Guidance for Precision Landing on Titan Using Optimization Techniques.

Ridma Ganganath*
Utah State University, Logan, Utah, 84321

I. Nomenclature

α	angle-of-attack, rad	A	state coefficient matrix	m	spacecraft mass, kg
β	angle of sideslip, rad	$\mathcal{A}$	aerodynamic frame	M	moment vector, N·m
γ_{gs}	glide-slope constraint, rad	a	parafoil height, m	n_i	i-th-axis unit vector
δ_a	asymmetric deflection, rad	$\mathbf{B}$	control coefficient matrix	$\mathbf{q}$	quaternion vector
δ_s	symmetric deflection, rad	$\mathcal{B}$	body frame	$\mathbf{r}$	position vector, m
ϵ	anhedral angle, rad	b	parafoil wingspan, m	$\mathbf{v}$	velocity vector, m/s
θ	pointing angle, rad	C_y	aerodynamic coefficient C_l	R	parafoil line length, m
μ	rigging angle, rad	$\mathbf{C_{B\|I}}$	direction cosine matrix	S_{ref}	reference area, m^2
ν	virtual control vector	c	parafoil chord, m	t	time, s
ρ	density, kg/m^3	F	force vector, N	t	parafoil thickness, m
χ	heading angle, rad	g	apparent	$\mathbf{u}$	control vector
τ	normalized trajectory time	$\mathbf{g}$	gravity acceleration, m/s^2	$\mathbf{x}$	state vector
Φ	state transition matrix	I	inertia tensor, kg·m^2	$[.X]$	skew-symmetric
ω	angular velocity vector, rad/s	$\mathcal{I}$	inertial frame	$\|\| \cdot \|\|$	vector norm

II. Introduction

Precision landing technology is becoming crucial for future planetary missions. This paper examines an autonomous spacecraft's Entry, Descent, and Landing (EDL) on a planetary surface, aiming for precision within several meters—a significant challenge. In this analysis, I specifically focused on the non-convex continuous-time minimum control effort guidance problem as presented in the study by Mazouz, Quadrelli, and Mooij (2021) [1], critically examining the methodologies and results discussed in their pioneering work on precision landing on Titan. Given Titan's unique geophysical characteristics, it is a primary focus for NASA's planetary science, especially its atmospheric and subsurface properties. The Space Exploration Technology Directorate recommends using a parafoil for landing on Titan due to its cost-efficiency, straightforward deployment, lightweight relative to its payload, and precise delivery capabilities. This study focuses on optimizing flight paths and developing guidance laws to tune parafoils for high-fidelity dynamics within Titan's dense and complex wind environments. These environments pose a nonlinear and nonconvex optimal control problem. The resolution of this six-degree-of-freedom (6DoF) control problem starts with an initially dynamically inconsistent trajectory, made feasible through a virtual control vector. The algorithm undergoes successive iterations to ensure the final solution adheres to the original nonlinear dynamics and kinematic conditions while complying with state and control constraints. A pointing constraint is also incorporated to meet Terrain Relative Navigation (TRN) requirements. This method is deemed robust and well-suited for autonomous interplanetary applications.

III. Dynamics Formulation

This section presents the dynamics and aerodynamics of the parafoil landing challenge, characterized by a six-degree-of-freedom (6DoF) nonconvex model. Additionally, the section defines several reference frames necessary for problem formulation.

$\mathcal{A}$: The aerodynamic frame defines two crucial angles: the angle-of-attack (α) and the side-slip angle

*Graduate Research Assistant, Department of Mechanical and Aerospace

(β). Their relative positioning is illustrated in Figure 1.

$\mathcal{B}$: The body frame is attached to the vehicle and rotates with it. This rotation, compared to the inertial frame, is represented by the angular rate vector (ω).

$\mathcal{I}$: The inertial frame is centered on the landing target, assuming a fixed position relative to Titan's surface with an Up-East-North alignment, making it effectively inertial.

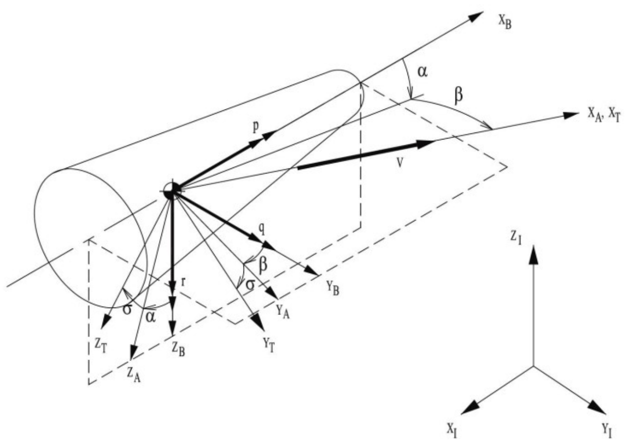


Fig. 1 The positive aerodynamic angles α and β describe how the body frame B is related to the aerodynamic frame $\mathcal{A}$.

A. Spacecraft Dynamics and Kinematics.

The spacecraft, modeled as a rigid parafoil-payload system, operates at low altitudes where it experiences a constant gravitational force $g_{\mathcal{I}}$. Its mass remains constant over time. The total force on the spacecraft, expressed in frame $\mathcal{I}$ as $\mathbf{F}_{\mathcal{I}}$, and both position ($\mathbf{r}_{\mathcal{I}} \in \mathbb{R}^3$) and velocity ($\mathbf{v}_{\mathcal{I}} \in \mathbb{R}^3$) are described in this same coordinate system. The translational dynamics are governed by these elements.

$$\dot{\mathbf{r}}_I(t) = \mathbf{v}_I(t) \tag{1}$$

$$\dot{\mathbf{v}}_I(t) = \frac{\mathbf{F}_I(t)}{m} + \mathbf{g}_I \tag{2}$$

The spacecraft utilizes a unit quaternion for attitude representation to avoid singularities. This represents the orientation of the B-frame with respect to the I-frame and is expressed as $q_{B|I} \in \mathbb{S}^3$.

$$q_{B|I}(t) = \begin{pmatrix} \cos\left(\frac{\xi}{2}\right) \\ \sin\left(\frac{\xi}{2}\right)\hat{n} \end{pmatrix} = \begin{pmatrix} q_1 & q_1 & q_2 & q_4 \end{pmatrix}^T$$

The attitude's parametrization utilizes three vector components on $\mathbb{S}^3$ and extends into $\mathbb{R}^4$ with a scalar. The Direction Cosine Matrix (DCM) $C_{I,\beta}$ denotes the quaternion attitude change from the B-frame to the I-frame.

$$C_{I,\mathcal{B}} = \begin{bmatrix} q_1^2 + q_1^2 - q_2^2 - q_4^2 & 2(q_1q_2 - q_1q_4) & 2(q_1q_4 + q_1q_2) \\ 2(q_1q_2 + q_1q_4) & q_1^2 - q_1^2 + q_2^2 - q_4^2 & 2(q_2q_4 - q_1q_1) \\ 2(q_1q_4 - q_1q_2) & 2(q_2q_4 + q_1q_1) & q_1^2 - q_1^2 - q_2^2 + q_4^2 \end{bmatrix}$$

Using DCM, $F_{\mathcal{I}}$ is determined by the net $\mathcal{B}$-frame force, predominantly aerodynamic.

