HYDRA: Rotorcraft Conceptual Sizing

Authors

Ananth Sridharan
Bharath Govindarajan

This is the analysis manual for the rotorcraft conceptual sizing analysis HYDRA - HYbrid Design and Rotorcraft Analysis, developed and evolved over several years at the University of Maryland, with inspiration drawn from the AHS Student Design Competition challenges. This manual contains a description of the theory and various operations performed by the sizing code for both conventional and novel Vertical-Lift aircraft.

The main advantages of HYDRA over other sizing codes are **flexibility**, **speed** and reliance on only **open-source tools**. With the majority of the code written in an interpreted language, i.e., Python, modules can be prototyped and added quickly. Subsequently select parts of the code can be ported to Fortran or C (i.e., compiled languages) and wrapped for execution sppeed. Using a combination of OpenMP, MPI and algorithmic acceleration, up to 2000 designs can be generated per second on a standard desktop computer with 8 cores.

Another advantage of HYDRA is the ability to set up input decks and call higher fidelity models (BEMT, FEA for airframe and wings) and the comprehensive analysis PRASADUM, and through the comprehensive analysis, the Maryland Free Wake (MFW). These higher-fidelity tools are integrated into the sizing loop to provide accurate estimates of rotor performance and component weights, and may be invoked as required.

Contents

1 VTOL sizing: Overview

The objective of rotorcraft sizing is to obtain a consistent combination of weights and performance for the various components in a VTOL platform. The interdependent nature of the design of various parts (e.g., engine, rotors, transmission, fuselage) necessitates methods that capture these dependencies to the required accuracy; otherwise components might be over-designed and heavier/bulkier than required (performance penalty) or under-designed, requiring a costly sequence of redesign operations for all parts in the detailed design phase.

Sizing for vertical-lift aircraft is not a new field; the fundamental methods have been adapted from fixed-wing sizing and augmented with VTOL-specific components, and refined over several decades. A notable example of a VTOL sizing algorithm is the method of Prof. Marat Tishchenko (MIL design bureau) and Dr. V.T. Nagaraj; this combination of component sizing laws and the iterative sizing sequence is one of the cornerstones of `HYDRA`.

Recent work by Dr. Wayne Johnson at NASA Ames produced several iterations of a continuously updated NASA code called `NDARC`. This code can size conventional and unconventional VTOL platforms, as well as a range of fixed-wing aircraft. Additionally, each major helicopter manufacturer has built up a database of manufactured part weights and performance, and have developed their respective in-house sizing tools. These sizing tools were originally developed for single main rotor helicopters powered by gas turbine engines, and have been gradually adapted to include eVTOLs (electric Vertical TakeOff and Landing platforms) and configurations with hybrid powerplants featuring Distributed Electric Propulsion (DEP).

1.1 The nature of VTOL sizing

Table 1: Conceptual sizing: drivers for group mass

Group name	Mass depends on
Fixed wing	Aspect ratio, wing loading $\rightarrow$ **take-off mass**
Rotors	Radius, solidity, RPM, torque, thrust $\rightarrow$ **take-off mass**
Empennage	Tail loading, pitch/yaw stability and control authority
Propeller	Vehicle drag $\rightarrow$ **take-off mass**
Fuselage	Dimensions, **take-off mass**
Landing gear	**Take-off mass**
Flight controls	Blade geometry, wing area, wing loading $\rightarrow$ **take-off mass**
Deicing	Wing area, rotor blade area $\rightarrow$ **take-off mass**
Battery	C-rating, specific energy, rotor power $\rightarrow$ **take-off mass**
Motor	Maximum torque/power $\rightarrow$ **take-off mass**
Avionics	Fixed mass group
Power wire	Layout of rotors and motor power limits $\rightarrow$ **take-off mass**
Signal wire	Layout of controls and sensors

To illustrate the interdependent nature of sizing laws for various components, consider the dominant drivers for the weights of various vehicle components in an eVTOL, shown in Table 1. The list of dependencies is illustrative and not exhaustive, but sufficient to highlight the key point: **determination of take-off mass requires solving for several cyclic dependencies**, i.e., the total take-off mass depends on *itself*. Two well-known methods have been successfully used to solve this class of problems with simultaneous nonlinear equations:

1. Iterative convergence

2. Optimization with compatibility constraints

The implementations of both approaches in `HYDRA` are detailed in the following sections.

1.1.1 Iterative convergence

Iterative convergence is a popular technique that is used to solve several classes of problems. It has been successfully applied to nonlinear algebraic equations in several variables, where analytical solutions are unavailable. The overall approach is shown in Fig. 1. Four primary types of models are required for conceptual sizing:

1. Weight

2. Performance

3. Sizing and geometry

4. Energy storage

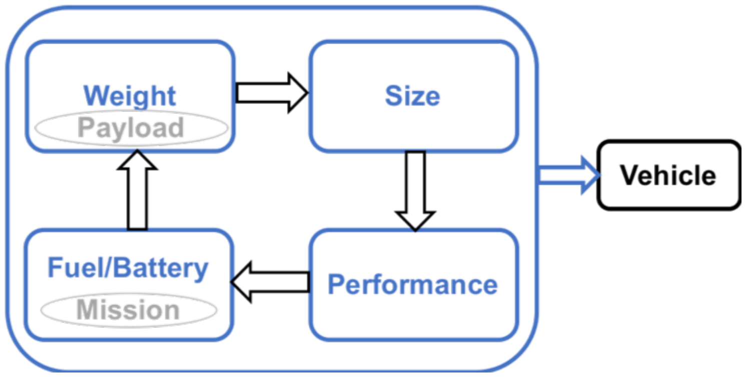


Figure 1: Iterative sizing: the classical approach

Typically, low-fidelity models are used to represent each group for conceptual sizing, because fine-grained resolution of each component is not possible at this stage due to the unavailability of design details. For VTOL sizing in particular, sizing operations can be executed in a particular sequence to obtain answers rapidly with fewer sizing iterations. One such sequence is shown in Fig. 2.

Each block shown in Fig. 2 consists of further sub-modules that are assembled together in the implementation. The key inputs for sizing are the three green blocks that feed information to the sizing of the vehicle: **aircraft layout**, **mission profile** and **model calibration data**.

The **aircraft layout** is perhaps the most significant assumption required to initiate sizing. Figure 3 shows a few candidate eVTOL configurations, with different permutations and combinations of rotor and wing technologies, as shown in Fig. 4. In `HYDRA`, the constituent rotor and wing technologies are defined as primitive types. Based on user inputs, these technology combinations are superimposed in `HYDRA` to obtain the appropriate configuration.

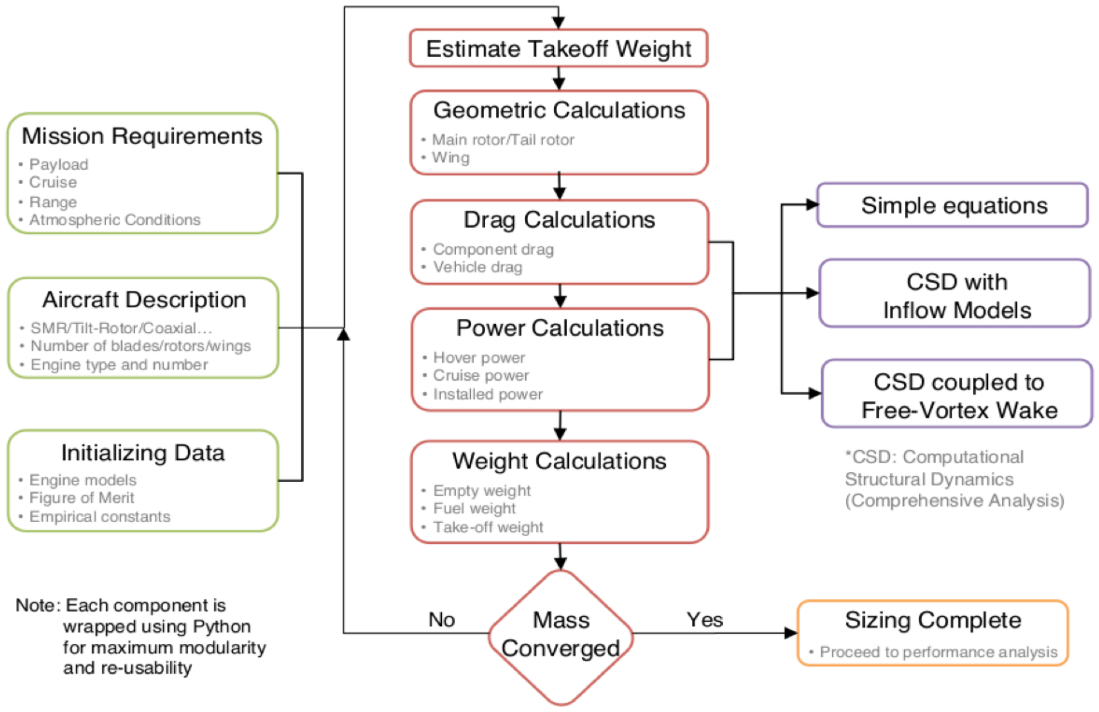


Figure 2: Iterative sizing: detailed breakdown of operations

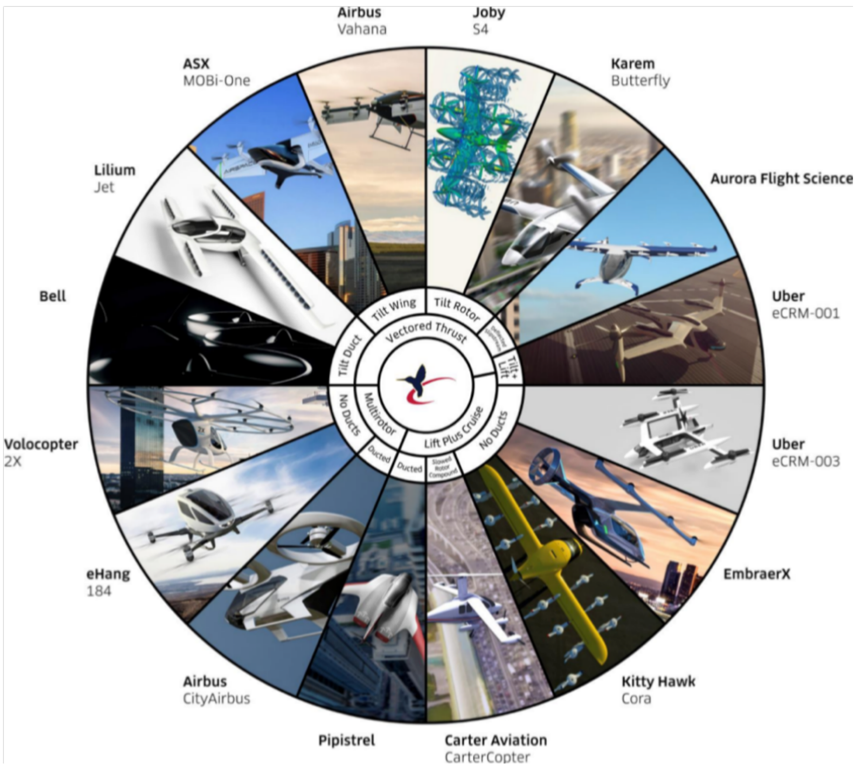


Figure 3: Variety in electric VTOL aircraft

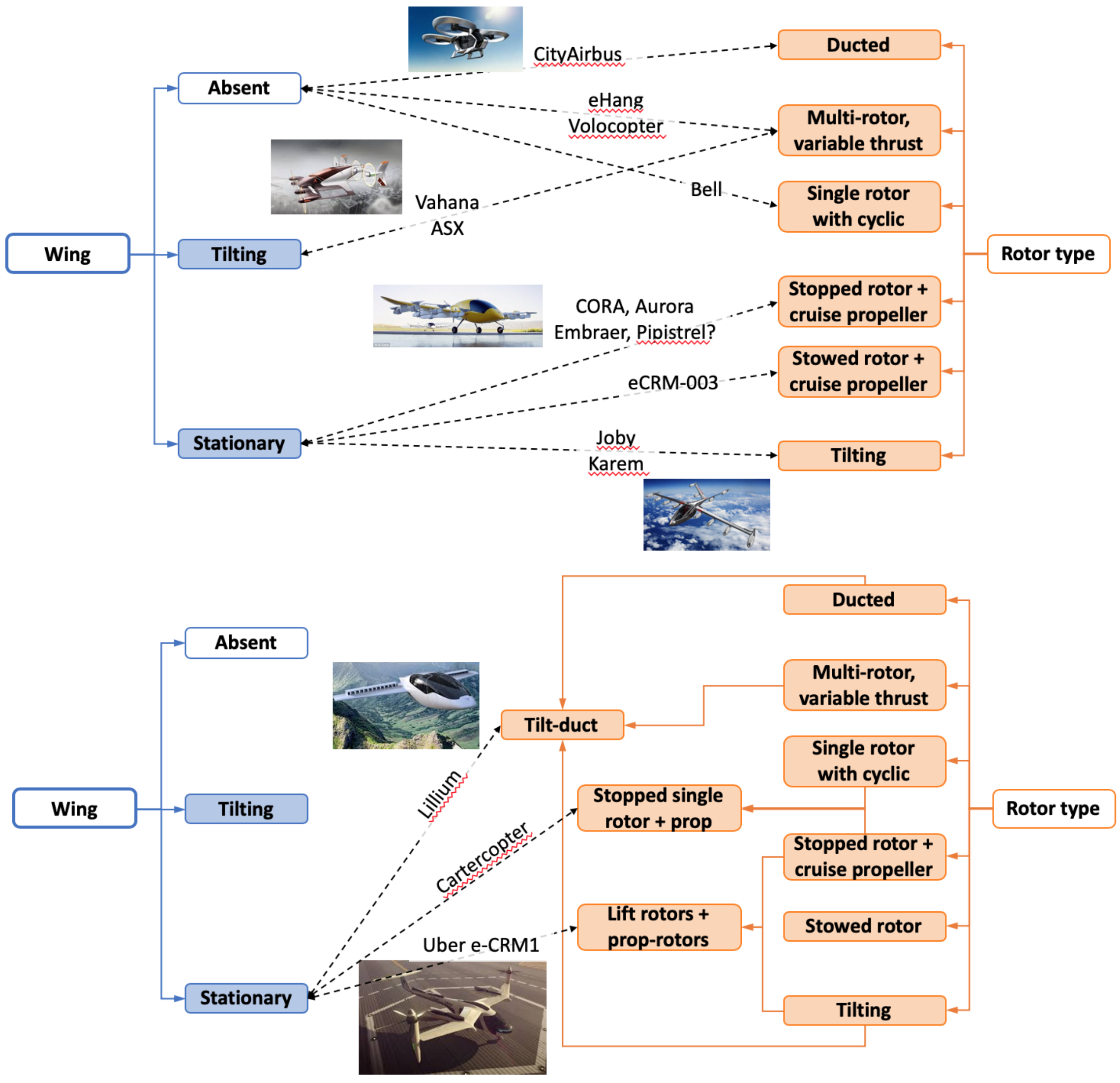


Figure 4: Assumbling eVTOLs from rotor and wing technologies

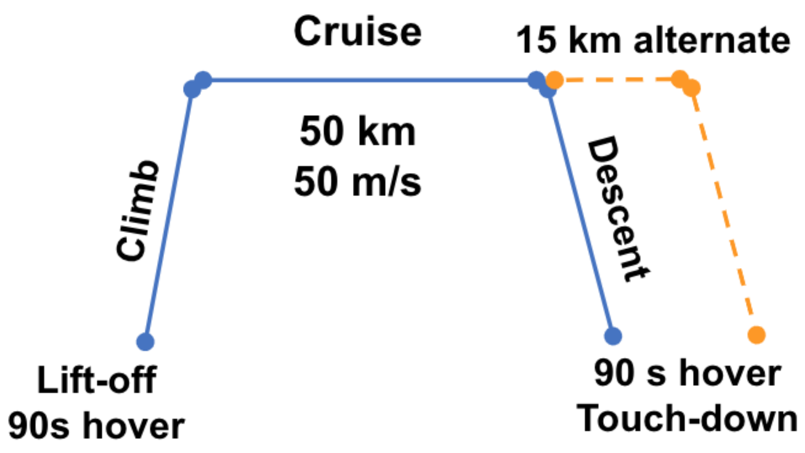


Figure 5: An example mission profile

The **mission profile** is specified as a sequence of **hover** and **cruise** segments. Cruise segments are defined through a specified airspeed and distance/time, while hover segments are defined through duration. Climb and descent segments are defined as cruise segments with different start and end altitudes. An example mission profile is shown in Fig. 5.

Finally, the **model calibration data** refers to calibration constants for parametric models of rotor performance, turboshaft engine performance, electric motors, fuselage drag vs. weight curve fits, and several other assumptions. Typically, up to 30 constants are required to calibrate different low-fidelity models accurately, defined in the **defaults.yaml** input file. In `HYDRA`, several parametric models have been systematically replaced with higher-fidelity counterparts and so lift the specific assumptions made to arrive at those constants.

Accelerated convergence

The implementation of iterative sizing shown in Fig. 1 requires several iterations (up

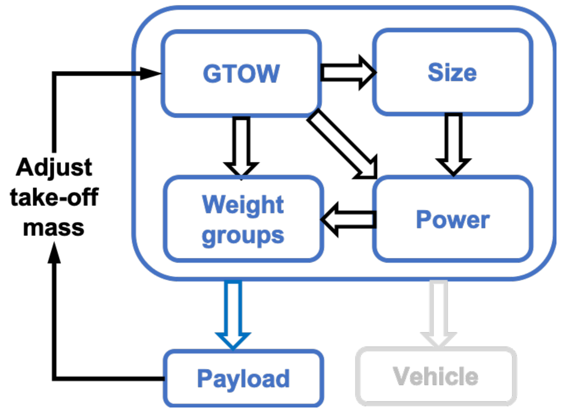


Figure 6: Accelerated convergence for iterative sizing

to 30) to converge take-off mass to the required tolerance. An alternate approach is implemented in HYDRA, resulting in significantly faster convergence for no loss in accuracy. This accelerated convergence technique is shown in Fig. 6.

In this method, the "inner loop" is executed in fixed take-off mass mode, and the constituent weight groups are evaluated. Finally, the remaining mass (take-off mass minus empty weight, minus energy storage) is assigned to payload mass, i.e.,

$$\mathrm{m_{pay}} = \mathrm{M} - \mathrm{m_{empty}} - \mathrm{m_{battery}} - \mathrm{m_{fuel}} \tag{1}$$

Here, M represents the take-off mass. Usually, mission profiles are specified together with the target payload $\mathrm{m_{target}}$. However, the "inner loop" described above runs once to calculate the payload mass $\mathrm{m_{pay}}$ for a given take-off mass M. Therefore, the take-off mass needs to be continuously adjusted to ensure that the "calculated" payload matches the target payload. One update scheme for the take-off mass (used in HYDRA) is

$$M_{i+1} \quad = \quad M_i - \frac{dM}{d\mathrm{m}_{\mathrm{pay}}}(\mathrm{m}_{\mathrm{pay}} - \mathrm{m}_{\mathrm{target}}) \tag{2}$$

The slope $\frac{dM}{d\mathrm{m}_{\mathrm{pay}}}$ is initially set to 3.0, and subsequently updated based on a finite-difference approximation as iterations proceed. This method usually converges in 3 – 5 updates for take-off mass (vs. 30 iterations for the baseline technique), resulting in 5 – 6 times faster convergence for no reduction in accuracy.

1.1.2 Constrained optimziation

The other technique that lends itself to VTOL conceptual sizing is constrained optimization. The primary advantage of this technique is that it allows for simultaneous minimization of a cost function (e.g. vehicle operating cost) while satisfying compatibility constraints (e.g., payload matching and landing footprint). If only sizing is required, the design variables are fed as prescribed inputs and compatibility constraints are specified as nonlinear algebraic constraints. The take-off weight is provided as a design variable and the payload is calculated as an output. The condition that the calculated payload ($\mathrm{m}_{\mathrm{pay}}$) be at least the target mission payload ($\mathrm{m}_{\mathrm{target}}$) is enforced as a nonlinear inequality constraint.

In HYDRA, two different optimizers are used for constrained minimization of the vehicle hourly operating cost: gradient-based optimization, and differential evolution. The constraints are landing footprint (based on wing span and fuselage length) and payload matching. For **concentual sizing**, gradient-based optimization requires an accurate initial condition to ensure that global minima (and not local minima) are identified at the end of optimization. In HYDRA, the gradient-based optimizer is initialized from several starting conditions, each of which are obtained from a parametric sweep over the available high-level design variables. For **differential evolution**, compatibility constraints are implemented indirectly as a "death penalty" - an analog of exterior penalty functions. The cost function is increased by a large number (1e6) for designs that violate any of the user-specified constraints. Additionally, `HYDRA` is programmed to automatically add and remove design variables based on the number of rotor and wing groups in the vehicle.

The optimizer in `HYDRA` is invoked after basic sizing over a limited parametric sweep is carried out. Both optimization techniques - gradient based and differential evolution - are used to minimize vehicle operating costs, and the resulting optimized designs are checked for uniqueness. These unique designs (obtained from optimization with different starting conditions) are ranked by operating cost, weight, battery/fuel or rated power based on a user-selected criterion. Further details are provided in the appropriate section later on in the documentation.

2 Drag models

2.1 Fuselage and empennage drag

The fuselage is modeled as a slender body of revolution with its length set by user inputs, and an equivalent diameter based on body width and height (also set by the user). The procedure to calculate skin friction drag and base drag are identical to those documented in NASA CR-3083: *Rotary-Wing Aerodynamics: Volume II* by C.N. Keys. Similiarly, the approach for estimating the drag of horizontal and vertical tail surfaces, as documented in NASA CR-3083, is implemented in `HYDRA`.

2.2 Edgewise rotor hub drag

The flat-plate area of each edgewise rotor hub is estimated based on a modified procedure outlined in NASA CR-3083, as follows:

$$f_{\text{hub}} = 1.2R^2(0.0006 + 0.00222N_{\text{b}}) \tag{3}$$

2.3 Spinner drag

The spinner is assumed to be a rotating body with a hemispherical nose and blunt base. As shown in Fig. 7, the drag coefficient of such a shape is 0.2 (referenced to cross-sectional area $\pi d^2/4$), for a length-to-diameter ratio of 2.5. For the calculations presented in this paper, a drag coefficient of 0.36 is assumed, to account for additional blockage from the airframe, vertical fins and interference effects. The radius of the spinner is 15% of the rotor radius.

2.4 Momentum drag for engine air intakes

For turboshaft engines operating at cruise speeds above 160 knots, the additional momentum drag is 0.3 ft^2 per intake.

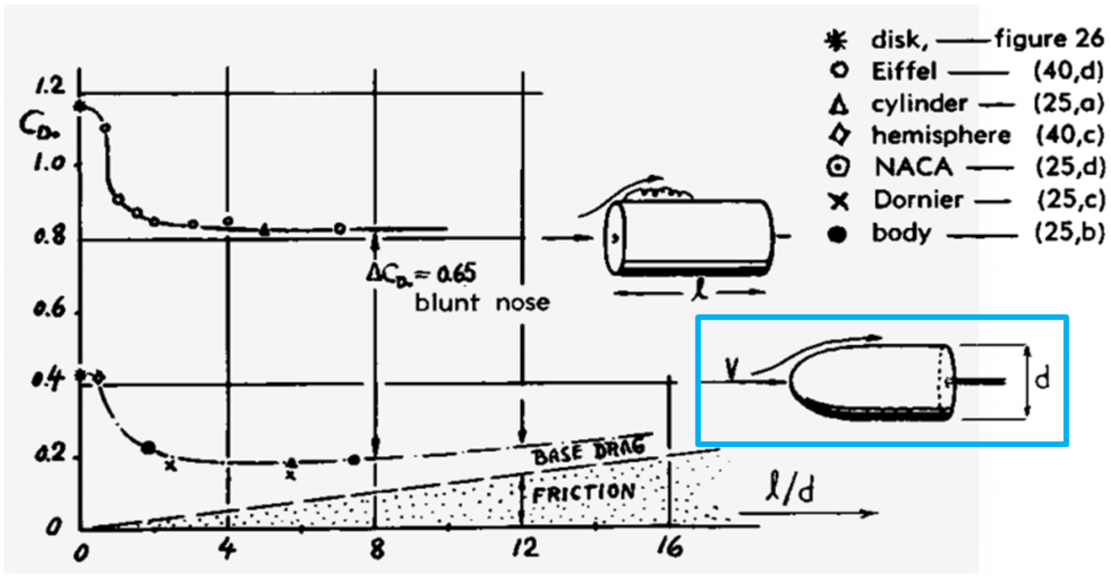


Figure 7: Spinner nose cone:drag coefficient, from *Fluid Dynamic Drag, Hoerner*

2.5 Landing gear drag

The drag of landing gear is modeled with a parametric representation (obtained from NASA CR-3083) as follows:

$$\log_e f_{\text{LG}} = y_0 + m \log_e(0.001W) \tag{4}$$

Here, f_{LG} is in square feet, and W is in lbs. The constants y_0 and m are set based on the type of landing gear.

1. **Retracted gear**: $f_{\text{LG}} = 0$
2. **Wheeled gear**: $y_0 = 0.5755$. The slope $m = 0.443$ for W $\geq$ 1000 lb, and zero for W $\leq$ 1000 lb.
3. **Skid gear**: y_0 =-0.11626. The slope m = 0.4979 for W $\geq$ 1000 lb, and zero for W $\leq$ 1000 lb.

2.6 Cooling drag

The cooling systems required to regulate engine temperatures creates additional drag, which is estimated using a parametric model given in NASA CR-3083.

$$f_{\text{cool}}(\text{ft}^2) \quad = \quad 10^{-4} P_{\text{ins}}(hp) \tag{5}$$

2.7 Mast fairing drag for coaxial rotors

The fairing for the exposed length of the rotor shaft between the upper and lower rotors of a coaxial system incurs an additional drag penalty, given by

$$f_{\text{mast}}(\text{ft}^2) \quad = \quad 0.0322 R^2 \tag{6}$$

2.8 Protrusion drag

The drag of protrusions is modeled as an additional 10% of the accumulated parasitic drag of all wetted surfaces.

3 Rotor and Wing Performance Models

3.1 Rotor performance model

Two performance models are used to calculate rotor power required in hover and axial flight. The first model is based on momentum theory, and the second model is based on blade element momentum theory.

3.1.1 Momentum Theory

For momentum theory-based predictions, rotor power is calculated based on assumed aerodynamic efficiency factors in hover and cruise. Imperial units (foot, pounds of force and horsepower) are assumed for these expressions. In hover, rotor power is given by

$$P_{\text{hover}}(hp) \quad = \quad \frac{T_R^{1.5}}{550\sqrt{2\rho A}\ \text{FM}} \tag{7}$$

Here, T is the thrust required in hover from each rotor, and is calculated assuming the segment weight is equally divided between all rotors. A vertical download factor (configuration-dependent) is included in the estimation of rotor thrust. The rotor disk area is A (ft^2), ρ is the ambient air density (slug/ft^3) and FM is the rotor figure of merit in hover.

In cruise, the power required by each rotor when operating in axial flight is

$$P_{\text{cruise}}(hp) \quad = \quad \frac{D}{N_{\text{R}}} \frac{V_\infty}{550\eta_{\text{p}}} \tag{8}$$

Here, D is the total vehicle drag in lbs when operating at a forward flight speed of V_∞ ft/s, which is overcome by each of the N_{R} rotors operating at a propeller efficiency of η_{p}.

3.1.2 Blade Element Momentum Theory (BEMT)

Certain configurations like the tilt-rotor or tilt-wing fly such that the rotors operate predominantly in axial flight. Therefore, the Blade Element Momentum Theory (BEMT) is used as a higher-fidelity replacement for momentum theory-based predictions. Though BEMT can resolve finer details like angle of attack variations along the span, it requires detailed inputs, namely airfoil tables and distribution of blade chord and twist along the span. These details are populated as described below.

