High-Altitude Research Blimp Flight Mechanics Simulation

Johnny Martin (Shaun) Lowis[a]
[a]*Department of Mechanical Engineering, University of Canterbury, Christchurch, New Zealand*
jml190@uclive.ac.nz — https://github.com/shaunlowis/hydroblimp

I. Executive Summary

A six DOF model was created for simulating the flight performance of a high-altitude blimp, with the simupy-flight Python package. The operating point for the vehicle is aimed at 30,000 [ft] elevation, or low stratosphere. The use case of the vehicle is for scientific research. The vehicle is filled with enough Helium to be neutrally buoyant at its operating point. The geometry and components of the vehicle were designed and areas for improvement noted.

Coefficients were sourced [6] and defined for the designed geometry and mass budget. An inertia tensor was derived and these values used as inputs to the 6-DOF model. The vehicle showed high stability at neutral operating conditions. Sources of instability in the model were found and examined. The initial condition was perturbed with a 2 N force in the y-direction of the body axis. The model showed instability, exiting after $\approx 1.6[\text{seconds}_{\text{model time}}]$, irrespective of simulation time, with high oscillations in the z direction.

II. Background

LTA craft often see use in hobbyist fields, but also see modern use in airshows [2] with omni-directional designs, or novel fixed-wing hybrid glider blimp designs [3].

Typically weather balloons are used for gathering atmospheric data, though airships have a rich history in this field, such as the Graf Zeppelin [13]. The key difference between a weather balloon and the blimp designed here is the addition of a propulsion system and control surfaces. This additional weight adds lateral maneuverability, with the possibility of re-use.

At this altitude, the vehicle will experience low stratospheric winds, which we will model as a perturbing force and investigate the vehicle's response and stability IV-B. This will let us comment on the vehicle's flying and handling IV-C.

The design uses two motors for differential thrust, allowing yaw control and two symmetrical horizontal stabilisers with elevators for roll and pitch control. The balloon has a rigid frame and contains the lifting gas.

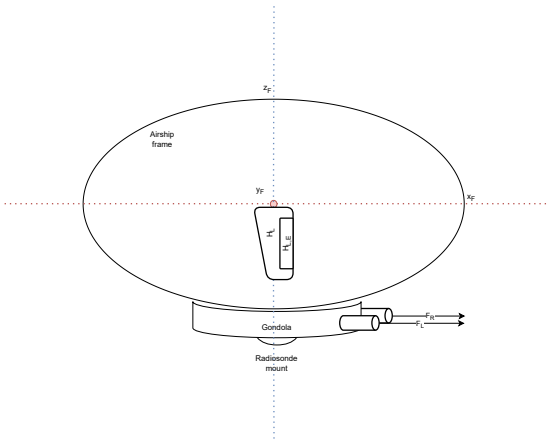


Fig. 1: Vehicle schematic.

Where:

$$\begin{aligned} H_L &= \text{Horizontal stabiliser, left} \\ H_{L,E} &= \text{Elevator, left} \\ F_L &= \text{Motor thrust, left} \\ y_F &= \text{Body-fixed y axis (+ve into page)} \\ x_F &= \text{Body-fixed x axis (+ve left)} \\ z_F &= \text{Body-fixed z axis (+ve down)} \end{aligned} \tag{1}$$

Where the right-side horizontal stabiliser, in Figure 1 is not shown, but is mounted symmetrically about the centre of mass, located at the centre of the body-fixed axes. This is a simplifying assumption, the actual centre of mass will depend on the specific mounting of the vehicle's components.

For scientific measurements, a radiosonde payload [12] weighing 60g has been factored into the mass budget.

The flight model is constructed with SimuPy Flight Vehicle Toolkit, with aerofoils designed using NACA's 4-digit framework [10] and modified in Onshape [7]. Code is available on the GitHub repository [8] for this assignment and also added as an appendix. When appropriate, this repository will be referenced in section headings using `/hydroblimp/sub-folder`, referring to the sub-folder of the repository relevant to that report section.

III. Methodology

A. Electronics - see flight_controller

For initial piloting, a Bluetooth capable ESP32 micro-controller was chosen, so the vehicle can be controlled from a mobile phone. The mission flight path is uploaded and activated automatically at the correct elevation. Other avionics consist of two servo's, two brushed motors, an ESC motor controller and a 6-axis inertial measurement unit. This is powered by a 2S LiPo battery.

B. Airframe

Aerofoils for 6DOF control and a gondola for housing the electronics and attachment point for the radiosonde were designed.

1) Aerofoils - see modeling/aerofoil_design.py: Using the dual motor setup for differential thrust has its advantages in terms of simplicity. In terms of stability, the two main drawbacks are that the magnitude of the yaw moment is limited by the distance to the centre of mass of the vehicle. The other drawback is that since the thrust is applied at some distance and at an angle to the centre of mass, a pitching moment (M) is created when thrust is applied. Equal thrust is illustrated in Figure 2. Also shown are equal and opposite weight (G_G), buoyancy (G_B) and drag W forces in the body-fixed coordinate system.