$$F_I = C_{I,\beta}F_B \tag{3}$$

Post $\mathcal{B}$-frame transformation, the generalized relation is established:

$$F_B = \sum_{i=1}^{n} F_i \tag{4}$$

We denote $\boldsymbol{\omega}_B \in \mathbb{R}^3$ for the $\mathcal{B}$-frame's angular velocity about the $\mathcal{I}$-frame, with its skew-symmetric matrices $[\boldsymbol{\omega}_B\times]$ and $\Omega(\boldsymbol{\omega}_B)$ detailed subsequently as:

$$[\boldsymbol{\omega}_B\times] \triangleq \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \quad \Omega(\boldsymbol{\omega}_B) \triangleq \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

Total spacecraft moment $M_B \in \mathbb{R}^3$ and inertia tensor I_B in the $\mathcal{B}$-frame are pivotal for dynamics and kinematics calculations.

$$\mathbf{I}_{\mathcal{B}}\dot{\boldsymbol{\omega}}_{\mathcal{B}}(t) = \mathbf{M}_{\mathcal{B}} - [\boldsymbol{\omega}_{\mathcal{B}}\times]\mathbf{I}_{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}} \tag{5}$$

$$\dot{q}_{\mathcal{B}|I}(t) = \frac{1}{2}\Omega(\boldsymbol{\omega}_{\mathcal{B}})q_{\mathcal{B}|I} \tag{6}$$

During landing, assumed the parafoil-payload's Center-of-Mass (CoM) is constant, ensuring a constant aerodynamic moment arm in the $\mathcal{B}$-frame. The resultant torque M_F is computed using Equation 7, where $[r_{F,\mathcal{B}}\times]$ is skew-symmetric.

$$M_{F,\mathcal{B}} = [r_{F,\mathcal{B}}\times]F_{\mathcal{B}} \tag{7}$$

The cumulative moment $M_{\mathcal{B}}$ results from summing all individual moments M_i.

$$M_{\mathcal{B}} = \sum_{i=1}^{n} M_i \tag{8}$$

B. Aerodynamics.

This section's nominal aerodynamic forces and moments are from a 6DoF model, sans wind effects, shown in Figure 2. Parafoil control is through aerodynamic force and moment alteration. Symmetric δ_s and asymmetric δ_a deflections manage longitudinal and lateral dynamics, derived from canopy's δ_r and δ_l.

$$\delta_s = \frac{1}{2}(\delta_r + \delta_l) \tag{9}$$

$$\delta_a = \delta_r - \delta_l \tag{10}$$

Aerodynamic forces (drag D, side force S, and lift L) begin with local dynamic pressure $\bar{q} = \frac{1}{2}\rho v_a^2$ and reference area S_{ref} of the vehicle. The parafoil and payload, subscripts 1 and 2, are the force sources, leading to the aerodynamic force equation. Such that:

$$\mathbf{F}_A = \begin{bmatrix} -D \\ -S \\ -L \end{bmatrix} = \bar{q}S_{\text{ref}} \begin{bmatrix} -C_D \\ -C_S \\ -C_L \end{bmatrix} \tag{11}$$

The parafoil's coefficient calculations are conducted as follows:

$$C_L = C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta_s}}\delta_s \tag{12}$$

$$C_D = C_{D_0} + C_{D_{\alpha^2}}\alpha^2 + C_{D_{\delta_s}}\delta_s \tag{13}$$

$$C_S = C_{S_\beta}\beta \tag{14}$$

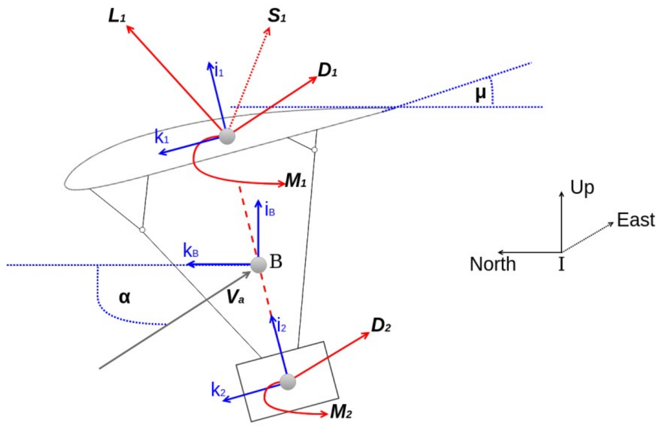


Fig. 2 Profile of the 6DoF parafoil-payload under aerodynamic forces and moments.

The coefficients depend on symmetric deflection, angle-of-attack, and side-slip angle, calculated as follows:

$$\alpha = \tan^{-1}\left(\frac{w}{u}\right), \quad \beta = \tan^{-1}\left(\frac{v}{\sqrt{u^2+w^2}}\right) \tag{15}$$

Velocity components are linked to local air velocity $\mathbf{V}_a = (u, v, w)^T$. Table 1 presents the aerodynamic coefficients and the payload contributes solely to drag, set at $C_{D_2} = 1.0$, necessitating a frame transformation from $\mathcal{A}$ to $\mathcal{B}$.

$$F_{\mathcal{B}} = C_{\mathcal{B},\mathcal{A}} F_{\mathcal{A}} \tag{16}$$

Where $C_{\mathcal{B},\mathcal{A}}$ can be computed as:

$$C_{\mathcal{A},\mathcal{B}} = C^T_{\mathcal{B},\mathcal{A}} = \begin{bmatrix} \cos\alpha\cos\beta & -\sin\beta & \sin\alpha\cos\beta \\ \cos\alpha\sin\beta & \cos\beta & \sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

Two elements contribute to the $\mathcal{B}$-frame aerodynamic force: the parafoil force $\mathbf{F}_1 = (L_1, S_1, D_1)^T$ and the payload force $\mathbf{F}_2 = (0, 0, D_2)^T$, as shown in Figure 2, yielding the aggregate force.

$$F_{\mathcal{B}} = F_1 + F_2 \tag{17}$$

Aerodynamic moments—roll L, pitch M, and yaw N can be demonstrated from Equation 18, requiring the parafoil's wingspan b and chord c, such that:

$$M_{\mathcal{B}} = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = \bar{q} S_{\text{ref}} \begin{bmatrix} C_l b \\ C_m c \\ C_n b \end{bmatrix} \tag{18}$$

Using the Up-East-North standard, $(p, q, r)^T = (\omega_x, \omega_y, \omega_z)^T$ informs moment coefficient computation based on following equations:

$$C_l = C_{l_\beta}\beta + C_{l_{\delta_a}}\delta_a + \frac{b}{2V}(C_{l_p} p + C_{l_r} r) \tag{19}$$

Parameters	Parafoil	Payload
Lift coefficients		
$C_{L,0}$	0.091	-
$C_{L,\alpha}$	0.90	-
C_{L,δ_s}	0.21	-
Drag coefficients		
$C_{D,0}$	0.25	-
C_{L,α^2}	0.12	-
C_{D,δ_s}	0.30	$C_{D_2} = 1.0$
Side force coefficient		
$C_{S,\beta}$	-0.23	-

Table 1 Up-to-dated aerodynamic forces and design parameters.

$$C_m = C_{m_0} + C_{m_\alpha}\alpha + \frac{c}{2V}C_{m_q}q \tag{20}$$

$$C_n = C_{n_\beta}\beta + C_{n_{\delta_a}}\delta_a + \frac{b}{2V}(C_{n_p}p + C_{n_r}r) \tag{21}$$

We can now quantify moments M_1 and M_2 using distance vectors r_1 and r_2 from the CoM to the parafoil and payload's force application points, respectively.

$$M_1 = [r_1 \times]F_1 + M_{\mathcal{A}} \tag{22}$$

Assumed that the parafoil accounts for moment derivatives, with the payload affecting M_2 solely through drag D_2.

$$M_2 = [r_2 \times]F_2 \tag{23}$$

The total nominal moment is calculated via summation in Equation 8, with coefficients and parameters in Table 2.