The rotor is assumed to be constructed from multiple user-specified airfoils. The lift and drag characteristics for each airfoil may be Reynolds-tabulated or mach-tabulated. Several combinations of blade chord and twist distribution are generated assuming linear taper and bilinear twist. The design variables for blade geometry are:

1. Blade taper ratio (1:1, 2:1 and 3:1)
2. Blade bilinear twist junction (30% – 70% span)
3. Twist at bilinear junction (-5 deg to -20 deg)
4. Twist at tip (-5 deg to -35 deg)

The rotor RPM in cruise (specified as a fraction of hover RPM) is added as a sizing variable. The BEMT analysis calculates the rotor power required for a given thrust, for all mission segments, exploring a parameter space of 360 designs (combinations of blade twist and taper distribution). The rotor geometry for minimum energy consumption over the mission is identified from the parametric sweep, and the aerodynamic efficiencies (figure of merit FM in hover and propeller efficiency η_{p} in cruise) are calculated for each mission segment.

Minimizing the energy required at the rotor ensures reduced fuel consumption, but not necessarily a good overall design. A cruise-dominated mission results in a prop-rotor with a potentially degraded hover figure of merit while achieving good aerodynamic cruise efficiency. Therefore, installed power (and therefore, engine and transmission weights) may increase, resulting in a lower payload fraction. To prevent unchecked increases in powerplant weights, another condition is introduced in the BEMT analysis: designs are considered “valid" if the hover figure of merit is greater than a user-specified threshold (e.g., 0.72), and only these valid designs are ranked to determine the “best” rotor efficiencies.

When BEMT is included in the sizing iterations, sizing proceeds in 3 steps:

1. Perform sizing with momentum theory and assumed efficiency factors in hover and cruise (FM=0.75, η_{p}=0.82)

2. If the design satisfies all constraints, perform BEMT analysis to calculate aerodynamic efficiencies for each segment based on thrust requirements; extract hover or cruise efficiencies (FM, η_{p}).

3. Perform sizing with momentum theory, using the calculated rotor aerodynamic efficiencies obtained in Step 2.

Figure 8 shows the comparison of thrust and power obtained from BEMT predictions with airfoil tables vs. sub-scale experiments for a variable-pitch and variable-RPM sub-scale prop-rotor. Good predictions were obtained at various cruise speeds ranging from 3 m/s to 15 m/s. For these cases, BEMT was found to be sufficient for accurate performance predictions.

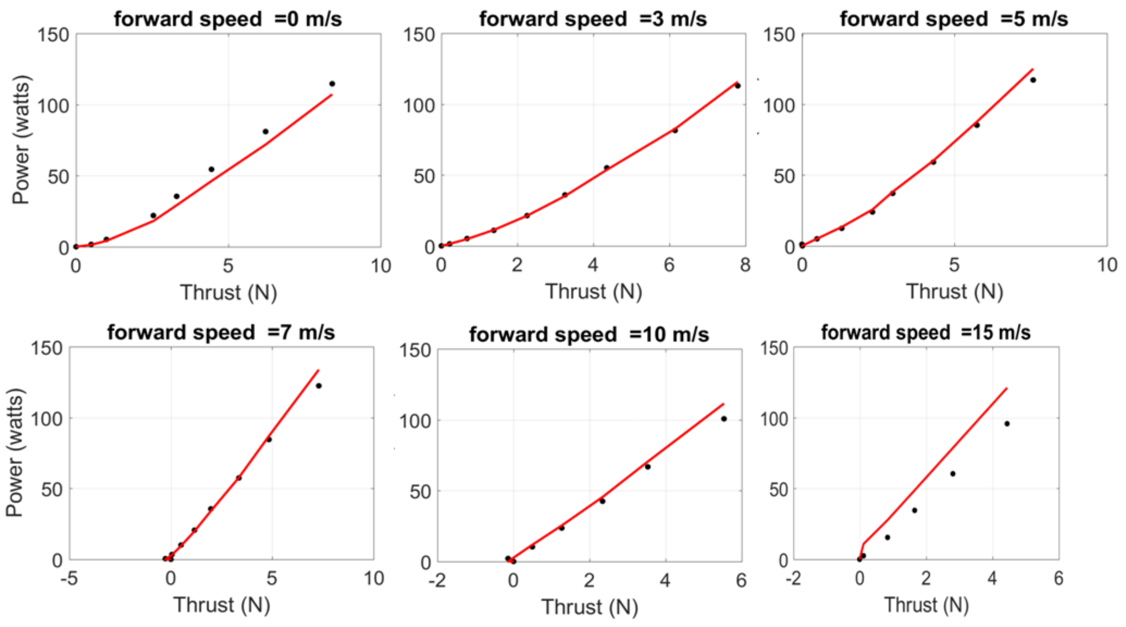


Figure 8: Comparison of BEMT predictions to experiments

3.2 Wing performance model

The drag coefficient of a fixed wing is given by

$$C_D = C_{Do} + \frac{1}{\pi ARe} C_L^2 \tag{9}$$

The lift coefficient C_L and aspect ratio AR are sizing design variables. The profile drag coefficient C_{Do} is an assumed value, usually 0.014 (including protrusions and interference). The Oswald efficiency e is calculated as

$$\begin{aligned} e &= (Q + P\pi AR)^{-1} \\ P &= 0.38 \times 0.02 \\ Q &= \frac{1.01}{s} \\ s &= 1 - 2\left(\frac{d_{\text{fus}}}{b_{\text{wing}}}\right)^2 \end{aligned}$$

The parameters are d_{fus} = equivalent fuselage diameter, and b_{wing} = wing span.

4 Weight models

There are three main contributors to vehicle take-off mass:

1. **Energy storage** - battery and/or fuel weight. This component may be traded-off against payload for mission flexibility, depending on available volume.

2. **Empty weight** - this weight group represents the airframe, wings, rotors, engines, transmission, motors, seats, avionics and other non-removable systems.

3. **Payload** - usually specified with the mission definition.

4.1 Engines, fuel and batteries

Though engines strictly belong in the empty weight category, the calculation of fuel weights are inextricably linked to the evaluation of engine performance and engine weights. Therefore, engine models are described in this section.

4.1.1 Engines

`HYDRA` has built-in models for fuel-burning engines of two types: gas turbine engines, and reciprocating engines (piston engines and aero-diesel engines).

Turboshaft engines

Based on statistical data for various production engines, a trend line was fitted with good accuracy ($R^2 = 0.92$) for the variation of engine weight $\mathrm{W}_{\text{engine}}$ (lb) with installed power $\mathrm{P_{ins}}$ (hp). The fitted engine weight model is given by

$$W_{\mathrm{engine}}\ (\mathrm{lb}) \quad = \quad 7.3874\ \mathrm{P_{ins}^{0.552}} \tag{10}$$

Another fit is shown in Fig. 9, in SI units with engines manufactured over the last fifty years.

The variation of engine Specific Fuel Consumption (SFC) at peak efficiency with installed power is

$$\mathrm{sfc_{base}}\ (\mathrm{lb/hp\text{-}hr}) \quad = \quad 1.549\ \mathrm{P_{ins}^{-0.161}} \tag{11}$$

Engines operating at sub-optimal conditions (i.e. at power settings lower than design power output) suffer from higher specific fuel consumption. This effect is modeled using a power law as

$$\mathrm{sfc} \quad = \quad \mathrm{sfc_{base}} \left(\frac{\mathrm{P_{req}}}{\mathrm{P_{ins}}}\right)^{-0.256} \tag{12}$$

Here, $\mathrm{P_{req}}$ is the power output required from the engine to produce a thrust T during a mission segment. Finally, the variation of available power (relative to installed power) with ambient temperature and pressure is given by

$$\mathrm{P_{avail}} \quad = \quad \mathrm{P_{ins}}\ [1 - K_T(\theta - 1)]\ [1 + K_D(\delta - 1)] \tag{13}$$

The terms θ and δ are ratios of ambient temperature and ambient pressure to their corresponding values at mean sea level (ISA). The constants K_T and K_D are obtained from curve fits of detailed engine performance curves.

Piston engines

The treatment of piston engine weights is similar to that followed for turboshaft engines. The variation of engine weight with installed power is given by

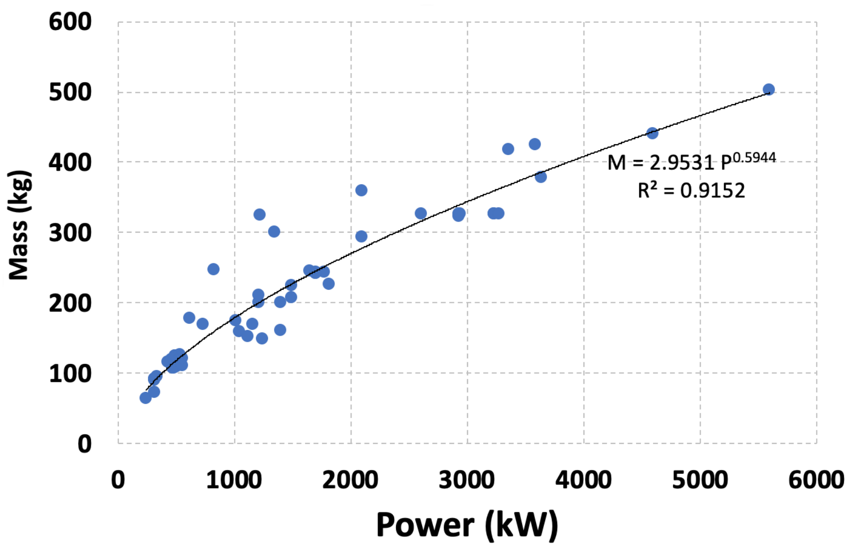


Figure 9: Statistics: turboshaft engine mass vs. installed power

$$W_{\text{engine}}\ (\text{lb}) \ = \ 2.367\ \text{P}_{\text{ins}}\ (\text{hp})^{0.916} \quad 35 \leq \ \text{P}_{\text{ins}}\ (\text{hp}) \leq 1000 \tag{14}$$

Another fit is shown in Fig. 10, in SI units with a larger sample size, including less efficient engines. The variation of engine SFC at its peak efficiency is

$$\begin{aligned} \text{sfc}_{\text{base}}(\text{lb/hp-hr}) \ &= 0.42, \quad \text{P}_{\text{ins}}\ (\text{hp}) \leq 4 \\ &= 0.594 - 0.0046\ \text{P}_{\text{ins}}\ (\text{hp}), \ \ 4 \leq \ \text{P}_{\text{ins}} \leq 56 \\ &= 0.52\ \text{P}_{\text{ins}}\ (\text{hp})^{-0.0972}, \ \ 56 \leq \ \text{P}_{\text{ins}} \leq 1000 \end{aligned}$$

4.1.2 Fuel, tank and fuel handling systems

After calculating the engine power required, the mission profile (flight segment duration) is used with engine power requirement and SFC estimates to calculate the fuel weight as

$$m_{\text{fuel}} \quad = \quad \Sigma_{i=1}^{\text{N}_{\text{SEG}}}\ sfc(i)\ P_{\text{eng}}(i)\ +\ m_{\text{unusable}} \tag{15}$$

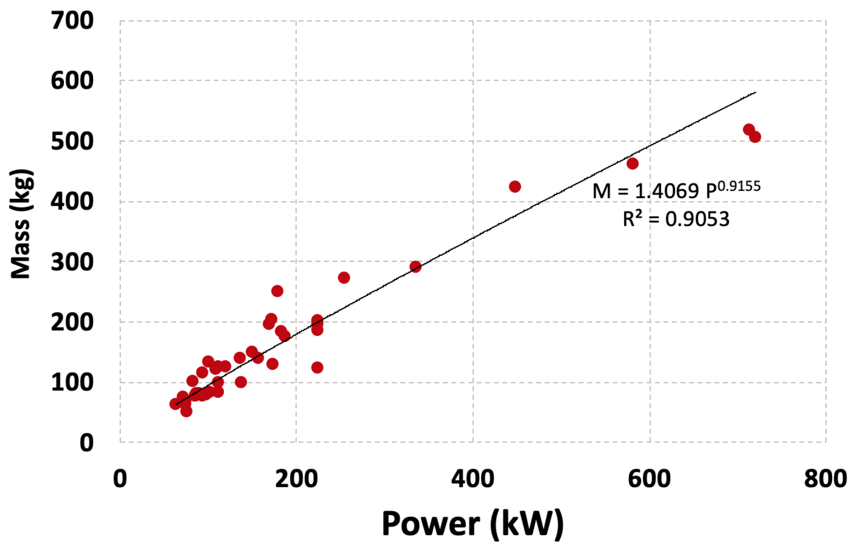


Figure 10: Statistics: reciprocating engine mass vs. installed power

An average density of 6.7 lb/gal is used for both turbine engine fuel and aviation gasoline. The fuel tank and plumbing mass is estimated from fuel volume using AFDD models as

$$W_{\text{tank}} \quad = \quad 0.4341\ V_{\text{tank}}^{0.7717}\ N_{\text{tank}}^{0.5897}\ f_{\text{cw}}\ f_{\text{bt}}^{1.9491} \tag{16}$$

The terms used in the equations are

V_{tank} is the tank volume in U.S. gallons (1 gal = 3.7.8 liters)

N_{tank} is the number of internal tanks

f_{cw} is the crashworthiness mark-up factor, usually 1.31

f_{bt} is the ballistic survivability mark-up factor (1.2 for military, 1.0 for civilian)

The weight of plumbing systems used for regulating fuel flow from tanks to engines is modeled as follows:

$$k0_{\text{plumb}} \quad = \quad \max(0.022M, 120)$$

$$k1_{\text{plumb}} = 0.025k0$$

$$W_{\text{plumb}} = k0_{\text{plumb}} + k1_{\text{plumb}}(0.01 * N_{\text{tank}} + 0.06 * N_{\text{engine}})\ \dot{V}^{0.866}$$

$\dot{V}$ is the peak fuel flow rate to the powerplant in lb/hr and the plumbing system weight is in pounds. M is the take-off mass in kg - the mixing of units and conventions is a result of AFDD models being specified in FPS units, while `HYDRA` is implemented in SI units.

For small-scale powerplants, the trendlines predict an over-estimate of fuel handling system weight. To handle these special cases, a maximum limiter is placed on the fuel system weights, with the limit set to 50% of fuel weight.

4.1.3 Battery models

Battery weights are accurately represented using a very simple statistical model. Figure 11 shows the variation of battery weight with rated energy capacity for commercially available small-scale batteries, where each data point represents a manufactured unit. Arguably, a battery density of 158 W-hr/kg is quite low and improvements can be made; this "conservative" design is typical of commercially available battery packs including the safety casing.

For full-scale platforms used in commercial operations, several conservative assumptions are used to incorporate various margins in sizing the battery.

1. The minimum depth of discharge is 7.5% of total energy capacity
2. The maximum state of charge near end of life for the battery is 80%, i.e., after cycling, the usable battery capacity is at most 80% of the rated energy. Thus,

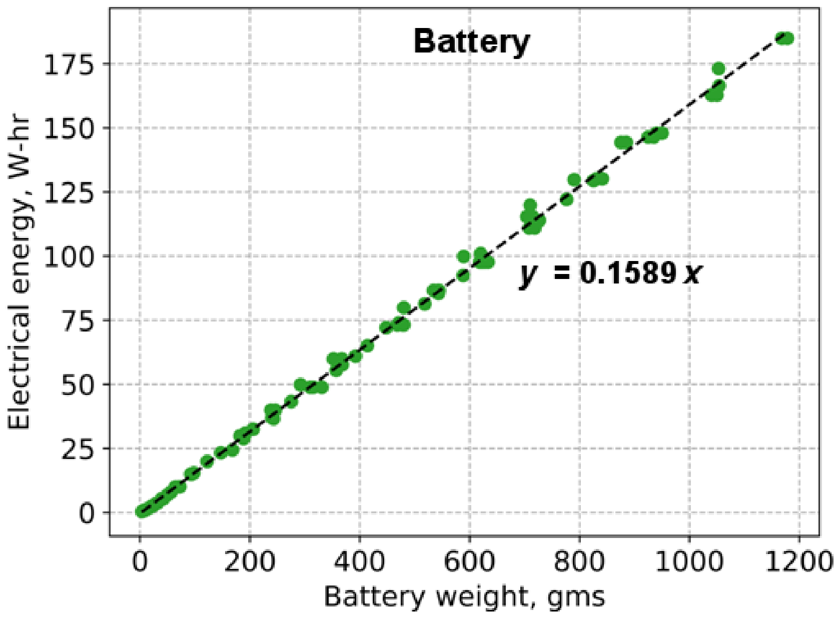


Figure 11: Small-scale batteries: weight statistics

the effective usable energy for battery sizing is **72.5% of the rated capacity**.

3. The average discharge efficiency of the battery reduces by 4% for every unit increase in average C-rating.

4. A specific energy of 240 Wh/kg is used to estimate the mass of the cells.

5. A pack integration factor of 0.75 (defined as battery mass to cell mass) is used to estimate the weight of battery management systems and casing.

6. A curve fit of temperature rise vs. C-rating (obtained from experimental data) is used to track the rise of cell temperature over the mission duration. A maximum limit of of 70^{o}C is used to iteratively add cells until thermal limits are satisfied.

4.1.4 Electric motor weights

Though electric motor weights are strictly driven by peak torque requirements, the data correlation between weight and peak rated power is also very good as shown in Fig. 12. For small-scale motors, the speed controller is sized based on the current drawn by the system assuming a 12 Volt source. The masses of several large-scale motors with controllers are plotted against rated power in Fig. 13. Motor + speed controller efficiencies range from 85% to 92% depending on the design point and operating RPM range. Motor and speed controller weights are estimated as

$$\begin{aligned} W_{\text{motor}}(lb) &= 0.74 \text{ P} && \text{P} \leq 13.4 \text{ hp} \\ W_{\text{ESC}}(lb) &= 1.047 \text{ P} && \text{P} \leq 13.4 \text{ hp} \\ W_{\text{motor}} + W_{\text{ESC}}(lb) &= 1.489 \text{ P}^{0.783} && 13.4 \text{ hp} < \text{P} \leq 350 \text{ hp} \end{aligned}$$

4.2 Electric Transmissions

Generators for electric transmissions are sized in a manner identical to electric drive motors. The components of an electric transmission are shown in Fig. 14.

The advantage of this hybrid system is that a mechanical transmission is not required for power transmission to the rotors; further the hybrid system can serve as a long-range drop-in upgrade for short-range battery-powered variants. The disadvantage of a turboshaft-electric hybrid system is that each additional component introduces additional points of failure. Adding redundancy and fail-safe architectures can incur a significant empty weight penalty.

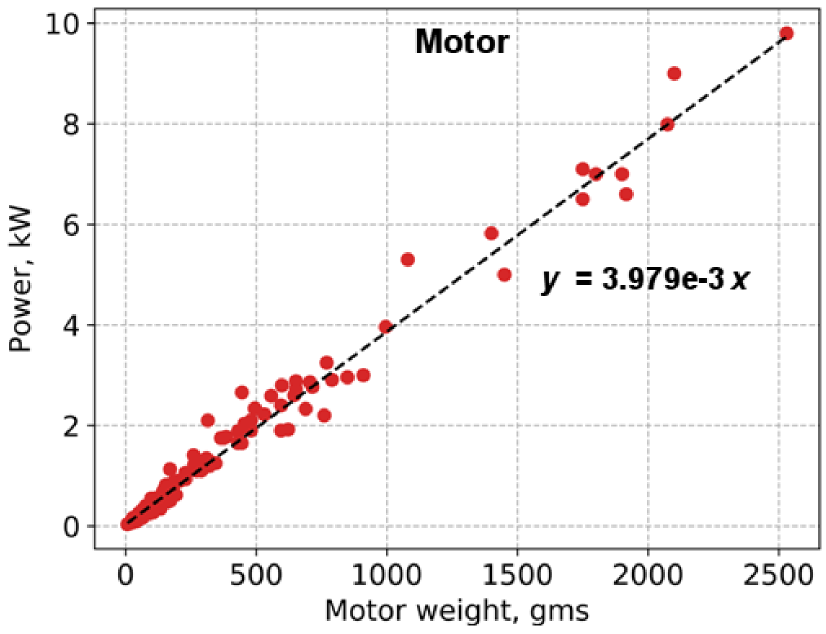


Figure 12: Statistics: weight of small BLDC motors

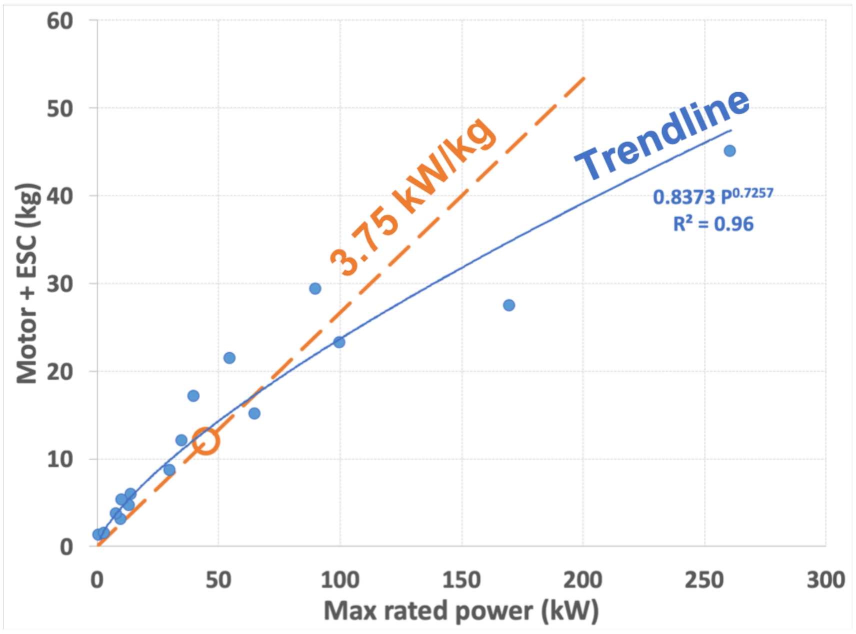


Figure 13: Statistics: large-scale DC motor + controller mass vs. installed power.

Manufacturing technology in 2018 can achieve 3.75 kW/kg of power density.

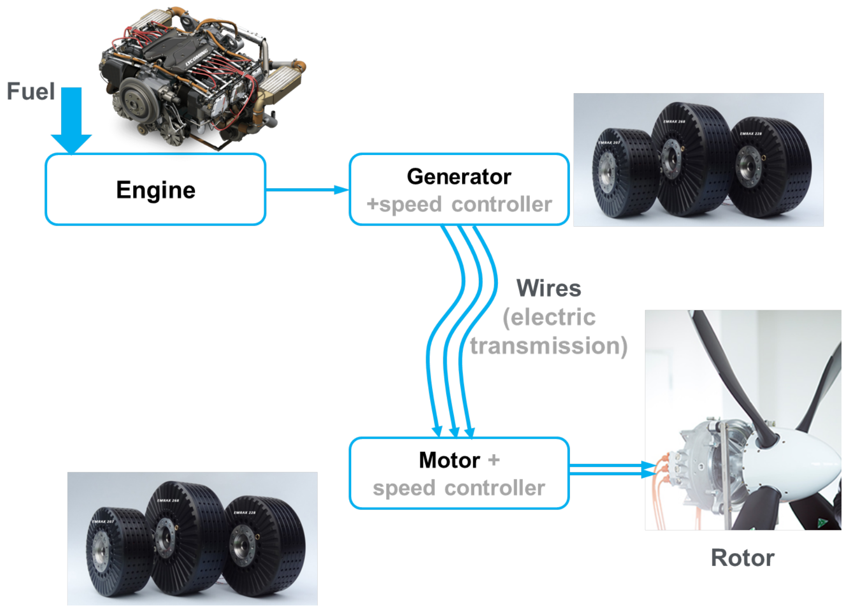


Figure 14: Turbine-generator hybrid powerplant for electric motor. 98% wire transmission efficiency, 92% generator efficiency

4.3 Wire weights

Two types of wires are considered in the sizing analysis: power cables and signal wires. The weights of electrical power cables are estimated from the vehicle rotor layout using a wire mass per unit length per unit power of 0.0057 kg/kW/m. Signal wires are calculated using a linear mass density of 0.17 kg/m and a knowledge of the rotor layout, together with an estimate of channel counts. Based on user inputs, doubly/triply redundant power cables and signal wires are automatically added to the appropriate weight groups for estimating vehicle empty mass.

At full-scale, a key driver of vehicle weight is the rotor design parameter set (tip speed, solidity, disk loading/radius, first rotating flap frequency). One such em-

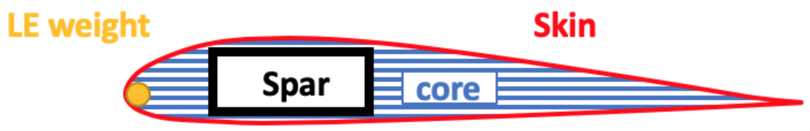


Figure 15: Blade cross-section

pirical parameter is the flap natural frequency. In the legacy approach for estimating rotor blade weight, the designer had to prescribe this parameter to perform sizing. However, underlying data for the flap frequency is restricted to articulated and hingeless rotors, and plentiful data is not available for stiffer rotors ($\nu_\beta \geq 1.08$/rev). For small-scale VTOL with very stiff rotor systems ($\nu_\beta \geq 1.4$/rev), extrapolation of the trend line beyond the range of available data for ν_β may yield erroneous estimates for rotor blade weight. For eVTOLs, a physics-based approach is implemented and used in `HYDRA`.

4.4 Physics based model for rotor blades

A schematic of the rotor airfoil is shown in Fig. 15. The cross-section consists of two categories of materials: non-structural masses, and load-bearing components. The non-structural masses are:

1. Paint: 0.15 mm thickness, density 1800 kg/cu.m, applied on surface
2. Glue: 0.25 mm thickness, density 1800 kg/cu.m, applied on skin, shear webs
3. Core: density of 52 kg/cu.m, encompasses cross-section area of 0.49 $\left(\frac{t}{c}\right) c$
4. Leading edge protection: 1.14mm thickness, density 8900 kg/cu.m, from lead-

ing edge to 25% chord on upper and lower surfaces.

Here, $\left(\frac{t}{c}\right)$ is the airfoil thickness-to-chord ratio, usually 0.1 to 0.12 and c is the blade airfoil chord. For the paint and glue, the chordwise location of the component center of gravity is $0.5c$. For the leading edge protection strip and core, the component chordwise center of gravity locations are 12.5% chord and 57% chord, respectively.

4.4.1 Skin sizing

The skin is sized to withstand a total torsion moment equivalent to an airfoil c_m=0.2 at the maximum rated rotor speed. The total root pitching moment in hover is

$$M_x = \frac{1}{6}\rho V_{\text{TIP}}^2 c^2 c_m R$$

The skin thickness is given by

$$t_{\text{skin}} = \frac{nM_x}{2A_{\text{cs}}\tau_{\text{skin}}}$$

Here, A_{cs} is the cross-section area of the airfoil, and τ_{skin} is the maximum allowable shear stress in the skin. The factor n accounts for additional overload and safety factors to account for for contingency conditions and fatigue margins.

4.4.2 Leading edge weight

The section chordwise center of gravity for the non-structural components (paint, leading edge protection and core) along with the skin is obtained from the respective weights and component centroid locations. If this center of gravity is behind the blade quarter-chord, a leading edge weight is added to avoid torsional divergence and flutter for the blade.

The CG of the non-structural components and skin is given by

$$\frac{x_1}{c} \quad = \quad \frac{m_{\text{LEP}}0.125 \;+\; m_{\text{skin}}0.5 \;+\; m_{\text{core}}0.57 \;+\; m_{\text{paint}}0.5 \;+\; m_{\text{glue}}0.5}{m_{\text{LEP}} \;+\; m_{\text{skin}} \;+\; m_{\text{core}} \;+\; m_{\text{paint}} \;+\; m_{\text{glue}}}$$

If $\frac{x_1}{c}$ is greater than 0.25, then a leading edge mass is added, equal to

$$m_{\text{LE}} \quad = \quad (m_{\text{LEP}} \;+\; m_{\text{skin}} \;+\; m_{\text{core}} \;+\; m_{\text{paint}} \;+\; m_{\text{glue}}) \left(4\frac{x_1}{c} \;-\; 1\right)$$

4.4.3 Spar caps and shear web sizing

The spar is placed so that the neutral axis and CG both lie at quarter-chord. A NACA-0012 section is used to estimate The elements that support shear and bending moment due to lift and drag and bending loads are the spar webs (shear and lag bending moment) and spar caps (flap bending moment and tension due to the centrifugal load).