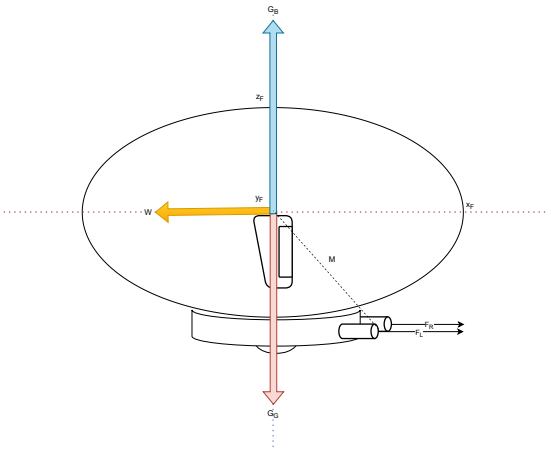
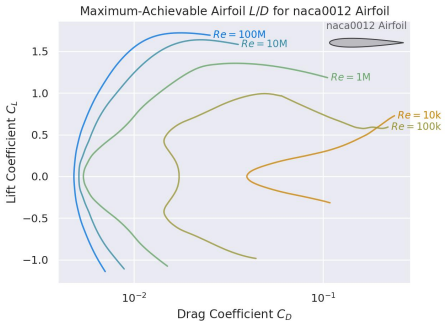
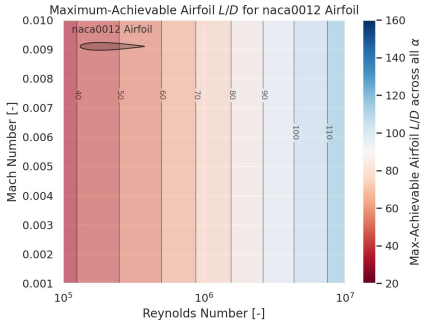


Fig. 2: Induced pitch, M, from applied thrust

These can be remedied by placing the motors level with the centre of mass, assuming a rigid structure for the balloon containing the lifting gas. Another solution is to design the horizontal stabilisers as asymmetric aerofoils, such that induced lift occurs at an angle, leading to a pitching moment in the opposite direction. These affects are noted here, but do not contribute in a large enough manner to justify changes to the existing design. A symmetrical aerofoil design was chosen, as the asymmetrical aerofoil would need to be designed for some nominal thrust, thereby not solving the problem for other operating points. Using the NACA [11] 4-digit aerofoil database [10], the 0012 aerofoil was chosen for its simplicity. To examine the performance and compute the lift and drag coefficients, C_L, C_D, the open-source neuralfoil aerofoil simulation framework [14] was used. Coefficients and lift to drag ratios for Mach 0.001 - 0.01, or 1.2 - 12km/h were plotted.



(a) Lift, drag coefficients for very low Mach range.

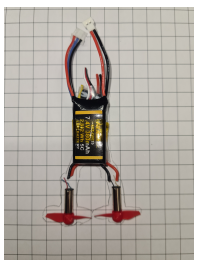


(b) Weight of components.

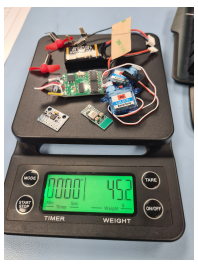
Fig. 3: Lift to drag ratio at a range of angles of attack and Reynolds numbers.

Seeing a linear L/D ratio in Figure 3 is unsurprising for a symmetric aerofoil, but serves as a good indication of the neuralfoil framework functioning. We can use the estimated C_D, C_L coefficients for our Vehicle constructor in our 6-DOF model.

2) Gondola: Here the additional weight is minimised as much as possible as this reduces the required balloon volume and lifting gas.



(a) Electronics on reference grid.



(b) Weight of components.

Fig. 4: Electronics housing and components.

C. Aerodynamic coefficients, moments of inertia

Aerodynamic coefficients for the blimp are sourced from chapter 56, (G. Leishman 2022) [6].

The `Vehicle` class requires moments of inertia in the body axis, a reference area S_A, lengths a_l, b_l, c_l, d_l for roll, pitch, yaw and Reynolds number respectively.

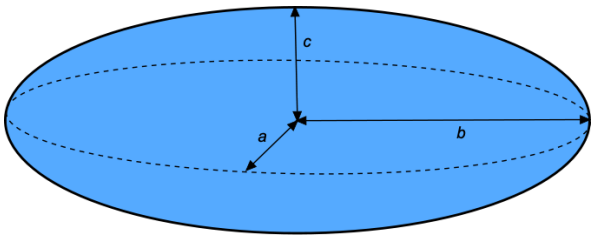


Fig. 5: Geometric Characteristics of Aerostats (G. Leishman 2022) [6]

For our vehicle, the required volume of helium to be neutrally buoyant, including the masses of the vehicle's components, at the troposphere was used to dimension the design. Further setting $a = c$, let us choose a radius, a, and a length b, for a given volume.

Neutral buoyancy is achieved when:

$$G_B = G_G \tag{2}$$

$$\rho_{\text{air}} g V = mg = 0.66848 \tag{3}$$

$$V_{\text{troposphere}} = \frac{m_{\text{total}} \times g}{\rho_{\text{troposphere}}} = 1.842[\text{m}^3] \tag{4}$$

$$b = \frac{3V_{\text{troposphere}}}{4\pi a^2} = 2.749[\text{m}] \tag{5}$$

For a = 0.4m and $\rho_{\text{troposphere}} = 0.3639$ [kg/ m^3], as per ISA. For simplicity, we will only evaluate the relevant coefficients when the balloon is fully inflated. At sea level, the volume will be lower. As part of the mass budget, 1 micron thick mylar is used. See *modeling/balloon_sizing.py*. Reference area, length and slenderness ratio are:

