m)", "e_d (m)"},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.005], 14],
    Epilog → {Text[Style["x_sr", 15], {65 000, 300}],
      Text[Style["x_mr", 15], {286 000, 300}], {Dotted,
       Line[{{-30 000, SumError / CntError}, {268 000, SumError / CntError}}],
       Line[{{80 657, 0}, {80 657, 1000}}], Line[{{80 657, 0}, {80 657, 1000}}]}},
    FrameTicks → {{0, 100 000, 200 000, 300 000}, Automatic, None, None},
    Joined → False];
```

■ D - Main program (2D Error reaching fixed target along xy plane)

```
Erroaux2D = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
NumberCycles = 0;

Npoints = 40; (* Cycle j - Angular sweep --- has extra point in zero *)
Npoints2 = 80; (* Cycle k - Radius sweep *)

For[j = 0, j ≤ Npoints, j++,
  For[k = 1, k ≤ Npoints2, k++,
    (* Initial conditions *)
    μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
    x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
    V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
    dt = 0.1; dt2 = dt / 2; t = 0.0;
    (*Error sweep*)
    xf = 105 000. + 185 000. * k / Npoints2 * Cos[2 j π / (2 * Npoints)] ;
    yf = 185 000 * k / Npoints2 * Sin[2 j π / (2 * Npoints)] ; zf = 3000.0;
    error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
    InitiateVariables[];
    memDateList = DateList[];
    While[ z0 ≥ zf,
         ControlAlgorithm[];
         For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
              RKT[];
              If[ z0 ≥ zf, StoreData[]];
              x0 = x1; y0 = y1;
      z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
    tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) +
       (zf - z0) (zf - z0)];
    InterpolateData[];

    Erroaux2D = Append[Erroaux2D, {xf, yf, error2}]; Erroaux2D =
     Append[Erroaux2D, {xf, -yf, error2}]; (*Symetric in y (about x axis)*)
    Elapsed = DateList[] - memDateList;
    Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
    NumberCycles = NumberCycles + 1;
    Print["Finished cycle: ", NumberCycles, " of ",
     (Npoints + 1) (Npoints2), " cycles in sweep. Calculated in ",
     Elapsedsec, " seconds. Previewed time left: ",
     Elapsedsec * ((Npoints + 1) * (Npoints2) - NumberCycles) / 60, " minutes or ",
     Elapsedsec * ((Npoints + 1) * (Npoints2) - NumberCycles) / 60 / 60, " hours."];
   ];];

If[Erroaux2D[[i, 3]] ≤ 200,
```

```
Erroaux2Da = {};
For[i = 1, i ≤ Length[Erroaux2D], i++,
  Erroaux2Da = Append[Erroaux2Da,
     {Erroaux2D[[i, 1]] / 1000, Erroaux2D[[i, 2]] / 1000, Erroaux2D[[i, 3]]}];
 ];

gc1 = ListContourPlot[Erroaux2Da, MaxPlotPoints → 300,
   ContourStyle -> Thickness[.005], InterpolationOrder → 10,
   PerformanceGoal → "Quality", FrameLabel → {"x  (km)", "y  (km)"},
   Frame → True, Contours → {30, 100, 180}, ContourShading → None,
   FrameStyle → Directive[Thickness[.005], 28],
   Epilog → {Text[Style["b)", 28], {-40, 150}]}];
gc1

GraphicsRow[{g2final, gc1}] // Print;
```

■ E1 - Main program (Typical Trajectory: Mid point HAC)

```
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
(* Initial conditions *)
μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100; V0 = Ventry;
χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
dt = 0.1; dt2 = dt / 2; t = 0.0;
(*Target HAC*)
xf = 180 000; yf = 90 000 ; zf = 3000.0;
error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
InitiateVariables[];

While[ z0 ≥ zf,
      ControlAlgorithm[];
      For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
           RKT[];
           If[ z0 ≥ zf, StoreData[]];
           x0 = x1; y0 = y1; z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
InterpolateData[];

graph = ParametricPlot3D[{fx[t] / 1000, fy[t] / 1000, fz[t] / 1000},
   {t, 0, tmax}, AxesLabel → {"x  (km)", "y  (km)", "z  (km)"},
   AxesStyle → Directive[Thickness[.005], 28],
   PlotStyle → Directive[Thickness[.003]]];
Show[graph, {Graphics3D[{Text[Style["a)", 28], {100, -20, 3}]}]}] // Print;
```

```
GraphicsRow[{
    Plot[fz[t] / 1000, {t, 0, tmax}, PlotRange → All, Frame → True,
     FrameLabel → {"t  (s)", "z  (km)"}, Epilog → {PointSize[Medium],
       Black, Point[{tmax, fz[tmax]}], Black, Point[{100, fz[100] / 1000}],
       Point[{200, fz[200] / 1000}], Text[Style["b)", 28], {50, 10}]},
     PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.005], 28]],
    ParametricPlot[{fx[t] / 1000, fy[t] / 1000}, {t, 0, tmax},
     Frame → True, FrameLabel → {"x  (km)", "y  (km)"},
     Epilog → {PointSize[Medium], Black, Point[{fx[tmax], fy[tmax]}],
       Black, Point[{fx[100] / 1000, fy[100] / 1000}],
       Point[{fx[200] / 1000, fy[200] / 1000}], Text[Style["c)", 28], {25, 20}]},
     PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.005], 28], PlotRange → All]} // Print;

GraphicsRow[{Plot[{fV[t], fVst[t]},
     {t, 0, tmax}, Frame → True, FrameLabel → {"t  (s)", "V  (m/s)"},
     PlotRange → {0, 1.2 Ventry}, PlotStyle → {{Thickness[.005], Black},
       {Thickness[.005], Black, Dashing[{Medium, Medium}]}},
     FrameStyle → Directive[Thickness[.005], 28],
     Epilog → Text[Style["d)", 28], {50, 1200}]],
    Plot[{fLDmg[t], fLDst[t], fLD[t], falfatarget[t]},
     {t, 0, tmax}, Frame → True, FrameLabel → {"t  (s)", "L/D"},
     PlotRange → {0, 10}, PlotStyle → {{Thick, Gray}, {Thick, Gray},
       {Thick, Black}, {Dashing[{Medium, Medium}], Thick, Black}},
     PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.005], 28],
     Epilog → Text[Style["e)", 28], {50, 9.2}]]} // Print;

GraphicsRow[{
    Plot[fα[t], {t, 0, tmax}, PlotRange → {0, 1.0}, Frame → True, FrameLabel →
      {"t  (s)", "α  (rad)"}, PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.005], 28],
     Epilog → Text[Style["f)", 28], {50, .9}]],
    Plot[fμ[t], {t, 0, tmax}, PlotRange → {-π / 2, π / 2}, Frame → True, FrameLabel →
      {"t  (s)", "μ  (rad)"}, PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.005], 28],
     Epilog → Text[Style["g)", 28], {50, 1.3}]]} // Print;
```

■ E2 - Main program (Typical Trajectory: Anti stall contribution)

```
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
(* Initial conditions *)
μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100; V0 = Ventry;
χ0 = -3 / 360 * 2 * π; γ0 = 0; machini = V0 / Vsom[z0]; (*-3/360*2*π*)
dt = 0.1; dt2 = dt / 2; t = 0.0;
(*Target HAC*)
xf = 40 000; yf = 2000 ; zf = 3000.0;
error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
InitiateVariables[];

While[ z0 ≥ zf,
      ControlAlgorithm[];
      For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
           RKT[];
           If[ z0 ≥ zf, StoreData[]];
           x0 = x1; y0 = y1; z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
InterpolateData[];