Parameters	Roll Coefficients	Pitch Coefficients	Yaw Coefficients
Parafoil	$C_{l,\beta} = -0.036$	$C_{m,0} = 0.25$	$C_{n,\beta} = -0.0015$
	$C_{l,\delta_a} = -0.0035$	$C_{m,\alpha} = -0.72$	$C_{n,\delta_a} = 0.0155$
	$C_{l,p} = -0.84$	$C_{m,q} = -1.49$	$C_{n,p} = -0.082$
	$C_{l,r} = -0.082$		$C_{n,r} = -0.27$

Table 2 Parameters for simulating aerodynamic moments.

Within the 6DoF framework, it is possible to calculate inertia moments, starting with the parafoil mass. The canopy volume, estimated as $V_{\text{vol}} = 0.09c^2$, leads to the parafoil's inertia tensor, calculated as:

$$\mathbf{I}_1 = \begin{bmatrix} \frac{V_{\text{vol}}\rho+m_1}{12}(b_{\text{inf}}^2 + \frac{V_{\text{vol}}^2}{cb}) & 0 & 0 \\ 0 & \frac{V_{\text{vol}}\rho+m_1}{12}(c^2 + \frac{V_{\text{vol}}^2}{cb}) & 0 \\ 0 & 0 & \frac{V_{\text{vol}}\rho+m_1}{12}(b_{\text{inf}}^2 + c^2) \end{bmatrix} \tag{24}$$

In the formula, ρ is atmospheric density, and $\varepsilon = \frac{b}{2R}$ represents the anhedral angle with suspension line length R. The inflated wingspan $b_{\text{inf}} = 2R \cdot \sin\varepsilon$, and to align the parafoil with the $\mathcal{B}$-frame, we transform the inertia tensor as shown in Figure 2 and can be derived as:

$$\mathbf{I}_{1\,1,B} = C_{1,B}\mathbf{I}_1 C_{1,B}^T \quad \text{with} \quad C_{1,B} = \begin{bmatrix} \cos\mu & 0 & \sin\mu \\ 0 & 1 & 0 \\ -\sin\mu & 0 & \cos\mu \end{bmatrix}$$

Assuming a cubic payload, we derive the inertia tensor $\mathbf{I}_2$ as follows:

$$\mathbf{I}_2 = \begin{bmatrix} \frac{m_2}{12}(b_2^2 + c_2^2) & 0 & 0 \\ 0 & \frac{m_2}{12}(a_2^2 + c_2^2) & 0 \\ 0 & 0 & \frac{m_2}{12}(a_2^2 + b_2^2) \end{bmatrix} \tag{25}$$

For the calculations of the aerodynamic properties, [2] has been referred. Next section, demonstrates the inertia and moments from fluid dynamic pressure on a moving body.

C. Perceived Aerodynamic Effects.

A moving body induces fluid motion, creating a pressure field. This significantly affects parafoil ram-air dynamics, approximating M_r ratio as:

$$M_r = \frac{m}{\rho S_{\text{ref}}^{2/3}} \tag{26}$$

For this problem, Titan's atmospheric density is approximated by:

$$\rho = 5.43 e^{-0.512 \cdot 10^{-3} h} \tag{27}$$

Calculated mass and inertia represent energy from acceleration. Apparent mass and inertia tensors referred to the Fig.3 and 4 are described as:

$$M_f = \begin{bmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{bmatrix} \quad \text{and} \quad I_f = \begin{bmatrix} I_A & 0 & 0 \\ 0 & I_B & 0 \\ 0 & 0 & I_C \end{bmatrix} \tag{28}$$

It is possible to calculate M_f and I_f using the Equations (29) and (30).

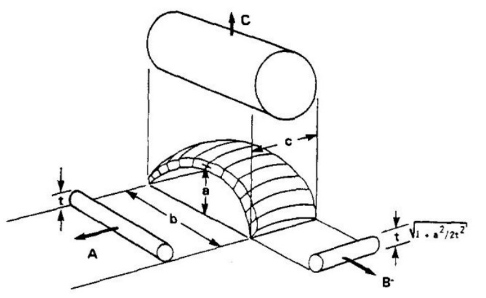


Fig. 3 First image

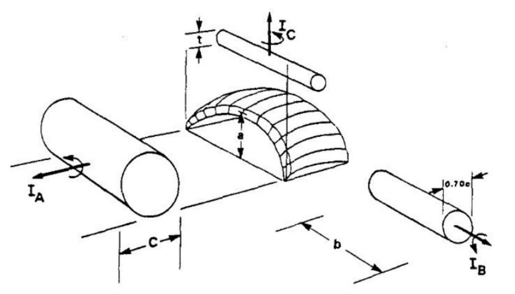


Fig. 4 Second image

Aspect ratio AR, thickness ratio t_*, and taper ratio a_* are defined by $AR = \frac{b}{c}$, $t_* = \frac{t}{c}$, and $a_* = \frac{a}{b}$, leading to the equations that follow:

$$\begin{aligned} A &= 0.667\rho\left(1 + \frac{8}{3}a_*^2\right)t^2 b \\ B &= 0.267\rho\left(1 + 2\frac{a_*^2}{t_*^2}\right)AR(1 - t_*)t^2 c \\ C &= 0.785\rho\sqrt{1 + 2 + a_*(1 - t_*)}\frac{AR}{1 + AR}c^2 b \end{aligned} \tag{29}$$

$$\begin{aligned} I_A &= 0.055\rho\frac{AR}{1 + AR}c^2 b^3 \\ I_B &= 0.0308\rho\frac{AR}{1 + AR}\left[1 + \frac{\pi}{6}(1 + AR)ARa_*^2 t_*^2\right]c^4 b \\ I_C &= 0.0555\rho(1 + 8a_*^2)t^2 b^3 \end{aligned} \tag{30}$$

Where **b** is the wingspan of the parafoil, **c** is the parafoil chord, **h** is the parafoil height and **t** be the thickness. Therefore, the discourse on dynamic formulation is conncluded. Subsequent is the exposition on the nonconvex challenge.

IV. Non-Convex Formulation and the Solution Strategy.

This section characterizes the parafoil landing problem as a non-convex optimization problem within a continuous temporal domain. Diverging from powered descent methods, control is enabled via electrically actuated canopy pulling. The approach is devised to minimum control effort problem.

$$0 < F_{\mathcal{A},\min} \leq \|F_{\mathcal{A}}\|_2 \leq F_{\mathcal{A},\max} \longrightarrow \text{subject to: } 0 \leq \delta_s \leq 1 \tag{31}$$

$$0 < M_{\mathcal{A},\min} \leq \|M_{\mathcal{A}}\|_2 \leq M_{\mathcal{A},\max} \longrightarrow \text{subject to: } 0 \leq \delta_a \leq 1 \tag{32}$$

The aerodynamic factors propel the parafoil's movement. Since aerodynamic force and moment cannot be assuredly zero, the issue presents as non-convex. The objective is to minimize control effort throughout the flight duration, t_f.

$$\min_{t_f} \|u(t)\| \tag{33}$$

The glide-slope constraint ensures an optimal trajectory by confining the spacecraft within a minimum slope-defined cone. A glide-slope angle γ_{gs} centralizes vehicle motion within the I-frame. Then constraint of a convex cone is applied as follows:

$$\mathbf{n}_1 \cdot \mathbf{r} \geq \tan(\gamma_{gs}) \left\|H_{23}^T \mathbf{r}\right\|_2 \quad \text{with} \quad H_{23} \triangleq [\mathbf{n}_2\ \mathbf{n}_3] \tag{34}$$