The limit load for eVTOL rotor blades is determined from the trim thrust required under OMI (One Motor Inoperative) conditions. For the “conventional” tilt-wing configuration (6 rotors on main wing, 2 rotors on horizontal tail) and the tandem tilt-wing configuration (4 rotors each on canard and wing), the maximum thrust scaling factor is approximately 150% the hover thrust. With rotor radius R, rotor speed Ω and peak torque at the blade root Q_b determined from the rotor geometry, the vertical force on the blade is obtained by dividing the maximum expected thrust level by the total number of blades as

$$F_z \quad = \frac{W}{N_R N_b} n_z \tag{17}$$

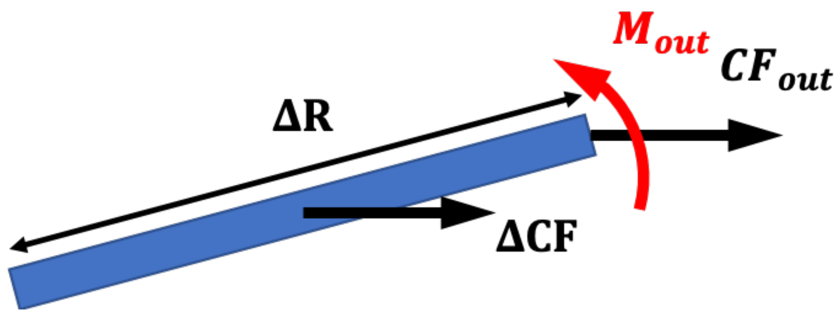


Figure 16: Blade spanwise segment

The terms n_z is a load factor that can incorporate safety margins, dynamic overshoot/landing and fatigue factors for blade loads. The lift distribution along the span is assumed to be quadratic (a structurally conservative loading profile - the optimum hovering rotor features a linear spanwise lift distribution, discounting tip loss effects). The blade is assumed to constructed with a pre-cone angle β_p to alleviate root bending moments, and then subdivided into $N = 5$ equal span segments. One such segment is shown in Fig. 16.

The dominant loads on the outboard end of the segment, that are carried by the spar caps, are the centrifugal force (from all segments outboard of the present one) and a flap bending moment due to (i) distributed lift on segments outboard of the present one, and (ii) a combination of pre-cone and centrifugal load on segments outboard of the present one. The blade spanwise segments are successively sized starting from the tip segment and ending in the root segment. A schematic of the rotor blade segment and cross-section is shown in Fig. 17.

The total centrifugal load and flap bending moment at the inboard end of the

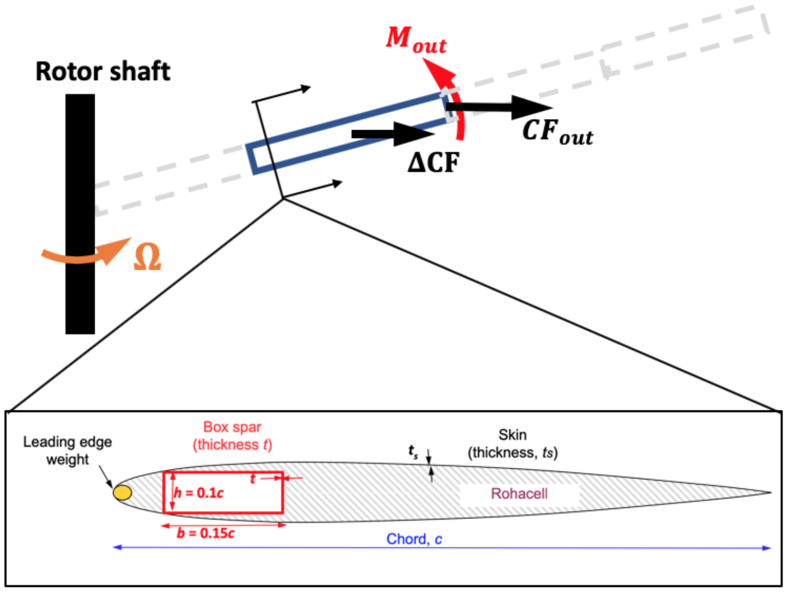


Figure 17: Rotor blade segment sizing: schematic

segment is

$$CF(x_{\text{in}}) \quad = \quad CF(x_{\text{out}}) \ + \ \Delta CF \tag{18}$$

$$M(x_{\text{in}}) \quad = \quad M(x_{\text{out}}) \ + \ \Delta M \tag{19}$$

The flap bending moment distribution due to lift, along the non-dimensional span coordinate x (0 at root, 1 at tip) is

$$M_{\text{lift}}(x) \quad = \quad \frac{1}{4}F_z R(x^4 - 4x + 3) \tag{20}$$

The additional centrifugal load and flap bending moment due to loads on the present segment are

$$\Delta CF \quad = \quad \frac{1}{2}(m_{\text{s}} \ + \ m_{\text{ns}})V_{\text{TIP}}^2 \ (x)\text{out}^2 \ - \ x_{\text{in}}^2)$$
$$\Delta M \quad = \quad -(CF_{\text{out}} + \frac{1}{2}\Delta CF)\Delta R\beta_p \ + \ M_{\text{lift}}(x_{\text{in}}) \ - \ M_{\text{lift}}(x_{\text{out}})$$

The spar mass per unit span is m_{s}) and the non-structural mass per unit span is

m_{ns}. The maximum axial stress in the spar caps at the inboard end of the present segment is

$$\sigma_{\text{xx}}(x_{\text{in}}) = \frac{CF}{A_{spar}} + \frac{M(x_{\text{in}})\left(\frac{h}{2}\right)}{2bt\left(\frac{h}{2}\right)^2}$$

The term A_{spar} is the area of the spar, given by $2(b+h)t$. The individual expressions can be rearranged to solve for the spar cap thickness t, as

$$\begin{aligned} t &= \frac{RHS}{LHS} \\ LHS &= \frac{1}{n}\sigma_{\text{max}} + \frac{1}{2}V_{\text{TIP}}^2(x_{\text{out}}^2 - x_{\text{in}}^2)\rho_{\text{spar}}\left[\Delta R\beta_p\frac{b+h}{bh} - 1\right] \\ RHS &= \frac{1}{2(b+h)}\left[CF(x_{\text{out}}) + \frac{1}{2}m_{\text{ns}}V_{\text{TIP}}^2(x_{\text{out}}^2 - x_{\text{in}}^2)\right] + M_{\text{lift}}(x_{\text{in}}) \\ &+ \frac{1}{bh}\left[M(x_{\text{out}}) - \Delta R\beta_p\left(CF(x_{\text{out}}) + \frac{1}{2}\Delta CF\right)\right] \end{aligned}$$

This formulation avoids iterative convergence inner loops for the rotor blade structure during sizing, improving computational efficiency.

After sizing the spar, the shear webs thickness is determined from the vertical bending load at the inboard end, given by

$$t_{\text{web}} = \max\left(\frac{1.5nV_z}{h\sigma_{\text{skin}}}, t_{\text{web}}\right)$$

Here, V_z is the vertical shear due to blade lift integrated from the inboard end of the segment (x=x_{in}) to the blade tip (x=1). After sizing one spanwise segment, the total axial load and flap bending moment at the inboard end of the segment are calculated using Eqns. 18,19 respectively. The loads at the inboard end of the present segment are then assigned as tip loads for the next segment, and the process is repeated for all segments in succession.

4.4.4 Root fitting sizing

The root fitting for the blade is assumed to extend from the shaft (x=0) to 10% of the blade span (x=0.1), and is made of 7075 Aluminum. The density is 2800 kg/cu.m, and a limiting axial stress of 20 MPa is used based on a fatigue life of 10^8 cycles. The construction of the root fitting is shown in Fig. 18.

The tube radius is assumed to be equal to 4× the airfoil thickness, and the thickness t_{Root} is evaluated to ensure that the maximum tensile stress in the material does not exceed the limiting stress (20 MPa). The tube thickness is solved analytically, using a formulation similar to the approach used for spar sizing. The expression for tube thickness is

$$\begin{aligned} t_{\text{Root}} &= \frac{RHS}{LHS} \\ RHS &= \frac{CF(x=0.1)}{2\pi r_{\text{Root}}\sigma_{\text{limit}}} + \frac{M_{\text{lift}}(x=0.1) - M_{\text{CF}}(x=0.1)}{3\pi r_{\text{Root}}^2\sigma_{\text{limit}}} \\ LHS &= 1 - \frac{\rho}{2\sigma_{\text{limit}}}\Omega^2(0.1R) \end{aligned}$$

The root fitting thickness is set to the larger of minimum gauge t_{min} or the calculated thickness above, whichever is larger.

4.4.5 Torsion frequency check

The skin thickness is used to calculate the first elastic torsion frequency for the rotating blade and ensure that it is above 3.3/rev at the hover RPM. If the torsion frequency is too low, then additional skin thickness is added until this requirement is satisfied, preserving the same spar design.

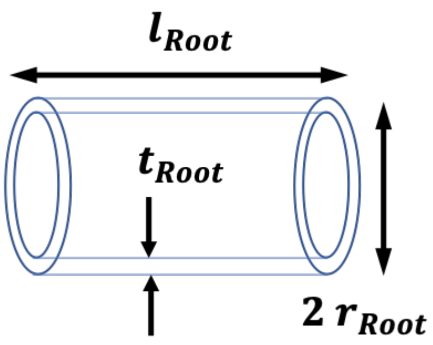


Figure 18: Blade root fitting

4.5 Flight controls

Two categories of actuators are modeled: fixed-wing, and rotary-wing.

4.5.1 Rotor controls

Unlike large rotorcraft, some eVTOLs feature low-bandwidth collective actuators and no cyclic or swashplate. In such designs, the collective pitch (e.g., for prop-rotors) is usually scheduled with airspeed, and RPM is used for feedback control. Sizing laws for these collective actuators are therefore markedly different from those used for full-scale helicopter controls. The rotor hub mass and collective pitch actuator masses scale with tip speed V_{TIP} (and blade chord c for the actuator) as

$$M_{\text{hub}}\ (\text{kg}) = 4.84 \left(\frac{V_{\text{TIP}}}{171}\right)^2 \left(\frac{c}{0.82}\right) \tag{21}$$

$$M_{\text{act}}\ (\text{kg}) = 1.47 \left(\frac{V_{\text{TIP}}}{171}\right)^2 \left(\frac{c}{0.82}\right) \tag{22}$$

The tip speed V_{TIP} is in m/s and mean blade chord c is in meters.

4.5.2 Fixed wing controls

Fixed wing flap and actuator weights are obtained from the AFDD model as

$$W_{\text{flaps}}\ (lb) \quad = \quad 0.01735 W^{0.644} S_{\text{wing}}^{0.41} \tag{23}$$

For this model, the weights are in lbs, including the take-off weight W. The term S_{wing} is the total plan-form area of all fixed wings, and the actuator weights W_{flaps} includes all actuators on all fixed wings.

4.5.3 Tilt actuators

For tilt-rotors and tilt-wing configurations, the masses of tilt actuators are estimated as the larger of 15.5% of wing mass, and 3.61% of the total mass being tilted by that actuator. For tilt-wings, both metrics are applicable; for tilt-rotors, the tilt actuator weight is set to 10% of the masses being tilted.

$$M_{\text{tilt}-\text{wing}} \quad = \quad \max(0.155 M_{\text{wing}}, 0.0361 M_{\text{tilt}}) \tag{24}$$

$$M_{\text{tilt}-\text{rotor}} \quad = \quad 0.10(M_{\text{hub}} + M_{\text{act}} + M_{\text{blades}} + M_{\text{motor}}) \tag{25}$$

4.6 Motor mount mass

The motor mount structure refers to the overhang beam connecting the motor and rotor hub, to a wing (or fuselage). For the Vahana tilt-wing configuration, the motor mount structure is the overhang beam used to offset the rotor hub ahead of the wing leading edge, as shown in Fig. 19. The motor mount is modeled as a cantilever beam of length **L** with a tip mass equal to the mass of the rotor hub, blades and

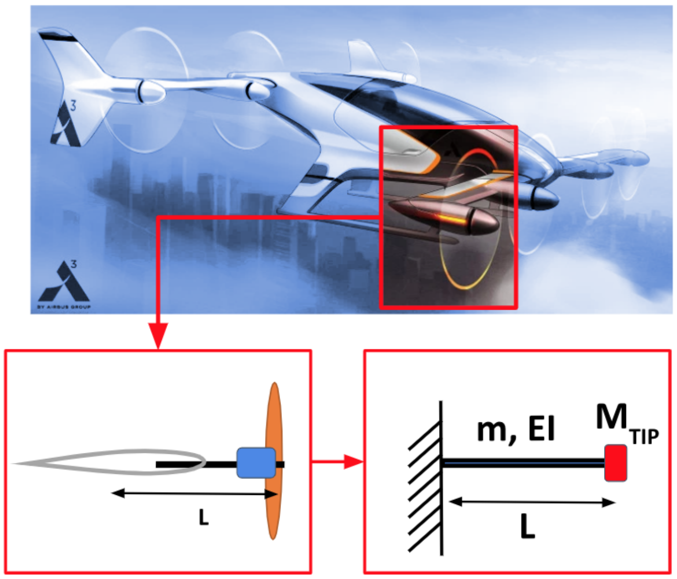


Figure 19: Motor mount model

motor. This structure is sized by a natural frequency requirement, i.e., **the first cantilever bending mode for this beam with a tip mass** must be sufficiently large so that the structural dynamics do not interact with flight dynamics (0 - 4 Hz). For eVTOLs, a target first bending frequency of 8.5 – 10 Hz is used to size the motor mount beam. The overhang length is set to the sum of wing three-quarter chord and one rotor radius.

For configurations such as CityAirbus (quad-ducted-coaxial rotors) with no fixed wings, the rotors are mounted directly on the fuselage using support arms. The support arms are treated as “motor mounts”, and the beam length is set to 120% rotor radius. Appendix A provides an analytical solution for sizing a cantilever beam with several lumped non-structural masses, distributed structural and non-structural masses based on a target frequency.

4.7 Wing weight model

Three models are available to estimate the weight of a fixed wing:

4.7.1 AFDD wing weight model

The weight of a fixed wing is estimated as

$$W_{\text{wing}} \quad = \quad 5.6641 \frac{\left(\frac{L_w}{1000}\right)^{0.847} n_z^{0.4} S_{\text{wing}}^{0.21} AR^{0.5}}{\left(\frac{t}{c}\right)^{0.0936}}$$

The wing weight is in lbs, L_w is the wing design lift in lbs, n_z is the design ultimate load factor, S_{wing} is the wing plan-form area in sq. ft, AR is the wing aspect ratio and $\left(\frac{t}{c}\right)$ is the wing thickness to chord ratio.

4.7.2 Frequency tuning: bending loads

The wing spar is idealized as a cantilever beam with lumped non-structural masses (rotor blades, hub and collective actuator assembly and motor), distributed non-structural masses (skin, ribs) and a hollow circular spar. A schematic of one half-wing is shown in Fig. 20.

The wing taper ratio $\lambda = \frac{c_{\text{TIP}}}{c_{\text{ROOT}}}$ is a design parameter with a default value of 0.75. A tapered wing shifts the non-structural skin mass inboard while strengthening the root, resulting in increased first bending frequency for the same mass. The skin thickness is set to include three layers of carbn fiber (45/0/45), resulting in an areal mass density of $\approx$ 4.9 kg/sq.m. During sizing, the rotor assembly mass and motor mass are used to calculate the wing spar thickness required to achieve a target first bending frequency using the Rayleigh-Ritz approximation detailed in Appendix A.

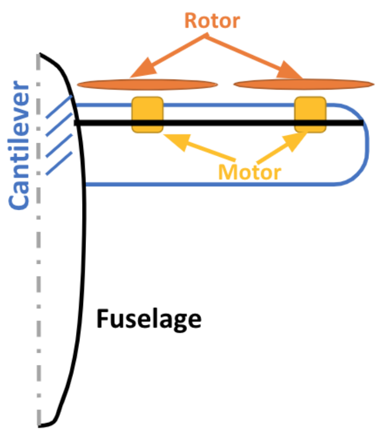


Figure 20: Wing structure: layout

To decouple wing structural dynamics from vehicle flight dynamics, a target first bending frequency of at least 4 Hz is desirable (for a take off mass of 2000 kg). Larger aircraft feature progressively relaxed natural frequency constraints, because the flight dynamic frequencies are lower for larger vehicles with correspondingly larger moments inertia.

4.7.3 Hover: shear loads

After determining the spar thickness based on natural frequency requirements, the wing structure is analyzed with static loads in hover with the tapered beam. Rotor thrusts are applied at the motor mount locations and the static bending stresses and shear stresses are calculated at five points between nodes (nodes are located at the root and motor mount locations). A load factor of 3.8 and safety margin of 1.5 are applied to ensure that the spar can support the loads in hover. This hover loading condition is more limiting than cruise flight with the same load factors and safety factors, because the bending moments due to distributed lift are lower than the corresponding bending moments in hover. The shear loading is used to estimate the number of $\pm$ 45 deg layers of carbon fiber, while the bending loads are supported by layers of uniaxial carbon fiber.

4.8 Airframe Weight Model

4.8.1 Statistical weight model

The AFDD82 fuselage weight model for helicopter is given by

$$w_{\text{fuselage}} = w_{\text{basic}} + w_{\text{press}} + w_{\text{cw}} \tag{26}$$

where w_{basic} is the basic weight of the fuselage, w_{press} is the weight from any pressurization constraints (set to none for the sample mission in this work) and w_{cw} is the weight addition for crashworthiness, which is assumed to be 6% of the basic weight as per AFDD standards. The basic weight is given by

$$w_{\text{basic}} = 5.896 f_{\text{ramp}} \left(\frac{W_{\text{GTOW}}}{1000}\right)^{0.4908} n_z^{0.1323} S_{\text{body}}^{0.2544} l^{0.61} \tag{27}$$

where f_{ramp} is the factor for a retractable ramp, n_z is the load factor, S_{body} is the wetted area of the fuselage (sq. ft) and l is the length of the fuselage (feet). While these terms can be defined for full-scale helicopters/tiltrotor, their definitions become increasingly challenging to interpret in the context of unconventional configurations such as an eVTOL, with very different load paths compared to helicopters.

4.8.2 Physics-based weight model

A physics-based model is used to estimate the weights of load-bearing components that transmit rotor hub forces and wing lift and drag to the vehicle center of mass. The airframe is subdivided into 3-D beam elements, each with bending, axial and torsion degrees of freedom. Rotor hub loads and wing lift/drag are modeled as point loads at the motor points, and distributed uniformly over the wing span, respectively. The external loads corresponding to each mission phase are applied on the structure, (rotor thrust, torque, wing lift and wing drag) and the resulting stresses and deflections are calculated and stored at all the nodes. Subsequently, the cross-section dimensions of the beams are iteratively resized to ensure; (i) Minimum factor of safety of 1.5 (based on Von-Mises stress) at a load factor of 3.5, (ii) A maximum deflection of 10% for any node (relative to its distance from the vehicle center). The cross-sections of all beam elements are assumed to be hollow circles with wall thickness equal to 15% outer radius. The only design parameter for a beam element is the outer radius of the cross-section. Beam cross-section radii for all elements are updated iteratively until the 3 design criteria are satisfied, and the weight of the airframe members is calculated using material density and final dimensions. The finite element analysis iterations are performed within the sizing loop. The parameterization for a quad-rotor layout is shown in Fig. 21.

Integration of FEA into sizing: workflow

The overall process is depicted in a flowchart shown in Fig. 22 and proceeds as follows

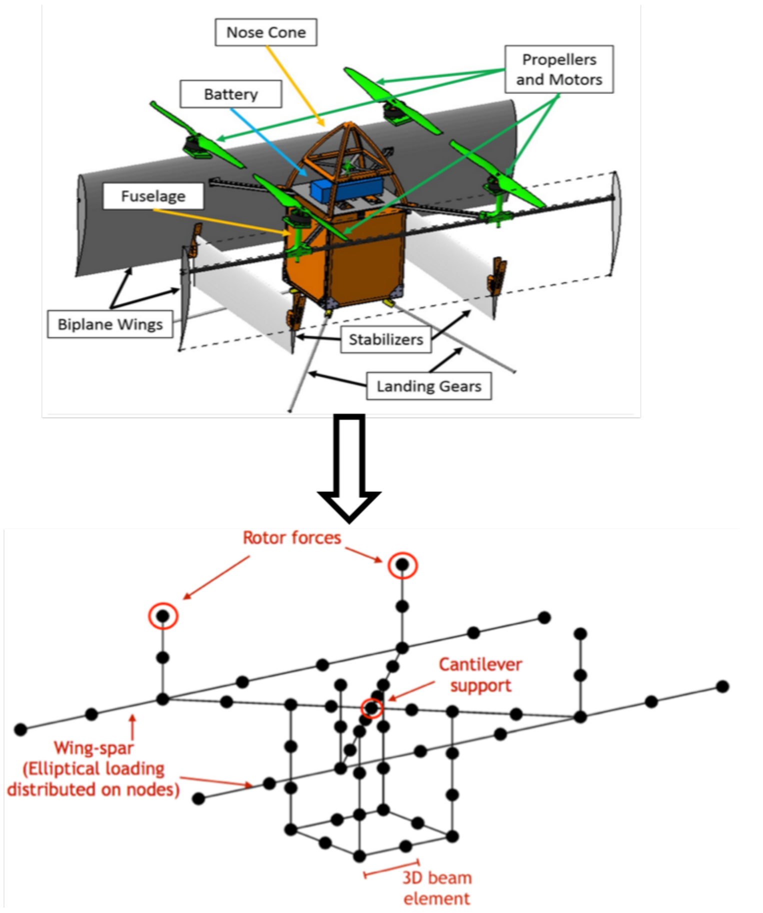


Figure 21: Quad-rotor biplane tail-sitter: layout to finite element representation

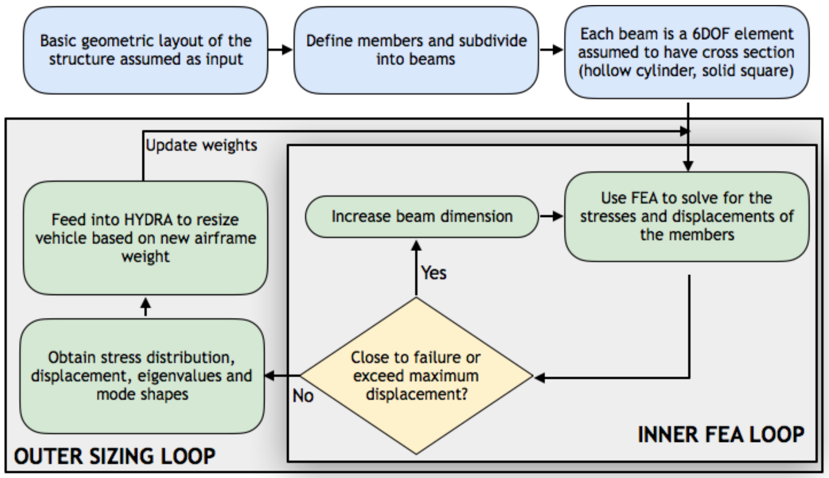

Basic geometric layout of the structure assumed as input
Define members and subdivide into beams
Each beam is a 6DOF element assumed to have cross section (hollow cylinder, solid square)
Update weights
Feed into HYDRA to resize vehicle based on new airframe weight
Increase beam dimension
Use FEA to solve for the stresses and displacements of the members
Yes
Close to failure or exceed maximum displacement?
No
Obtain stress distribution, displacement, eigenvalues and mode shapes
INNER FEA LOOP
OUTER SIZING LOOP

Figure 22: Workflow for including physics-based airframe weight model in sizing

1. **Initialize:** A geometric layout of the airframe is chosen, and beam elements are defined. The present work assumes the beam cross section to be simple shapes such as a hollow cylinder or a solid square.

2. **Inner FEA loop:** Point forces and moments are applied to the structure based on rotor and wing loads, such as rotor thrust, rotor torque and wing lift. The von-Mises stress (σ_{VM}), the corresponding factor of safety and deflection of the various nodes are computed as

$$\begin{aligned} \sigma_{\mathrm{VM}} &= [(\sigma_{xx}-\sigma_{yy})^2+(\sigma_{yy}-\sigma_{zz})^2 \\ &+ (\sigma_{zz}-\sigma_{xx})^2+6(\sigma_{yz}^2+\sigma_{xy}^2+\sigma_{xz}^2)]^{1/2} \end{aligned} \tag{28}$$

   where σ_{ij} are components of the stress tensor and the factor-of-safety (FOS) is

$$\mathrm{FOS}=\min(\sigma_{VM})_i/\sigma_{\mathrm{yield}} \qquad \forall \;\; i \; \in N \tag{29}$$

   where N is the total number of beam elements.

3. **Update cross-section dimensions:** The mathematical constraints imposed for convergence are

$$|\mathrm{FOS}_{\mathrm{tar}}-\Delta\mathrm{FOS}| \leq \min(\mathrm{FOS}) \tag{30}$$

   where $\mathrm{FOS}_{\mathrm{tar}}$ is the target FOS, set to 1.5 and ΔFOS is the allowable band, set to 0.1. The axial and bending deflections of each node are normalized by the distance of the node from the vehicle center of gravity. These normalized deflections must not exceed a pre-determined fraction (0.08) to ensure structural rigidity. Depending on the type of condition encountered, two types of cross-section updates are calculated:

(a) The beam cross-section is scaled up by the term $(\mathrm{FOS}/\mathrm{FOS}_{\mathrm{tar}})^{1/3})$ if the factor of safety is not sufficiently large. The cube root in the expression stems from the fact that stress is inversely proportional to cross-section dimension under static load.

(b) If the deflection of any one node is too large, the dimensions of all cross-sections are increased by 10%. Increasing the sizes of all the member cross-sections is required because adding stiffness to the root end of a longer beam may be more effective than stiffening an outboard member (in the present workflow, no apriori assumptions are made about the load path, and so to guarantee deflections are restricted, all members are stiffened).

The larger of the two calculated dimensions for each structural member is provided to the FEA for the next iteration. The stresses and deflections are recalculated, and members resized until the structure achieves the required factor of safety and deflection limits.

The airframe weight is computed by multiplying the total volume of all the beam elements with the material density (Aluminum, 2,700 kg/m^3 or carbon, 1650 kg/m^3). This converged airframe weight from static finite element analysis replaces the fuselage weight from the AFDD empty weight formulae in the iterative sizing process.

The structural analysis used in in sizing the airframe structure also provides estimates of the natural frequencies and the mode-shapes. This information can be used in advanced stages of design to ensure sufficient separation between the airframe, blade natural frequencies and operating RPM range of the rotor(s).

4.9 Weight margin

A weight margin is included in the analysis to admit additional expansion of component performance (and therefore weight) at later stages of design. Therefore, **10% of the vehicle empty weight** is allocated for a weight margin.

4.10 Fixed weight groups

Fixed empty weight groups consist of mission-specific components and do not scale with vehicle dimensions or system performance during sizing. These fixed weight groups are

1. Carpets, cabin trim and noise insulation
2. Avionics, sensors and flight computers
3. Airconditioning and heaters
4. Seats
5. Low voltage battery, anti-collision lights, air data systems
6. High voltage bus and power distributors

5 Analysis Organization

`HYDRA` is a rotorcraft sizing code with the following features:

1. **Sizing**: obtain consistent vehicle weights and performance for a given payload and mission profile.

2. **Optimization** Optimize a given design (e.g., for annual operating cost)

3. **Sensitivity Analysis** For a given design, identify sensitivity of weights and performance metrics to (i) underlying assumptions (e.g., engine efficiency), and (ii) changing design variables (e.g., rotor tip speed)

The main focus of this document is to explain the usage of the analysis, and implementation of the various performance, weight and cost models used to size the rotorcraft. The theory, if any, will be explained side-by-side with the corresponding code segment.

5.1 Programming Language and Syntax

`HYDRA` is written in `Python3`, with heavy emphasis on classes to contain and compartmentalize information. Because `Python2` support expires in 2019, it is recommended to use `Python3` specifically.