$$A_{\text{ref}} = V^{\frac{2}{3}} = 1.503 \tag{6}$$

$$L_{\text{ref}} = V^{\frac{1}{3}} = 1.226 \tag{7}$$

$$\lambda = \frac{b}{a} = 6.87 \tag{8}$$

Lift, drag and moment coefficients at $V_{\text{inf}} = 2$[m/s], $\rho_{\text{troposphere}} = 0.3639$, and Reynold's number:

$$\text{Re} = \frac{\rho V_{\text{inf}} L_{\text{ref}}}{\mu_{air}} = 28077 \tag{9}$$

$$C_D = \frac{2D}{\rho_{\text{inf}} V_{\text{inf}}^2 V^{\frac{2}{3}}} \tag{10}$$

At zero angle of attack an approximation holds, from measurements recorded in a wind tunnel [5]:

$$C_D \approx \frac{0.045}{Re^{\frac{1}{6}}} \times (4\lambda^{\frac{1}{3}} + 6\lambda^{\frac{-7}{6}} + 24\lambda^{\frac{-8}{3}}) \approx 0.068386$$

$$C_L = \frac{2L}{\rho_{\text{inf}} V_{\text{inf}}^2 V^{\frac{2}{3}}} = 0.6111 \tag{11}$$

$$C_{Ma} = \frac{2M_a}{\rho_{\text{inf}} V_{\text{inf}}^2 V} \tag{12}$$

And elevator deflection changes our lift and drag coefficients by:

$$\Delta C_L = C_1 \alpha + C_2 \delta_E \tag{13}$$

$$\Delta C_D = D_1 \Delta C_L + D_2 \delta_E \tag{14}$$

Where $C_1 \approx 0.000146$ and $C_2 \approx 0.01$ per degree of α, δ_E and $D_1 \approx 0.01$ and $D_2 \approx 0.02$ are reasonable values given wind tunnel measurements from NACA report 394 [5].

Moments of inertia:

The vehicle is left-right symmetrical, so $I_{xy} = I_{yz} = 0$. This gives us an inertia tensor defined as:

$$\overrightarrow{I} = \begin{bmatrix} I_x & 0 & -I_{xz} \\ 0 & I_y & 0 \\ -I_{xz} & 0 & I_z \end{bmatrix} \tag{15}$$

We use the body axes as defined in Figure 1 and Equation 1. For an ellipsoid, the inertia tensor only has non-zero entries on the diagonal, due to symmetry.

For a general ellipsoid, the cartesian coordinates are

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \tag{16}$$

Using the same dimensions as shown in Figure 5, this can be parameterised with:

$$x = ar \sin\theta \sin\phi \tag{17}$$

$$y = br \sin\theta \cos\phi \tag{18}$$

$$z = cr \cos\theta \tag{19}$$

The Jacobian for these coordinates is $abcr^2 \sin\theta$, which gives us the integral:

$$I_{ij} = \rho_0 \int \mathrm{d}r\ \mathrm{d}\theta\ \mathrm{d}\phi\ abcr^2 \sin\theta [r^2(\sin^2\theta(a^2\cos^2\phi + b^2\sin^2\phi) + c^2 \cos^2\theta)\delta_{ij} - x_i x_j] \tag{20}$$

For simplicity, density is assumed as constant, so some error for the true flight vehicle would arise here. Setting $\rho_0 = \frac{3M}{4\pi abc}$ then yields the inertia tensor, for our vehicle $(a = c)$:

$$\overrightarrow{I} = \frac{M}{5} \begin{bmatrix} b^2 + c^2 & 0 & 0 \\ 0 & a^2 + c^2 & 0 \\ 0 & 0 & a^2 + b^2 \end{bmatrix} \tag{21}$$

$$\overrightarrow{I} = \begin{bmatrix} 0.1055 & 0 & 0 \\ 0 & 0.00438 & 0 \\ 0 & 0 & 0.1055 \end{bmatrix} \tag{22}$$

D. Model setup

The modeling approach is outlined in Figure 6.

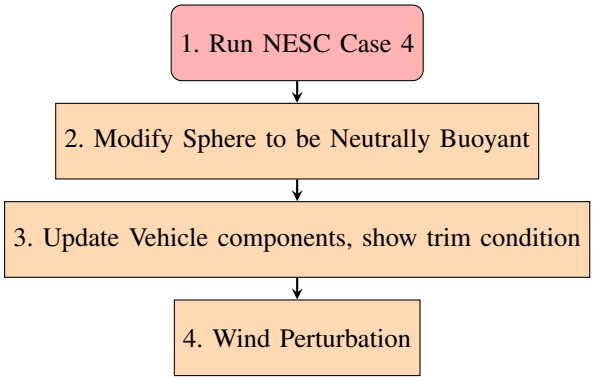


Fig. 6: Modeling approach.

1) Run NESC Case 4: We started our approach with a simple implementation of the Vehicle and Planet instances, using a sphere and basic gravity with no wind respectively. Rework of deprecated code in simupy-flight [1] and simupy was required before a test case was executable. Substituting `np.Inf` with `np.inf` in `simupy/block_diagram.py` and `np._float` with `np.float64` in `simupy_flight/__init__.py` and `simupy/block_diagram.py` fixed the issues.

For some cases, such as checking neutral buoyancy, running the model for longer than the example 30 seconds was of interest. This was achieved by modifying the simulation section with additional integrator options, see *modeling/nesc_case4.py*

2) Modify Sphere to be Neutrally Buoyant: To implement buoyancy, we needed to add in a buoyancy force to the vehicle. This was done with the Vehicle's `input_force_moment` function.