GraphicsRow[{Plot[fz[t] / 1000, {t, 0, tmax}, PlotRange → All, Frame → True,
     FrameLabel → {"t  (s)", "z  (km)"}, Epilog → {PointSize[Medium], Black,
       Point[{tmax, fz[tmax] / 1000}], Black, Point[{100, fz[100] / 1000}],
       Point[{200, fz[200] / 1000}], Text[Style["a)", 28], {25, 10}]},
     PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.003], 28]],
    ParametricPlot[{fx[t] / 1000, fy[t] / 1000}, {t, 0, tmax}, AspectRatio → 1,
     Frame → True, FrameLabel → {"x  (km)", "y  (km)"}, Epilog →
      {PointSize[Medium], Black, Point[{fx[tmax] / 1000, fy[tmax] / 1000}],
       Black, Point[{fx[100] / 1000, fy[100] / 1000}],
       Point[{fx[200] / 1000, fy[200] / 1000}], Text[Style["b)", 28], {5, 2}]},
     PlotStyle → Directive[Thickness[.005], Black],
     FrameStyle → Directive[Thickness[.005], 28], PlotRange → All]}] // Print;
```

■ E3 - Main program (Typical Trajectory: Energy contribution)

```
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

(*First run with Energy on*)
(* Initial conditions *)
```

```
μ1 = 0.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0; (*0. 2 π/360.*)
x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 3300; V0 = Ventry;
χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
dt = 0.1; dt2 = dt / 2; t = 0.0;
(*Target HAC*)
xf = 180 000; yf = 90 000 ; zf = 3000.0;
error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
InitiateVariables[];
While[ z0 ≥ zf,
      ControlAlgorithm[];
      For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
           RKT[];
           If[ z0 ≥ zf, StoreData[]];
           x0 = x1; y0 = y1; z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
InterpolateData[];
g4A1 = Plot[fz[t] / 1000, {t, 0, tmax}, PlotRange → All, Frame → True,
   FrameLabel → {"t  (s)", "z  (km)"}, Epilog → {PointSize[Medium], Black,
     Point[{tmax, fz[tmax] / 1000}], Black, Point[{100, fz[100] / 1000}],
     Point[{200, fz[200] / 1000}], Text[Style["a)", 28], {50, 10}]},
   PlotStyle → Directive[Thickness[.005], Black],
   FrameStyle → Directive[Thickness[.003], 28]];
g4A2 = ParametricPlot[{fx[t] / 1000, fy[t] / 1000}, {t, 0, tmax},
   Frame → True, FrameLabel → {"x  (km)", "y  (km)"},
   Epilog → {PointSize[Medium], Black, Point[{fx[tmax] / 1000, fy[tmax] / 1000}],
     Black, Point[{fx[100] / 1000, fy[100] / 1000}],
     Point[{fx[200] / 1000, fy[200] / 1000}], Text[Style["b)", 28], {25, 20}]},
   PlotStyle → Directive[Thickness[.005], Black],
   FrameStyle → Directive[Thickness[.005], 28], PlotRange → All];

(*Second run with Energy off*)
Energybankon = 0;
(* Initial conditions *)
μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 3300; V0 = Ventry;
χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
dt = 0.1; dt2 = dt / 2; t = 0.0;
(*Target HAC*)
xf = 180 000; yf = 90 000 ; zf = 3000.0;
error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
InitiateVariables[];
While[ z0 ≥ zf,
      ControlAlgorithm[];
      For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
       If[control == 1,
         RKT[];
         If[ z0 ≥ zf, StoreData[];, control = 0];
         x0 = x1; y0 = y1; z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];];
tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
InterpolateData[];
g4B1 = Plot[fz[t] / 1000, {t, 0, tmax}, PlotRange → All, Frame → True,
```

```
    FrameLabel → {"t  (s)", "z  (km)"}, Epilog → {PointSize[Medium], Black,
      Point[{tmax, fz[tmax] / 1000}], Black, Point[{100, fz[100] / 1000}],
      Point[{200, fz[200] / 1000}], Text[Style["a)", 28], {50, 10}]},
    PlotStyle → Directive[Thickness[.005], Black, Dashing[{Medium, Medium}]],
    FrameStyle → Directive[Thickness[.003], 28]];
g4B2 = ParametricPlot[{fx[t] / 1000, fy[t] / 1000}, {t, 0, tmax},
    Frame → True, FrameLabel → {"x  (km)", "y  (km)"},
    Epilog → {PointSize[Medium], Black, Point[{fx[tmax] / 1000, fy[tmax] / 1000}],
      Black, Point[{fx[100] / 1000, fy[100] / 1000}],
      Point[{fx[200] / 1000, fy[200] / 1000}], Text[Style["b)", 28], {25, 20}]},
    PlotStyle → Directive[Thickness[.005], Black, Dashing[{Medium, Medium}]],
    FrameStyle → Directive[Thickness[.005], 28], PlotRange → All];

g41 = Show[{g4A1, g4B1}];

g42 = Show[{g4A2, g4B2}];

GraphicsRow[{g41, g42}] // Print;

DiagnosysPack[]
```

■ F1 - Main program (Sensitivity: Control Time Trajectories)

```
g5 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;
Npoints = 5;

memControlTime = {.1, 1., 10., 20., 40.}
For[i = 1, i ≤ Npoints, i++,
  memDateList = DateList[];
  ControlTime3 = memControlTime[[i]];
  (* Reinitiate the control time set in the default *)
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime3 && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i, " of ", Npoints,
   " cycles. Calculated in ", Elapsedsec, " seconds. Previewed time left: ",
   Elapsedsec * (Npoints - i) / 60, " minutes or ",
   Elapsedsec * (Npoints - i) / 60 / 60, " hours. Error: ", error2, " m."];

  g5A = ParametricPlot3D[{fx[t] / 1000, fy[t] / 1000, fz[t] / 1000},
    {t, 0, tmax}, AxesLabel → {"x  (km)", "y  (km)", "z  (km)"},
    PlotStyle → Directive[Thickness[.001], Black],
    BoxRatios → {1, 1, .5}, AxesStyle → Directive[Thickness[.003], 28]];
  g5 = Append[g5, g5A]
 ];
```

```
dot = ListPointPlot3D[{{xf / 1000, yf / 1000, zf / 1000}},
    PlotStyle → {Black, AbsolutePointSize[10]}];
Texta = Graphics3D[{Text[Style["a)", 28], {180, 10, 25}]}];
aa = Show[{g5, dot, Texta}]
```

■ F2 - Main program (Sensitivity: Control Time Error)

```
g6 = {};
g6A = {};
g6B = {};
g6C = {};

(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = .1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

Npoints = 200;
TimeControlMax = 40;

(* First run with all banks on *)
For[i = 1, i ≤ Npoints, i++,
  memDateList = DateList[];
  ControlTime2 = i / Npoints * TimeControlMax;
  (* Reinitiate the control time set in the default *)
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;
  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime2 && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i, " of ", Npoints, " cycles. Calculated in ",
   Elapsedsec, " seconds. Previewed time left: ", Elapsedsec * (Npoints - i) / 60,
   " minutes or ", Elapsedsec * (Npoints - i) / 60 / 60, " hours."];

  g6A = Append[g6A, {ControlTime2, error2}];
 ];
```

```
g6Afull = ListPlot[g6A, Frame → True,
   PlotRange → {{0, 40}, {0, 300}}, FrameLabel → {"T_con (s)", "e_d (m)"},
   PlotStyle → Directive[Thickness[.005], Black],
   FrameStyle → Directive[Thickness[.003], 28], Joined → True,
   Epilog → {Text[Style["b)", 28], {5, 250}]}];
Show[{g6Afull}]