Some parts of the code are written in `Fortran`. Three fortran compilers have been tested and verified to work with the compiled parts of the code: GNU fortran, intel fortran and PGI fortran. Among these three compilers, `gfortran` and `pgfortran` are free, while `ifort` requires academic or commercial licenses. The main advantage of `pgfortran` is the availability of support for `CUDA-Fortran`, while executables generated with `ifort` consistently run faster than those generated with `gfortran` or `pgfortran` for the target applications.

`HYDRA` also features a built-in Blade Element Momentum Theory (BEMT) solver, also implemented in `Fortran90`. The BEMT solver is parallelized with `OpenMP`. Additionally, the sizing and optimization stages of `HYDRA` are parallelized with `MPI`, and can be executed in parallel.

5.2 Pre-Requisites and Installation Notes

A **Fortran90 compiler** is required. Due to the use of free-format coding, derived types and modules, *a Fortran77 compiler is not sufficient.*

If you want to try running the code on Windows, there are some manual steps required to help certain Python modules recognize the OpenMP libraries. For the time being, consult Ananth for installation details on Windows.

Additional **Python3** and its modules (freely available) are required to run HYDRA and its various postprocessors:

1. **matplotlib, numpy, scipy, pyyaml, python3-tk, f90wrap**
2. Optional: mpi4py

Finally, **CMake** version 3.0 or higher is required to create Platform-independent Makefiles that compile the Fortran90 routines. To use integrated LaTeXrendering for fonts in plots, you may also need the **texlive-fonts-extra** package.

5.3 Source Code Directories

All source code to be compiled is located in **src**/, under various subfolders. Each subfolder contains a collection of subroutines or functions that pertain to specific types of operations.

5.3.1 Python code: src/Python

This subfolder contains various classes and functions written in Python that pertain to sizing a vehicle, in four sub-folders.

1. **Stage_0**/: This folder contains the "main" class that defines the vehicle in **hydraInterface.py**. It contains the various high-level driver functions pertaining to sizing and optimization. The folder also contains two other files, **dict2obj.py** (convert dictionary to class) and **footprint.py** (calculates vehicle footprint).

2. **Stage_1**/: This folder contains the workhorse routines that perform sizing with weight, performance and cost estimation.

3. **Stage_3**/: This folder contains Python-wrapped routines for higher-fidelity fuselage and wing weight models, BEMT rotor performance models and the associated interfaces to setup the inputs and extract outputs for the sizing analysis.

4. **Postprocessing**/: This folder contains functions that perform postprocessing analysis, including optimization, design perturbation and sensitivity studies,

noise estimation, battery status, profile, power curve, and pie charts for the breakdown of vehicle weight, operating cost and parasitic drag.

5.3.2 NETLIB FILES: Source/fea/Source/mathops/

This subfolder contains the ODE Solver `dassl`, algebraic equation solver `hybrd` and various dependencies needed for linear algebra and matrix operations. These files have not be modified, except the algebraic equation solver `hybrd.f`. This file alone has been changed to obtain the Jacobian in parallel using OpenMP.

Parts of the sizing analyis, FEA-based fuselage/wing weight estimation and BEMT-based rotor performance estimation are written in Fortran90 to improve execution speed. The source code for these operations are contained in **src/bemt/Source/**, **src/fea/Source/** and **src/sizing/Source/**. Within each of these folders, there are three sub-folders:

5.3.3 Python-integrated routines: Source/pyint_files/

This subfolder contains routines that can be invoked from Python, using `f2py` and James Kermode's `f90wrap` adaptation for derived types.

5.3.4 Python-accesible modules: Source/pyint_modules/

The files in this folder contain module definitions used to define universal constants (e.g., 1, 0, π) to the required precision. Using James Kermode's `f2py-f90wrap` interface generator, `Python` can access and modify these module variables directly without having to painstakingly break down derived types into its primitive constituents (arrays, integers, real numbers and logicals). The advantage of this automation and abstraction helps avoid programming errors, reduces the number of arguments passed between `Python` and `Fortran` and allows for code feature expansion without significant additional effort.

5.3.5 Python-invisible files: Source/pyinv_files/

The files in this folder are work-horse routines that can be called only by the routines in **Source/pyint_files/**, but not directly by `Python`. These routines, together with those in Source/pyint_files/, are compiled into shared object files. Specific routines (present in Source/pyint_files/) in these shared objects can be called by the `Python` interface.

6 First-time setup

6.1 Installing pre-requisites

For Debian-based Linux distributions, you can get `Python3`, CMake and gfortran using the command

sudo apt-get install python3 cmake gfortran texlive-fonts-extra

For MacOS, similar commands can be executed with brew.

After installing `Python3`, install various modules using `pip`

sudo python3 -m pip install numpy

sudo python3 -m pip install scipy

In a similar manner, install the other modules with a terminal.

6.2 Compiling source code

After installing the pre-requisites, create the directory **cmake/build/** as follows

cd cmake/

mkdir build/

cd build/

Then, execute the cmake command and compile the code

FC=gfortran cmake ../../

make -j

6.3 MacOS-specific issues

Hopefully, there are no issues and the build process executes successfully. However, if it does not, and you are reading this, chances are that you are using Windows, or Mac. For Windows, there is a long and complicated series of steps documented here: `https://github.com/jameskermode/f90wrap/issues/73`

For MacOS, the system may not recognize the `f90wrap` and `f2py-f90wrap` commands even if the **f90wrap** module is installed - this issue was noticed for OSX High Sierra and v3.7 of `Python`. In case this situation arises, you may have to create symbolic links to these two commands in `/usr/local/bin/`. For example, Ananth had to create a symlink as follows:

1. Identify where **f90wrap** is installed: use the following command in terminal

   python3 -m site

   If `Python3` is installed correctly, you will see an output that looks like this:

```
sys.path = [
    '/usr/local/bin',
    '/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/
        Versions/3.7/lib/python37.zip',
    '/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/
        Versions/3.7/lib/python3.7',
    '/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/
       Versions/3.7/lib/python3.7/lib-dynload',
    '/usr/local/lib/python3.7/site-packages',
    '/usr/local/lib/python3.7/site-packages/RBF-2018.10.31-py3.7-
       macosx-10.13-x86_64.egg',
    '/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/
       Versions/3.7/lib/python3.7/site-packages',
    '/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/
       Versions/3.7/lib/python3.7/site-packages/RBF-2018.10.31-py3.7-
       macosx-10.13-x86_64.egg',
]
USER_BASE: '/Users/ananthsridharan/Library/Python/3.7' (doesn't exist)
USER_SITE: '/Users/ananthsridharan/Library/Python/3.7/lib/python/site-
    packages' (doesn't exist)
ENABLE_USER_SITE: True
```

The results of this command show that **f90wrap** may be found in

```
/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/Versions
    /3.7/bin/
```

To verify that the **f90wrap** and **f2py-f90wrap** commands can indeed be located in that folder, use the **ls** command as follows: (without the linebreak)

ls /usr/local/Cellar/python/3.7.0/Frameworks/
Python.framework/Versions/3.7/bin/*f90wrap

You should see an output that looks like this:

```
/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/Versions
   /3.7/bin/f2py-f90wrap
/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/Versions
   /3.7/bin/f90wrap
```

Now, we're ready to use the **ln** command to create symlinks.

2. First, create a link for f90wrap:

   ln -s /usr/local/Cellar/python/3.7.0/Frameworks/ Python.framework/Versions/3.7/bin/f90wrap /usr/local/bin

3. Next, create a link for f2py-f90wrap:

   ln -s /usr/local/Cellar/python/3.7.0/Frameworks/ Python.framework/Versions/3.7/bin/f2py-f90wrap /usr/local/bin

   Then, try deleting the build directory and redo all the compilation steps.

7 Input files

This section of the documentation details the inputs files and conventions that that are necessary to perform sizing. The two main input files are **input.yaml** and **defaults.yaml**.

7.1 input.yaml

This input file specifies the mission profile, vehicle configuration, propulsion type and masses of fixed weight groups (e.g., crew, payload, mission equipment). An example input.yaml file is shown below in various segments, and each input and its units are explained.

`HYDRA` inputs and outputs are handled using **yaml** files through the **pyyaml** package. The yaml input files are converted to `Python` dictionaries when read, allowing the inputs to be specified in any order. The main dictionaries in **input.yaml** are **Sizing**, **Configuration**, **Mission** and **Aircraft**. All four dictionaries must be present in **input.yaml**. An example file is shown below with the **main dictionary names** and **sub-dictionary names** highlighted.

```
Sizing:
   Rotors:
      All_rotors:
         Nb:         [3]                   # per rotor
         Vtip:       [170]                 # hover tip speed, m/s
         ctsigma:    [0.139]
```

```
  Wings:
    Main_wing:
      nwing:          [1]
      aspectratio:    [5,6,7,8]
      cl:             [0.4,0.5,0.6]
      liftfraction:   [0.9]
      nrotors:        [6]
    Tail:
      nwing:          [1]
      aspectratio:    [6]
      cl:             [0.3]
      nrotors:        [2]
  Fuselage:
    nrotors:     0
# fidelity options for weight and performance models
  ifea:              False
  use_bemt:          False

Configuration:
  Rotors:
    All_rotors:   'All_motors' # motor group to drive this rotor group
  Wings:
    Main_wing:    'All_rotors' # what rotor group to use on this wing
    Tail:         'All_rotors' #
```

```
# note: 4k95 = 1219.2 m, ISA + 27.92 C, 6k95 = 1828.8 m, ISA + 11.88 C

# flight mode: 0-idle, 1-hover, 3-cruise

Mission:

   nsegments:        4

   flight_mode:      ['hover','cruise', 'cruise','hover' ]

   time_seg:         [   1.5,      0,       0,    1.5  ]

   start_altitude:   [   0.0,    00.0,      0,      0  ] # m

   end_altitude:     [   0.0,     0.0,      0,      0  ] # m

   delta_temp_isa:   [   0.0,     0.0,      0,      0  ] # centrigrade

   cruise_speed:     [     0,     98,      98,      0  ] # knots

   distance:         [     0,     50,      15,      0  ] # in km

   add_payload:      [     0,      0,       0,      0  ] #

   segment_type:     [  'all',   'all','reserve', 'all' ] #

   sizing_order:     [     1,      2,       0,      0  ] #

   # fixed_GTOW:       1400.0

Aircraft:

   aircraftID: 2

   # 1: SMR, 3: Coax, 4: Quadrotor (needs to be improved), 5: custom


   # payload, crew (kg)

   mass_payload:   320.0      # 250 kg payload + 70 kg margin

   mass_crew:       0

   avionics:       79.2
```

```
common_equipment: 24.0     # HVAC systems - common for all PAX

common_per_pax:  00.0

pax_count:        0          # passengers (uses pax -> baggage map)


nrotor:           8         # total number of rotors

npropeller:       0         # Cruise propeller count

engineType:     'electric_motor'    # engine parameters
```

Each main dictionary and its relevant inputs are discussed below.

7.1.1 Sizing

This dictionary specifies the values of high-level design variables to use for sizing the vehicle. Up to three sub-dictionaries (**Rotor, Wings and Fuselage**) and two logical inputs for using higher-fidelity models constitute the sizing dictionary.

7.1.1.1 Rotors

The **Rotors** dictionary consists of sub-dictionaries, with each sub-dictionary corresponding to a rotor. Several instances of this rotor unit may exist on the vehicle; this detail is provided in the **Configuration** dictionary, detailed in the following sub-section.

Each **Rotor** group is determined by several sizing design variables. In the present example, the **Rotor** sub-dictionary is

```
Rotors:

  All_rotors:
```

```
Nb:       [3]                  # per rotor
Vtip:     [170, 180, 190]    # hover tip speed, m/s
ctsigma:  [0.13, 0.139]
```

This definition shows that there is one rotor group used in the vehicle called ‘All_rotors’, with three design variables to size rotors in this group:

1. Number of blades N_b - single or multiple integers
2. Tip speed V_{TIP} in m/s - list of floats
3. Hover blade loading $C_T/\sigma = \text{T}/(\rho N_b c R V_{\text{TIP}}^2)$ - list of floats

Rotor radius: If the rotor radius is specified, then it is set to the input value. If the disk loading is specified, rotor radius is calculated based on the vehicle weight, rotor thrust share and hover down-load (specified in **defaults.yaml**).

If the rotor disk loading in hover is not specified (keyword **DL**, lb/sq.ft) and the rotor radius (keyword **radius**, meters) is also not specified, the rotor is assumed to be mounted on a wing, and the size is set based on wing span (calculated in fixed wing sizing) and rotor tip clearance (specified in **defaults.yaml**). After calculating rotor radius, the tip speed is used to identify the mean **rotor blade chord**. If the rotor geometric solidity (keyword **solidity**) is specified, then the hover blade loading is calculated. Otherwise, the the hover blade loading (keyword **ctsigma**) must be specified, and the geometric solidity is calculated. Some additional inputs are rotor cruise RPM to hover RPM ratio (keyword **RPM_ratio**), and first flap frequency in hover (keyword **fl_freq**). If these inputs are not specified, then default values of

$\Omega_C/\Omega_H = 0.5$ and ν_β=1.1/rev are assumed.

7.1.1.2 Wings

The **Wings** dictionary consists of sub-dictionaries, with each sub-dictionary corresponding to a fixed wing type. Several instances of this fixed wing unit may exist on the vehicle; this detail is specified by the parameter **nwing**. In the example shown above, the **Wings** dictionary is

```
Wings:
   Main_wing:
      nwing:           [1]
      aspectratio:     [5,6,7,8]
      cl:              [0.4,0.5,0.6]
      liftfraction:    [0.9]
      nrotors:         [6]
   Tail:
      nwing:           [1]
      aspectratio:     [6]
      cl:              [0.3]
      nrotors:         [2]
```

This input specifies one fixed wing called "Main_wing", and another called "Tail". The number of wings is specified using the parameter **nwing**, which is usually 1. The input **nwing** =[2] is used for tandem-wing designs, where both the canard and main wing are of identical construction. The parameter **aspectratio** is the wing aspect ratio b^2/S_{wing}, **cl** is the wing cruise lift coefficient and **liftfraction** is the fraction of vehicle weight carried by the wing group in cruise. Finally, the parameter **nrotors** specifies the number of rotors mounted on a wing of this group. Here, 6 rotors are mounted on the main wing, and 2 rotors are mounted on the tail. Each input parameter can be specified as a list of values to investigate.

7.1.1.3 High-fidelity model switches

The parameter **ifea** is a logical input, used to specify if the FEA-based weight model is to be used in the sizing loop to estimate the airframe structural weight. The other parameter **use_bemt** indicates whether the Blade Element Momentum Theory (BEMT) should be used to refine rotor performance estimates in hover (and cruise for prop-rotors). In this example, both options are not enabled.

7.1.2 Vehicle configuration

```
Configuration:
   Rotors:
      All_rotors:   ‘All_motors’ # motor group to use to drive this rotor
   Wings:
      Main_wing:    ‘All_rotors’ # what rotor group to use on the main wing
      Tail:         ‘All_rotors’ # what rotor group to use on the tail
```

This dictionary specifies the vehicle configuration, and associations between wings/-fuselage and rotors, and drive motors for each rotor group. In the example given above, the configuration dictionary is repeated below. The first sub-dictionary for the configuration specifies that rotors belonging to the group "All_rotors" are driven by motors specified by a group called "All_motors". The rotor group name corresponds to the string used to specify the rotor design variables in the **Sizing** section. The second sub-dictionary for the vehicle configuration specifies that on the fixed wing called 'Main_wing', the rotors mounted on this structure are of the type "All_rotors". Similarly, the "Tail" fixed wing features rotors of type "All_rotors".

7.1.3 Mission profile

The mission profile from **input.yaml** is repeated below for convenience:

```
Mission:

   nsegments:         4

   flight_mode:       ['hover','cruise', 'cruise','hover' ]

   time_seg:          [   1.5,      0,        0,    1.5   ]

   start_altitude:    [   0.0,    00.0,       0,      0   ] # m

   end_altitude:      [   0.0,     0.0,       0,      0   ] # m

   delta_temp_isa:    [   0.0,     0.0,       0,      0   ] # centrigrade

   cruise_speed:      [     0,      98,      98,      0   ] # knots

   distance:          [     0,      50,      15,      0   ] # in km

   add_payload:       [     0,       0,       0,      0   ] # extra payload

   segment_type:      [  'all',   'all','reserve', 'all' ]

   sizing_order:      [     1,       2,       0,      0   ] #

   # fixed_GTOW:        1400.0
```

The parameter **nsegments** specifies the number of mission segments. Each segment type can be 'hover' or 'cruise', specified by the parameter **flight_mode**. For hover segments, the duration is specified by the parameter **time_seg**. For cruise segments, either the segment duration can be specified, or the **distance** can be provided as an input and the segment velocity **cruise_speed** will be calculated. If the distance is provided as a non-zero input value, the cruise segment duration input is ignored. For climb segments, the start and end altitudes **start_altitude**, **end_altitude** are specified in meters.

The parameter **add_payload** is used to inform the analysis that the payload has changed at the end of a mission segment (e.g., cargo picked up/dropped off or passenger disembarked/entered the vehicle). The input parameter **segment_type** specifies the type of mission segment - 'all' indicates the segment is used in day-to-day operations, while 'reserve' indicates that the segment is used for sizing, nut not operating cost calculations. The parameter **sizing_order** is a list of integers, with zero specifying that the segment is not used for sizing, and a non-zero integer specifying that the segment is used to size a particular component. Usually, the first hover segment and first cruise segment are used for sizing various components. The final line is an optional input specifying the parameter **fixed_GTOW**. If this line is present, it instructs the analysis to run sizing in fixed take-off weight mode, and estimate the payload based on the remaining mass.

7.1.4 Aircraft

The aircraft specification dictionary is repeated below for convenience. The payload mass, crew mass, common equipment and avionics mass are fixed mass groups, specified in kg. The parameter **common_per_pax** is the mass of equipment that is multiplied by the number of passengers, specified by **pax_count**. Based on the number of passengers, additional payload is added internally with the following map: 150 kg for first passenger, 125 kg for next two passengers and 120 kg for the fourth and fifth passengers.

The parameters **nrotor** and **npropeller** are outdated inputs. Finally, **engineType** specifies the powerplant that is used to store and produce mechanical energy. Valid entries are *'electric_motor'* (with Lithium ion battery), *'turboshaft'* (with mechanical transmission), *piston* (with mechanical transmission), *turbo_electric* (turboshaft with generator and electric motors) and *piston_electric* (piston engine with generator and electric motors).

```
Aircraft:

  aircraftID: 2             # 1: SMR, 3: Coax, 4: Quadrotor, 5: custom

  mass_payload:   320.0     # 250 kg payload + 70 kg margin

  mass_crew:       0

  avionics:       79.2

  common_equipment: 24.0    # HVAC systems - common for all PAX

  common_per_pax: 00.0

  pax_count:       0        # passengers (uses pax -> baggage map)


  nrotor:          8        # total number of rotors

  npropeller:      0        # Cruise propeller count

  engineType:     'electric_motor'    # engine parameters
```

7.2 defaults.yaml

This input file contains the sizing constraints, motor efficiencies, powerplant details, calibration factors for empirical/reduced-order weight and peformance models and cost models. Also included are "technology factors", i.e., multipliers applied to weight predictions for each component. The file is too large to be printed verbatim as a whole unit; instead the text is subdivided into dictionary-sized segments, and each dictionary in this input file is detailed below.

7.2.1 Sizing dictionary

```
Sizing:
  Constraints:
    max_rotor_radius:  12.0 # m
    max_ct_sigma   :    0.14
    max_gtow       :  5000.0 # kg
```

Constraints used to size the vehicle are specified in this dictionary. The first parameter **max_rotor_radius** is the maximum rotor radius (for single main rotor helicopters) or the maximum vehicle footprint (for eVTOL). The parameter **max_ct_sigma** is the upper limit on rotor blade loading in hover. Finally, **max_gtow** is the upper limit on take-off mass imposed for the vehicle. These constraints are imposed during optimization as well as iterative sizing. If any of these three constraints are violated during fixed-point iterations, the sizing loop is terminated and the design is marked “invalid”.

7.2.2 Empirical parameters

This dictionary contains several sub-dictionaries, each of which are detailed below. A sample dictionary is broken into sub-dictionaries and explained below.

7.2.2.1 Transmission

```
Empirical:                      # Empirical modeling parameters
  Transmission:
    eta:            1.00      # transmission efficiency
```

The parameter **eta** is a fraction between 0 and 1, quantifying the efficiency of the electrical/mechanical transmission. Both wires and driveshafts/gears feature upwards of 98% efficiency.

7.2.2.2 Electric motors

```
  Motors:                                   #motor efficiencies
    All_motors:
      hover_efficiency: 0.90
      cruise_efficiency: 0.85
```

Presently, the empirical parameters used for motor sizing are **hover_efficiency** and **cruise_efficiency**. In reality, prop-rotors may operate at significantly lower cruise RPMs, resulting in different electrical-to-mechanical energy conversion efficiencies for the electric motors and speed controllers.

7.2.2.3 Battery

```
Battery:
  Cell:
    sp_energy:  240.0     # measured in W-hr/kg
    Tmax:        70.0     # max cell temperature, deg C
    energy_vol: 632.0     # energy density, Watt-hours/liter
    volume:       0.01708 # volume of a cell unit, liters
  Pack:
    SOH:          0.8     # state of health; 0 = gone; 1 = brand new
    DOD_min:      0.075   # minimum depth of discharge;
    integ_fac:    0.75    # battery pack integration factor for mass
    vol_fac:      0.3     # battery volume integration factor
  Force_sizing:   ‘energy’ # ignore cell count or temperature effects
```

The **battery** sub-dictionary features parameters to model individual cells, as well as the battery pack. The cell parameters are **sp_energy** (maximum rated energy stored per unit cell mass), **Tmax** (maximum rated cell temperature), **energy_vol** (rated energy per unit cell volume) and **volume** (unit cell volume in liters).

The battery pack is quantified by the following parameters

1. State of health **SOH** - the maximum energy that can be stored in the pack, as a fraction of its rated energy. This parameter is usually less than unity because charge/discharge cycling of cells results in reduced energy storage capacity.

2. Minimum depth of discharge **DOD_min** - the minimum energy capacity that the pack must retain to avoid permanent set and reduced energy capacity in individual cells.

3. Pack mass integration factor **integ_fac** - the ratio of battery cell mass to pack mass, to account for the battery casing and power management systems.

4. Pack volume factor **vol_fac** - the ratio of cell mass to battery pack mass, to account for additional components introduced by the mass integration factor, as well as clearances for cell cooling.

Finally, the battery sizing option **Force_sizing** is character input that directs the battery sizing module to ignores thermal effects and pack voltage constraints; if this input is present, only energy-based sizing is performed.

7.2.2.4 Aerodynamics

The **aerodynamics** dictionary is used to specify rotor hover and cruise efficiencies, interference losses, wing efficiencies and body drag. A sample dictionary is shown below.

```
Aerodynamics:
  Rotors:
    hover_dwld_factor:  0.015
    cd0:                0.012
    induced_power_factor: 1.18
    FM:                 0.75
    kint:               1.02
    hover_thrust:       ‘equal’
  Wings:
    oswald:         0.8
    cd0:            0.014
  Propellers:
    eta:            0.85
  Body:
    flat_plate_factor: 0.88    # means use drag build-up model
```

The Rotors sub-dictionary contains modeling constants to quantify hover efficiency

1. **hover_dwld_factor** is the additional fraction of nominal rotor lift share that the rotor has to produce in hover to overcome vertical down-load due to the rotor wake impinging on the structure of the vehicle
2. **cd0** is the average profile drag coefficient of the rotor blade airfoil section
3. **induced_power_factor** is the multiplier for ideal rotor power to account for non-ideal losses
4. **FM** is the rotor hover figure of merit. If the hover figure of merit is given as non-zero, then **FM** is used to estimate rotor shaft power in hover. If the value of figure of merit is given as zero, then the profile drag coefficient and induced power factor are used to estimate rotor hover power.

5. **kint** is the rotor power scaling factor to account for interference losses.

6. **hover_thrust** is a character input that ignores the calculated rotor lift share, and distributes the hover thrust equally across all rotors in the system. This input is primarily present for backward-compatibility, and should not be used extensively except for debugging.

The empirical parameters used to model wing performance are the Oswald efficiency factor **oswald** (quantifies additional induced drag for non-elliptical span loading) and the mean profile drag coefficient **cd0**. The efficiencies of dedicated cruise propellers and prop-rotors in cruise is quantified by the **Propeller** parameter **eta**. Finally, the scaling factor to estimate body drag from weight using a weight-drag trendline is the parameter **flat_plate_factor**. If this input is zero, then the analysis uses a component drag build-up to estimate parasitic drag.

7.2.2.5 Geometry

```
Geometry:
   fuselage_width:  1.00        # fuselage width in meters
   fuselage_length: 7.00        # fuselage length in meters
   clearance:       0.15        # rotor tip clearance / radius ratio
```

The three geometry parameters used for sizing are the fuselage width at the widest point **fuselage_width** (meters), airframe length **fuselage_length** (meters) and the rotor clearance parameters **clearance**. This final parameter is the in-plane clearance between a rotor plane and other rotor planes/vehicle fuselage.

7.2.2.6 Technology factors

The term technology factor refers to scaling factors ("multipliers") used to increase or decrease component empty weights to account for improvements in lightweight manufacturing. A sample dictionary is shown below.

```
Tech_factors:
   Weight_scaling:
      rotor:          1.0          # rotor blades and hub
      wing:           0.9         # wings
      empennage:      1.0          # tail surfaces/winglets
      fuselage:       1.0          # airframe structure
      landing_gear:   0.4          #
      fuel_system:    1.0          # fuel pumps for engines
      drive_system:   1.0          #  transmission
      flight_control: 1.0          #
      anti_icing:     0.0          #
      powerplant:     0.76         # motors and engines
      fuel:           1.0          # fuel
      battery:        1.0          # battery
```

7.2.3 Acquisition cost model

There are two types of costs associated with the rotorcraft that are modeled in `HYDRA`: acquisition cost (component purchase) and operating cost. The dictionary **Acquisition** contains three sub-dictionaries: **Fixed_cost**, **Scaling_cost** and **Beta_acq_factors**. Each of these dictionaries is detailed below with examples.

7.2.3.1 Fixed acquisition cost

```
Acquisition:
   Fixed_cost:
      sense_avoid:   189817.0           # USD
      avionics:      145807.0           # USD
      interiors:      45152.0           # USD, air conditioning/heater/HUD
      testing:         6400.0           # USD
```

This sub-dictionary contains inputs for the cost of vehicle components that do not vary with vehicle size. These groups include the sense and avoid system, avionics, interiors and component testing.

7.2.3.2 Scaling_cost

This sub-dictionary contains inputs for the cost of vehicle components that scale with the mass of each component. Several components fall under this category, particularly airframe, wings, landing gear, rotor blade structures, transmission lines and motors. A sample dictionary is shown below.

```
   Scaling_cost:
      final_assem_line:  90.07          # USD/kg of take-off mass
      BRS:                10.885        # USD/kg of take-off mass
      fuselage:         2807.0          # USD/kg of fuselage weight
```

```
    landing_gear:    1725.0          # USD/kg of landing gear strl. weight
    wing_structure:  3779.1          # USD/kg of wing   structural weight
    motors:          2669.0          # USD/kg of drive motor mass
    power_dist:        31.0          # USD/kW of installed power
    rotor_blade:    77605.0          # USD/sq.m of plan-form area
    rotor_hub:      14133.0          # USD/kg: hub+collective actuator
    wires:             20.3          # USD/kg of wire weight
    tilt_actuator:   2868.0          # USD/kg of tilt actuator weight
    wing_flap:       2619.0          # USD/kg of wing flap/aileron
```

7.2.3.3 Acquisition cost scaling factors

This dictionary deals with cost multipliers that account for reduction of component prices associated with mass-production. A value less than one indicates that the component is less expensive when manufactured in bulk.

```
Beta_acq_factors:                    # acquisition cost multipliers
    sense_avoid:       0.5
    avionics:          0.75
    interiors:         1.0           # air conditioning/heater/HUD
    testing:           1.0
    final_assem_line:  0.5
    BRS:               0.75
    fuselage:          0.2
    landing_gear:      0.2
```

```
      wing_structure:    0.2
      motors:            0.2
      power_dist:        1.0
      rotor_blade:       0.2
      rotor_hub:         0.2
      wires:             1.0
      tilt_actuators:    0.75
      wing_flaps:        0.75
```

7.2.4 Operations

Operating costs are of two types: **annual** and **hourly** operating costs. These two components of the operating cost model are specified as sub-dictionaries under **Operations**. Additionally, the constants required to obtain **battery** life cycle costs are also specified. Finally, **vertiport** details are also specified and the cost of an eVTOL trip is compared to the cost of using a **taxi** for traveling the same distance. Each of these sub-dictionaries are detailed below.