3) Update Vehicle components, show trim condition: Earlier we defined the aerodynamic coefficients and moments of inertia of our vehicle. These were inputted to the Vehicle class, alongside a small thrust from the motors.

4) Wind Perturbation: A perturbing force is shown in NESC example 7, as a bulk wind speed. This was added and the vehicle's response examined.

IV. Results and Discussion

A. Trim condition at lower stratosphere

NESC case 4 was chosen as a starting point as it shows a dropped sphere over non-rotating, spherical Earth. From this case we can verify the gravity model and develop our buoyancy force as an additional input.

Simupy uses a block diagram approach to modeling, where you can connect the elements of your dynamical system in a coupled manner to its BlockDiagram class [9]. The connection order is important as it determines the output dimensions of the simulation. In Figure 7 we visualised the model output. The planet class was initialised with constant winds and spherical gravity:

```
planet = simupy_flight.Planet(
gravity=simupy_flight.get_spherical_gravity(
    ↪ simupy_flight.earth_spherical_gravity_constant
),
winds=simupy_flight.get_constant_winds(),
atmosphere=simupy_flight.atmosphere_1976,
planetodetics=simupy_flight.Planetodetic(
    a=20902255.199 / ft_per_m, omega_p=0.0,
    ↪ f=0.0
),
)
```

However oscillations in wind speed occur at $t > 25$, which we noted for our future model runs.

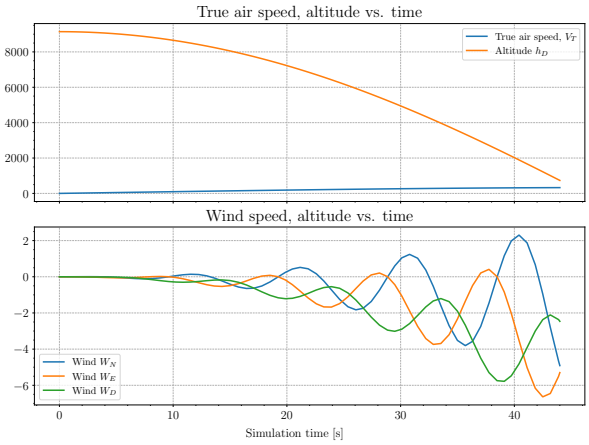


Fig. 7: NESC case 4 */modeling/nesc_case4.py*.

To implement a buoyancy force, we alter the Vehicle class to have an input force. Since we set $G_G = G_B = mg$, we add this to the model using the `input_force_moment` method.

The model's implementation of standard gravity did not seem to coincide with expected values for a simple spherical Earth 8. Using sea-level gravity of 9.81 [m/s] for the buoyancy force resulted in the vehicle gaining altitude over time, whereas the true gravity value of 9.794 undershot. To move forward with our model, an average was taken and this was noted.

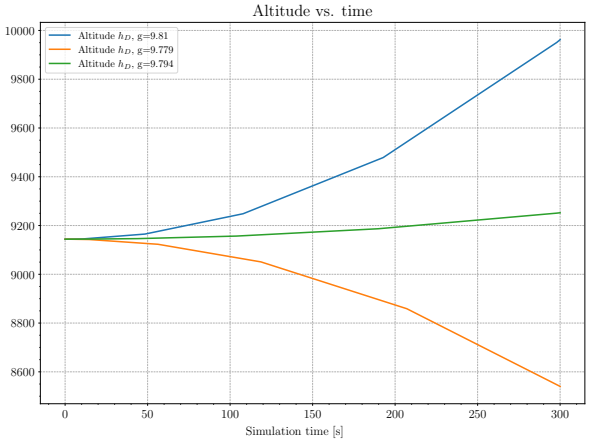


Fig. 8: Different values of gravity for model.

Trimmed flight occurs when the resultant forces and pitching moments in each direction are zero.

We inputted all coefficients and moments defined in Section III-C in *modeling/trim_condition_static.py* via the `base_aero_coeffs` function of the `Vehicle` class and the respective moments and reference areas and lengths of its attributes.

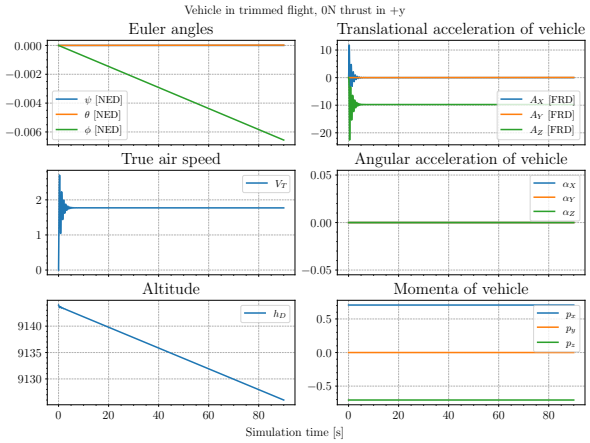


Fig. 9: Output of *modeling/trim_condition_static.py*.

The vehicle showed good stability for zero thrust, with constant momenta and accelerations, demonstrating a trim condition. We note the small decrease in altitude arising from the mismatch in the $G_G = G_B$ relation mentioned earlier.