To meet TRN constraints, spacecraft attitude is defined by the tilt angle θ, the X-axis divergence of the $\mathcal{B}$ and $\mathcal{I}$ frames as:

$$\cos\theta = \mathbf{n}_1 \cdot C_{I,B}\mathbf{n}_1 = 1 - 2(q_2^2 + q_3^2) \tag{35}$$

To prevent too much tilt, a cap on θ, $\theta_{\max}$, is set, shaping the convex constraint.

$$\cos\theta_{\max} \leq 1 - 2(q_2^2 + q_3^2) \tag{36}$$

To prevent stalling, the angle-of-attack α is bounded by a maximum limit. Than constraint stated as:

$$\mathbf{v} \cdot \mathbf{n}_2 \leq \tan(\alpha_{\max})\mathbf{v} \cdot \mathbf{n}_1 \tag{37}$$

Aerodynamic angles are considered outcomes of motion by being defined as states, not controls. A convex limit on the maximum angular rate, $\omega_{\max}$, is enforced as

$$\|\boldsymbol{\omega}_B\|_2 \leq \omega_{\max} \tag{38}$$

The sequence's boundary conditions ensure a soft landing by mandating zero positional and touchdown velocity deviations at termination, within the guidance frame's 3D Euclidean space. The initial attitude is nominally set, aiming for zero at termination. Problem summary below, outlines the entire non-linear, continuous-time framework for optimizing control effort in six-degree-of-freedom landing guidance.

Problem Summary: Non-convex continuous time minimum control effort guidance problem [1].

Cost Function:

$$\min_{t_f} \|u(t)\| \quad \text{subject to:}$$

Dynamics:

$$\dot{r}_I(t) = v_I(t)$$
$$\dot{v}_I(t) = \frac{F_I(t)}{m} + g_I$$
$$I_B\dot{\omega}_B(t) = M_B - [\omega_B \times] I_B \omega_B$$
$$\dot{q}_{B|I}(t) = \frac{1}{2}\Omega(\omega_B) q_{B|I}$$

State Constraints:

$$\tan\gamma_{gs} \|H_{23}^T r_I\|_2 \le n_1 \cdot r_I$$
$$cos\theta_{max} \le \ 1 - 2(q_2^2 + q_3^2)$$
$$\|\omega_B\|_2 \le \omega_{max}$$
$$v \cdot n_2 \le \tan(\alpha_{max}) v \cdot n_1$$

Control Constraints:

$$0 < F_{\mathcal{A},min} \le \|F_{\mathcal{A}}\|_2 \le F_{\mathcal{A},max} \text{—> subject to: } 0 \le \delta_s \le 1$$
$$0 < M_{\mathcal{A},min} \le \|M_{\mathcal{A}}\|_2 \le M_{\mathcal{A},max} \text{—> subject to: } 0 \le \delta_a \le 1$$

Boundary Conditions:

$$r_I(0) = r_{I,0},\ v_I(0) = v_{I,0},\ \omega_B(0) = \omega_{B,0},\ q_{B|I}(0) = [1 \quad 0 \quad 0 \quad 0]^T$$
$$r_I(t_f) = 0,\ v_I(t_f) = 0,\ \omega_B(t_f) = 0,\ q_{B|I}(t_f) = [1 \quad 0 \quad 0 \quad 0]^T$$

In this optimization approach, we employ YALMIP [3] tools and the GUROBI [4] solver within MATLAB [5], to formulate and solve a spacecraft trajectory optimization problem[6], where the spacecraft's dynamics are governed by both translational and rotational motion equations under constraints such as glide-slope, tilt limitations, and control bounds, with the objective of minimizing the cumulative squared norm of aerodynamic forces and moments applied over the spacecraft's journey from an initial to a terminal condition.

In the formulation of the optimization problem, instead of directly enforcing norm constraints, equivalent constraints are implemented by squaring and comparing the sums of the squares of the components of the aerodynamic forces and moments to their respective squared bounds, thus simplifying the mathematical representation and computational handling within the solver.

Using Eqns. 11 and 18, we computed FA_{min}, FA_{max}, MA_{min}, and MA_{max}. Moreover, the velocity of the wind is opposite to that of the spacecraft itself, which was one of the important assumptions we considered (i.e., $\beta = 0$) when exploring the problem.

V. Simulation.

This section displays the outcomes of simulations for the Titan parafoil descent guidance. The simulations were conducted using MATLAB [5], particularly with the aid of YALMIP [3] and GUROBI [4] tools. Details such as simulation settings, algorithm specifics, and initial conditions are outlined in Table 3 below. Fig. 5 and 6 demonstrates that both position and velocity reach zero upon landing, confirming the algorithm's compliance with the established state and control constraints for a soft landing. It is observed that,from the Fig. 7 and 8 further corroborate the scenario by showing the spacecraft's position and velocity progressively decrease to zero, indicating a controlled descent and compliance with the designated end-state conditions for a gentle touchdown on Titan's surface.

The quaternion attitude is constrained to maintain a unit quaternion of one, as observed in Fig. 9. Fig. 12 and 13 illustrates the spacecraft's movement both within and outside of the plane. The 2D trajectory, particularly from the up-east perspective, clearly shows that at the end of the mission, the spacecraft is oriented in the opposite direction to the wind, a requirement imposed on the system. Furthermore, the 3D trajectory displayed in Fig. 13 further demonstrates the three-dimensional path taken by the spacecraft, which shows a controlled descent curve responding dynamically to aerodynamic forces, aligning with the mission's end-state constraints for a targeted and safe landing.

Simulation Parameters	**Description**	**Value**
g_T	Titan gravity	1.352 m/s^2
m	Spacecraft mass	201.4 kg
$S_{\text{ref}}{}^*$	Parafoil surface area	7.25 m^2
t_f	Time-of-flight	76 s
$\alpha_{\max}$	Maximum angle-of-attack	45°
γ_{gs}	Glide-slope angle	20°
$\theta_{\max}$	Maximum tilt angle	90°
$\omega_{\max}$	Maximum angular rate	30/s
Algorithm Parameters		
w_v	Virtual control weight factor	10^5
w_Δ^i	Trust region weight factor	10^{-3}
v_{tol}	Virtual control tolerance	10^{-9}
Δ_{tol}	Trust region tolerance	10^{-3}
$i_{\max}$	Maximum number of iterations	10
K	Temporal nodes	70
Starting Conditions		
r_0	Starting position	$[9, -7, 183]^T$ m
v_0	Starting velocity	$[9, 0, -8]^T$ m/s

Table 3 Parameters and Initial Conditions for Titan Parafoil Descent Simulation.

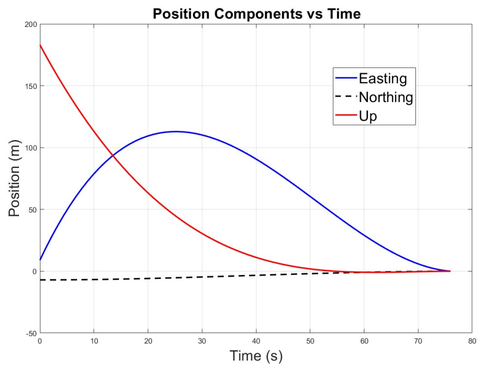


Fig. 5 Position Profile.

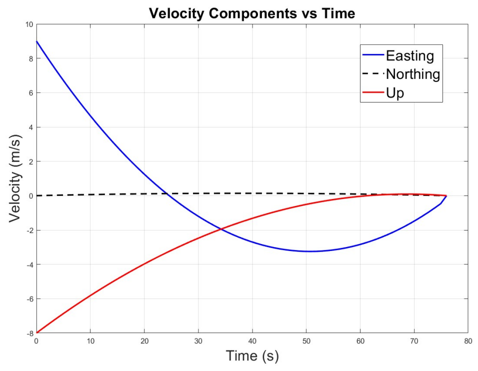


Fig. 6 Velocity Profile.