7.2.4.1 Annual costs

```
Operations:
   Annual:
      Flight_hours:      1500            # flight hours per year
      Liability:         22000           # USD liability insurance per year
      Inspection:        7700         # USD, per year
```

```
Insurance_percent: 4.5            # insurance, % of acquisition
Depreciation_percent: 10          # % of acquisition cost depr./year
Pilot:             280500         # USD, pilot cost to company/year
Training:          9900           # USD, pilot training/year
```

The number of flight hours per year (**Flight_hours**) is used to calculate the equivalent cost per flight hour from the annual fixed costs incurred in ensuring vehicle airworthiness. The annual costs incurred are **Liability**, **Inspection**, insurance and depreciation (specified as a percentage of acquisition cost, **Insurance_percent** and **Depreciation_percent** respectively). For a piloted vehicle, additional costs are incurred for pilot salary and overhead (**Pilot**) as well as continuous training (**Training**).

7.2.4.2 Hourly costs

This dictionary specifies the maintenance and inspection costs associated with operating an air vehicle. The three elements in this dictionary are

1. Frame inspection (**Frame_maintenance**): specified as a cost in currency per flight hour
2. Rotor blades, collective actuator and hub inspection (**Rotor_inspection**): specified as a cost in currency per flight hour per unit assembly
3. Electric motor inspection (**Motor_inspection**): specified as a cost in currency per flight hour per motor unit.

```
Hourly:
    Frame_maintenance:   37.35          # $/flight hr
    Rotor_inspection:     1.0           # $/flight hr/rotor
    Motor_inspection:     0.625         # $/flight hr/rotor
```

7.2.4.3 Battery costs

This dictionary specifies the number of charge/discharge **Cycles** that a battery can withstand before its energy reduces to the design state of health specified in **Sizing** → **Battery** → Pack → **SOH**. Additionally, the purchase price of a battery pack as well as the battery recharge cost are specified as cost per unit rated energy/cost per unit of energy (**Cost_per_kwh**, **Electricity** respectively).

```
Battery:
    Cycles:           900
    Cost_per_kwh:     180.0             # battery cost/rated energy, $/kWh
    Electricity:        0.20            # electricity/unit energy, $/kWh
```

7.2.4.4 Vertiport

The relevant operational details at the vertiport are the landing tariffs (**Landing_fees**) per touchdown, the distance from the vertiport to the final passenger destination (**Ground_distance** in km) and the additional time spent in commuting to and from the vertiport changing modes of transport (**Padding_time**).

```
Vertiport:
    Landing_fees:      20.0             # landing fee per flight, USD
    Padding_time:      26.0             # min, curb -> UAM + UAM -> curb
    Ground_distance:    2.0            # last leg distance in km
```

7.2.4.5 Taxi details

The details of a **taxi** that can compete with a short-range aircraft are the tariff per unit distance traveled (**Distance_rate** in currency per km), a time tariff (**Time_rate** in currency per minute) and the additional time required to change from another mode of transport to the taxi (**Padding_time**, minutes).

```
  Taxi:
    Distance_rate:      0.55           # Taxi price in USD per km
    Time_rate:          0.36           # USD/minute of taxi ride
    Padding_time:      15.0            # airport gate to curb, minutes
```

7.2.5 Redundant systems

The dictionary **Redundancies** specifies components that feature doubly or triply redundant backups for flight controls, power cables and avionics. The redundancy factors are used to proportionally increase the component empty weights and associated group costs.

```
Redundancies:
  wing_flap:             1.0
  tilt_actuator:         2.0
  wires:                 1.0
  avionics:              1.0
```

8 Output files

This section of the documentation details the output file generated at the end of sizing. The file pattern is **log<XYZ>.yaml**, where <XYZ> is an integer representing a unique design ID number. The output is in plain text formatted as a YAML file, similar to the input files. These files can be read an automatically converted to `Python` dictionaries using the **pyyaml** package. All outputs are stored under the **outputs/log/** subdirectory in the run folder.

8.1 log<XYZ>.yaml

An example output log file is detailed dictionary-by-dictionary below, and the relevant outputs are explained.

8.1.1 vehicle summary

```
# ----------------------------------------------------

# Mission and Vehicle Log

# ----------------------------------------------------

vehicle:

   aircraftID:              2

   take_off_mass:           1734.67 # [kg]

   power_installed:         963.0 # [kW]

   Cruise_LbyD:             11.80 #
```

The **vehicle** dictionary contains four high-level vehicle metrics. The **aircraftID** is an outdated output, that is a copy of the input value given in **input.yaml**. The other metrics given in this dictionary are the take-off mass in kg, sum of maximum rated motor power values in kiloWatts and the vehicle lift-to-drag ratio in the cruise sizing segment.

8.1.2 Rotor

The **Rotor** dictionary provides details of the rotor geometry and performance in the hover sizing segment. Each sub-dictionary under the rotor dictionary corresponds to a unique rotor group. For each rotor group, the following outputs are specified:

1. **nrotor**: number of rotors of this type in the vehicle
2. **nblade**: number of blades per rotor in this rotor type
3. **disk_loading**: rotor hover disk loading in pounds per square foot
4. **aspect_ratio**: blade aspect ratio (radius to chord ratio)
5. **radius**: rotor radius in meters
6. **chord**: mean rotor blade chord in meters
7. **tip_speed**: rotor hover tip speed in m/s
8. **hover_FM**: rotor figure of merit in the hover sizing segment
9. **cruise_rpm_ratio**: rotor cruise RPM divided by rotor hover RPM
10. **eta_xmsn**: transmission efficiency, 0 to 1

11. **solidity**: rotor geometric solidity

12. **cd0**: rotor blade airfoil drag coefficient at zero lift

13. **ipf**: rotor induced power factor in hover (non-ideal losses)

14. **hvr_dwld**: hover download divided by rotor lift share

15. **hvr_ct_sigma**: rotor blade loading in hover sizing segment

16. **prop_eta**: prop-rotor efficiency in cruise sizing segment

```
Rotor:

   set0:

      nrotor:            8

      nblade:            3

      disk_loading:      16.76 # [lb/ft2]

      aspect_ratio:      5.86

      radius:            0.93 # [m]

      chord:             0.158 # [m]

      tip_speed:         170.0 # [m/s]

      hover_FM:          0.750

      cruise_rpm_ratio:  0.500

      eta_xmsn:          1.000

      solidity:          0.16299

      cd0:               0.012

      ipf:               1.234

      hvr_dwld:          0.015
```

```
hvr_ct_sigma:      0.139

prop_eta:          0.850
```

8.1.3 Wings

This dictionary contains details of the wing geometry and operating condition in the cruise sizing segment. If multiple wing groups are present in the system, then details of each wing group are listed as sub-dictionaries. For each wing group, the following details are written:

1. **nwing**: the number of fixed wings of this type present in the vehicle.
2. **span**: the tip-to-tip dimension in meters
3. **chord**: the average wing chord in meters
4. **structure_wt**: the weight of the structure for one wing
5. **wires_wt**: the weight of wires running along one wing
6. **aspect_ratio**: the wing aspect ratio
7. **oswald**: the wing Oswald efficiency
8. **cd0**: the wing profile drag coefficient
9. **cl**: the operating lift coefficient for the cruise sizing segment
10. **lift_fraction**: lift fraction for this wing group in the cruise sizing segment
11. **rotors_per_wing**: the number of rotors mounted along a wing of this group

12. **rotor_group_id**: the group identifier for rotors mounted on this wing

```
Wings:
  set0:
    nwing:             1
    span:              11.086 # [m]
    chord:             2.217 # [m]
    structure_wt:      117.303 # [kg, each]
    wires_wt:          66.506 # [kg, each]
    aspect_ratio:      5.000
    oswald:            0.800
    cd0:               0.014
    cl:                0.400
    lift_fraction:     0.900
    rotors_per_wing:     6
    rotor_group_id:      0
  set1:
    nwing:             1
    span:              4.674 # [m]
      ... (pattern repeats)
      ...
```

8.1.4 Costs

The costs dictionary contains a breakdown of the vehicle acquisition cost and operating cost per flight hour, as well as the results of comparisons with a ground taxi. The individual sub-dictionaries are detailed below.

8.1.4.1 Acquisition cost breakdown

The first row **Frame_acquisition** is the acquisition cost of the aircraft in millions of currency. The following dictionary shows two values per line: the first value is the cost of the component in currency, and the second column shows the fraction of the component's acquisition cost as a percentage of the total acquisition cost.

```
Costs:
   Frame_acquisition: [0.779632] # [Millions of USD]
   acquisition_cost_breakdown:
      BRS            : [     14161.395 ,  1.816] # [USD, % acquisition cost]
      avionics       : [    109355.250 , 14.027] # [USD, % acquisition cost]
      final_assem_line: [    42876.657 ,  5.500] # [USD, % acquisition cost]
      fuselage       : [     47190.123 ,  6.053] # [USD, % acquisition cost]
      interiors      : [     45152.000 ,  5.791] # [USD, % acquisition cost]
      landing_gear   : [     20874.300 ,  2.677] # [USD, % acquisition cost]
      motors         : [    117455.191 , 15.065] # [USD, % acquisition cost]
      power_dist     : [     29854.461 ,  3.829] # [USD, % acquisition cost]
      rotor_blade    : [     54432.887 ,  6.982] # [USD, % acquisition cost]
```

```
    rotor_hub      : [     20923.226 ,  2.684] # [USD, % acquisition cost]

    sense_avoid    : [     94908.500 , 12.174] # [USD, % acquisition cost]

    testing        : [      6400.000 ,  0.821] # [USD, % acquisition cost]

    tilt_actuators : [     37893.206 ,  4.860] # [USD, % acquisition cost]

    wing_flaps     : [     29202.352 ,  3.746] # [USD, % acquisition cost]

    wing_structure : [    106832.064 , 13.703] # [USD, % acquisition cost]

    wires          : [      2120.209 ,  0.272] # [USD, % acquisition cost]
```

8.1.4.2 Annual operating costs

This dictionary details the breakdown of annual operating costs for maintaining airworthiness. The field **Fixed_operating_costs** shows the aggregated annual costs in currency. The **fixed cost breakdown** dictionary is organized in a manner similar to the previous section: the first column shows the cost in currency, and the second column shows the cost as a percentage of the annual cost.

```
Fixed_operating_costs: [142746.614190] # [USD]

fixed_cost_breakdown:

    depreciation   : [     77963.182   , 54.616] # [USD, % fixed cost]

    inspection     : [      7700.000   ,  5.394] # [USD, % fixed cost]

    insurance      : [     35083.432   , 24.577] # [USD, % fixed cost]

    liability      : [     22000.000   , 15.412] # [USD, % fixed cost]
```

8.1.4.3 Variable operating costs

This section details the costs that depend on the number of flight hours that the vehicle is operated for. The field **Variable_operating_costs** details the currency spent per flight hour in operating costs due to maintenance, energy usage and inspections/cleaning. The variable operating costs are broken down into the individual cost elements in the **variable_cost_breakdown** dictionary. The first number in the two-column rows is the cost in currency per flight hour, and the second number is the cost of that contributing element expressed as a percentage of the **Variable_operating_costs** field.

```
Variable_operating_costs: [290.210] # [USD/hr]
variable_cost_breakdown:
   battery_use    : [        51.327  , 17.686] # [USD/hr, % variable cost]
   electricity    : [        31.895  , 10.990] # [USD/hr, % variable cost]
   fixed_annual   : [        95.164  , 32.792] # [USD/hr, % variable cost]
   frame_overhaul  : [       37.350  , 12.870] # [USD/hr, % variable cost]
   landing_fees   : [        61.474  , 21.183] # [USD/hr, % variable cost]
   vpf_overhaul   : [        13.000  ,  4.480] # [USD/hr, % variable cost]
UAM_time:           [        62.589] # [minutes  ]
Taxi_time:          [        80.261] # [minutes  ]
UAM_trip_cost:      [       101.662] # [USD      ]
Taxi_trip_cost:     [        50.994] # [USD      ]
Time_value:         [         2.867] # [$/min saved]
```

The last five fields are miscellaneous data that compare the performance and cost of an eVTOL with a ground taxi. **UAM_time** is the time taken to travel from an airport gate to a vertiport, fly to another vertiport and finally travel the last leg by ground taxi (i.e., door-to-door time from airport to final destination through an "air taxi"). The field **Taxi_time** is the door-to-door time from airport to final destination using ground transport only. **UAM_trip_cost** is the total cost of operating a combined air taxi and ground taxi for the final leg, and the corresponding cost for the ground taxi-only option is **Taxi_time_cost**. The time value is defined as the price difference between the two transport options, divided by the time difference, or *time value*.

8.1.5 Weight breakdown

This dictionary details the breakdown of vehicle weights. Three main weight categories are listed: **battery** (and/or **fuel**), **payload** and **empty mass**. For rows with two columns, the first column is the component mass in kg, and the second column is the mass of the component expressed as a percentage of take-off mass. For rows with a single column, the number corresponds to a component's mass in kg.

For the **empty weight** dictionary, each components is listed under sub-dictionaries. If the empty weight group is a dictionary with additional breakdowns within the component, those details are also printed, along with a field called **total** - the accumulated mass of components within that group. These breakdowns are plotted in pie charts by the postprocessing script **piethon.py**.

```
Weights:
  empty_weight:
    total:          [ 952.1       ,  54.9] # [kg, %GTOW]
    alighting_gear :
      total        : [  60.5       ,   3.5] # [kg, % GTOW]
      fairing      : [  26.0] # [kg]
      structure    : [  34.5] # [kg]
    avionics       : [  79.2       ,   4.6] # [kg, %GTOW]
    common_equip   : [  24.0       ,   1.4] # [kg, %GTOW]
    emergency_sys  : [  40.4       ,   2.3] # [kg, %GTOW]
    empennage      :
      total        : [  23.9       ,   1.4] # [kg, % GTOW]
      vtail        : [  23.9] # [kg]
    fuselage       :
      total        : [  84.1       ,   4.8] # [kg, % GTOW]
      bulkhead     : [  12.4] # [kg]
      canopy       : [  12.2] # [kg]
      keel         : [  14.9] # [kg]
      skin         : [  44.6] # [kg]
    powerplant     :
      total        : [ 220.0       ,  12.7] # [kg, % GTOW]
      motor_group0 : [ 220.0] # [kg]
    rotor          :
      total        : [ 143.6       ,   8.3] # [kg, % GTOW]
```

```
        group0actuatr: [ 26.9] # [kg]
        group0blades : [ 73.7] # [kg]
        group0hub    : [  43.0] # [kg]
    wing          :
        total      : [ 171.9        ,   9.9] # [kg, % GTOW]
        actuators  : [  14.9] # [kg]
        mounts     : [  22.0] # [kg]
        structure  : [ 117.4] # [kg]
        tilters    : [  17.6] # [kg]
    wires         :
        total      : [ 104.4        ,   6.0] # [kg, % GTOW]
        power_wire : [  74.3] # [kg]
        signal_wire : [  30.1] # [kg]
battery:          [462.58        ,  26.7] # [kg]
payload:          [320.02        ,  18.4] # [kg]
```

8.1.6 Volumes

The volume output section is under development, and additional fields will be introduced in future versions of HYDRA. At present, the dictionary contains:

```
Volumes:
    fuselage_width:          1.00 # [m]
    passenger_count:            0 # [people]
    battery_volume:         0.439 # [cu.m]
```

This dictionary outputs the fuselage width (specified in **defaults.yaml**) in meters, number of passengers (specified in **input.yaml**) and the calculated battery volume in cubic meters.

8.1.7 Parasitic drag

This dictionary details the vehicle flat plate area (**flat_plate_area**) in square meters. If the (**Empirical** → **Aerodynamics** → **Body** → flat_plate_factor input in **defaults.yaml** is set to zero, the component drag build-up for the vehicle parasitic drag is also printed in a sub-dictionary, **flat_plate_breakdown**. The entries in this sub-dictionary are the parasitic drag of each component expressed as a percentage of the total flat plate area.

```
Parasitic_drag:
   flat_plate_area:   0.422 # [sq.m]
   flat_plate_breakdown:
      LG              : [ 41.0] # [%]
      base            : [  7.0] # [%]
      fus             : [ 18.9] # [%]
      prot            : [  9.1] # [%]
      spin            : [ 20.1] # [%]
      vt              : [  3.8] # [%]
```

8.1.8 Mission details

This dictionary provides relevant details for the vehicle performance during various mission segments. The field **total_energy** is the energy in kWh required to complete the mission, which has **nsegments** segments. The segment types are given in the list **flight_mode**, with the segment altitudes given in meters by the fields **start_alt** and **end_alt**. The variable **delta_temp_ISA** is the temperature offset at sea level for the sizing mission relative to the temperature in the International Standard Atmosphere model. The segment rate of climb is specified in feet per minute, while cruise speed is specified in knots. Segment duration is specified by the field **time**, and **rotor power required** for each segment is listed in kilowatts. The corresponding **battery power draw** is also listed in kW, and the battery **C-rating** is listed in 1/hr. The **cell temperature** (if the model is used in sizing) is shown at the end of each segment in deg C. Ambient **density** is specified in kg/cu.m. The vehicle mass at the beginning of each segment is listed in **segment_mass**. The variable **segment_type** is a character string; 'all' indicates the segment is used for sizing and regular operations for operating cost evaluation; 'reserve' indicates the segment is used for sizing only, but not for regular mission operating cost. The ground distance covered by the vehicle in kilometers is output in the field **segment_distance**.

```
Mission:

   total_energy:       67.07 # [kW-hr]

   nsegments:          4
```

```
flight_mode:        ['hover  ','cruise ','cruise ','hover  ']
start_alt:          [       0 ,       0 ,       0 ,       0 ] #[m]
end_alt:            [       0 ,       0 ,       0 ,       0 ] #[m]
delta_temp_ISA:     [    0.00 ,    0.00 ,    0.00 ,    0.00 ] #[C]
rate_of_climb:      [       0 ,       0 ,       0 ,       0] #[ft/min]
cruise_speed:       [    0.00 ,   98.00 ,   98.00 ,    0.00 ] #[knots]
time:               [    1.50 ,   16.53 ,    4.96 ,    1.50 ] #[min]
rotor_power_reqd:   [  486.76 ,  123.49 ,  123.49 ,  486.76 ] #[kW]
battery_power_draw: [  540.84 ,  145.28 ,  145.28 ,  540.84 ] #[kW]
C-rating:           [    4.96 ,    1.33 ,    1.33 ,    4.96 ] #[1/hr]
cell temperature:   [    0.00 ,    0.00 ,    0.00 ,    0.00 ] #[deg C]
density:            [  1.226 ,   1.226 ,   1.226 ,   1.226] #[kg/cu.m]
segment_mass:       [  2000.9 ,  2000.9 ,  2000.9 ,  2000.9 ] #[kg]
segment_type:       ['all    ','all    ','reserve ','all    ']
segment_distance:   [     0.00,    50.00,    15.00,     0.00]
```

8.1.9 Battery details

This dictionary is used to output the details pertaining to the battery. The **rated_capacity** is the rated energy in kWh of all vehicle battery packs. The **state_of_health** is the multiplier for the battery pack rated energy; the product of these two numbers is the maximum energy that the pack can store at the pack's end-of-life after several charge/discharge cycles. The parameter **depth_discharge** is a fraction ranging from 0 to 1, which indicates the smallest energy fraction that the battery can be discharged to, without damaging the cells. In this example, the battery rated capacity before cycling is 109.07 kWh, and the design state of health is 0.8, i.e., at end of life, the maximum energy that can be stored in the battery is $0.8 \times 109.07 = 87.25$ kWh. The minimum depth of discharge is 0.075, i.e., the battery can be discharged until $0.075 \times 109.07 = 8.18$ kWh of energy is remaining, without permanently damaging the cells.

```
Battery:
   rated_capacity:    109.07
   state_of_health:   0.800
   depth_discharge:   0.075
```

8.2 log<XYZ>.txt

The other output generated by `HYDRA` is the final converged design and its details stored as a **pickle** module. The various postprocessing scripts in `HYDRA` load the vehicle class in its converged state for various sensitivity studies and to extract details. To load the file corresponding to design ID = 0, use this template:

```
import pickle
fname = 'outputs/log/log0.yaml'
with open(fname,'rb') as f:
    design = pickle.load(f)
```

The variable **design** is an instance of the **hydraInterface** class with all methods and variables required to perform sizing.

8.3 summary.dat

A high-level summary of all designs is provided in **summary.dat**. The first column is the unique design identifier; the second column is the take-off mass in kg, the third column is the installed power in kW, the fourth column is the mass sum of configuration fuel and battery (kg); the fifth column is the empty mass in kg; the sixth column is the payload mass in kg; the seventh column is the vehicle operating cost in currency per flight hour; the final column is an integer - 0 indicates the design is invalid, while 1 indicates the design satisfies all the constraints.

```
 Design ID #    Weight (kg)    Power (kW) Fuel/Batt (kg) Empty Wt (kg)  Payload (kg) Op Cost ($/hr) Valid design
           0    1734.67        963.05         462.579      952.07          320.02        290.21050        1
           1    1872.70        1171.23        525.369      1027.33         320.00        305.44722        1
           2    2049.93        1411.57        609.178      1120.75         320.00        325.62067        1
           3    1607.40        810.49         399.572      887.79          320.04        275.44891        1
           4    1702.75        967.71         441.230      941.50          320.02        285.65621        1
           5    1826.42        1143.83        497.869      1008.55         320.00        299.42785        1
 ...
```

9 Running HYDRA

1. Create a run directory under **cases/**

2. Create input files **input.yaml**, **defaults.yaml**. If multiple input values of each design parameter are specified, the analysis generates all possible combinations of all specified design parameters, and launches multiple sizing operations.

3. Run sizing on a parametric sweep: copy **xrun.py** from a sample run directory to the current run folder, then use the following command:

   python3 xrun.py (serial mode)

   mpirun -n 8 python3 xrun.py (parallel mode, 8 threads)

   The various cases are automatically subdivided by HYDRA , analyzed and the summaries are collated in the summary.dat output file.

4. Sort valid designs: use the routine process_data.py routine:

   python3 process_data.py

   This command will generate a file called **best_design.dat** in the same format as summary.dat, with valid designs ranked in order of ascending cost, take-off weight, installed power, or empty weight depending on the user-specified choice in process_data.py.

10 Postprocessing

Several postprocessing scripts are provided in the **Postprocessing/** folder in the sample run directories. Copy these scripts to the run directory with results to postprocess. Examples of these commands and sample outputs are shown below:

1. **Pie chart generator**: the script **piethon.py** generates pie charts for the breakdown of empty weight, annual operating costs, hourly operating costs and parasitic drag. Invoke it using the command

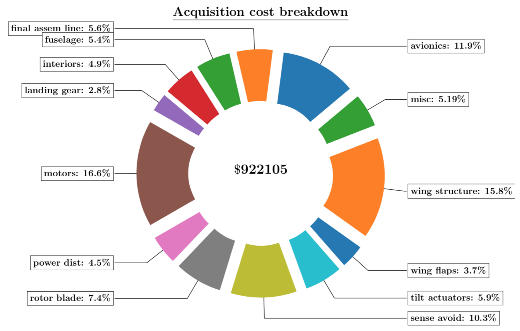
   python3 Postprocessing/piethon.py <XYZ>

   Here, <XYZ> is the integer unique identifier for a design. For example, the command **python3 Postprocessing/piethon.py 10** generates pie charts for a vehicle with the design ID 10. The relevant images are compiled into a PDF called **costs_design_10.pdf** for the sample input. The individual pie charts are shown in Fig. 23

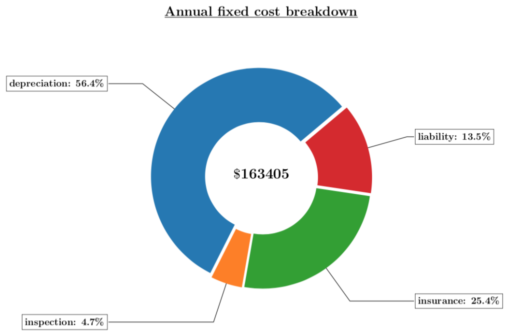
2. **Battery charge profile**: `HYDRA` features the ability to visualize the battery state of charge as a function of time along the mission profile. This postprocessor can be invoked with the command

   python3 Postprocessing/battery_draw.py 10

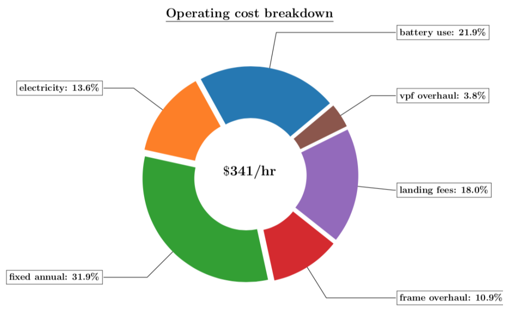
   Here, 10 is the unique design ID for the sized vehicle that needs to be postprocessed. The resulting plot is stored in a PDF called **battery_design_10.pdf**, and shown in Fig. 24. The first subplot shows the estimated vehicle range for



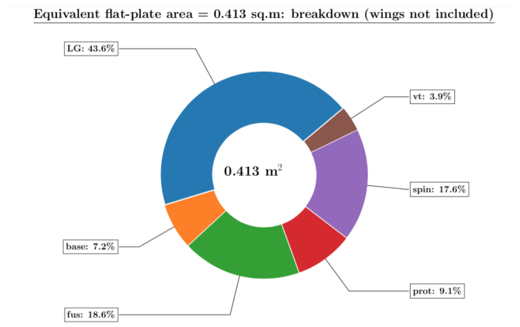
(a) Acquisition cost



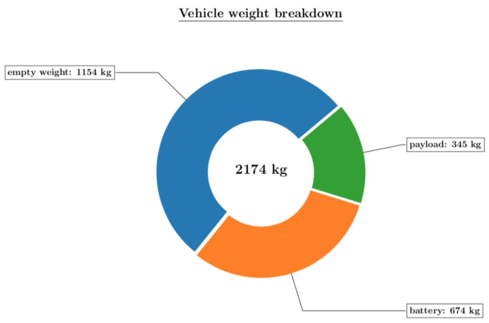
(b) Annual cost



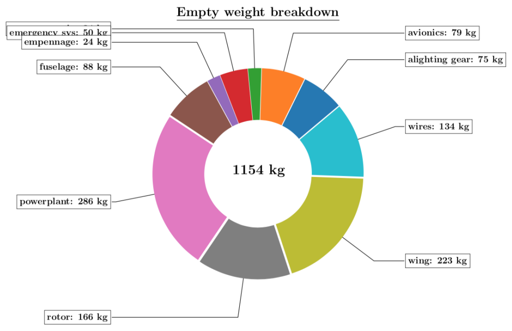
(c) Operating cost



(d) Parasitic drag breakdown



(e) Vehicle weight



(f) Empty weight

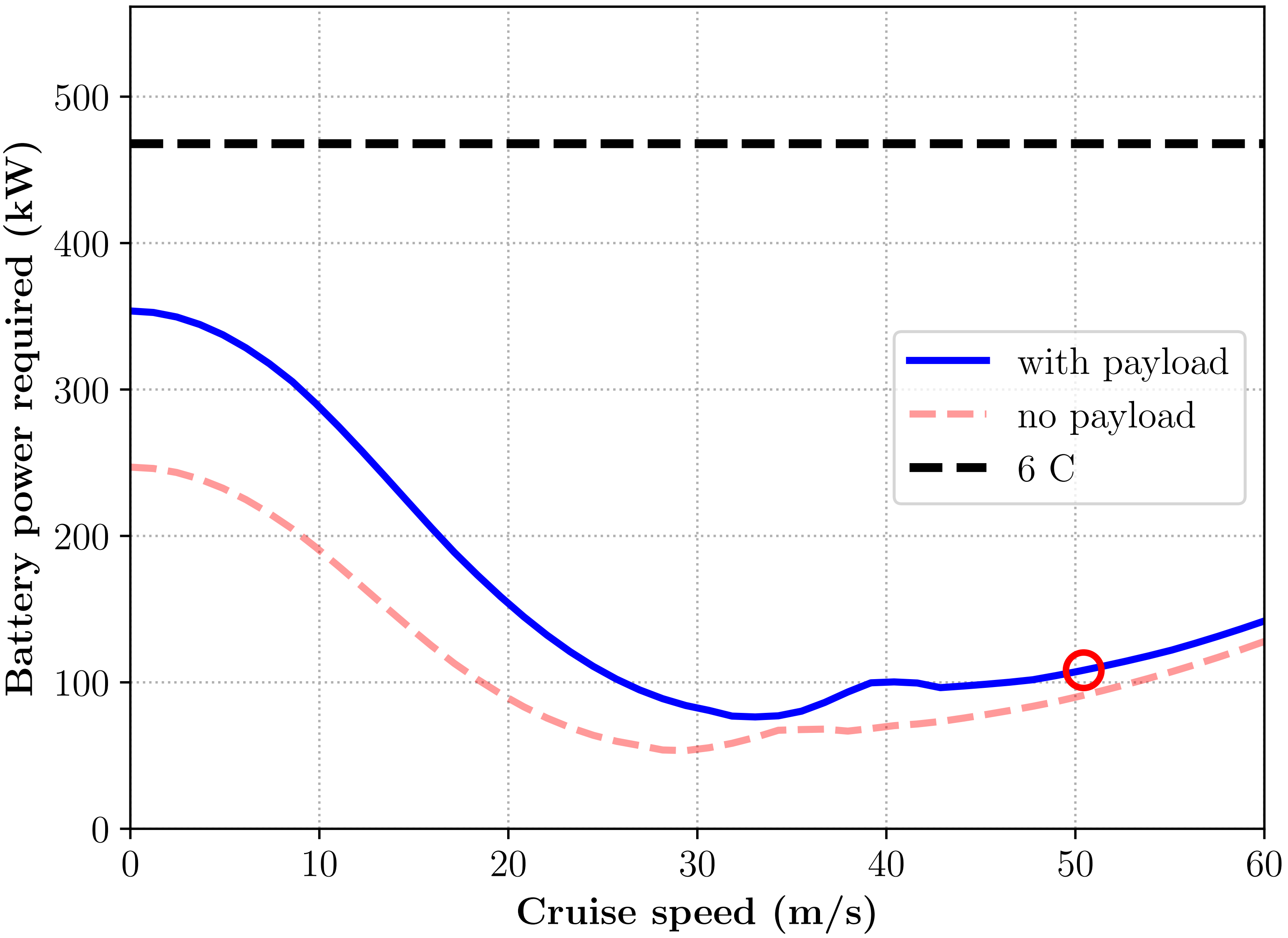
Figure 23: Pie charts generated by piethon.py postprocessor for a single design

a "perfect battery" starting the mission at its theoretical maximum state of charge. The second subplot shows the same "ideal battery" charge, along with the estimated battery state of charge used in HYDRA for sizing the vehicle.