GraphicsRow[{aa, g6Afull}] // Print;
```

■ G1 - Main program (Sensitivity: Error with $\gamma 0$)

```
g7 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

Npoints = 90;

For[i = -Npoints, i ≤ Npoints, i++,
  memDateList = DateList[];
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;

  γs = γ0 = i * π / 360; (* Reinitiate default initial γ0 *)

  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i, " of ",
   2 * Npoints + 1, " cycles. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (2 * Npoints - i) / 60,
   " minutes or ", Elapsedsec * (2 * Npoints - i) / 60 / 60, " hours."];

  g7 = Append[g7, {γs, error2}];
 ];

g7A = Table[{g7[[i, 1]] * 180 / π, g7[[i, 2]] / 1000}, {i, 1, Length[g7]}];

Print[N[ArcTan[(3 - 30) / Sqrt[90^2 + 180^2]] * 360 / 2 / π]]
```

```
-7.64141
```

```
g7Agraph = ListPlot[g7A, Frame → True, FrameLabel → {"γ0  (degree)", "ed  (km)"},
  FrameTicks → {{-90, -60, -30, 0, 30, 60 , 90}, Automatic},
  PlotStyle → Directive[Thickness[.005], Black],
  FrameStyle → Directive[Thickness[.005], 28], Joined → True,
  Epilog → {Text[Style["a)", 28], {80, 25}], Text[Style["γT", 14], {-20, 160}],
    {Dotted, Line[{{-7.64, 0}, {-7.64, 200}}]}}]
```

■ G2 - Main program (Sensitivity: Error with $\chi 0$)

```
g8 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

Npoints = 360 ;

For[i = 0, i ≤ Npoints, i++,
  memDateList = DateList[];
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;

  χs = χ0 = i * 2 π / 360; (* Reinitiate default initial γ0 *)

  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i, " of ", Npoints + 1, " cycles. Calculated in ",
   Elapsedsec, " seconds. Previewed time left: ", Elapsedsec * (Npoints - i) / 60,
   " minutes or ", Elapsedsec * (Npoints - i) / 60 / 60, " hours."];

  g8 = Append[g8, {χs, error2}];
 ];
```

```
Finished cycle: 0 of 721 cycles. Calculated in 75.518450
  seconds. Previewed time left: 453.11070 minutes or 7.5518450 hours.
```

```
$Aborted
```

```
g8A = Table[{g8[[i, 1]] * 180 / π, g8[[i, 2]] / 1000}, {i, 1, Length[g8]}];
```

```
Print[N[ArcCos[180 / Sqrt[90^2 + 180^2]] * 360 / 2 / π]]
```

```
26.5651
```

```
g8Agraph = ListPlot[g8A, Frame → True, FrameLabel → {"χ0  (degree)", "ed  (km)"},
  FrameTicks → {{0, 60, 120, 180, 240, 300 , 360}, Automatic},
  PlotStyle → Directive[Thickness[.005], Black],
  FrameStyle → Directive[Thickness[.005], 28], Joined → True,
  PlotRange → {{0, 360}, All}, Epilog → {Text[Style["b)", 28], {340, 25}],
    Text[Style["χT", 14], {55, 150}], {Dotted, Line[{{26.5, 0}, {26.5, 200}}]}}]
```

```
GraphicsRow[{g7Agraph, g8Agraph}] // Print;
```

■ G3 - Main program (Sensitivity: Error with z0)

```
g9 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

Npoints = 200;

For[i = -Npoints, i ≤ Npoints, i++,
  memDateList = DateList[];
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1; α1 = 30 / 360 * 2 * π; α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;

  zs = z0 = 40 000 + i * 100; (* Reinitiate default initial z0 *)

  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1; γ0 = γ1; χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i, " of ",
   2 * Npoints + 1, " cycles. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (2 * Npoints - i) / 60,
   " minutes or ", Elapsedsec * (Npoints - i) / 60 / 60, " hours."];

  g9 = Append[g9, {zs / 1000, error2 / 1000}];
 ];
g9graph = ListPlot[g9, Frame → True, FrameLabel → {"z₀  (km)", "e_d  (km)"},
  PlotStyle → Directive[Thickness[.005], Black],
  FrameStyle → Directive[Thickness[.005], 28], Joined → True,
  Epilog → {Text[Style["c)", 28], {39, 5}], Text[Style["z₀", 28], {29, 20}],
    {Dotted, Line[{{30, 0}, {30, 100}}]}}, PlotRange → {{20, 40}, All}]
```

```
DiagnosysPack[]
```

■ G4 - Main program (Sensitivity: Error with V0)

```
g10 = {};
(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

Npoints = 100;

For[i = -Npoints, i ≤ Npoints * 2, i++,
  memDateList = DateList[];
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1;  α1 = 30 / 360 * 2 * π;  α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;

  Vs = Ventry = V0 = 1100 + i * 10; (*  Reinitiate default initial V0 *)

  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1;  γ0 = γ1;  χ0 = χ1; α0 = α1; μ0 = μ1;];];
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i + Npoints, " of ",
   3 * Npoints + 1, " cycles. Calculated in ", Elapsedsec,
   " seconds. Previewed time left: ", Elapsedsec * (3 * Npoints - i) / 60,
   " minutes or ", Elapsedsec * (3 * Npoints - i) / 60 / 60, " hours."];

  g10 = Append[g10, {Vs, error2 / 1000}];
 ];
```

```
g10graph = ListPlot[g10, Frame → True,
  PlotRange → {{0, 3100}, {0, 120}}, FrameLabel → {"V0  (m/s)", "ed  (km)"},
  PlotStyle → Directive[Thickness[.005], Black],
  FrameStyle → Directive[Thickness[.005], 28],
  Joined → False, Epilog → {Text[Style["d)", 28], {2850, 15}],
    Text[Style["V0", 28], {950, 60}], {Dotted, Line[{{1100, 0}, {1100, 300}}]}}]

GraphicsRow[{g9graph, g10graph}]
```

■ G5 - Main program (Sensitivity: Impact on Structural Limits)

```
g11A = {};
g11Amem = {};
g11B = {};
g11Bmem = {};

g12A = {};
g12Amem = {};
g12B = {};
g12Bmem = {};

(* Algorithm *)
Antistallon = 1; Energybankon = 1; alfaprecision = 0.0001; alfastall = π / 4;
μmax = 70. / 360 * 2 * π; wfreq = 2 π * 1 / 120; ControlTime = 0.1; control = 1;
(* Constraints *)
VEnergy = 200; glim = 0.; Eloadlim = 5.; MhfluxLim = 5 * 10^5; α1min = 1.5 * 2 π / 360;
α1max = alfastall; tstag = 0; γ0t = 1.42; γ1t = -7 * 10^(-5); RN = 1;

Npoints = 3;

For[i = 1, i ≤ Npoints, i++,
  memDateList = DateList[];
  (* Initial conditions *)
  μ1 = 0. × 2 π / 360.; μ0 = μ1;  α1 = 30 / 360 * 2 * π;  α0 = α1; m1 = 0.0;
  x0 = 0.; y0 = 0.; z0 = 30 000.; Ventry = 1100;
  V0 = Ventry; χ0 = 0.0; γ0 = -3 / 360 * 2 * π; machini = V0 / Vsom[z0];
  dt = 0.1; dt2 = dt / 2; t = 0.0;
  (*Target HAC*)
  xf = 180 000; yf = 90 000 ; zf = 3000.0;
  error = 100.0; dis0 = dist[x0, y0, z0, xf, yf, zf]; dis1 = dis0;