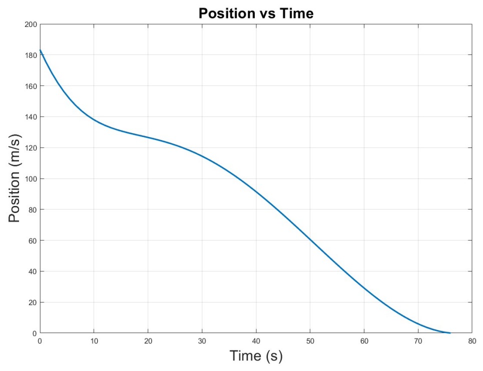


Fig. 7 Position vs. Time plot.

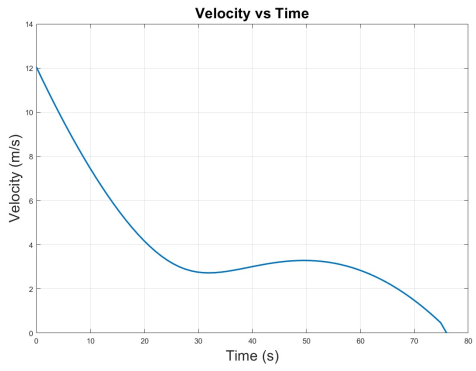


Fig. 8 Velocity vs. Time plot.

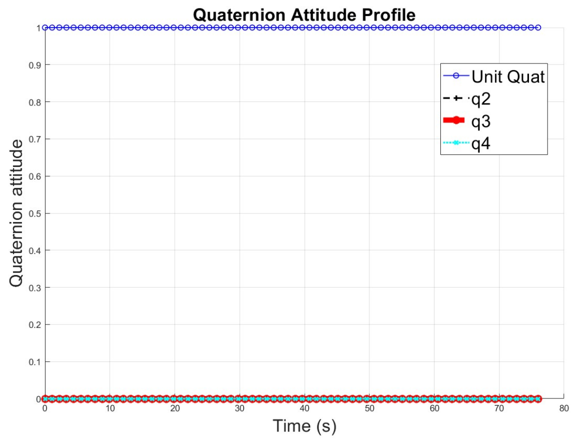
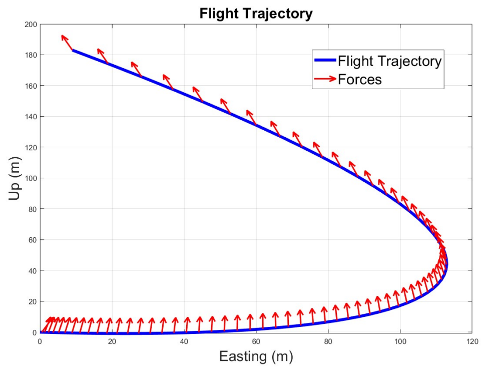


Fig. 9 Quaternion attitude profile.



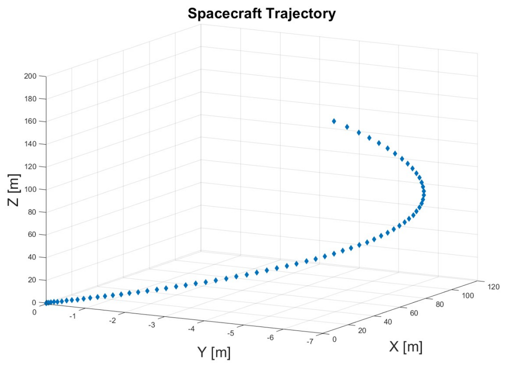


Fig. 10 Trajectory visualization (in-plane). Fig. 11 Trajectories visualization (out-of-plane).

Finally, we conclude with a sensitivity analysis that offers substantial understanding of the impact that variations in initial conditions may have on the flight path, control efforts, and orientation. Such analysis is instrumental in assessing the resilience of the parafoil navigation algorithm. The simulations enable the identification of potential risk factors that could affect the precision of the landing or the overall flight trajectory. The parameters assessed in the sensitivity analysis are detailed in Table 4. For this study, initial state conditions were adjusted by either increasing or decreasing them by increments of 5%, 10%, and 20% from their nominal values. Notably, absolute values were employed in the simulations rather than relative percentages. We will now review the outcomes for each scenario, examining time histories of position and velocity, as these metrics yield the most insightful data from the simulations.

Table 4 Parameters of sensitivity and their respective percentages.

Parameter	Description	Sensitivity interval
r_0	Initial position	±20%
v_0	Initial velocity	±20%

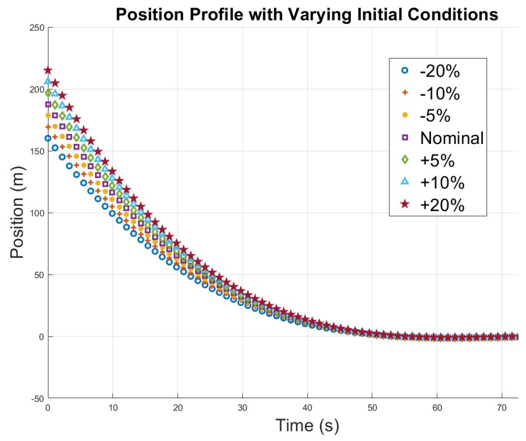


Fig. 12 Position profile Variation.

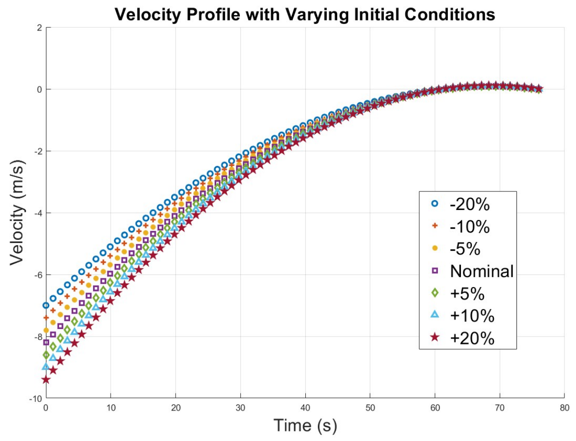


Fig. 13 Velocity profile Variation.

The results from the sensitivity simulations are illustrated in Figures 12 to 13. Regarding the sensitivity to the initial position, depicted in Figure 12, the data suggests the algorithm consistently performs well despite variations in starting conditions. It appears capable of reaching the target across a spectrum of initial positions. As for the sensitivity to initial velocity, showcased in Figures 12, the findings align closely with the previous initial position analysis.

The overarching inference is that the algorithm robustly directs the parafoil towards the intended destination, despite a broad array of initial states in both position and velocity. For real-world flight applications, a guidance algorithm that minimizes error is advisable to simplify the optimal control challenge.

VI. Conclusion

This study successfully addressed a non-convex optimization problem to model the terminal descent of a Titan parafoil, centering on a guidance strategy presented as a finite horizon optimal control problem. YALMIP within Matlab facilitated all simulation activities and GUROBI has been used as the solver. The methodology ensures a soft landing, complying with the defined mission constraints. The implementation of the glide-slope constraint was particularly effective in preventing a rapid descent that could lead to an undesirable penetration below the surface. Additionally, the pointing constraint was rigorously applied to meet the demands of Terrain Relative Navigation (TRN). It has been determined that the algorithm adeptly navigates the parafoil towards its destination across a diverse array of initial conditions related to both position and velocity.