B. Response to perturbation at operating point

When applying a small force in the Y direction, by setting `FY=2` in the `input_force_moment` function of the `Vehicle` class, simulating thrust from the motors, the model showed severe instability. The integrator exited after a little over a second of model time, even with the `nsteps` input argument raised to 5000, which was sufficient for the NESC F16 test case.

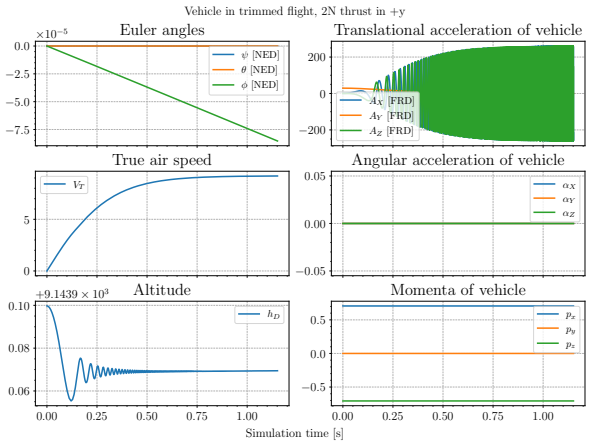


Fig. 10: Output of *modeling/perturbation.py*.

Several methods of adding the perturbation was tested. Below a description and relevant lines of code in *modeling/perturbation.py* are presented:

- Using a callback `perturbation_force` function [116-117, 138]
- Overloading provided simupy-flight `get_constant_force_moments` function [128-137]
- Adding an initial condition directly [184]

Where the first two methods resulted in the same output and the final method crashed the program, as the Vehicle class' initial conditions are forced to zero, then not implemented.

C. Handling/Flying Qualities of vehicle.

LTA craft are known for their stability [2] and simple control [4] [15]. Generally, LTA have some known pitch oscillation for maneuvers as their thrust moment is at some distance to their centre of gravity and buoyancy [6]. While oscillations were seen in the body z-direction, this is not likely to be the same phenomena and is probably due to model instability. From the design of the vehicle, it is likely to be quite maneuverable in the body x and pitch directions from thrust vectoring, but its roll and pitch maneuvering is likely to be poor. This is due to small control surfaces to induce moments about these axes. Due to the vehicle's neutral buoyancy, it will be susceptible to wind gusts. For the operating point, a sufficient propulsion system, for orientation and resisting wind displacement should be considered.

V. Conclusion

An LTA blimp composed of control surfaces, a lifting gas, electronics and a payload was designed III. Its aerodynamic coefficients and moments of inertia were estimated III-C and a trim condition was demonstrated IV-A. Inputting these into the simupy-flight modeling framework was challenging, with refactoring of the modeling software required III-D1, alongside noted instability in wind speed and the implemented gravity model. A perturbation was introduced into the initial condition of the model via the `Vehicle` class IV-B, but resulted in the model solver exiting early in various different implementation methods. Finally, the flying qualities of the vehicle were discussed, with some notable qualities of the vehicle being its stability and responses to external moments.

VI. Appendix

The python scripts referenced above are added here. Imports and plotting related code is omitted for brevity. See [8], in the `modeling` folder for full versions and better legibility, formatting.

A. *balloon_sizing.py*

```
# Sea level gravity
g = 9.8066

# at sea level:
rho_atm = 1.225  # kg/m^3
rho_helium = 0.178  # kg/m^3

# ISA defines density at the tropopause
↪  (11,000m) as 0.3639 kg/m^3
rho_tropopause = 0.3639

# component masses [kg]
total_mass = sum(
    [
        # servos
        ((4.8 + 5) / 1000),
        # ESP32
        (3.7 / 1000),
        # Accelerometer
        (1.1 / 1000),
        # LiPo battery
        (17.8 / 1000),
        # Motors
        ((2 + 2.2) / 1000),
        # Voltage regulator
        (2 / 1000),
        # Motor ESC
        (6.9 / 1000),
        # 3-D printed parts, output from
        ↪  Prusa Slicer
        # Aerofoil & gondola
        (17 / 1000),
        # Density of mylar is: 1400 kg/m3
        # use a mylar thickness of 1 micron
        # Approximating a sphere;
        ↪  4pi/3*(r2-r1)*rho_mylar
        1400 * ((4 * np.pi) / 3) * (1 *
        ↪  10**-6),
    ]
)

# We use: A = Vballoon * rho_air * g = Gg
# Thus Vballoon = Gg / rho_air

v_balloon = (total_mass * g) / rho_atm
v_balloon_tropopause = (total_mass * g) /
↪  rho_tropopause
```

B. *neutral_case4.py*

```
af = asb.Airfoil("naca0012")

re = np.geomspace(1e5, 1e7, 50)
# This is about 1.2 -> 12 kmh at sea level
mach = np.linspace(0.001, 0.010, 50)
alpha = np.linspace(-3, 7, 50)

Re, Mach, Alpha = np.meshgrid(re, mach,
↪  alpha)

aero_f = af.get_aero_from_neuralfoil(
    alpha=Alpha.flatten(),
    Re=Re.flatten(),
    mach=Mach.flatten(),
)
aero = {k: np.reshape(v, Re.shape) for k, v
↪  in aero_f.items()}
```