  Vs = Ventry = V0 = 1100 * (1 + i / 2); (*  Reinitiate default initial V0 *)

  InitiateVariables[];
  While[ z0 ≥ zf,
       ControlAlgorithm[];
       For[t1 = dt, t1 ≤ ControlTime && z0 ≥ zf, t1 += dt,
            If[z0 > 0, RKT[]];
            If[z0 ≥ zf, StoreData[]];
            x0 = x1; y0 = y1;
    z0 = z1; V0 = V1;  γ0 = γ1;  χ0 = χ1; α0 = α1; μ0 = μ1;];];
```

```
  tmax = t; error2 = Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0) + (zf - z0) (zf - z0)];
  InterpolateData[];

  Elapsed = DateList[] - memDateList;
  Elapsedsec = Elapsed[[6]] + 60 * Elapsed[[5]];
  Print["Finished cycle: " , i, " of ", Npoints,
   " cycles. Calculated in ", Elapsedsec, " seconds. Previewed time left: ",
   Elapsedsec * (Npoints - i) / 60, " minutes or ",
   Elapsedsec * (Npoints - i) / 60 / 60, " hours. Flight time: ", tmax];

  g11A = Plot[fTstag[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t   (s)", "T_stag   (K)"}, PlotRange → {{0, tmax}, {0, 12 000}},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.005], 28]];
  g11Amem = Append[g11Amem, g11A];
  g11B = Plot[fmhf[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t   (s)", "ϕ_nose   (W/m^2)"}, PlotRange →
     {{0, tmax}, {0, 550 000}}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.005], 28]];
  g11Bmem = Append[g11Bmem, g11B];

  g12A = Plot[fml[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t   (s)", "n_load"}, PlotRange → {{0, tmax}, {0, 6}},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.005], 28]];
  g12Amem = Append[g12Amem, g12A];
  g12B = Plot[fglim[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t   (s)", "N_g"}, PlotRange → {{0, tmax}, {-5, 1}},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.005], 28]];
  g12Bmem = Append[g12Bmem, g12B];
 ];
a11 = Show[{g11Amem}, Epilog → {Text[Style["a)", 28], {460, 2000}]}];
a12 = Show[{g11Bmem},
   Epilog → {Text[Style["ϕ_Max", 28], {400, 450 000}], Text[Style["b)", 28],
      {460, 100 000}], {Dotted, Line[{{0, 500 000}, {500, 500 000}}]}}];
a13 = Show[{g12Amem }, Epilog → {Text[Style["n_Max", 28], {400, 5.5}],
     Text[Style["c)", 28], {460, 4}], {Dotted, Line[{{0, 5}, {500, 5}}]}}];
a14 = Show[{g12Bmem}, Epilog → {Text[Style["-g_lim", 28], {400, -2.5}], ,
     Text[Style["d)", 28], {460, -1}], {Dotted, Line[{{0, -3}, {500, -3}}]}}];

GraphicsRow[{a11, a12}] // Print;
GraphicsRow[{a13, a14}] // Print;
```

Support Procedures (run first)

- **1 - Initiate variables**
- **2 - Atmosphere: Density, Pressure and Temperature**

```
RE = 6.371 × 10^6; (*Earth radius*)
g[z_] := 9.80665 (RE / (RE + z))^2;
ρ[z_] := Module[{T0, λ0, P0, R, Ma, rho, P, T},
   R = 8.31432;
   Rsp = 287.04;
   Ma = 0.0289644;
   If[z ≤ 0., z = 0.;];
   (*1*)If[z ≥ 0. && z ≤ 11016., T0 = 288.15; λ0 = -0.0065; P0 = 101325.0;
    T = T0 + λ0 z; P = P0 (T0 / T)^(g[z] Ma / (R λ0)); rho = P / (Rsp T)];
   (*2*)If[z > 11016. && z ≤ 20063., T0 = 216.65; λ0 = 0.0; P0 = 22632.10;
    T = T0; P = P0 E^(-g[z] Ma (z - 11016.) / (R T)); rho = P / (Rsp T)];
   (*3*)If[z > 20063. && z ≤ 32162., T0 = 216.65; λ0 = 0.0010; P0 = 5474.89;
    T = T0 + λ0 (z - 20063.); P = P0 (T0 / T)^(g[z] Ma / (R λ0)); rho = P / (Rsp T)];
   (*4*)If[z > 32162. && z ≤ 47359., T0 = 228.65; λ0 = 0.0028; P0 = 868.02;
    T = T0 + λ0 (z - 32162.); P = P0 (T0 / T)^(g[z] Ma / (R λ0)); rho = P / (Rsp T)];
   (*5*)If[z > 47359. && z ≤ 51412., T0 = 270.65; λ0 = 0.0; P0 = 111.91;
    T = T0; P = P0 E^(-g[z] Ma (z - 47359.) / (R T)); rho = P / (Rsp T)];
   (*6*)If[z > 51412. && z ≤ 71802., T0 = 270.65; λ0 = -0.0028; P0 = 66.94;
    T = T0 + λ0 (z - 51412.); P = P0 (T0 / T)^(g[z] Ma / (R λ0)); rho = P / (Rsp T)];
   (*7*)If[z > 71802., T0 = 214.65; λ0 = -0.0020; P0 = 3.96;
    T = T0 + λ0 (z - 71802.); P = P0 (T0 / T)^(g[z] Ma / (R λ0)); rho = P / (Rsp T)];
   rho
  ];
Tem[z_] := Module[{T0, λ0, P0, R, Ma, rho, P, T},
   R = 8.31432;
   Ma = 0.0289644;
   If[z ≤ 0., z = 0.;];
   (*1*)If[z ≥ 0. && z ≤ 11016.,
    T0 = 288.15; λ0 = -0.0065; P0 = 101325.0; T = T0 + λ0 z;];
   (*2*)If[z > 11016. && z ≤ 20063., T0 = 216.65;
    λ0 = 0.0; P0 = 22632.10; T = T0;];
   (*3*)If[z > 20063. && z ≤ 32162., T0 = 216.65; λ0 = 0.0010;
    P0 = 5474.89; T = T0 + λ0 (z - 20063.);];
   (*4*)If[z > 32162. && z ≤ 47359., T0 = 228.65; λ0 = 0.0028;
    P0 = 868.02; T = T0 + λ0 (z - 32162.);];
   (*5*)If[z > 47359. && z ≤ 51412., T0 = 270.65; λ0 = 0.0; P0 = 111.91; T = T0;];
   (*6*)If[z > 51412. && z ≤ 71802.,
    T0 = 270.65; λ0 = -0.0028; P0 = 66.94; T = T0 + λ0 (z - 51412.);];
   (*7*)If[z > 71802., T0 = 214.65; λ0 = -0.0020;
    P0 = 3.96; T = T0 + λ0 (z - 71802.);];
   (* estava errado If[z> 71802.,T0=214.65;λ0=-0.0020;
      P0=3.96;T=T0+λ0(71802.- z);];*)
   T
```