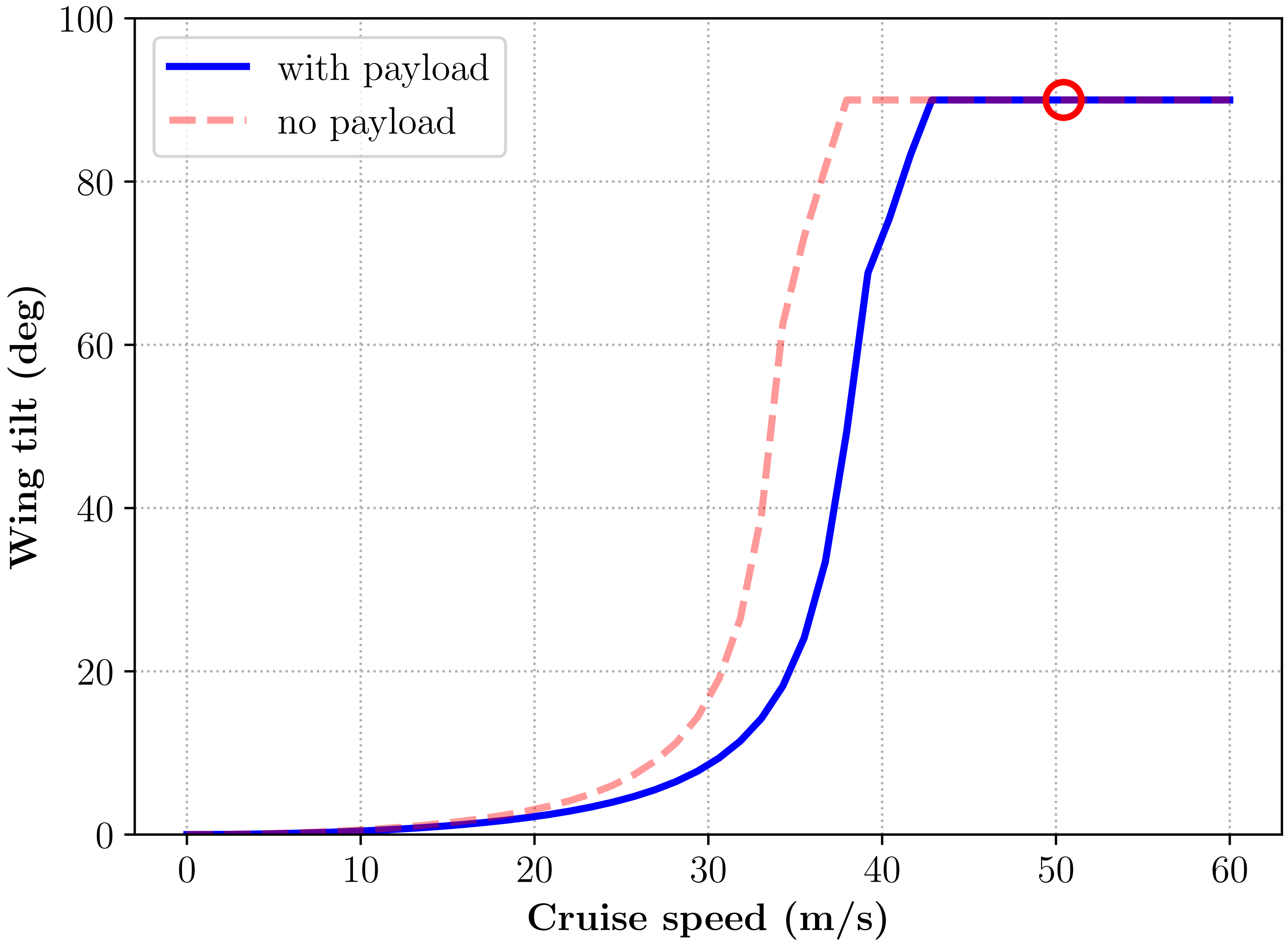
3. **Vehicle performance**: To automatically generate the (approximate) power curve, payload-range and payload-endurance trade offs, HYDRA features a performance postprocessor that can be invoked as follows:

   ```
   python3 Postprocessing/performance.py 28
   ```

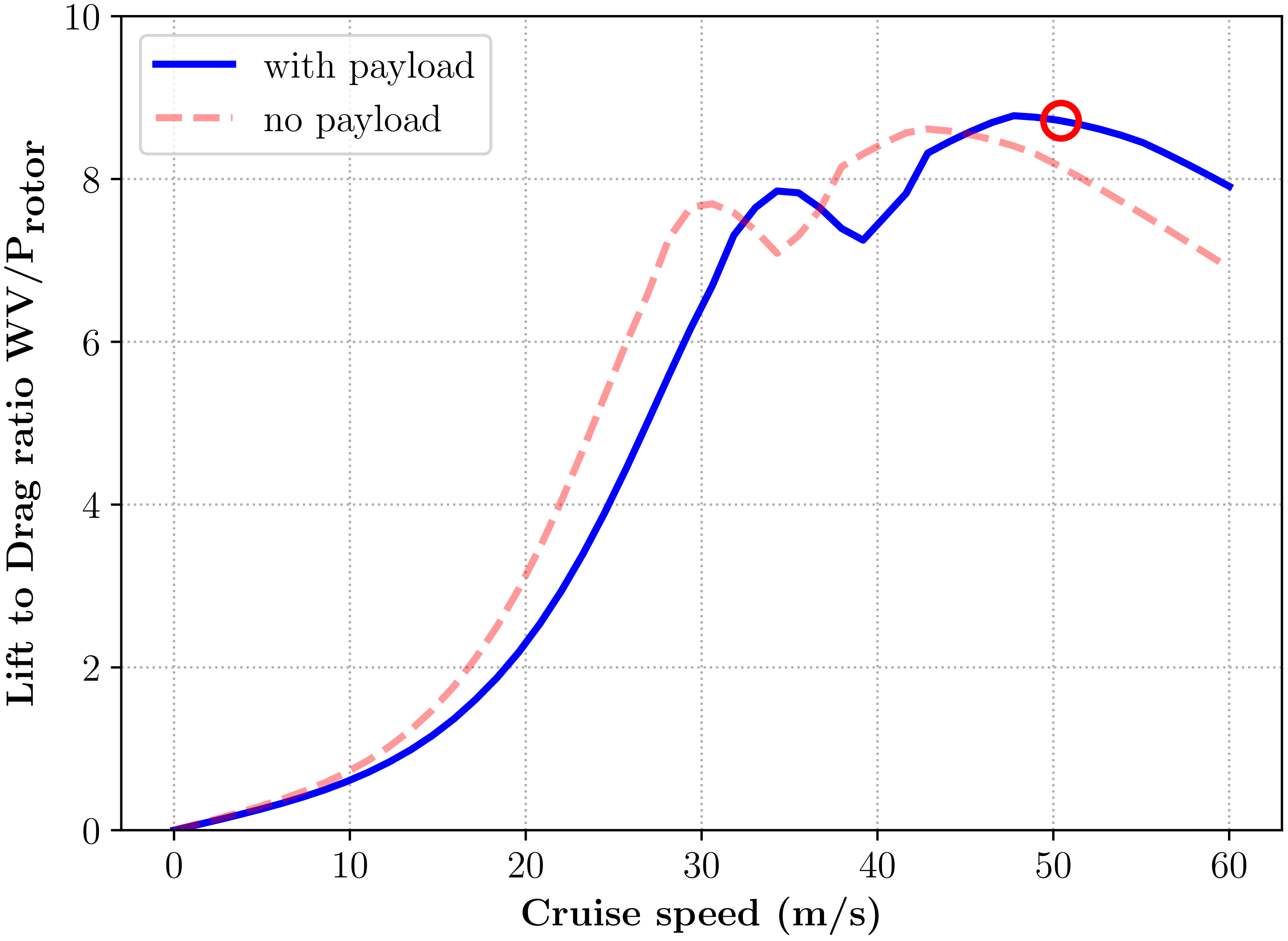
   Here, 28 is the unique design identifier that will be analyzed at various airspeeds and longitudinal trim will be performed. The outputs from the postprocessor are shown in the following pages.



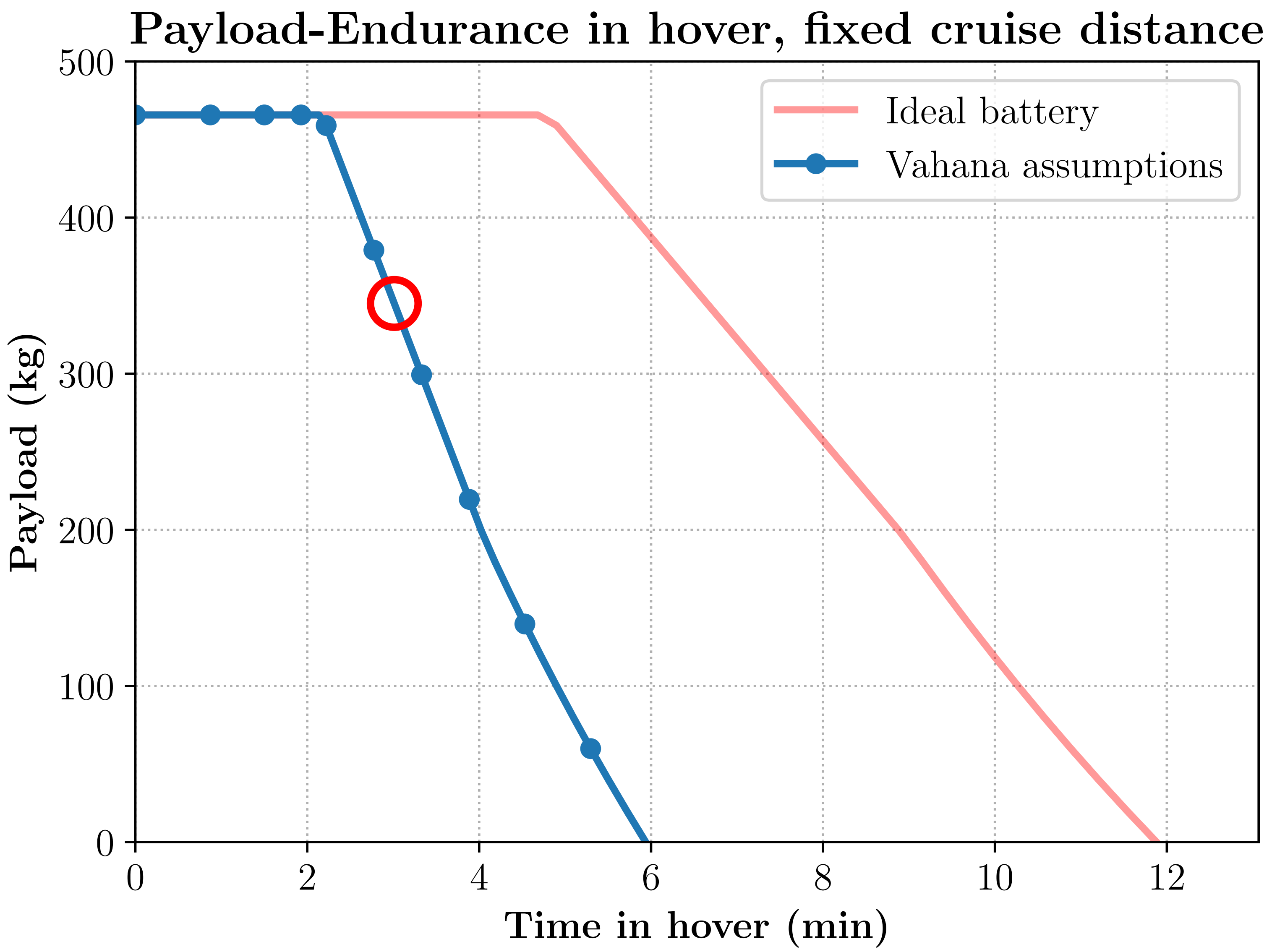
Battery power required (kW)
Cruise speed (m/s)
0
100
200
300
400
500
0
10
20
30
40
50
60
with payload
no payload
6 C

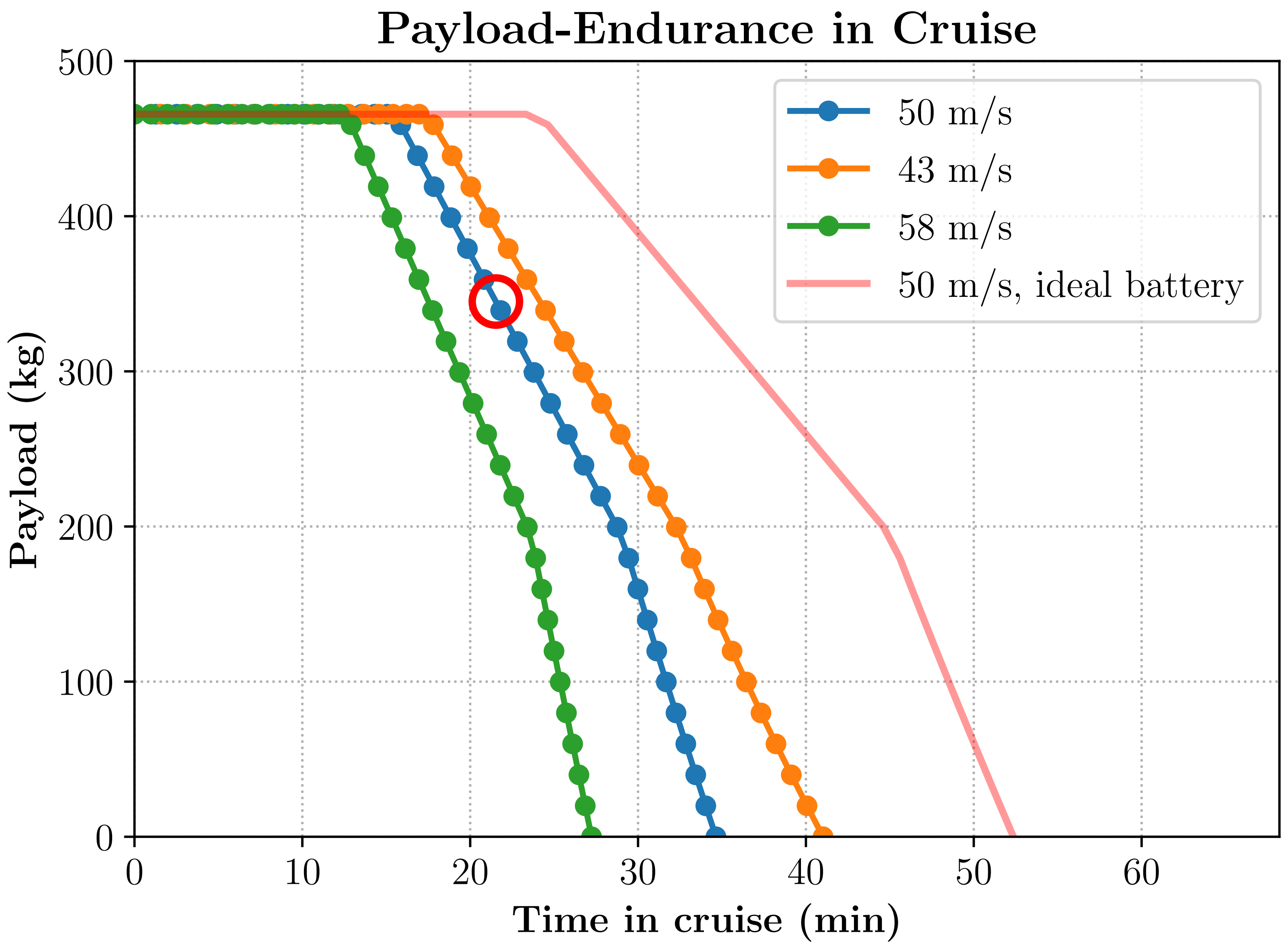


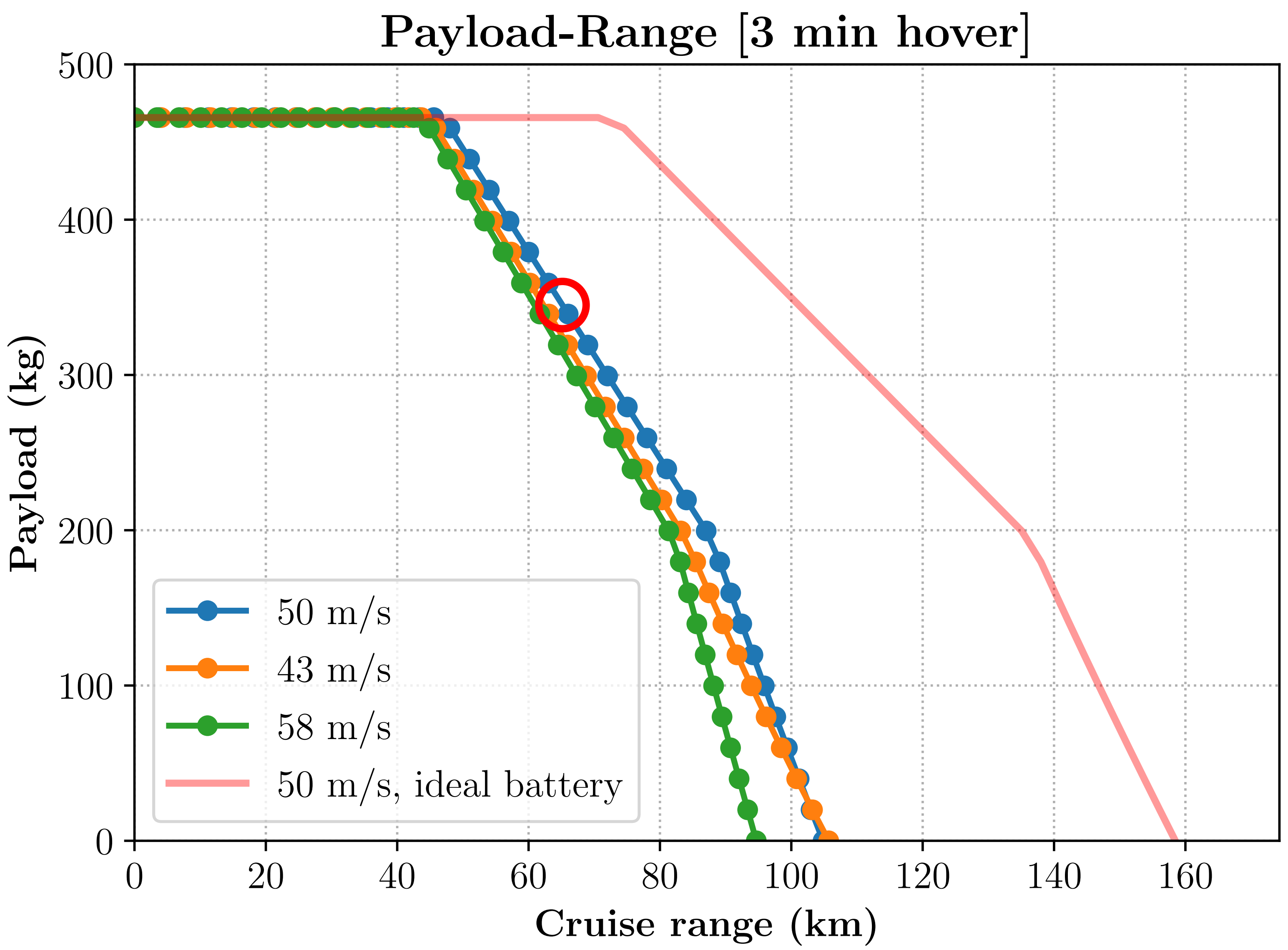
with payload
no payload
Wing tilt (deg)
Cruise speed (m/s)
0
20
40
60
80
100
0
10
20
30
40
50
60



10
8
6
4
2
0
0
10
20
30
40
50
60
Lift to Drag ratio WV/P_{rotor}
Cruise speed (m/s)
with payload
no payload


Payload-Endurance in hover, fixed cruise distance
Payload (kg)
Time in hover (min)
Ideal battery
Vahana assumptions
0
100
200
300
400
500
0
2
4
6
8
10
12


Payload-Endurance in Cruise
Payload (kg)
Time in cruise (min)
0
100
200
300
400
500
0
10
20
30
40
50
60
50 m/s
43 m/s
58 m/s
50 m/s, ideal battery


Payload-Range [3 min hover]
Payload (kg)
Cruise range (km)
500
400
300
200
100
0
0
20
40
60
80
100
120
140
160
50 m/s
43 m/s
58 m/s
50 m/s, ideal battery

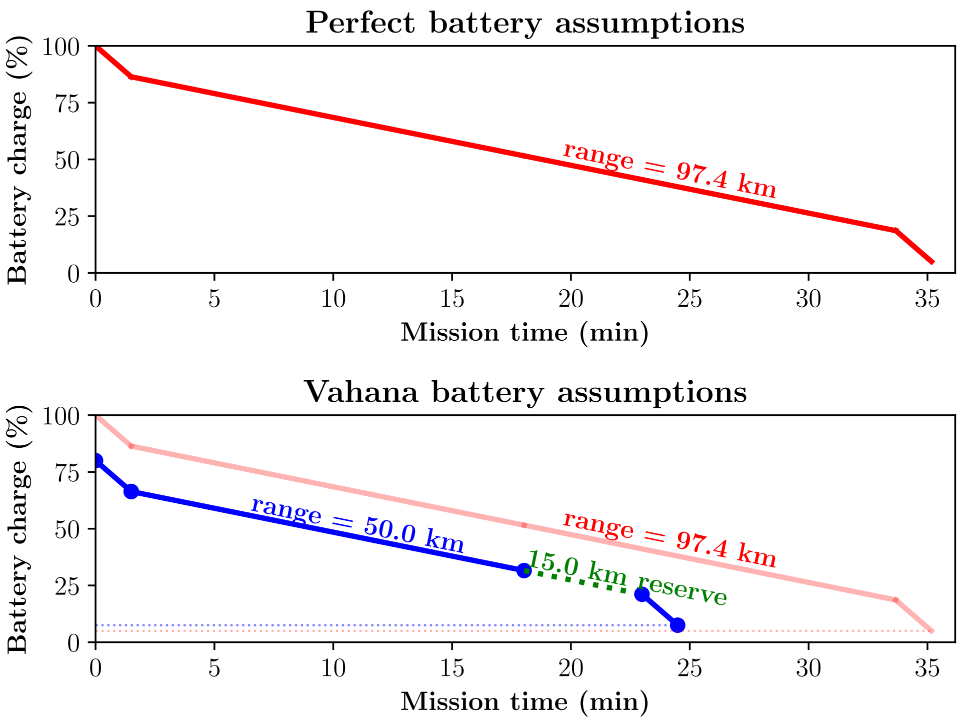
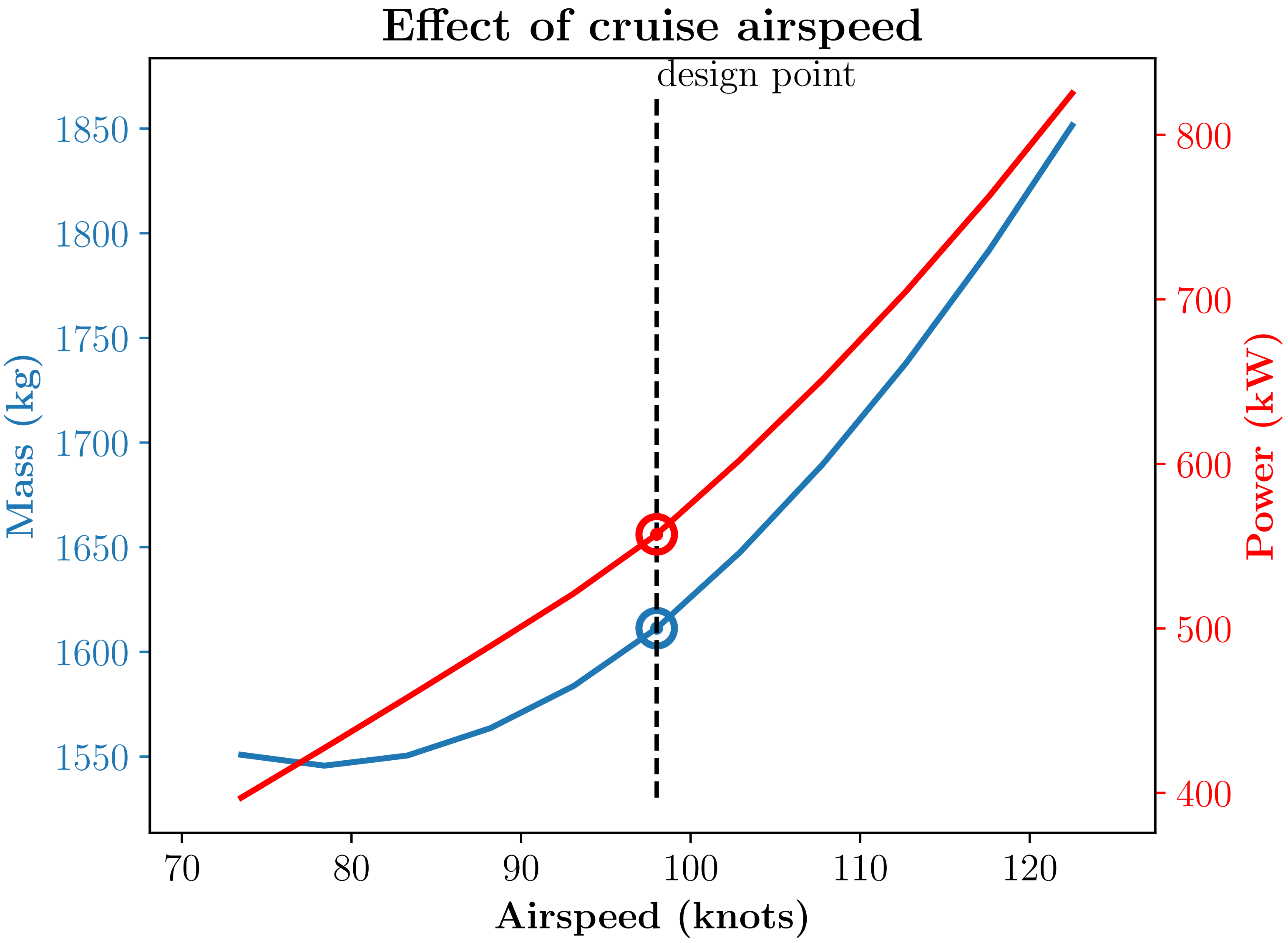