C. *aerofoil_design.py*

```
planet = simupy_flight.Planet(

    ↪  gravity=simupy_flight.get_spherical_gravity(

        ↪  simupy_flight.earth_spherical_gravity_constant
    ),
    winds=simupy_flight.get_constant_winds(),
    atmosphere=simupy_flight.atmosphere_1976,
    planetodetics=simupy_flight.Planetodetic(
        a=20902255.199 / ft_per_m,
        ↪  omega_p=0.0, f=0.0
    ),
)

Ixx = 3.6 * kg_per_slug / (ft_per_m**2)  #
↪  slug-ft2
Iyy = 3.6 * kg_per_slug / (ft_per_m**2)  #
↪  slug-ft2
Izz = 3.6 * kg_per_slug / (ft_per_m**2)  #
↪  slug-ft2
Ixy = 0.0 * kg_per_slug / (ft_per_m**2)  #
↪  slug-ft2
Iyz = 0.0 * kg_per_slug / (ft_per_m**2)  #
↪  slug-ft2
Izx = 0.0 * kg_per_slug / (ft_per_m**2)  #
↪  slug-ft2
m = 1.0 * kg_per_slug  # slug

x = 0.0
y = 0.0
z = 0.0

S_A = 0.1963495 / (ft_per_m**2)
b_l = 1.0
c_l = 1.0
a_l = b_l

# Height for initial conditions
h_ic = 30_000 / ft_per_m
R = 6371 * 1000
g_0 = 9.807  # gravity at sea level

g_ic = g_0 * ((R / (R + h_ic)) ** 2)

# Add in buoyancy force.
vehicle = simupy_flight.Vehicle(
    base_aero_coeffs=(
    simupy_flight.get_constant_aero(
    CD_b=0.068386
    )),
    input_force_moment=(
    simupy_flight.get_constant_force_moments(
    FZ=-m * g_ic
    )),
    m=m,
    I_xx=Ixx,
    I_yy=Iyy,
    I_zz=Izz,
    I_xy=Ixy,
    I_yz=Iyz,
    I_xz=Izx,
    x_com=x,
    y_com=y,
    z_com=z,
    x_mrc=x,
    y_mrc=y,
    z_mrc=z,
    S_A=S_A,
    a_l=a_l,
    b_l=b_l,
    c_l=c_l,
    d_l=0.0,
)

# Now add buoyancy force, can't just use
↪  g=9.81 as the gravity model is complex:
```

```
# gravity_model(p_x,p_y,p_z).ravel()
# This is callable as a method though:

BD = BlockDiagram(planet, vehicle)
BD.connect(planet, vehicle,
↪  inputs=np.arange(planet.dim_output))
BD.connect(vehicle, planet,
↪  inputs=np.arange(vehicle.dim_output))

lat_ic = 0.0 * np.pi / 180
long_ic = 0.0 * np.pi / 180

V_N_ic = 0.0
V_E_ic = 0.0
V_D_ic = 0.0

psi_ic = 0.0 * np.pi / 180
theta_ic = 0.0 * np.pi / 180
phi_ic = 0.0 * np.pi / 180

# omega_X_ic = 10.0 * np.pi / 180
# omega_Y_ic = 20.0 * np.pi / 180
# omega_Z_ic = 30.0 * np.pi / 180

omega_X_ic = 0.0 * np.pi / 180
omega_Y_ic = 0.0 * np.pi / 180
omega_Z_ic = 0.0 * np.pi / 180

planet.initial_condition =
↪  planet.ic_from_planetodetic(
    long_ic, lat_ic, h_ic, V_N_ic, V_E_ic,
    ↪  V_D_ic, psi_ic, theta_ic, phi_ic
)
planet.initial_condition[-3:] = omega_X_ic,
↪  omega_Y_ic, omega_Z_ic


DEFAULT_INTEGRATOR_OPTIONS = {
    "name": "dopri5",
    "rtol": 1e-6,
    "atol": 1e-12,
    "nsteps": 500,
    "max_step": 0.0,
}

with benchmark() as b:
    res_fine_gravity = BD.simulate(
    300,
    integrator_options=(
    DEFAULT_INTEGRATOR_OPTIONS)
    )
```