```
  ];
Pre[z_] := Module[{T0, λ0, P0, R, Ma, rho, P, T},
   R = 8.31432;
   Ma = 0.0289644;
   If[z ≤ 0., z = 0.;];
   (*1*)If[z ≥ 0. && z ≤ 11016., T0 = 288.15; λ0 = -0.0065;
    P0 = 101325.0; T = T0 + λ0 z; P = P0 (T0 / T)^(g[z] Ma / (R λ0));];
   (*2*)If[z > 11016. && z ≤ 20063., T0 = 216.65; λ0 = 0.0;
    P0 = 22632.10; T = T0; P = P0 E^(-g[z] Ma (z - 11016.) / (R T));];
   (*3*)If[z > 20063. && z ≤ 32162., T0 = 216.65; λ0 = 0.0010; P0 = 5474.89;
    T = T0 + λ0 (z - 20063.); P = P0 (T0 / T)^(g[z] Ma / (R λ0));];
   (*4*)If[z > 32162. && z ≤ 47359., T0 = 228.65; λ0 = 0.0028; P0 = 868.02;
    T = T0 + λ0 (z - 32162.); P = P0 (T0 / T)^(g[z] Ma / (R λ0));];
   (*5*)If[z > 47359. && z ≤ 51412., T0 = 270.65; λ0 = 0.0;
    P0 = 111.91; T = T0; P = P0 E^(-g[z] Ma (z - 47359.) / (R T));];
   (*6*)If[z > 51412. && z ≤ 71802., T0 = 270.65; λ0 = -0.0028;
    P0 = 66.94; T = T0 + λ0 (z - 51412.); P = P0 (T0 / T)^(g[z] Ma / (R λ0));];
   (*7*)If[z > 71802., T0 = 214.65; λ0 = -0.0020; P0 = 3.96;
    T = T0 + λ0 (z - 71802.); P = P0 (T0 / T)^(g[z] Ma / (R λ0));];
   P
  ];

Vsom[z_] := Sqrt[1.4 Tem[z] Rsp];

p1 = Plot[ρ[z], {z, 0, 80000}, PlotRange → All,
  Frame → True, FrameLabel → {"Altitude (m)", "ρ (kg/m^3)"}]

p2 = Plot[Tem[z], {z, 0, 80000}, PlotRange → All,
  Frame → True, FrameLabel → {"Altitude (m)", "T (K)"}]

p3 = Plot[Pre[z], {z, 0, 80000}, PlotRange → All,
  Frame → True, FrameLabel → {"Altitude (m)", "P (Pa)"}]

GraphicsRow[{p1, p2, p3}]

Plot[Vsom[z], {z, 0, 80000}, PlotRange → All,
 Frame → True, FrameLabel → {"Altitude (m)", "V_Sound (m/s)"}]
```

3 - Aerodynamics: Max Glide, Lift, Drag, Bissetrix Method

```
Sur = 249.9; (*Surface*)
m = 82500.0; (*mass*)
a1 = -0.053; a2 = 2.73; a3 = -1.55; b1 = -1.01;
b2 = 1.1; d3 = 1.79; e1 = -1.4; e2 = 1.5;
f1 = 0.028; f2 = 1.4; Mc = 1.25;
CLift[α_, M_] :=
  (a1 + a2 α + a3 α α) (0.5 (1 + Sqrt[Abs[1 - (M / Mc)^2]]))^(b1 + α b2);
CDrag[α_, M_] := (0.01 + f1 M^f2 + d3 α α)
   (0.5 (1 + Sqrt[Abs[1 - (M / Mc)^2]]))^(e1 + α e2);
Amaxg[M_] := If[M ≤ 1.25, 0.0906 + 0.0573 M + 0.0071 M M,
   0.1070 + 0.0577 M - 0.0037 M M];
Newton[a_, xin_, M_, amg_, as_] :=
  Module[{x0 = xin, y0 = amg, y1 = as, eps = 10.0, i1 = 0, final = {0, 0}},
   While[eps > alfaprecision,
    i1++;
    If[((CLift[x0, M]) / (CDrag[x0, M]) - a) > 0, y0 = x0, y1 = x0;];
    x0 = (y0 + y1) / 2.0;
    eps = Abs[y1 - y0];
    final = {(y0 + y1) / 2.0, i1};
   ]; final];
Newton2[a_, αin_, M_, amg_, as_] :=
  Module[{x0 = αin, y0 = amg, y1 = as, eps = 10.0, i3 = 0, final = {0, 0}},
   While[eps > alfaprecision,
    i3++;
    If[(((CLift[x0, M] * Cos[x0] + CDrag[x0, M] * Sin[x0]) *
          (ρ[z0] * Sur * V0^2) / (2 * m * g[z0])) - a) < 0, y0 = x0, y1 = x0;];
    x0 = (y0 + y1) / 2.0;
    eps = Abs[y1 - y0];
    final = {(y0 + y1) / 2.0, i3};
   ]; final];
dist[x0_, y0_, z0_, x1_, y1_, z1_] :=
  Sqrt[(x1 - x0) (x1 - x0) + (y1 - y0) (y1 - y0) + (z1 - z0) (z1 - z0)];
```

4 - Control Algorithm : Attack and Bank Angles

```
(*======================== ATTACK ANGLE ========================*)
ControlAlgorithm[] := Module[{},
   (*Attack angle heading control *)
   gaux = (zf - z0) / Sqrt[(xf - x0) (xf - x0) + (yf - y0) (yf - y0)];
   clcdaux = -1. / (gaux);
   alfamaxg = Amaxg[V0 / Vsom[z0]];
   Iterateslinc = 0;
   Iteratesloadc = 0;

   (* Equilibrium approach*)
   If[clcdaux ≥ CLift[alfamaxg, V0 / Vsom[z0]] / CDrag[alfamaxg, V0 / Vsom[z0]],
        α1 = alfamaxg;,
        If[clcdaux ≤
       (CLift[alfastall, V0 / Vsom[z0]]) / CDrag[alfastall, V0 / Vsom[z0]],
```

```
        α1 = alfastall;,
             α1x = Newton[clcdaux, α0, V0 / Vsom[z0], alfamaxg, alfastall];
             α1 = α1x[[1]];
             Iterateslinc = α1x[[2]];];];
If[dist[x0, y0, z0, xf, yf, zf] < error && α0 == alfastall, α1 = alfastall;];
(* Fine tune to avoid "panic" inside the handover sphere *)

(* Load Factor Constraint in attack - IMPOSES A MAXIMUM ANGLE*)
EloadTest =
 (CLift[α1, V0 / Vsom[z0]] * Cos[α1] + CDrag[α1, V0 / Vsom[z0]] * Sin[α1]) *
  (ρ[z0] * Sur * V0^2) / (2 * m * g[z0]);
α1maxx = {0, 0};
If[EloadTest > Eloadlim,
         α1maxx = Newton2[Eloadlim, α1, V0 / Vsom[z0], 0, alfastall];
         α1max = α1maxx[[1]];
         α1 = Max[0, α1max];
         Iteratesloadc = α1maxx[[2]];

 Eload = (CLift[α1, V0 / Vsom[z0]] * Cos[α1] + CDrag[α1, V0 / Vsom[z0]] * Sin[α1]) *
   (ρ[z0] * Sur * V0^2) / (2 * m * g[z0]);
     , Eload = EloadTest;];

(* Heat Flux Constraint in attack - IMPOSES A MINIMUM ANGLE *)
MhfluxTest = 1.83 * 10^(-4) * (ρ[z0] / 1)^(1 / 2) * V0^3 * Cos[α1];
If[MhfluxTest > MhfluxLim,

 α1min = ArcCos[MhfluxLim / (1.83 * 10^(-4) * (ρ[z0] / 1)^(1 / 2) * V0^3)];
         α1 = Min[α1min, alfastall];
         Mhflux = 1.83 * 10^(-4) * (ρ[z0] / 1)^(1 / 2) * V0^3 * Cos[α1];
     , Mhflux = MhfluxTest;];
tstag =
 Tem[z0] * (1 + γ0t * (V0 / Vsom[z0])^2) / (1 - γ1t * (V0 / Vsom[z0])^2 * Tem[z0]);

(*======================== BANK ANGLE ========================*)