The meticulous arrangement of nomenclature, the careful delineation of the spacecraft's dynamics and kinematics, and the incorporation of non-convex optimization problem into the simulation framework illustrate the complexity and depth required for such advanced research. An interesting observation is the interplay between the various physical parameters—such as the glide-slope constraints, pointing angle requirements, and the virtual control tolerances—that must be precisely calibrated to ensure the parafoil's successful descent and landing. The sensitivity analysis underscores the robustness of the algorithm, providing confidence in its ability to cope with a range of initial conditions. This document serves as a testament to the comprehensive nature of simulation studies in planetary exploration and the innovative methodologies utilized to surmount the challenges of extraterrestrial landings.

References

[1] Mazouz, R., Quadrelli, M. B., and Mooij, E., "Convex Optimization Guidance for Precision Landing on Titan," *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics Inc. (AIAA), 2021, pp. 11–15 & 19–21 January 2021. https://doi.org/10.2514/6.2021-1345.

[2] Quadrelli, M. B., Schutte, A., Rimani, J., and Ermolli, L., "Aero Maneuvering Dynamics and Control for Precision Landing on Titan," *2019 IEEE Aerospace Conference*, 2019, pp. 1–16. https://doi.org/10.1109/AERO.2019.8742230.

[3] Löfberg, J., "YALMIP : A Toolbox for Modeling and Optimization in MATLAB," *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[4] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," , 2023. URL https://www.gurobi.com.

[5] Inc., T. M., "MATLAB version: 9.13.0 (R2022b)," , 2024. URL https://www.mathworks.com.

[6] Harris, M. W., and Rose, M. B., *Optimal Spacecraft Guidance*, Utah State University, Logan, UT, 2023.

Appendix

A. Calculations.

```
u = 0.5;  % east-west component in m/s
v = 0.2;  % north-south component in m/s
w = -0.1; % vertical component in m/s
sref = 7.25;  % Reference area m^2
rho = 1.8; %kg/m^3
b=3.07; %m
c=1.02; %m
h=0.14*c ; %m
rho1= 5.43 * exp(-0.512 * 10^-3*h);  % Air density at given altitude on Titan
alpha_max = 45*pi/180;       %rad

% Calculate the magnitude of the velocity vector
V = sqrt(u^2 + v^2 + w^2);

% Calculate dynamic pressure
qbar = 0.5 * rho * V^2;

% Set control surface deflection
ds = 1; %(FA_max)              %  ds = 0(For FA_min);
da = 1; %(MA_max)              %  da = 0(For MA_min);
p = 0.1; % Roll rate in rad/s
q = 0.1; % Pitch rate in rad/s
r = 0.1; % Yaw rate in rad/s

% Define aerodynamic coefficients
CL0 = 0.091;
CLalpha = 0.9;
CLds1 = 0.21;
```

```
CD0 = 0.25;
CDalpha2 = 0.12;
CDds = 0.3;
CSbeta = -0.23;
% Calculate angles alpha and beta
alpha = inv(w/u);
%beta = atan(v / sqrt(u^2 + w^2));
beta=0;

Cl_beta=-0.036; Cl_da=-0.0035; Cl_p=-0.84;Cl_r=-0.082;
Cm_0=0.25; Cm_alpha=-0.72; Cm_q=-1.49;
Cn_beta=-0.0015; Cn_da=0.0155; Cn_p=-0.082; Cn_r=-0.27;

% Calculate coefficients of Lift, Drag, and Side force
CL = CL0 + 0*CLalpha * alpha_max + CLds1 * ds;
CD = CD0 + 0*CDalpha2 * alpha_max^2 + CDds * ds;
CS = CSbeta * 0;

Cl=Cl_beta *beta +Cl_da *da +Cl_p * (b/(2*V)) * p +Cl_r *(b/(2*V))*r;
Cm=Cm_0 +Cm_alpha *alpha +Cm_q *(c/(2*V))*q;
Cn=Cn_beta *beta +Cn_da *da +Cn_p *(b/(2*V)) *p + Cn_r *(b/(2*V))*r;

% Calculate aerodynamic forces
FA = qbar * sref * [-CD; -CS; -CL];
MA = qbar * sref * [Cl*b; Cm*c ; Cn*b];

% Calculate the norm of the aerodynamic force vector
normFA = norm(FA);
normMA = norm(MA);

% Display the aerodynamic force vector and its norm
disp('Aerodynamic Force Vector FA:');
disp(FA);
disp('Norm of the Aerodynamic Force Vector FA:');
disp(normFA);

% Display the aerodynamic force vector and its norm
disp('Aerodynamic Force Vector FA:');
disp(MA);
disp('Norm of the Aerodynamic Force Vector FA:');
disp(normMA);
```