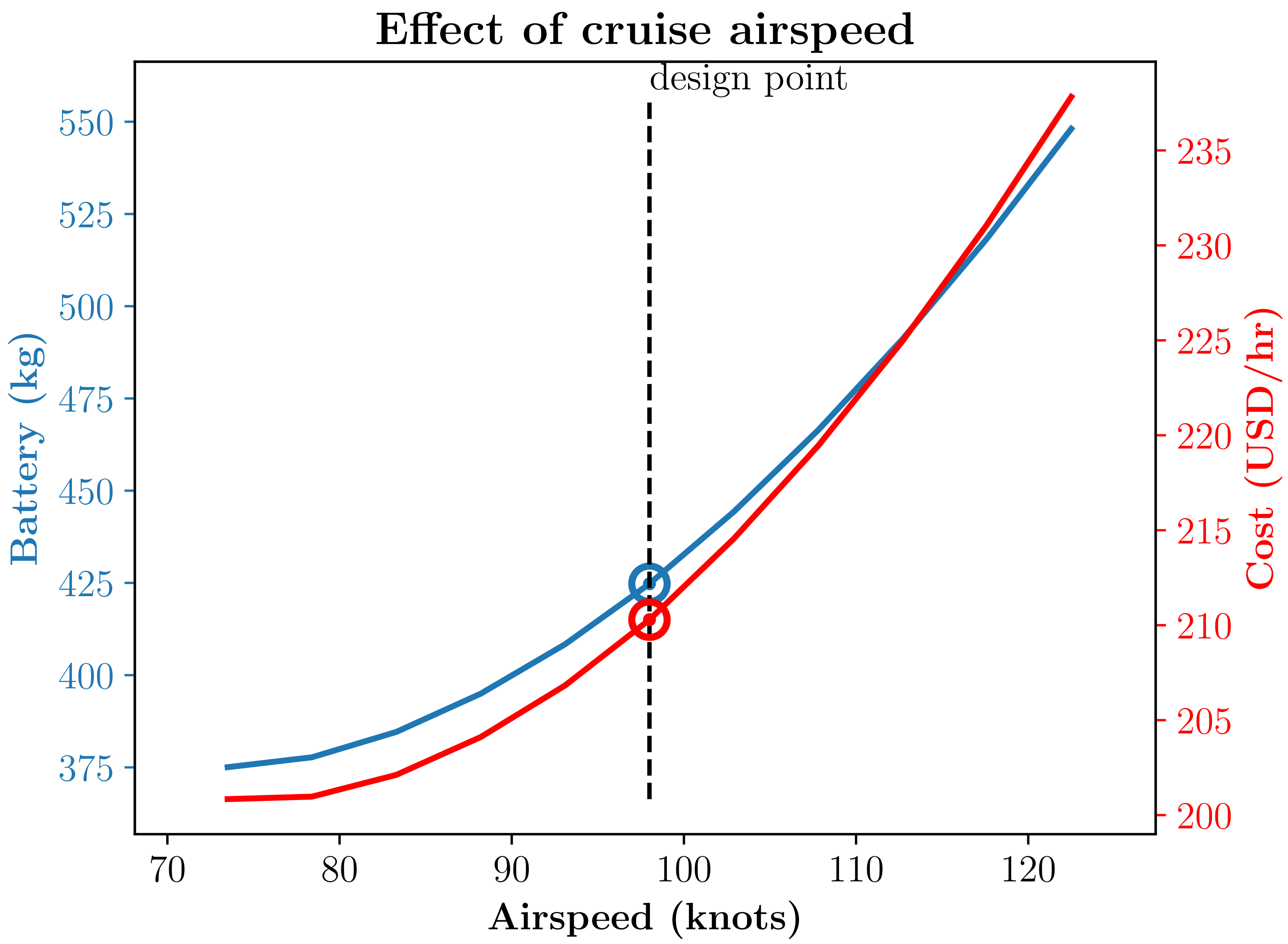
Figure 24: Battery state of charge along mission profile

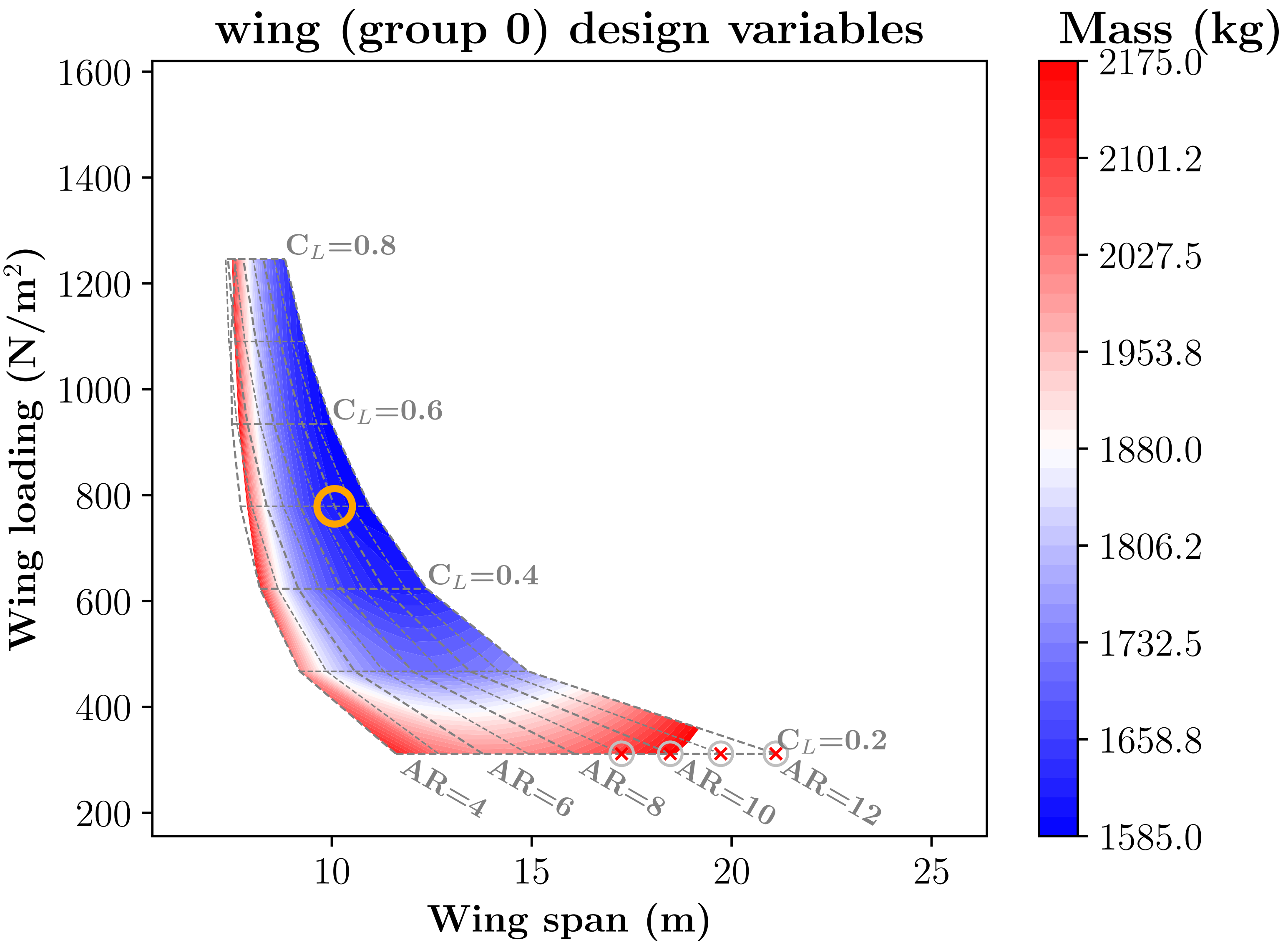
4. **Sensitivity analysis**: single and two-perturbation sensitivity studies can be launched using the following script:

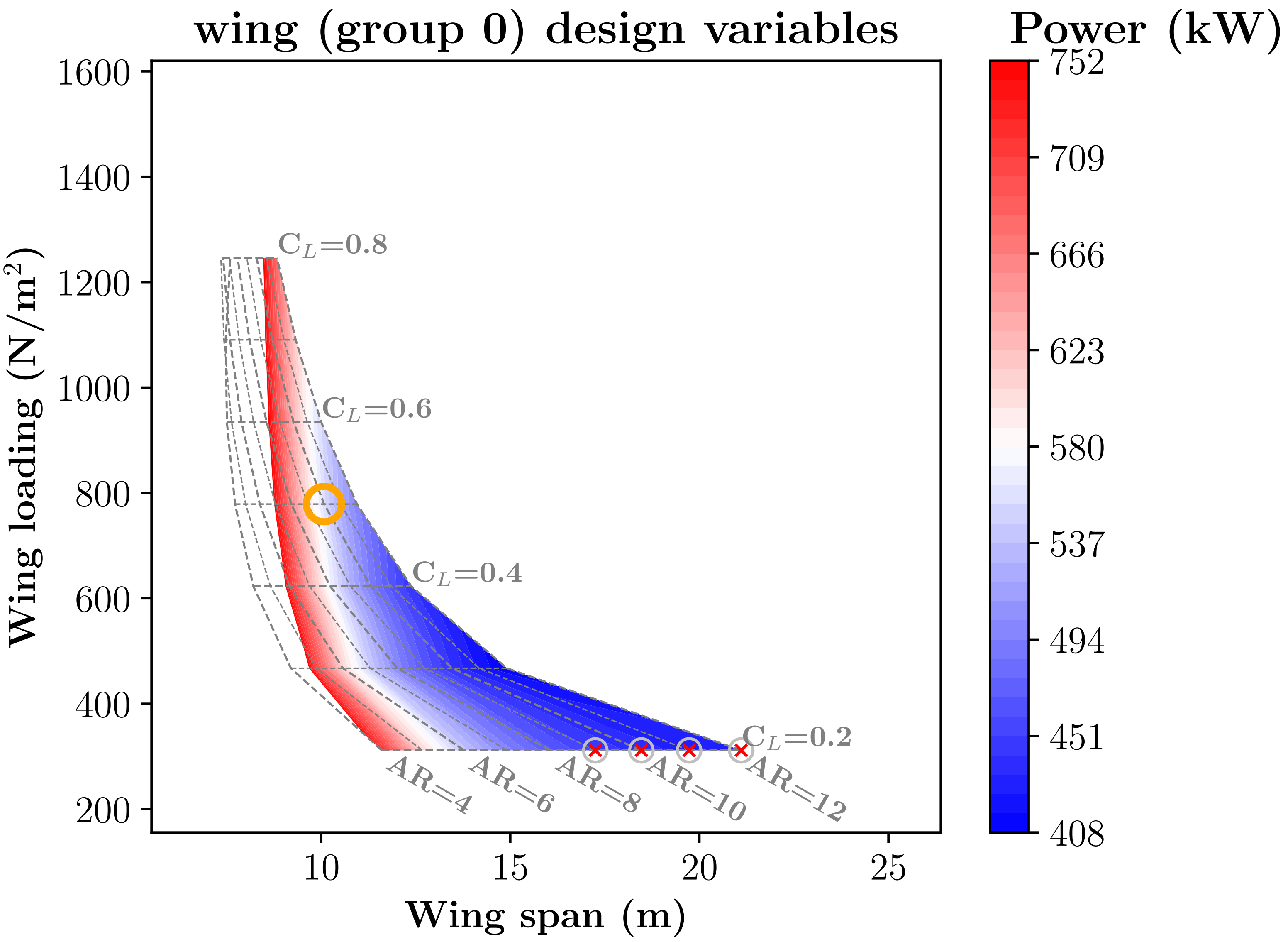
   ```
   python3 Postprocessing/sensitivity_wrapper.py 32
   ```

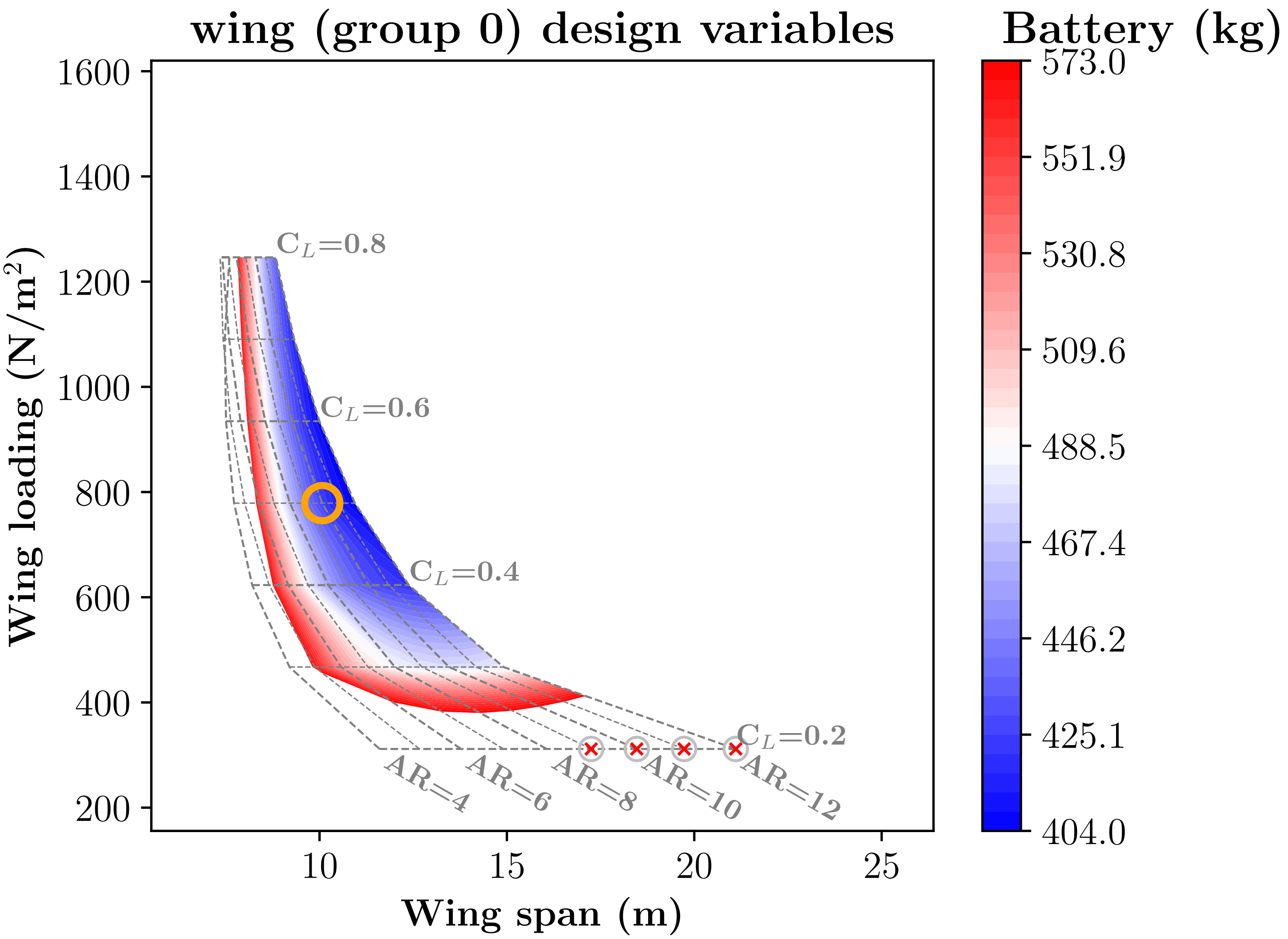
   Here, 32 is the unique identifier for a converged vehicle design. The resulting plots from the sensitivity of the converged design to perturbations in target airspeed (for the same mission range) are shown in the first two plots. The next four plots show the effect of changing the cruise wing loading and aspect ratio on vehicle take-off mass, installed power, battery power and operating cost as contour plots, with carpet axes. The baseline design point is marked as a circle, and invalid designs are marked with a red cross (×) inside a gray circle ○. The final two plots show the effect of changing the rotor hover blade loading and hover tip speed on take-off mass, installed power, battery mass and operating cost. These results are stored in a PDF named sensitivities_design_32.pdf.

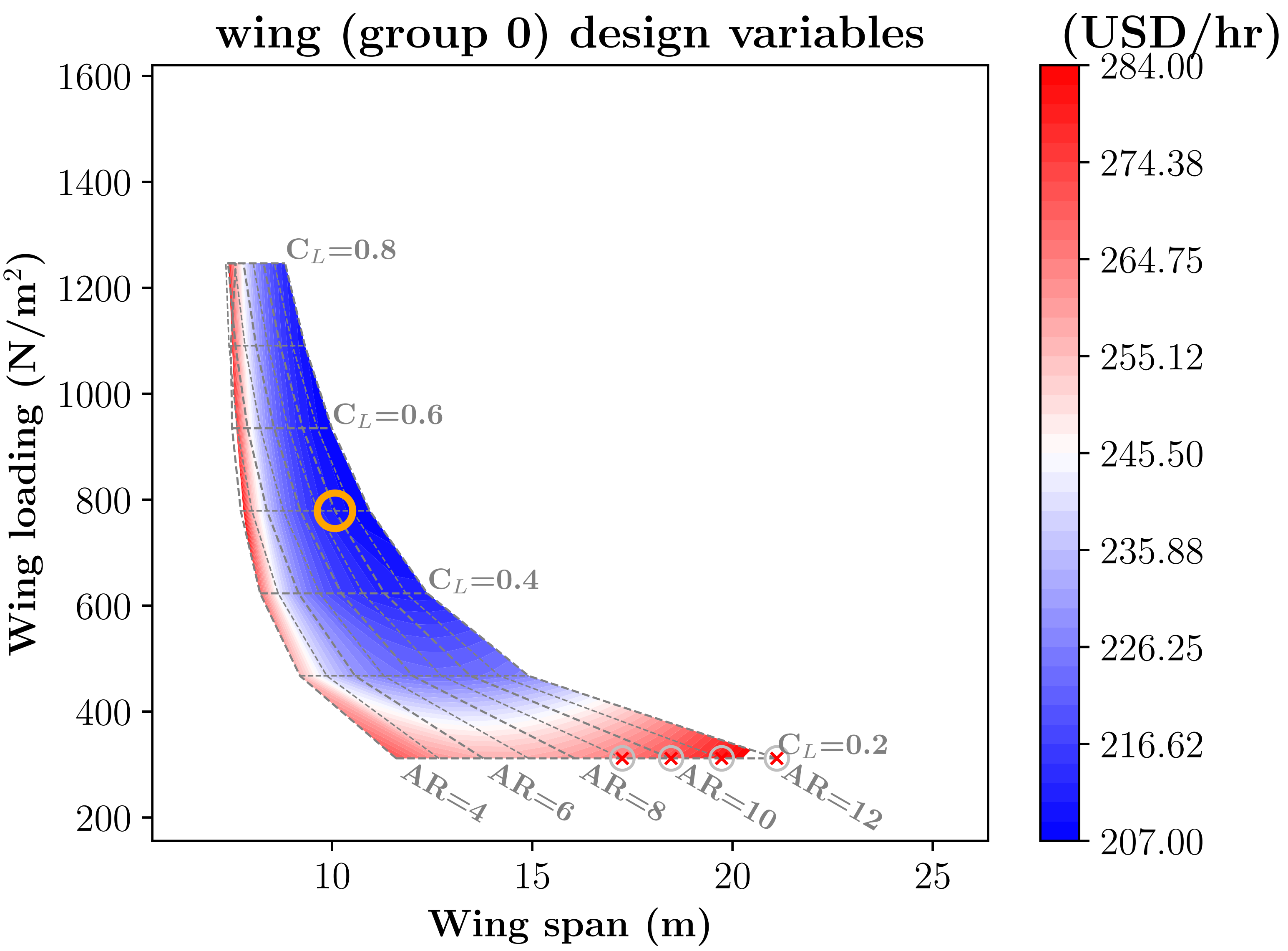

Effect of cruise airspeed
design point
Mass (kg)
Power (kW)
Airspeed (knots)
1550
1600
1650
1700
1750
1800
1850
400
500
600
700
800
70
80
90
100
110
120

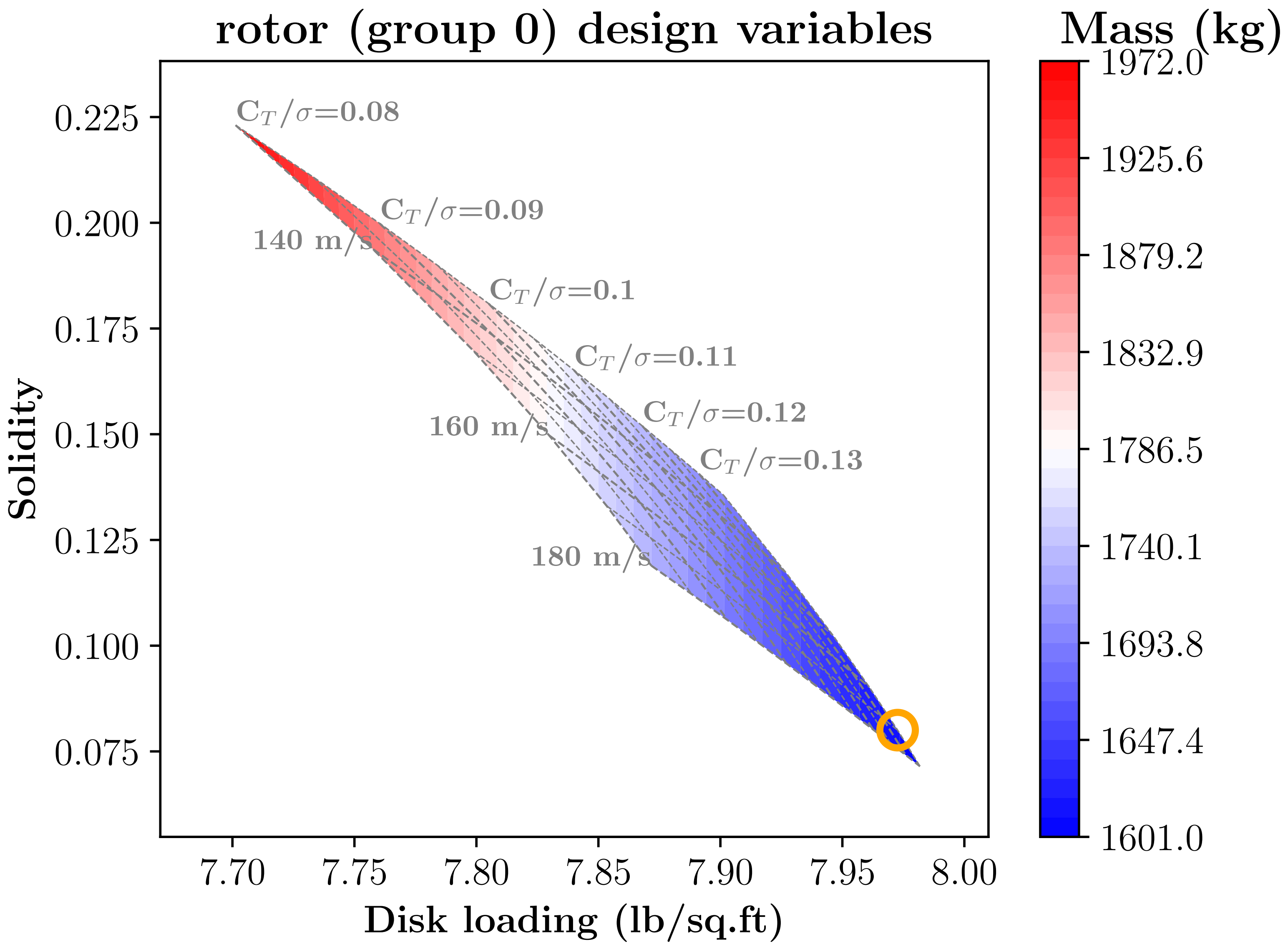

Effect of cruise airspeed
design point
Battery (kg)
Cost (USD/hr)
Airspeed (knots)
375
400
425
450
475
500
525
550
200
205
210
215
220
225
230
235
70
80
90
100
110
120

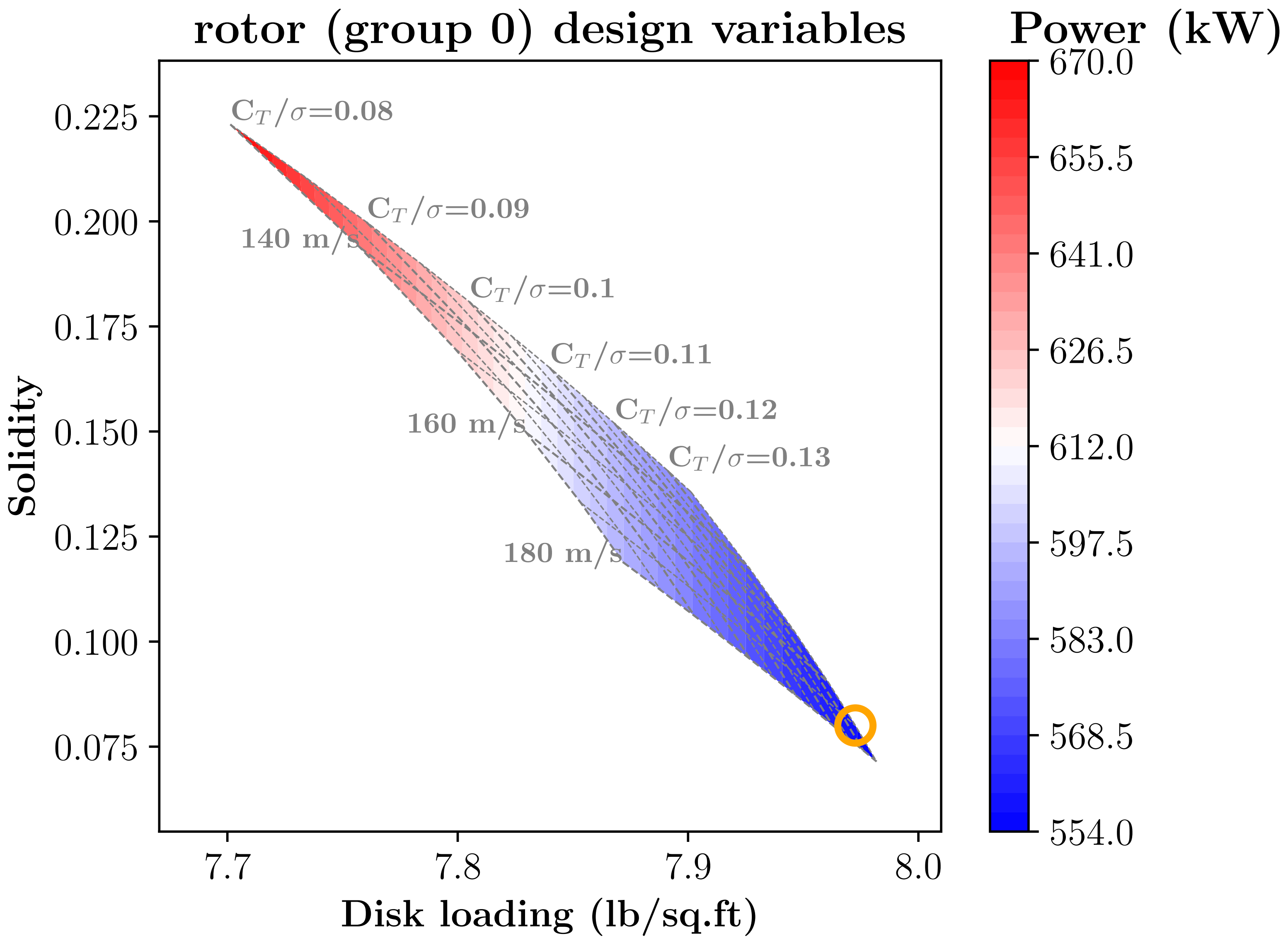

wing (group 0) design variables
Mass (kg)
Wing loading (N/m^2)
Wing span (m)
1600
1400
1200
1000
800
600
400
200
10
15
20
25
C$_L$=0.8
C$_L$=0.6
C$_L$=0.4
C$_L$=0.2
AR=4
AR=6
AR=8
AR=10
AR=12
2175.0
2101.2
2027.5
1953.8
1880.0
1806.2
1732.5
1658.8
1585.0