(*Bank angle heading control*)
arg = ((xf - x0) (V0 Cos[χ0] Cos[γ0]) + (yf - y0) (V0 Sin[χ0] Cos[γ0])) /
  Sqrt[((xf - x0)^2 + (yf - y0)^2) (V0 Cos[γ0])^2];
aux1 = Re[ArcCos[arg]];
ss = Sign[((xf - x0) (V0 Sin[χ0] Cos[γ0]) - (yf - y0) (V0 Cos[χ0] Cos[γ0]))];
If[ss == 0, ss = 1];
maux1 = -aux1 * ss;
If[dist[x0, y0, z0, xf, yf, zf] < error && Abs[maux1] > μmax, maux1 = 0;];

(* Anti-stall in bank angle*)
m1 = 0.0;
If[α1 == alfastall,
     arg = CLift[alfastall, V0 / Vsom[z0]] / CDrag[alfastall, V0 / Vsom[z0]];
     m1 = Re[ArcCos[-1 / (gaux * arg)]] * Antistallon;];
If[m1 > 0 && Sign[maux1mem] * Sign[maux1] == -1, maux1 = -1 * maux1];
(* Avoid yo-yo in anti-stall *)
```

```
If[dist[x0, y0, z0, xf, yf, zf] < error, m1 = 0;];
maux1mem = maux1;

(*Bank angle energy control*)
arg = Abs[2 * m * g[z0] Cos[γ0] / (ρ[z0] V0 V0 Sur CLift[α1, V0 / Vsom[z0]])];
aux2 = Re[ArcCos[arg]];
Vstnb = Sqrt[g[z0]] / (deltad2 + deltal2)^(1 / 4);
γstnb = -ArcTan[CDrag[α1, V0 / Vsom[z0]] / CLift[α1, V0 / Vsom[z0]]];
If[ (γ0 ≥ γstnb) && V0 - Vstnb ≥ VEnergy && V0 / Vsom[z0] ≥  Mc,
 maux2 = aux2 * Energybankon, maux2 = 0.0];

If[maux2mem == 0 && maux2 ≠ 0, Tsst = t;]; (*Start well the S-Turn*)
If[maux2mem == 0 && maux2 ≠ 0 && Sign[μ0] * Sign[Sin[wfreq * (t - Tsst + dt) + ϕ0]] ==
    -1, ϕ0 = ϕ0 + π;]; (*Start well the S-Turn*)
SturnSign = Sign[Sin[wfreq * (t - Tsst + dt) + ϕ0]];

(*Combine Anti-Stall,Energy and Heading Bank Angles*)
If[Abs[maux2] > Max[Abs[maux1], Abs[m1]],
 signal = SturnSign; maux2mem = maux2;,
 signal = Sign[maux1]; maux2mem = 0.0;];
μ1 = Max[Abs[maux1], Abs[maux2], Abs[m1]] * signal;

(*Limit m to operating area*)
If[Abs[μ1] > μmax, μ1 = Sign[μ1] * μmax];

];
```

■ 5 - Motion Equation: Numerical Integration with RKT 4th Order

```
RKT[] := Module[{},
   k1x = V0 Cos[χ0] Cos[γ0];
   k1y = V0 Sin[χ0] Cos[γ0];
   k1z = V0 Sin[γ0];
   k1V = -g[z0] Sin[γ0] - ρ[z0] V0  V0 Sur CDrag[α1, V0 / Vsom[z0]] / (2 m);
   k1γ = -g[z0] Cos[γ0] / V0 + Cos[μ1] ρ[z0]  Sur V0 CLift[α1, V0 / Vsom[z0]] / (2 m);
   k1χ = Sin[μ1] ρ[z0]  Sur V0 CLift[α1, V0 / Vsom[z0]] / (2 m Cos[γ0]);

   k2x = (V0 + dt2 k1V) Cos[χ0 + dt2 k1χ] Cos[γ0 + dt2 k1γ];
   k2y = (V0 + dt2 k1V) Sin[χ0 + dt2 k1χ] Cos[γ0 + dt2 k1γ];
   k2z = (V0 + dt2 k1V) Sin[γ0 + dt2 k1γ];
   k2V = -g[z0 + dt2 k1z] Sin[γ0 + dt2 k1γ] - ρ[z0 + dt2 k1z] (V0 + dt2 k1V)
       (V0 + dt2 k1V) Sur CDrag[α1, (V0 + dt2 k1V) / Vsom[z0 + dt2 k1z]] / (2 m);
   k2γ = -g[z0 + dt2 k1z] Cos[γ0 + dt2 k1γ] / (V0 + dt2 k1V) + Cos[μ1] ρ[z0 + dt2 k1z]
       Sur (V0 + dt2 k1V) CLift[α1, (V0 + dt2 k1V) / Vsom[z0 + dt2 k1z]] / (2 m);
   k2χ = Sin[μ1] ρ[z0 + dt2 k1z]  Sur (V0 + dt2 k1V)
     CLift[α1, (V0 + dt2 k1V) / Vsom[z0 + dt2 k1z]] / (2 m Cos[γ0 + dt2 k1γ]);

   k3x = (V0 + dt2 k2V) Cos[χ0 + dt2 k2χ] Cos[γ0 + dt2 k2γ];
   k3y = (V0 + dt2 k2V) Sin[χ0 + dt2 k2χ] Cos[γ0 + dt2 k2γ];
   k3z = (V0 + dt2 k2V) Sin[γ0 + dt2 k2γ];
   k3V = -g[z0 + dt2 k2z] Sin[γ0 + dt2 k2γ] - ρ[z0 + dt2 k2z] (V0 + dt2 k2V)
```

```
    (V0 + dt2 k2V) Sur CDrag[α1, (V0 + dt2 k2V) / Vsom[z0 + dt2 k2z]] / (2 m);
k3γ = -g[z0 + dt2 k2z] Cos[γ0 + dt2 k2γ] / (V0 + dt2 k2V) + Cos[μ1] ρ[z0 + dt2 k2z]
    Sur (V0 + dt2 k2V) CLift[α1, (V0 + dt2 k2V) / Vsom[z0 + dt2 k2z]] / (2 m);
k3χ = Sin[μ1] ρ[z0 + dt2 k2z] Sur (V0 + dt2 k2V)
  CLift[α1, (V0 + dt2 k2V) / Vsom[z0 + dt2 k2z]] / (2 m Cos[γ0 + dt2 k2γ]);

k4x = (V0 + dt k3V) Cos[χ0 + dt k3χ] Cos[γ0 + dt k3γ];
k4y = (V0 + dt k3V) Sin[χ0 + dt k3χ] Cos[γ0 + dt k3γ];
k4z = (V0 + dt k3V) Sin[γ0 + dt k3γ];
k4V = -g[z0 + dt k3z] Sin[γ0 + dt k3γ] - ρ[z0 + dt k3z] (V0 + dt k3V)
    (V0 + dt k3V) Sur CDrag[α1, (V0 + dt k3V) / Vsom[z0 + dt k3z]] / (2 m);
k4γ = -g[z0 + dt k3z] Cos[γ0 + dt k3γ] / (V0 + dt k3V) + Cos[μ1] ρ[z0 + dt k3z]
    Sur (V0 + dt k3V) CLift[α1, (V0 + dt k3V) / Vsom[z0 + dt k3z]] / (2 m);
k4χ = Sin[μ1] ρ[z0 + dt k3z] Sur (V0 + dt k3V)
  CLift[α1, (V0 + dt k3V) / Vsom[z0 + dt k3z]] / (2 m Cos[γ0 + dt k3γ]);

x1 = x0 + dt (k1x + 2. k2x + 2.0 k3x + k4x) / 6.0;
y1 = y0 + dt (k1y + 2. k2y + 2.0 k3y + k4y) / 6.0;
z1 = z0 + dt (k1z + 2. k2z + 2.0 k3z + k4z) / 6.0;
V1 = V0 + dt (k1V + 2. k2V + 2.0 k3V + k4V) / 6.0;
γ1 = γ0 + dt (k1γ + 2. k2γ + 2.0 k3γ + k4γ) / 6.0;
χ1 = χ0 + dt (k1χ + 2. k2χ + 2.0 k3χ + k4χ) / 6.0;