B. Algorithm.

```
%% Constants and Initialization
g_I = [0; 0; -1.352]; % Titan gravity in m/s^2
m = 201.4; % Spacecraft mass in kg
N = 70; % Number of temporal nodes
tf = 76; % Time-of-flight in seconds
dt = tf / (N - 1); % Time step
W_vc = 1e5; % Virtual control weight factor
gamma_gs = 20*pi/180;; % Glide-slope angle in rad
```

```
theta_max = 90*pi/180;; % Maximum tilt angle in rad

% Initial conditions
r_I0 = [9; -7; 183]; % Starting position in meters
v_I0 = [9; 0; -8]; % Starting velocity in m/s

% Control limits
F_A_min_sq =251.7137;
F_A_max_sq = 1.1215e+03;
M_A_min_sq =F_A_min_sq*7;
M_A_max_sq = F_A_max_sq*7;
omega_max=30*pi/180;
alpha_max=45*pi/180;

% Spacecraft inertia matrix
I_B = diag([50, 50, 50]); % Inertia matrix of the spacecraft
n1 = [1;0;0];
n2= [0;1;0];
n3=[0;0;1];
H_23=[n2, n3];

%% Variables
F_A = sdpvar(3, N-1); % Aerodynamic forces
M_A = sdpvar(3, N-1); % Moments
delta_s = sdpvar(1, N-1); % Scaling factors
delta_a = sdpvar(1, N-1);
r_I = sdpvar(3, N); % Position
v_I = sdpvar(3, N); % Velocity
q_BI = sdpvar(4, N); % Orientation quaternions
omega_B = sdpvar(3, N); % Angular velocity


%% Dynamics and Constraints

% Dynamics and Constraints
constraints = [];
%Objective = [];


Constraints = [r_I(:,1) == r_I0, v_I(:,1) == v_I0, q_BI(:,1) == [1;0;0;0], ...
    omega_B(:,1) == zeros(3,1), r_I(:,N) == zeros(3,1), v_I(:,N) == zeros(3,1), ...
    omega_B(:,N) == zeros(3,1), q_BI(:,N) == [1;0;0;0]];
for i = 1:N-1

    glide_slope_constraint = tan(gamma_gs) * sqrt(2) <= n1' * r_I(:,i);
    Constraints = [Constraints, glide_slope_constraint];
end

for i = 1:N
    tilt_constraint = cos(theta_max) <= 1 - 2*(q_BI(2,i)^2 + q_BI(3,i)^2);
    Constraints = [Constraints, tilt_constraint];
end

for i = 1:N
```

```
% Calculate the dot products
dot_product_n1 = n1' * v_I(:,i);  % v     n1
dot_product_n2 = n2' * v_I(:,i);  % v     n2

% Set up the constraint
velocity_constraint = dot_product_n2 <= tan(alpha_max) * dot_product_n1;

% Add it to the YALMIP Constraints
Constraints = [Constraints, velocity_constraint];
end


for i = 1:N-1
    % Dynamics
    Constraints = [Constraints, ...
        r_I(:,i+1) == r_I(:,i) + dt * v_I(:,i), ...
        v_I(:,i+1) == v_I(:,i) + dt * (F_A(:,i) / m + g_I)];
    % omega_B(:,i+1) == omega_B(:,i) + dt * inv(I_B) * (M_A(:,i) - cross(omega_B(:,i),
    % q_BI(:,i+1) == q_BI(:,i) + 0.5 * dt * quatmultiply(q_BI(:,i)', [0; omega_B(:,i)]

    omega_cross_matrix = skew(omega_B(:,i));

    % Rotational dynamics constraint:
    omega_dot = inv(I_B) * (M_A(:,i) - omega_cross_matrix * (I_B * omega_B(:,i)));
    Constraints = [Constraints, ...
        omega_B(:,i+1) == omega_B(:,i) + dt * omega_dot];

    % Quaternion dynamics constraint:
    % q _B|I(t) = 1/2 *    ( _B ) * q_B|I
    % Discretized using Euler integration:
    % q_B|I(i+1) = q_B|I(i) + dt * 0.5 * quat_rate_matrix(omega_B(:,i)) * q_B|I(i)
    q_dot = 0.5 * quat_rate_matrix(omega_B(:,i)) * q_BI(:,i);
    Constraints = [Constraints, ...
        q_BI(:,i+1) == q_BI(:,i) + dt * q_dot];

    % Angular velocity norm constraint
    Constraints = [Constraints, ...
        sum(omega_B(:,i).^2) <= omega_max^2];
    % Control magnitude constraints
    Constraints = [Constraints, ...
        sqrt(F_A(1,i)^2 + F_A(2,i)^2 + F_A(3,i)^2) >= F_A_min_sq, ... % Lower bound fo
        sqrt(F_A(1,i)^2 + F_A(2,i)^2 + F_A(3,i)^2) <= F_A_max_sq, ... % Upper bound fo

        sqrt(M_A(1,i)^2 + M_A(2,i)^2 + M_A(3,i)^2) >= M_A_min_sq, ...
% Lower bound for M_A
        sqrt(M_A(1,i)^2 + M_A(2,i)^2 + M_A(3,i)^2) <= M_A_max_sq, ... % Upper bound fo

        0 <= delta_s(i) <= 1,...
        0 <= delta_a(i) <= 1];

end

Objective = 0; % Initialize the objective function
for k = 1:N-1
```

```
        Objective = Objective + norm(F_A(:,k), 2)^2 + norm(M_A(:,k), 2)^2; % Add the squar
        Objective = Objective + W_vc * (delta_s(k)^2 + delta_a(k)^2); % W_vc is a weight f
    end

%% Solver Setup and Execution
options = sdpsettings('solver', 'gurobi','verbose',1);
sol = optimize(Constraints, Objective, options);

%% Post-Processing
if sol.problem == 0
    % Successful optimization
    disp('Optimization succeeded');
    r_val = value(r_I);
    v_val = value(v_I);
    F_val = value(F_A);
    F_A = double(F_A);
    for i = 1:size(F_A,2)
        F_A_norm(i) = norm(F_A(:,i));
    end

    % Create a scatter plot for the trajectory
    figure;
    scatter3(r_val(1,:), r_val(2,:), r_val(3,:), 'filled');
    xlabel('X [m]');
    ylabel('Y [m]');
    zlabel('Z [m]');
    title('Spacecraft Trajectory');
    grid on; % Optionally add a grid
else
    % Handle errors
    disp(['Problem solving the optimization: ' yalmiperror(sol.problem)]);
end

%% Plots

F_norm = F_val ./ vecnorm(F_val);

% Scale factor for better visualization of arrows
scale_factor = 10; % Adjust this scale factor as needed for your specific plot

% Determine the number of arrows you want to plot
num_arrows = size(F_norm, 2); % This should match the number of columns in F_norm

% Plot the trajectory in 2D
figure;
plot(r_val(1,:), r_val(3,:), 'b-'); % Trajectory in Easting vs Up plane
hold on;



quiver(r_val(1,1:num_arrows), r_val(3,1:num_arrows), ...
    scale_factor * F_norm(1,:), scale_factor * F_norm(3,:), ...
    'r', 'AutoScale', 'off');
```

```
% Formatting the plot
xlabel('Easting (m)');
ylabel('Up (m)');
title('Flight Trajectory');
legend('Flight Trajectory', 'Forces');
grid on;
hold off;

% Adjust axis limits and aspect ratio if needed
axis equal;
xlim([min(r_val(1,:)) max(r_val(1,:))]);
ylim([min(r_val(3,:)) max(r_val(3,:))]);
%%

% Constants for the scenarios
scenarios = {'-20%', '-10%', '-5%', 'Nominal', '+5%', '+10%', '+20%'};
colors = lines(length(scenarios)); % Generate a set of colors
lineStyles = {'-', '--', '-.', ':', '-', '--', '-.'}; % Define line styles for each sc



% Create the plot
figure; hold on; grid on;
for i = 1:length(scenarios)
    plot3(r_val(1,:,i), r_val(2,:,i), r_val(3,:,i), ...
        'Color', colors(i,:), 'LineStyle', lineStyles{i}, ...
        'DisplayName', scenarios{i});
end

% Axis labels and title
xlabel('x (m)');
ylabel('y (m)');
zlabel('z (m)');
title('Trajectory with Sensitivity to Initial Conditions');

% Adjust the view and legend
view(3);
legend('show');
hold off;


%%


% Extract the numerical values of the quaternion after optimization
q_BI_val = value(q_BI);

% Create a time vector for plotting
time_vector = linspace(0, tf, length(q_BI_val));

% Plotting each quaternion component vs. time
figure;
hold on; % Allows multiple plots on the same figure
plot(time_vector, q_BI_val(1, :), 'b-', 'LineWidth', 1, 'Marker', 'o', 'DisplayName',
plot(time_vector, q_BI_val(2, :), 'k--', 'LineWidth', 2, 'Marker', '+', 'DisplayName',
plot(time_vector, q_BI_val(3, :), 'r-.', 'LineWidth', 6, 'Marker', '*', 'DisplayName',
plot(time_vector, q_BI_val(4, :), 'c:', 'LineWidth', 2, 'Marker', 'x', 'DisplayName',
hold off;

% Adding labels and title
xlabel('Time (s)');
ylabel('Quaternion attitude');
title('Quaternion Attitude Profile');

% Add legend
legend show; % This will show the display names set for each plot

% Optional: Add grid lines for better readability
grid on;

%%
r_val = value(r_I); % Position values from the optimization result
v_val = value(v_I); % Velocity values from the optimization result

% Create a time vector
time_vector = linspace(0, tf, N);

% Extract individual components of the position
easting = r_val(1, :);
northing = r_val(2, :);
up = r_val(3, :);

% Extract individual components of the velocity
velocity_easting = v_val(1, :);
velocity_northing = v_val(2, :);