D. *trim_condition_static.py*

```
planet = simupy_flight.Planet(
    gravity=simupy_flight.earth_J2_gravity,
    winds=simupy_flight.get_constant_winds(),
    atmosphere=simupy_flight.atmosphere_1976,
    planetodetics=simupy_flight.Planetodetic(
    a=simupy_flight.earth_equitorial_radius,

    ↪  omega_p=simupy_flight.earth_rotation_rate,
    f=simupy_flight.earth_f,
    ),
)


# Inertia tensor is:
#   I = M/5 * (
#                b^2+c^2    0       0
#                   0    a^2+c^2    0
#                   0       0    a^2+b^2
#             )
# We have a = c = 0.4 and b = 2.749 [m], M =
↪  0.06836 [kg]

# This gives:
#   I = M/5 * (
#                b^2+a^2    0       0
#                   0      2a^2     0
#                   0       0    a^2+b^2
#             )

#     = 0.06836/5 * (
#                2.749^2+0.4^2    0       0
#                   0      2*0.4^2     0
#                   0       0    0.4^2+2.749^2
#             )

# This gives:
#   I = (
#        0.1055     0       0
#           0    0.00438    0
#           0       0     0.1055
#       )

# Units are kg/m^2
Ixx = 0.1055
Iyy = 0.00438
Izz = 0.1055
Ixy = 0.0
Iyz = 0.0
Izx = 0.0
m = 0.06836

# Stay the same
x = 0.0
y = 0.0
z = 0.0

S_A = 1.503
a_l = 1.226
b_l = 1.226
c_l = 1.226
d_l = 1.226

# Height for initial conditions
h_ic = 30_000 / ft_per_m
R = 6371 * 1000
g_0 = 9.807  # gravity at sea level

g_ic = g_0 * ((R / (R + h_ic)) ** 2)

# See report for justification
g_avg = (g_ic + 9.81) / 2

# See report for derivations.
vehicle = simupy_flight.Vehicle(

    ↪  base_aero_coeffs=simupy_flight.get_constant_aero(
        CD_b=0.068386,
        CL_b=0.6111,
    ),
    #
    ↪  input_force_moment=simupy_flight.get_constant_force_m
    #     # FZ=-m * g_avg
    #     FY=2
    # ),
    m=m,
    I_xx=Ixx,
    I_yy=Iyy,
    I_zz=Izz,
    I_xy=Ixy,
    I_yz=Iyz,
    I_xz=Izx,
    x_com=x,
    y_com=y,
    z_com=z,
    x_mrc=x,
    y_mrc=y,
    z_mrc=z,
    S_A=S_A,
    a_l=a_l,
    b_l=b_l,
    c_l=c_l,
```

```
    d_l=d_l,
)

BD = BlockDiagram(planet, vehicle)
BD.connect(planet, vehicle,
↪  inputs=np.arange(planet.dim_output))
BD.connect(vehicle, planet,
↪  inputs=np.arange(vehicle.dim_output))

lat_ic = 0.0 * np.pi / 180
long_ic = 0.0 * np.pi / 180

V_N_ic = 0.0
V_E_ic = 0.0
V_D_ic = 0.0

psi_ic = 0.0 * np.pi / 180
theta_ic = 0.0 * np.pi / 180
phi_ic = 0.0 * np.pi / 180

omega_X_ic = 0.0 * np.pi / 180
omega_Y_ic = 0.0 * np.pi / 180
omega_Z_ic = 0.0 * np.pi / 180

planet.initial_condition =
↪  planet.ic_from_planetodetic(
    long_ic, lat_ic, h_ic, V_N_ic, V_E_ic,
    ↪  V_D_ic, psi_ic, theta_ic, phi_ic
)
planet.initial_condition[-3:] = omega_X_ic,
↪  omega_Y_ic, omega_Z_ic


DEFAULT_INTEGRATOR_OPTIONS = {
    "name": "dopri5",
    "rtol": 1e-6,
    "atol": 1e-12,
    "nsteps": 5000,
    "max_step": 0.0,
}

with benchmark() as b:
    res = BD.simulate(
    90,

    ↪  integrator_options=DEFAULT_INTEGRATOR_OPTIONS
    )
```

E. perturbation.py

```
planet = simupy_flight.Planet(
    gravity=simupy_flight.earth_J2_gravity,
    winds=simupy_flight.get_constant_winds(),
    atmosphere=simupy_flight.atmosphere_1976,
    planetodetics=simupy_flight.Planetodetic(

        ↪  a=simupy_flight.earth_equitorial_radius,

        ↪  omega_p=simupy_flight.earth_rotation_rate,
        f=simupy_flight.earth_f,
    ),
)


# Inertia tensor is:
#   I = M/5 * (
#               b^2+c^2    0       0
#                  0    a^2+c^2    0
#                  0       0    a^2+b^2
#             )
# We have a = c = 0.4 and b = 2.749 [m], M =
↪  0.06836 [kg]

# This gives:
#   I = M/5 * (
#               b^2+a^2    0       0
#                  0      2a^2     0
#                  0       0    a^2+b^2
#             )

#     = 0.06836/5 * (
#                2.749^2+0.4^2    0       0
#                   0      2*0.4^2     0
#                   0       0    0.4^2+2.749^2
#             )

# This gives:
#   I = (
#        0.1055     0       0
#           0    0.00438    0
#           0       0     0.1055
#       )

# Units are kg/m^2
Ixx = 0.1055
Iyy = 0.00438
Izz = 0.1055
Ixy = 0.0
Iyz = 0.0
Izx = 0.0
m = 0.06836

# Stay the same
x = 0.0
y = 0.0
z = 0.0

S_A = 1.503
a_l = 1.226
b_l = 1.226
c_l = 1.226
d_l = 1.226

# Height for initial conditions
h_ic = 30_000 / ft_per_m
R = 6371 * 1000
g_0 = 9.807  # gravity at sea level

g_ic = g_0 * ((R / (R + h_ic)) ** 2)

# See report for justification
g_avg = (g_ic + 9.81) / 2

# Callback function:
def perturbation_force(*args):
    return np.array([0.0, 2.0, 0.0, 0.0, 0.0,
    ↪  0.0])


# See report for derivations.
vehicle = simupy_flight.Vehicle(

    ↪  base_aero_coeffs=simupy_flight.get_constant_aero(
        CD_b=0.068386,
        # CL_b=0.0,
        CL_b=0.6111,
    ),
    # Overload default input:
    #
    ↪  input_force_moment=simupy_flight.get_constant_force_m
    #     # FZ=-m * g_avg
    #     # FY=2
    #     FX=0,
    #     FY=2,
    #     FZ=0,
    #     MX=0,
    #     MY=0,
    #     MZ=0,
    # ),
    input_force_moment=perturbation_force,
    m=m,
    I_xx=Ixx,
    I_yy=Iyy,
    I_zz=Izz,
```

```
    I_xy=Ixy,
    I_yz=Iyz,
    I_xz=Izx,
    x_com=x,
    y_com=y,
    z_com=z,
    x_mrc=x,
    y_mrc=y,
    z_mrc=z,
    S_A=S_A,
    a_l=a_l,
    b_l=b_l,
    c_l=c_l,
    d_l=d_l,
)

BD = BlockDiagram(planet, vehicle)
BD.connect(planet, vehicle,
↪  inputs=np.arange(planet.dim_output))
BD.connect(vehicle, planet,
↪  inputs=np.arange(vehicle.dim_output))

lat_ic = 0.0 * np.pi / 180
long_ic = 0.0 * np.pi / 180

V_N_ic = 0.0
V_E_ic = 0.0
V_D_ic = 0.0

psi_ic = 0.0 * np.pi / 180
theta_ic = 0.0 * np.pi / 180
phi_ic = 0.0 * np.pi / 180

omega_X_ic = 0.0 * np.pi / 180
omega_Y_ic = 0.0 * np.pi / 180
omega_Z_ic = 0.0 * np.pi / 180

planet.initial_condition =
↪  planet.ic_from_planetodetic(
    long_ic, lat_ic, h_ic, V_N_ic, V_E_ic,
    ↪  V_D_ic, psi_ic, theta_ic, phi_ic
)
planet.initial_condition[-3:] = omega_X_ic,
↪  omega_Y_ic, omega_Z_ic

# Adding initial condition directly:
# vehicle.initial_condition[5] =
↪  np.array([2])


DEFAULT_INTEGRATOR_OPTIONS = {
    "name": "dopri5",
    "rtol": 1e-6,
    "atol": 1e-12,
    "nsteps": 50000,
    "max_step": 0.0,
}

with benchmark() as b:
    res = BD.simulate(90,
    ↪  integrator_options=DEFAULT_INTEGRATOR_OPTIONS)
```