If[γ1 > π / 2, γ1 = -π + γ1, If[γ1 < -π / 2, γ1 = π + γ1]];
If[χ1 > 2 π , χ1 = χ1 - 2 π, If[χ1 < 0, χ1 = χ1 + 2 π]];

dis1 = dist[x1, y1, z1, xf, yf, zf];
dis0 = dist[x0, y0, z0, xf, yf, zf];

(*g Limit Constraints *)
glim = ((V1 - V0) / dt) / g[0];

(* Accuracy Test RKT*)
Accuracytest = (((k1V + 2. k2V + 2.0 k3V + k4V) / 6.0 +
        1 / (2 m) * ρ[z1] * Sur * CDrag[α1, V1 / Vsom[z1]] * V1^2)^2 +
     (V1 * (k1γ + 2. k2γ + 2.0 k3γ + k4γ) / 6.0 - 1 / (2 m) * ρ[z1] * Sur *
         CLift[α1, V1 / Vsom[z1]] * V1^2 * Cos[μ1])^2) / g[z1]^2;

t = t + dt;
];
```

■ 6 - Motion Equation: Store motion data for analysis

```
StoreData[] := Module[{},
   Tx = Append[Tx, {{t}, x1}];
   Ty = Append[Ty, {{t}, y1}];
   Tz = Append[Tz, {{t}, z1}];
   TV = Append[TV, {{t}, V1}];
   Tγ = Append[Tγ, {{t}, γ1}];
   Tχ = Append[Tχ, {{t}, χ1}];
   TMach = Append[TMach, {{t}, V1 / Vsom[z1]}];
   Tα = Append[Tα, {{t}, α1}];
   Tμ = Append[Tμ, {{t}, μ1}];
   TμH = Append[TμH, {{t}, maux1}];
   TμA = Append[TμA, {{t}, m1}];
   TμE = Append[TμE, {{t}, maux2}];

   TLD = Append[TLD, {{t}, CLift[α1, V1 / Vsom[z1]] / CDrag[α1, V1 / Vsom[z1]]}];
   TLDmg = Append[TLDmg, {{t}, CLift[Amaxg[V1 / Vsom[z1]], V1 / Vsom[z1]] /
       CDrag[Amaxg[V1 / Vsom[z1]], V1 / Vsom[z1]]}];
   TLDst = Append[TLDst, {{t}, CLift[alfastall, V1 / Vsom[z1]] /
       CDrag[alfastall, V1 / Vsom[z1]]}];
   Talfatarget = Append[Talfatarget, {{t}, clcdaux}];
   Tml = Append[Tml, {{t}, Eload}];
   Tmhf = Append[Tmhf, {{t}, Mhflux}];
   Tglim = Append[Tglim, {{t}, glim}];
   Ttstag = Append[Ttstag, {{t}, tstag}];
   TerrorRKT = Append[TerrorRKT, {{t}, Accuracytest - 1}];
   Iterateslin = Append[Iterateslin, {{t}, Iterateslinc}];
   Iteratesload = Append[Iteratesload, {{t}, Iteratesloadc}];

   (*Equilibrium velocity*)
   deltal2 = (ρ[z1] Sur CLift[α1, V1 / Vsom[z1]] / (2 m))^2;
   deltad2 = (ρ[z1] Sur CDrag[α1, V1 / Vsom[z1]] / (2 m))^2;
   Vst = Sqrt[g[z1]] / (deltad2 + deltal2 * Cos[μ1]^2)^(1 / 4);
   TVst = Append[TVst, {{t}, Vst}];
  ];
```

■ 7 - Motion Equation: Interpolation data for graphs

```
InterpolateData[] := Module[{},
   fx = ListInterpolation[Tx];
   fy = ListInterpolation[Ty];
   fz = ListInterpolation[Tz];
   fV = ListInterpolation[TV];
   fγ = ListInterpolation[Tγ];
   fχ = ListInterpolation[Tχ];
   fMach = ListInterpolation[TMach];
   fα = ListInterpolation[Tα];
   fμ = ListInterpolation[Tμ];
   fμH = ListInterpolation[TμH];
   fμA = ListInterpolation[TμA];
   fμE = ListInterpolation[TμE];
   fLD = ListInterpolation[TLD];
   fLDmg = ListInterpolation[TLDmg];
   fLDst = ListInterpolation[TLDst];
   falfatarget = ListInterpolation[Talfatarget];
   fVst = ListInterpolation[TVst];
   fml = ListInterpolation[Tml];
   fmhf = ListInterpolation[Tmhf];
   fglim = ListInterpolation[Tglim, InterpolationOrder → 0];
   fTstag = ListInterpolation[Ttstag];
   fTerrorRKT = ListInterpolation[TerrorRKT];
   fIterateslin = ListInterpolation[Iterateslin];
   fIteratesload = ListInterpolation[Iteratesload];
  ];
```

■ 8 - Output: Diagnosis pack per target point

```
DiagnosysPack[] := Module[{}, (*Long Pack*)
   Print[
    "*****************************************************************"];
   Print["Ventry = ", Ventry];
   Print["Antistall_on: ", Antistallon, "  Energybank_on: ", Energybankon];
   Print["x_final = ", x1, "  y_final = ", y1, "  z_final = ", z1];
   Print["x_target = ", xf, "  y_target = ", yf, "  z_target = ", zf];
   Print["Current Speed = ", V0];
   Print["Falling time = ", tmax, "    Error =",
    error2, " Steps error = ", error2 / (V0 * ControlTime)];
   Print["Last Trajectory Angle = ", γ1 * 360 / (2 π), " º"];
   Print[
    "*****************************************************************"];
   (*Invariant*)
   Print["Equation Invariant Test"];
   Plot[fTerrorRKT[t] * 100, {t, 0, tmax}, PlotRange → {-4, 4},
     Frame → True, FrameLabel → {"t(s)", "Error % "}] // Print;

   SumErrorRKT = 0; CntErrorRKT = 0;
   For[i = 1, i < Length[TerrorRKT], i++,
```

```
 SumErrorRKT = SumErrorRKT + Abs[Part[TerrorRKT, i][[2]]];
 CntErrorRKT = CntErrorRKT + 1;];

Print["RKT Invariant error statistics - Total Error: ",
 SumErrorRKT, ", # Points: ", CntErrorRKT, ", Avg Error: ",
 SumErrorRKT / CntErrorRKT, "(or ", SumErrorRKT / CntErrorRKT * 100, " %)"];