velocity_up = v_val(3, :);

% Extract the magnitudes of position and velocity at each time step
position_magnitudes = vecnorm(r_val); % Euclidean norm (magnitude) of position vector
velocity_magnitudes = vecnorm(v_val); % Euclidean norm (magnitude) of velocity vector

% Plot Position Components (Easting, Northing, Up) vs Time
figure;
plot(time_vector, easting, 'b-', 'LineWidth', 2); hold on; % Easting in blue
plot(time_vector, northing, 'k--', 'LineWidth', 2);        % Northing in black dashed
plot(time_vector, up, 'r-', 'LineWidth', 2);               % Up in red
hold off;

% Formatting the plot
xlabel('Time (s)');
ylabel('Position (m)');
title('Position Components vs Time');
legend('Easting', 'Northing', 'Up');
grid on;

% Velocity (m/s) vs Time (s)
figure;
plot(time_vector, velocity_magnitudes, 'LineWidth', 2);
```

```
title('Velocity vs Time');
xlabel('Time (s)');
ylabel('Velocity (m/s)');
grid on;

% Velocity (m/s) vs Time (s)
figure;
plot(time_vector, position_magnitudes, 'LineWidth', 2);
title('Position vs Time');
xlabel('Time (s)');
ylabel('Position (m/s)');
grid on;

% Plot Velocity Components (Easting, Northing, Up) vs Time
figure;
plot(time_vector, velocity_easting, 'b-', 'LineWidth', 2); hold on; % Easting velocit
plot(time_vector, velocity_northing, 'k--', 'LineWidth', 2);
% Northing velocity in black dashed
plot(time_vector, velocity_up, 'r-', 'LineWidth', 2);
% Up velocity in red
hold off;

% Formatting the plot
xlabel('Time (s)');
ylabel('Velocity (m/s)');
title('Velocity Components vs Time');
legend('Easting', 'Northing', 'Up');
grid on;

%%
% Make sure you have run the solver before this step
% Retrieve the numerical values from the optimization variables
r_val = value(r_I); % Position values
v_val = value(v_I); % Velocity values
omega_B_val = value(omega_B); % Angular velocity values

% Since we have N time points and N-1 control/action points,
% we need two time vectors, one for the state variables and one for the control variab
state_time_vector = linspace(0, tf, N);   % Time vector for state variables (position
control_time_vector = linspace(0, tf, N); % Time vector for control variables (angula

% Extract the individual components of the velocity and angular velocity for plotting
velocity_magnitudes = vecnorm(v_val); % Compute magnitude of velocity vectors
omega_x = omega_B_val(1, :);
omega_y = omega_B_val(2, :);
omega_z = omega_B_val(3, :);



% Plotting Angular Velocity Components vs Time
figure;
plot(control_time_vector, omega_x, 'b-', 'LineWidth', 2); hold on;
plot(control_time_vector, omega_y, 'k--', 'LineWidth', 2);
plot(control_time_vector, omega_z, 'r-', 'LineWidth', 2); hold off;
```

```
xlabel('Time (s)');
ylabel('Angular velocity (rad/s)');
title('Angular Velocity Components vs Time');
legend('\omega_x', '\omega_y', '\omega_z');
grid on;


%%

% Example to extract the numerical values of F_A after optimization
F_A_val = value(F_A); % Ensure that the optimization problem has been solved

% Calculate the norms of the thrust vectors at each time step
thrust_norms = vecnorm(F_A_val); % This computes the Euclidean norm of each column

% Create a time vector for plotting
time_vector = linspace(0, tf, N-1); % Adjust according to how F_A is defined

% Plotting the thrust norm vs. time
figure;
plot(time_vector, thrust_norms, 'LineWidth', 2);
title('Forces vs. Time');
xlabel('Time (s)');
ylabel('Force (N)'); % Adjust the unit based on your F_A definition
grid on; % Optional: add a grid to the plot for better visibility


% Extract the necessary values if not already done
r_val = value(r_I); % Position data from optimization
v_val = value(v_I); % Velocity data from optimization

% Create the trajectory plot
figure;
plot(r_val(1, :), r_val(3, :), 'b-'); % Assuming r_val(1,:) is Easting and r_val(3,:)
hold on;

% Calculate a suitable scaling factor for the arrows
scale_factor = max(range(r_val(1, :))) / max(vecnorm(v_val)); % Example scale factor

% Add arrows to indicate velocity direction and magnitude
quiver(r_val(1, 1:end-1), r_val(3, 1:end-1), v_val(1, :), v_val(3, :), scale_factor, '

% Label the plot
xlabel('Easting (m)');
ylabel('Up (m)');
title('Flight Trajectory');

% Optional: Add a legend
legend('Flight Trajectory');

% Ensure the axes are equal to prevent distortion
axis equal;
grid on; % Add grid lines for better readability
hold off;
```

```
%%
% Extract individual components of the position for Z-axis (altitude)
altitude = r_val(3, :);

% Time vector corresponding to the number of points in the altitude data
time_vector = linspace(0, tf, N);

% Define the scenarios and their respective colors and markers
scenarios = {'-20%', '-10%', '-5%', 'Nominal', '+5%', '+10%', '+20%'};
colors = lines(numel(scenarios)); % Generate distinct colors for each scenario
markers = {'o', '+', '*', 's', 'd', '^', 'p', 'h'}; % Ensure there are enough markers

% Plotting each scenario
figure; hold on; % Hold on to add all scenarios to the same plot
for i = 1:numel(scenarios)
    % Generate random data for the altitude as an example (replace this with your actu
    % Here I'm simulating the variations by multiplying by a factor for each scenario
    factor = 1 + (i - numel(scenarios)/2) * 0.05;
    simulated_altitude = altitude * factor; % Use your actual altitude data for each

    scatter(time_vector, simulated_altitude, 'Marker', markers{i}, ...
        'MarkerEdgeColor', colors(i, :), 'DisplayName', scenarios{i});
end
xlabel('Time (s)');
ylabel('Position (m)');
title('Position Profile with Varying Initial Conditions');
legend('Location', 'eastoutside');
grid on;
hold off;

% Save the figure if needed
saveas(gcf, 'position_profile.png');




% After running the optimization
% v_val = value(v_I); % Velocity values from the optimization result

% Extract individual components of the velocity for Z-axis (vertical velocity)
vertical_velocity = v_val(3, :);

% Time vector corresponding to the number of points in the vertical velocity data
time_vector = linspace(0, tf, N);

% Define the scenarios and their respective colors and markers
scenarios = {'-20%', '-10%', '-5%', 'Nominal', '+5%', '+10%', '+20%'};
colors = lines(numel(scenarios)); % Generate distinct colors for each scenario
markers = {'o', '+', '*', 's', 'd', '^', 'p', 'h'}; % Ensure there are enough markers

% Plotting each scenario
figure; hold on; % Hold on to add all scenarios to the same plot
for i = 1:numel(scenarios)
    % Generate random data for the vertical velocity as an example (replace this with
    % Here I'm simulating the variations by multiplying by a factor for each scenario
    factor = 1 + (i - numel(scenarios)/2) * 0.05;
    simulated_vertical_velocity = vertical_velocity * factor; % Use your actual verti

    scatter(time_vector, simulated_vertical_velocity, 'Marker', markers{i}, ...
        'MarkerEdgeColor', colors(i, :), 'DisplayName', scenarios{i});
end
xlabel('Time (s)');
ylabel('Velocity (m/s)');
title('Velocity Profile with Varying Initial Conditions');
legend('Location', 'eastoutside');
grid on;
hold off;

% Save the figure if needed
saveas(gcf, 'velocity_profile.png');
%%

function S = skew(v)
S = [  0    -v(3)   v(2);
      v(3)    0    -v(1);
     -v(2)   v(1)    0  ];
end

function omegaMatrix = quat_rate_matrix(omega)
% omega is a 3x1 vector [omega_x; omega_y; omega_z]
omega_x = omega(1);
omega_y = omega(2);
omega_z = omega(3);

omegaMatrix = [0,  -omega_x, -omega_y, -omega_z;
               omega_x, 0,    omega_z,  -omega_y;
               omega_y, -omega_z, 0,  omega_x;
               omega_z, omega_y,  -omega_x, 0];
end
```