REFERENCES

[1] Brian McCormick, et al. (2015) Github - nasa/simupy-flight. [Online]. Available: https://github.com/nasa/simupy-flight

[2] M. Burri, L. Gasser, M. Käch, M. Krebs, S. Laube, A. Ledergerber, D. Meier, R. Michaud, L. Mosimann, L. Müri, C. Ruch, A. Schaffner, N. Vuilliomenet, J. Weichart, K. Rudin, S. Leutenegger, J. Alonso-Mora, R. Siegwart, and P. Beardsley, "Design and control of a spherical omnidirectional blimp," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1873–1879.

[3] H. Cheng, Z. Sha, Y. Zhu, and F. Zhang, "Rgblimp: Robotic gliding blimp - design, modeling, development, and aerodynamics analysis," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7273–7280, 2023.

[4] S. Cho, Q. Tao, P. Varnell, S. Maxon, and F. Zhang, "Autopilot design of a class of miniature autonomous blimps enabled by switched controllers," *International Journal of Intelligent Robotics and Applications*, vol. 6, pp. 385–396, 9 2022. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=0f11cbf5-9df6-3432-8682-6ba3677262c4

[5] f. given i=H, given=Ira, "Airship model tests in the variable density wind tunnel." [Online]. Available: https://ntrs.nasa.gov/citations/19930091467

[6] f. given i=J, given=Gordon and Embry-Riddle Aeronautical University, *Introduction to Aerospace Flight Vehicles*. Embry-Riddle Aeronautical University. [Online]. Available: https://eaglepubs.erau.edu/introductiontoaerospaceflightvehicles/chapter/lth/

[7] S. Lowis, "Blimp," 2024. [Online]. Available: https://cad.onshape.com/documents/bee8e34993cbbd3d01a73270/w/f7f296099beac5cb308246c7/e/c67e3bb3622a29c3bdaf1189

[8] ——, "GitHub - shaunlowis/hydroblimp: making a hydrogen generator and a remote control mini blimp," 2024. [Online]. Available: https://github.com/shaunlowis/hydroblimp

[9] B. W. L. Margolis. (2017) Api documentation — simupy 1.0.0 documentation. [Online]. Available: https://simupy.readthedocs.io/en/latest/api/api.html

[10] NACA. Naca0012h for vawt from sandia report sand80-2114 (naca0012h-sa). [Online]. Available: http://airfoiltools.com/airfoil/details?airfoil=naca0012h-sa

[11] NASA. Naca airfoils - nasa. [Online]. Available: https://www.nasa.gov/image-article/naca-airfoils/

[12] NOAA's National Weather Service. Radiosonde observation. [Online]. Available: https://www.weather.gov/upperair/factsheet

[13] O. M. Raspopov, S. N. Sokolov, I. M. Demina, R. Pellinen, and A. A. Petrova, "The first aeromagnetic survey in the arctic: results of the graf zeppelin airship flight of 1931." *History of Geo & Space Sciences*, vol. 4, pp. 35–46, 1 2013. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=710df47c-33cc-3f50-aa9c-7d6df3fa3693

[14] P. Sharpe, "NeuralFoil: An airfoil aerodynamics analysis tool using physics-informed machine learning," https://github.com/peterdsharpe/NeuralFoil, 2023.

[15] Y. Wang, G. Zheng, D. Efimov, and W. Perruquetti, "Disturbance compensation based control for an indoor blimp robot," pp. 2040–2046, 5 2019. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=c52ff684-4d32-38fd-a1d4-e3fa35707791