(*Iterates*)
Print["# of Iterates control (performance)"];
GraphicsRow[{Plot[fIterateslin[t], {t, 0, tmax}, PlotRange → All,
    Frame → True, FrameLabel → {"t(s)", "# Iterates Attack Linear"},
    AxesStyle → Directive[Thickness[.003], 18],
    PlotStyle → Directive[Thickness[.003]]],
   Plot[fIteratesload[t], {t, 0, tmax}, PlotRange → All, Frame → True,
    FrameLabel → {"t(s)", "# Iterates Attack Angle Load"}]},
  AxesStyle → Directive[Thickness[.003], 18]] // Print;
(*Trajectory*)
Print["Trajectory"];
Lista = Table[{fx[t], fy[t], fz[t]}, {t, 0., tmax}];
ListPointPlot3D[Lista, AxesStyle → Directive[Thickness[.003], 18],
  PlotStyle → Directive[Thickness[.003]],
  AxesLabel → {"x(m)", "y(m)", "z(m)"}] // Print;
ParametricPlot3D[{fx[t], fy[t], fz[t]}, {t, 0, tmax}, AxesLabel →
   {"x(m)", "y(m)", "z(m)"}, AxesStyle → Directive[Thickness[.003], 18],
  PlotStyle → Directive[Thickness[.003]]] // Print;
GraphicsRow[{
   Plot[fz[t], {t, 0, tmax}, PlotRange → All,
    Frame → True, FrameLabel → {"t  (s)", "z  (m)"},
    Epilog → {PointSize[Medium], Black, Point[{tmax, fz[tmax]}],
      Black, Point[{100, fz[100]}], Point[{200, fz[200]}]},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 18]],
   ParametricPlot[{fx[t], fy[t]}, {t, 0, tmax}, AspectRatio → 1,
    Frame → True, FrameLabel → {"x  (m)", "y  (m)"},
    Epilog → {PointSize[Medium], Black, Point[{fx[tmax], fy[tmax]}],
      Black, Point[{fx[100], fy[100]}], Point[{fx[200], fy[200]}]},
    PlotStyle → Directive[Thickness[.005], Black], FrameStyle →
     Directive[Thickness[.005], 18], PlotRange → All]}] // Print;
(*Other trajectory*)
Print["Trajectory support"];
GraphicsRow[{Plot[{fV[t], fVst[t]}, {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "v(m/s)"}, PlotRange → {0, 1.2 Ventry},
    PlotStyle → {{Thickness[.005], Black}, {Thickness[.005], Black, Dotted}},
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[fV[t] / fVst[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "v/v* (m/s)"}, PlotRange → {0, fV[0] / fVst[0]},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;
GraphicsRow[{Plot[fMach[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "Ma"}, PlotRange → {{0, tmax}, {0, 1.2 machini}},
    PlotStyle → Directive[Thickness[.005], Black],
```

```
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[fγ[t], {t, 0, tmax}, Frame → True, FrameLabel → {"t(s)", "γ(rad)"},
    PlotRange → All, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;
(*Control Summary*)
Print["Control History"];
GraphicsRow[{
   Plot[fα[t], {t, 0, tmax}, PlotRange → {0, 1.0}, Frame → True, FrameLabel →
     {"t  (s)", "α  (rad)"}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 18]],
   Plot[fμ[t], {t, 0, tmax}, PlotRange → {-π / 2, π / 2},
    Frame → True, FrameLabel → {"t  (s)", "μ  (rad)"},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 18]]}] // Print;
(*Bank Detail (ABS) and normal*)
Print["Bank Angle Zoom-in"];
Show[{
   Plot[Abs[fμH[t]], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "μH(rad)"}, PlotRange → {-π / 2, π / 2},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[Abs[fμA[t]], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "μA(rad)"}, PlotRange → {-π / 2, π / 2},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[Abs[fμE[t]], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "μE(rad)"}, PlotRange → {-π / 2, π / 2},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;
GraphicsRow[{
   Plot[fμH[t], {t, 0, tmax}, Frame → True, FrameLabel → {"t(s)", "μH(rad)"},
    PlotRange → {-π / 2, π / 2}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[fμA[t], {t, 0, tmax}, Frame → True, FrameLabel → {"t(s)", "μA(rad)"},
    PlotRange → {-π / 2, π / 2}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[fμE[t], {t, 0, tmax}, Frame → True, FrameLabel → {"t(s)", "μE(rad)"},
    PlotRange → {-π / 2, π / 2}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;
(*Attack Detail*)
Print["Attack Angle Zoom-in"];
Plot[{fLDmg[t], fLDst[t], fLD[t], falfatarget[t]}, {t, 0, tmax},
  Frame → True, FrameLabel → {"t(s)", "L/D"}, PlotRange → {0, 10},
  PlotStyle → {{Dotted, Black}, {Dotted, Black}, Red, Green},
  PlotStyle → Directive[Thickness[.005], Black],
  FrameStyle → Directive[Thickness[.003], 14]] // Print;
(*Structural Limits*)
Print["Structural Limits Zoom-in"];
GraphicsRow[{Plot[fTstag[t], {t, 0, tmax}, Frame → True, FrameLabel →
     {"t(s)", "T_stag(K°)"}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]], Plot[fmhf[t],
```

```
    {t, 0, tmax}, Frame → True, FrameLabel → {"t(s)", "ϕnoose  (W/m²)"},
    PlotRange → All, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;
 GraphicsRow[{Plot[fml[t], {t, 0, tmax}, Frame → True,
    FrameLabel → {"t(s)", "nload"}, PlotRange → All,
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[fglim[t], {t, 0, tmax}, Frame → True, FrameLabel → {"t(s)", "Ng"},
    PlotRange → All, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;

 Print["Extra"];
 GraphicsRow[{Plot[{fV[t], fVst[t]}, {t, 0, tmax}, Frame → True,
    FrameLabel → {"t  (s)", "V  (m/s)"}, PlotRange → {0, 1.2 Ventry},
    PlotStyle → {{Thickness[.005], Black}, {Thickness[.005], Black, Dotted}},
    FrameStyle → Directive[Thickness[.003], 18]],
   Plot[{fLDmg[t], fLDst[t], fLD[t], falfatarget[t]}, {t, 0, tmax},
    Frame → True, FrameLabel → {"t  (s)", "L/D"}, PlotRange → {0, 10},
    PlotStyle → {{Gray}, {Gray}, {Thick, Black}, {Dashed, Black}},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 18]]}] // Print;
];
```

■ 8A - Output: Mini Diagnosis pack per target point

```
DiagnosysPackMini[] := Module[{}, (*Quick Pack*)
 Print[
  "*****************************************************************"];
 Print["Ventry = ", Ventry];
 Print["Antistall_on: ", Antistallon, "  Energybank_on: ", Energybankon];
 Print["xfinal = ", x1, "  yfinal = ", y1, "  zfinal = ", z1];
 Print["xtarget = ", xf, "  ytarget = ", yf, "  ztarget = ", zf];
 Print["Current Speed = ", V0];
 Print["Falling time = ", tmax, "    Error =",
  error2, " Steps error = ", error2 / (V0 * ControlTime)];
 Print["Last Trajectory Angle = ", γ1 * 360 / (2 π), " °"];
 Print[
  "*****************************************************************"];

 Print["Control History"];
 GraphicsRow[{
   Plot[fα[t], {t, 0, tmax}, PlotRange → {0, 1.0}, Frame → True, FrameLabel →
     {"t  (s)", "α (rad)"}, PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]],
   Plot[fμ[t], {t, 0, tmax}, PlotRange → {-π / 2, π / 2},
    Frame → True, FrameLabel → {"t  (s)", "μ (rad)"},
    PlotStyle → Directive[Thickness[.005], Black],
    FrameStyle → Directive[Thickness[.003], 14]]}] // Print;
];
```

Reference: Functions and procedures List

Block 1 – InitiateVariables[]
Block 2 – Not Applicable (is a set of Functions)
Block 3 – Not Applicable (is a set of Functions)
Block 4 – ControlAlgorithm[]
Block 5 – RKT[]
Block 6 – StoreData[]
Block 7 – InterpolateData[]
Block 8 – DiagnosysPack[]
Block 8 A – DiagnosysPackMini[]