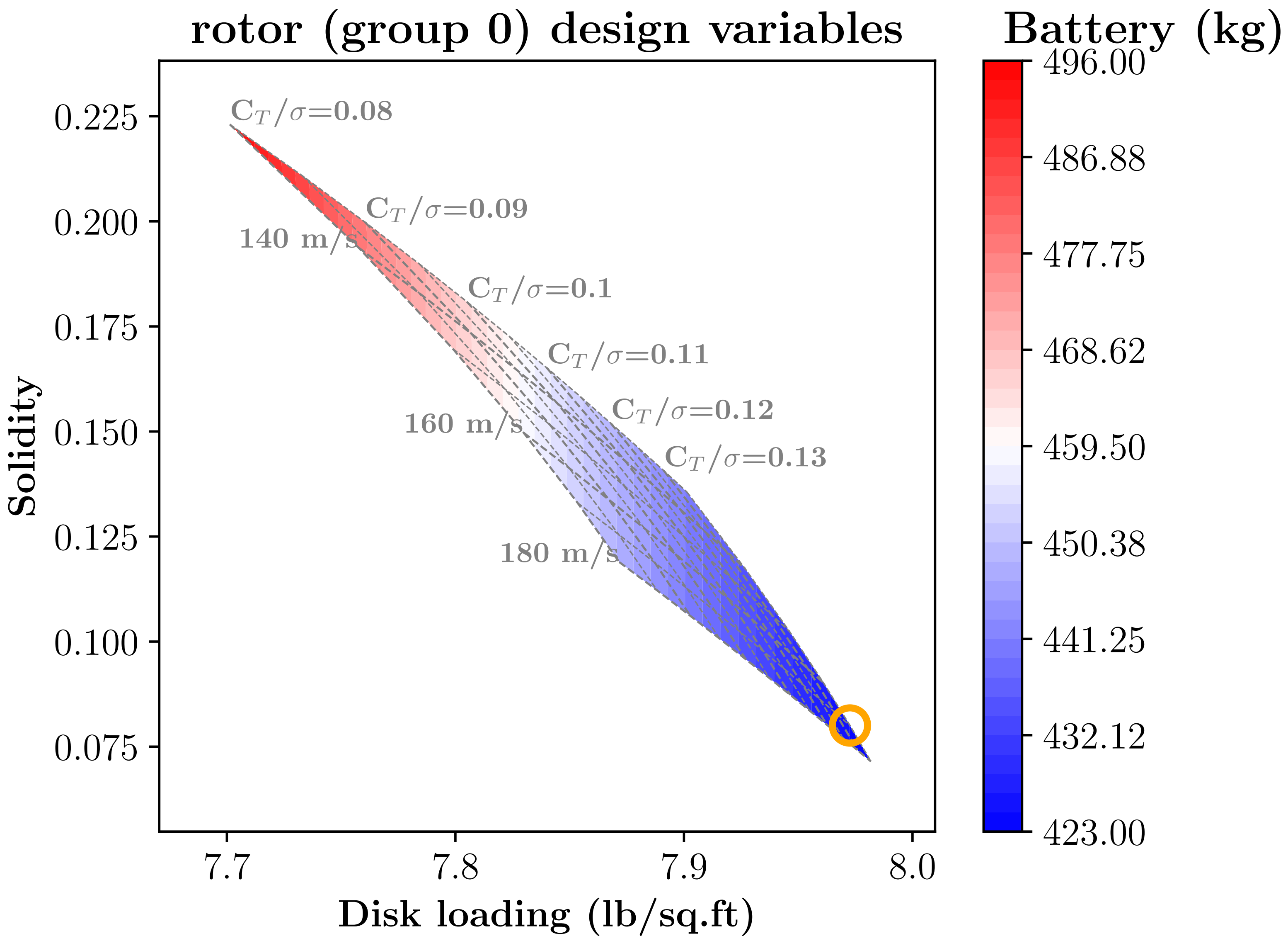

wing (group 0) design variables
Power (kW)
752
709
666
623
580
537
494
451
408
Wing loading (N/m^2)
Wing span (m)
1600
1400
1200
1000
800
600
400
200
10
15
20
25
C$_L$=0.8
C$_L$=0.6
C$_L$=0.4
C$_L$=0.2
AR=4
AR=6
AR=8
AR=10
AR=12

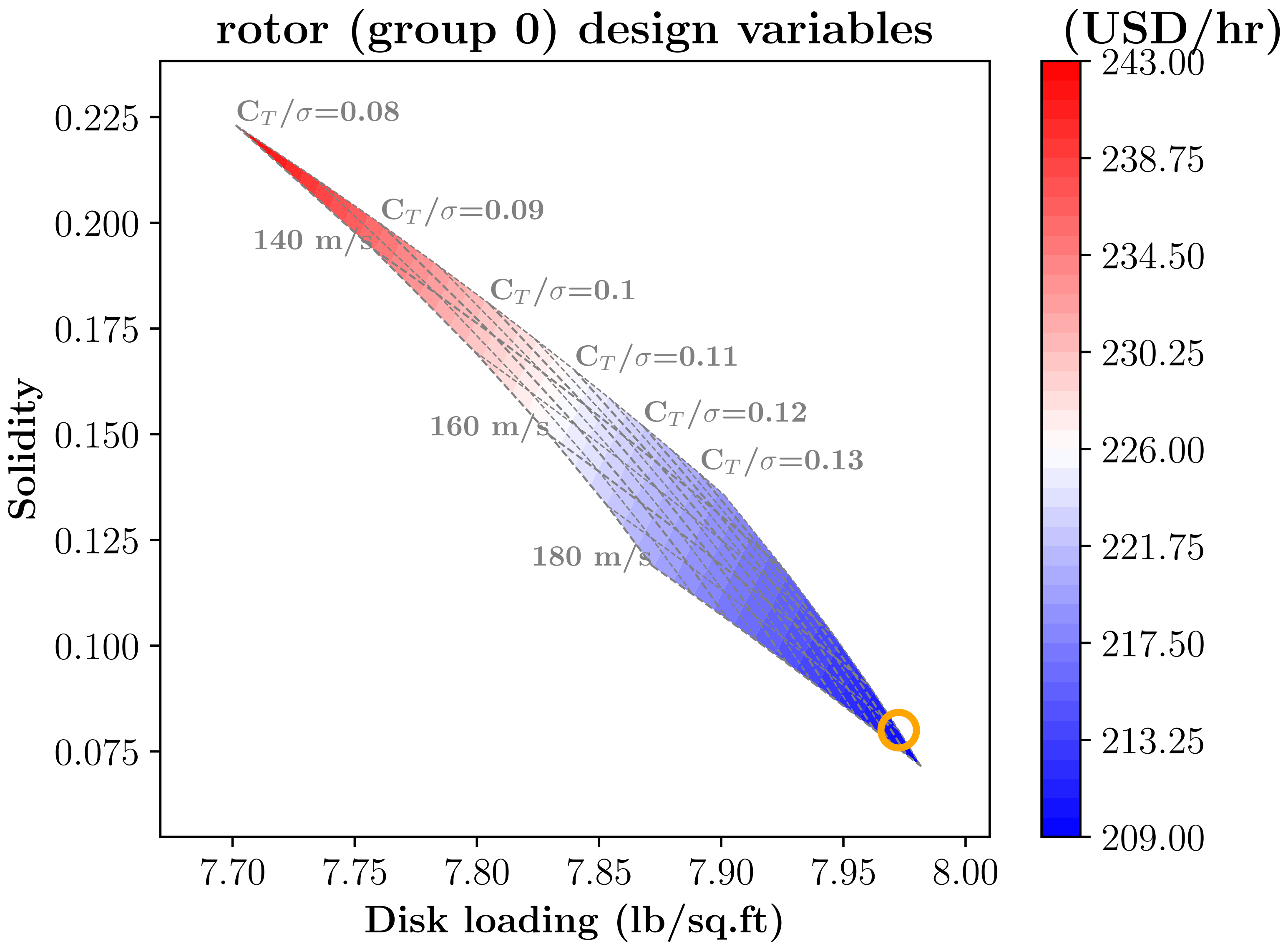

wing (group 0) design variables
Battery (kg)
Wing loading (N/m²)
Wing span (m)
C_L=0.8
C_L=0.6
C_L=0.4
C_L=0.2
AR=4
AR=6
AR=8
AR=10
AR=12
1600
1400
1200
1000
800
600
400
200
10
15
20
25
573.0
551.9
530.8
509.6
488.5
467.4
446.2
425.1
404.0


wing (group 0) design variables
(USD/hr)
Wing loading (N/m^2)
Wing span (m)
1600
1400
1200
1000
800
600
400
200
10
15
20
25
284.00
274.38
264.75
255.12
245.50
235.88
226.25
216.62
207.00
C$_L$=0.8
C$_L$=0.6
C$_L$=0.4
C$_L$=0.2
AR=4
AR=6
AR=8
AR=10
AR=12


rotor (group 0) design variables
Mass (kg)
1972.0
1925.6
1879.2
1832.9
1786.5
1740.1
1693.8
1647.4
1601.0
Solidity
0.225
0.200
0.175
0.150
0.125
0.100
0.075
Disk loading (lb/sq.ft)
7.70
7.75
7.80
7.85
7.90
7.95
8.00
C_T/σ=0.08
C_T/σ=0.09
C_T/σ=0.1
C_T/σ=0.11
C_T/σ=0.12
C_T/σ=0.13
140 m/s
160 m/s
180 m/s


rotor (group 0) design variables
Power (kW)
670.0
655.5
641.0
626.5
612.0
597.5
583.0
568.5
554.0
Solidity
0.225
0.200
0.175
0.150
0.125
0.100
0.075
Disk loading (lb/sq.ft)
7.7
7.8
7.9
8.0
C_T/σ=0.08
C_T/σ=0.09
C_T/σ=0.1
C_T/σ=0.11
C_T/σ=0.12
C_T/σ=0.13
140 m/s
160 m/s
180 m/s


rotor (group 0) design variables
Battery (kg)
496.00
486.88
477.75
468.62
459.50
450.38
441.25
432.12
423.00
C_T/σ=0.08
C_T/σ=0.09
C_T/σ=0.1
C_T/σ=0.11
C_T/σ=0.12
C_T/σ=0.13
140 m/s
160 m/s
180 m/s
Solidity
0.225
0.200
0.175
0.150
0.125
0.100
0.075
7.7
7.8
7.9
8.0
Disk loading (lb/sq.ft)


rotor (group 0) design variables
(USD/hr)
Solidity
Disk loading (lb/sq.ft)
C_T/σ=0.08
C_T/σ=0.09
C_T/σ=0.1
C_T/σ=0.11
C_T/σ=0.12
C_T/σ=0.13
140 m/s
160 m/s
180 m/s
0.225
0.200
0.175
0.150
0.125
0.100
0.075
7.70
7.75
7.80
7.85
7.90
7.95
8.00
243.00
238.75
234.50
230.25
226.00
221.75
217.50
213.25
209.00

5. **Optimization**: After running **xrun.py**, use the following commands to invoke sizing optimization:

   python3 process_data.py #⇒ filter out invalid designs, rank

   python3 Postprocessing/optimize_driver.py # serial mode

   mpirun -n 8 python3 Postprocessing/optimize_driver.py # parallel

   This command launches two types of optimization: (a) gradient-based optimization (several threads), with several initial conditions based on valid designs identified by the **process_data.py** command, and (b) differential evolution (first thread), with a latin-hypercube initial sampling. The optimized designs (generated by both methods) are aggregated and checked for uniqueness and validity. These unique designs are written to the file **optim_summary.dat**, with the syntax identical to **summary.dat** and **best_design.dat**. Additionally, the file **optim_dvars_summary.dat** shows the values of the design variables that yield the optimum design. After optimization is complete, the outputs/log/ folder will feature files with the pattern **optim_log_<XYZ>.yaml** and **optim_log_<XYZ>.txt**, with patterns similar to the regular output files of the same format. The process_data.py command can be called with an additional argument 'optim' as follows:

   python3 process_data.py optim

   This command sorts the various optimized unique designs in **optim_summary.dat** from best to worst and writes the results in a file called **optim_ranked.dat**.

A Cantilever beam dynamics

Consider an eVTOL wing with multiple electric motors and rotors mounted along the spar, shown in Fig. 25. The wing structure is idealized as a cantilever beam with non-structural lumped masses placed along the span, shown in Fig. 26.

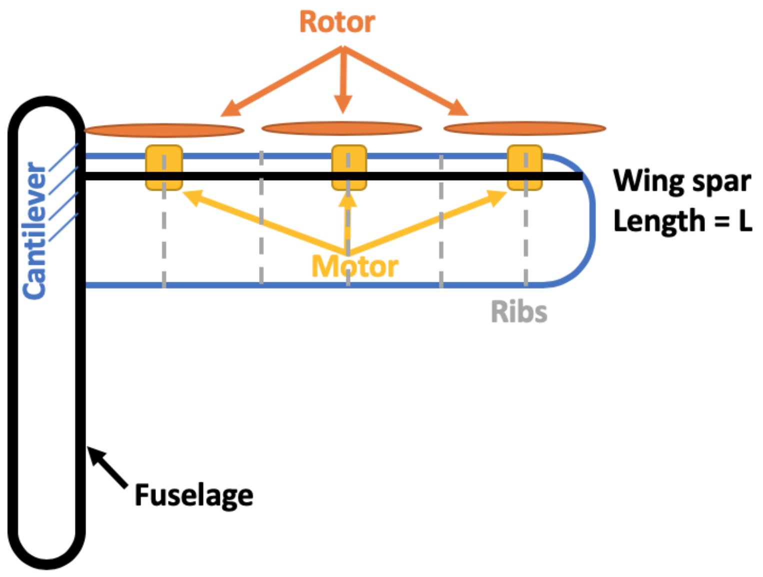


Figure 25: Rotor layout for an example eVTOL wing

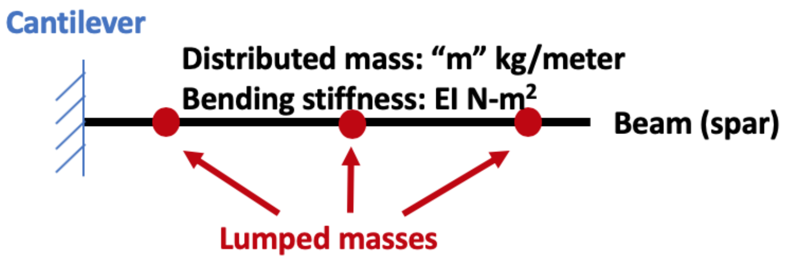
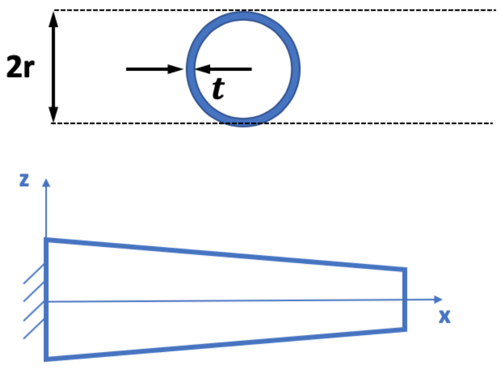


Figure 26: Idealization of eVTOL wing structure

To expand the generality of the analysis, consider a tapered circular spar running along the length of the wing. The cross-section of the spar is shown in



$$r(x) = r_1 + \frac{r_2 - r_1}{L} x$$

$$r(x) = r_1 + r' x$$

Figure 27: Linearly tapered circular spar

Fig. 27. The tube thickness is equal to t, and is uniform along the spar length. The radius of the circular cross-section varies from r_1 at the root of the beam to r_2 at the tip. The mass per unit span and bending stiffness of the spar is given by

$$\begin{aligned} m(x) &= 2\pi\rho r(x)t \\ EI_{yy}(x) &= E\pi r(x)^3 t \end{aligned}$$

The first bending natural frequency is estimated using a Rayleigh-Ritz approximation, with a mode shape set by the static deflection of the cantilever due to a tip-load, i.e.,

$$w''(x) = P(L - x) \tag{31}$$

Integrate twice along the span and apply the cantilever boundary condition to obtain the deflected bending slope as

$$w'(x) = \frac{P}{E\pi t}\left[\frac{a_2}{r(x)^2} + \frac{a_1}{r(x)} + a_0\right] \tag{32}$$

The constants are

$$\begin{aligned} a_2 &= -\frac{L}{2r'} - \frac{r_1}{2r'^2} \\ a_1 &= -\frac{1}{r'^2} \\ a_0 &= \frac{L}{2r'r_1^2} - \frac{1}{2r'^2 r_1} \end{aligned}$$

Integrate the bending slope along the span to obtain the deflected mode shape as

$$w(x) = \frac{P}{E\pi t}\left[b_0 + b_1 x + \frac{b_m}{r(x)} + b_l \log_e r(x)\right] \tag{33}$$

The constants are

$$\begin{aligned} b_0 &= -\frac{a_2}{r_1 r'} - \frac{a_1}{r'}\log_e(r_1) \\ b_1 &= a_0 \\ b_m &= -\frac{a_2}{r'} \\ b_l &= \frac{a_1}{r'} \end{aligned}$$

A.1 Potential energy

The maximum potential energy is given by

$$U = \frac{1}{2}\int_0^L \frac{P^2(L-x)^2}{EI_{yy}(x)}dx$$

Substitute for the bending stiffness to obtain

$$U = \frac{1}{2}\frac{P^2}{E\pi t}[A+B+C](\lambda)$$

The term λ is the spar taper ratio, given by

$$\lambda = \frac{r_2}{r_1} \tag{34}$$

The sum $A+B+C$ is well-represented by a cubic polynomial in λ as

$$A+B+C \quad \approx \quad (1.6261 \ - \ 0.4826\lambda \ + \ 2.1197\lambda^2 \ - \ 0.5957\lambda^3)\frac{1}{K^3}$$

The constant K is the inverse of the spar aspect ratio, given by

$$K \quad = \quad \frac{2(\frac{t}{c})}{AR_{\text{wing}}} \tag{35}$$

Here, $\frac{t}{c}$ is the wing thickness to chord ratio, and AR_{wing} is the wing aspect ratio.

A.2 Kinetic energy

The maximum kinetic energy of the beam has contributions from three types of masses:

1. **Structural masses**: This term deals with the kinetic energy of the spar structure. This contribution to kinetic energy is

$$KE_s \quad = \quad \int_0^L \frac{1}{2}\ 2\pi r(x)t\rho\ w(x)^2\ \omega_n^2\ dx \quad = \quad \frac{P^2}{E^2\pi t}\rho\omega_n^2(RHS)$$

   The term RHS is a function of the taper ratio λ, length L and spar taper ratio inverse K only. Further, the dependence on wing taper λ can be approximated using a cubic polynomial in λ. Thus, the kinetic energy coefficient RHS is approximated as

$$RHS \quad \approx \quad \frac{L^2}{K^5}(0.04823 \ + \ 0.269\lambda \ + \ 0.15178\lambda^2 \ + \ 0.36917\lambda^3)$$

2. **Non-structural distributed masses**: This term deals with the kinetic energy due to the skin and other components (e.g., wires) that do not contribute

significantly to bending stiffness. The kinetic energy term is

$$KE_{\text{NS}} \quad = \quad \int_0^L \frac{1}{2}\, m_{\text{NS}}(x)\; w(x)^2\; \omega_n^2\; dx$$

The non-structural mass due to wing skin is given by

$$m_{\text{NS}}(x) \quad = \quad \frac{2M_{\text{NS}}}{A(\frac{t}{c})} r(x)$$

The term $\frac{M_{\text{NS}}}{A}$ is the non-structural mass per unit plan-form area of the skin. Define an intermediate quantity β given by

$$\beta \quad = \quad \frac{M_{\text{NS}}}{A(\frac{t}{c})}$$

The expression for kinetic energy of non-structural masses simplifies to

$$KE_{\text{NS}} \quad = \quad \beta\, \frac{P^2\omega_n^2}{E^2\pi^2 t^2}\; RHS$$

3. **Non-structural lumped mass**: This term refers to the kinetic energy of the electric motor/speed controller unit, rotor hubs and rotor blades. These discrete lumped masses are mounted along the wing spar, and the kinetic energy of these masses is given by

$$KE_{\text{lump}} \quad = \quad \sum_i^{N_{\text{M}}} \frac{1}{2}\; M_i\; w(x_i)^2 \quad = \quad \frac{P^2\omega_n^2}{E^2\pi^2 t^2}\Delta RHS$$

The coefficient ΔRHS is

$$\Delta RHS \quad = \quad \sum_i^{N_{\text{M}}} \frac{M_i}{2}\left[b_0 \;+\; b_1 x_i \;+\; b_l \log_e r(x_i) \;+\; \frac{b_m}{r(x_i)}\right]^2$$

A.3 Frequency tuning

The goal is to determine the spar mass by setting a target first natural frequency ω_n for the cantilever beam with distributed and lumped non-structural masses. Equating the maximum total kinetic energy and maximum potential energy for the first bending mode, we obtain

$$\frac{1}{2}\frac{P^2}{E^2\pi t}\,LHS \quad = \quad \omega_n^2\left[\frac{P^2}{E^2 t}(RHS) \;+\; \frac{P^2}{E^2\pi^2 t^2}\beta(RHS) \;+\; \frac{P^2}{E^2\pi^2 t^2}\Delta RHS\right] \tag{36}$$

Solve for spar thickness as

$$t \quad = \quad \left(\frac{E\pi}{2\omega_n^2}LHS \;-\; \rho\pi RHS\right)^{-1}\left[\beta(RHS) \;+\; \Delta RHS\right]$$

The total spar mass is therefore

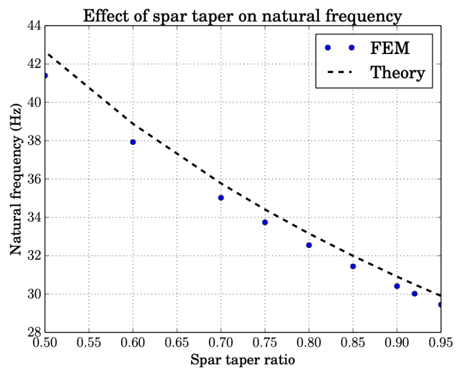
$$M_{\text{spar}} \quad = \quad \pi\rho\; t(1+\lambda)r_1 \tag{37}$$

For validation of the Rayleigh-Ritz solution, several cases were evaluated with and without lumped non-structural masses.

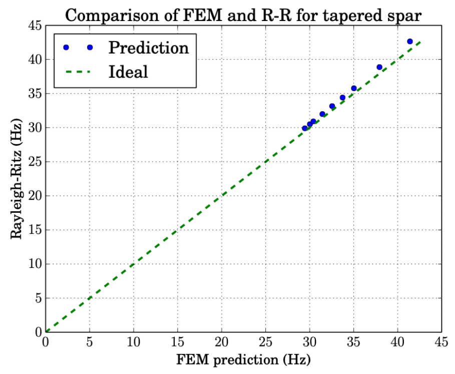
A.4 Validation

A.4.1 Case 1: cantilever beam

Consider a cantilever beam with a hollow circular cross-section of constant wall thickness, and the cross-section radius is linearly tapered from root to tip, with a taper ratio of λ varying from 0.5 to 1.2. The beam length is 2.9 m, mean tube radius is 0.1411 m. The first bending natural frequency predictions from the Rayleigh-Ritz approximation are compared against a beam Finite Element Method analysis. These two frequencies are plotted in Fig. 28. The Rayleigh-Ritz approximation exhibits about 2% over-prediction, but otherwise captures the trends accurately. In this example, no non-structural masses were considered.

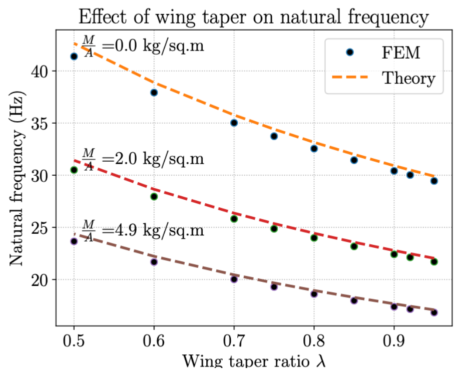


(a) Natural frequency vs. taper ratio

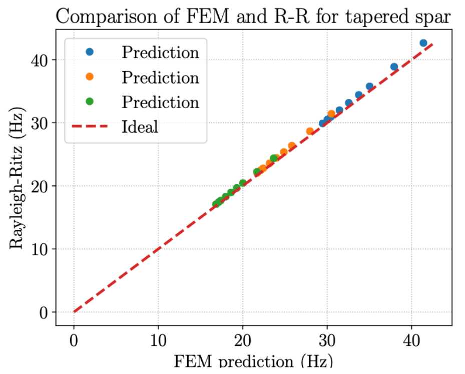


(b) Rayleigh-Ritz vs. FEM

Figure 28: Natural frequency: Rayleigh-Ritz vs. FEM predicted natural frequencies for a tapered cantilever beam

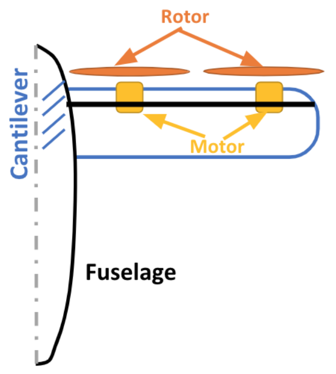


(a) Natural frequency vs. taper ratio

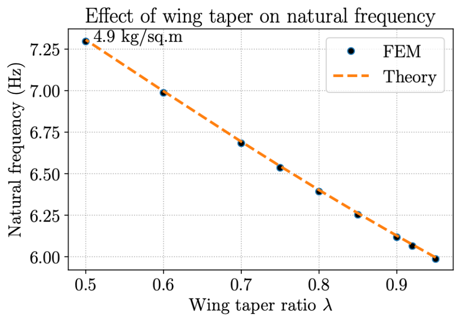


(b) Rayleigh-Ritz vs. FEM

Figure 29: Natural frequency: Rayleigh-Ritz vs. FEM for a tapered cantilever beam with distributed non-structural masses



(a) Layout of eVTOL wing



(b) Rayleigh-Ritz vs. FEM

Figure 30: Natural frequency: Rayleigh-Ritz vs. FEM for an example eVTOL wing

A.4.2 Case 2: distributed non-structural mass

This case is representative of an aircraft half-wing with skin, and no lumped masses. The non-structural masses are represented as a mass per unit area, i.e., $\frac{M}{A}$ kg/m^2. Three different values of the non-structural mass per unit wing area were evaluated: 0, 2.0 and 4.9 kg/m^2. The resulting natural frequencies were calculated from both Rayleigh-Ritz and FEM, and are plotted in Fig. 29. Excellent agreement is obtained.

A.4.3 Case 3: half-wing with lumped masses

This case is representative of an eVTOL wing. The dimensions of the beam are the same as in the previous examples, and the distributed skin mass per unit area is 4.9 kg/m^2. Two lumped masses (21.3 kg, 27 kg) are placed along the beam at $0.5L$ and L, respectively. The wing layout and natural frequencies are shown in Fig. 30.