HYDRA: Rotorcraft Conceptual Sizing

Authors

Ananth Sridharan
Bharath Govindarajan

This is the user 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 platforms.

The key features of HYDRA 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 desktop workstation. The use of Pythonenables powerful built-in text parsing abilities, resulting in more intuitive interfaces.

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.

This release of HYDRA is written in Python 3, so that the baseline analysis can be easily modified for custom applications where required. The compute-heavy modules are implemented in **Fortran90** to increase the speed of the calculations, and wrapped so that they can be called from Python.

Contents

1 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.

1.1 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.

1.2 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.

1.3 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.

1.3.1 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.

1.3.2 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:

1.3.3 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.

1.3.4 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 con-

stituents (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.

1.3.5 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.

2 First-time setup

2.1 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.

2.2 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

2.3 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.

3 Input files

This section of the documentation details the inputs files and conventions that that are necessary to perform sizing. `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**. Two example files are shown below with the **main dictionary names** and **sub-dictionary names** highlighted. The two main input files are **input.yaml** and **defaults.yaml**.

3.1 Single main rotor helicopter

Input file 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). The input file is subdivided into multiple segments, and each input and its units are explained in the following sections. The following input file is used to size a single main rotor/tail rotor helicopter with a mechanical transmission and turboshaft engines. An example single main rotor helicopter is shown in Fig. 1.

Figure 1: UH-60 Blackhawk: single main rotor helicopter

3.1.1 Sizing dictionary

```
Sizing:
   Rotors:
      SMR:
         DL:         [10.03]        # lb/ft2
#       radius:     [6.607]         # radius in meters
         Nb:         [4]
         Vtip:       [213.36] # m/s
         solidity:   [0.1178]
         tip_mach:   [0.95]
         flap_freq:  [1.035]
         cruise_rpm_ratio: [1.0]
   Fuselage:
      nrotors:        [1]
      liftfraction:   [1.0]
```

```
   Powerplant:
      TSEngine:
         type: 'turboshaft'
         num: 2
         redundancy: 0      # how many units can fail, group still works
   Transmission:
      Trans1:
         type:      'mechanical'
         eta:       0.98
#fidelity options for weight and performance models
   ifea:              False
   use_bemt:          False
```

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

3.1.1.1 Rotors

Rotors consists of sub-dictionaries, with each sub-dictionary corresponding to a rotor. In the present example, there is only one rotor that is sized by user-given inputs. The rotor that is sized is the main rotor, called "SMR" (Single Main Rotor). The actual name of the rotor can be any string, i.e., `SMR` is not a keyword, but `Rotors` is a keyword. If neither of these keywords are specified, then the rotor group must be mounted on wings, and its diameter is determined from geometric

requirements (largest rotor that fits on all wings that the rotor is mounted on, with adequate rotor-rotor clearance and rotor-fuselage spacing).

```
   Rotors:

      SMR:

         DL:         [10.03]        # lb/ft2
#       radius:     [6.607]         # radius in meters
         Nb:         [4]

         Vtip:       [213.36] # m/s

         solidity:   [0.1178]

         tip_mach:   [0.95]

         flap_freq:  [1.035]

         cruise_rpm_ratio: [1.0]
```

"SMR" is parameterized using the following **keywords** shown in Table 1. The last keyword **radius** is grayed out, because it is not used in the present example (the corresponding line in the input file is commented out, indicated by the '#' symbol). The **radius** keyword may be specified as an alternate to rotor disk loading.

3.1.1.2 Fuselage

The **Fuselage** dictionary is shown below, and the keywords used to define the fuselage are shown in Table 2.

```
   Fuselage:

      nrotors:        [1]

      liftfraction:   [1.0]
```

Table 1: Keywords for sizing a helicopter rotor

Keyword	Meaning	Value in example
DL	Disk loading	10.03 lb/ft^2
Nb	Number of blades	4
Vtip	Rotor hover tip speed	213.36 m/s
solidity	Geometric solidity	0.1178
tip_mach	Maximum blade tip Mach number	0.95
flap_freq	First flap natural frequency in hover	1.035/rev
cruise_rpm_ratio	Rotor cruise RPM/hover RPM	1.0
radius	Rotor radius	6.607 m

Table 2: Keywords for sizing a fuselage

Keyword	Meaning	Value in example
nrotors	Number of rotors mounted on this fuselage	1
liftfraction	Lift fraction carried by fuselage-mounted rotors	1.0

3.1.1.3 Powerplant

The **Powerplant** definition is shown below. There is only one powerplant group, called 'TSEngine' (TurboShaft Engine).

```
Powerplant:
   TSEngine:
      type: 'turboshaft'
      num: 2
      redundancy: 0      # how many units can fail, group still works
```

Here, 'TSEngine' is not a keyword: it is only the name given to the powerplant group, just as 'SMR' is the name of the rotor group. This powerplant group is defined using the keywords shown in Table 3. Here, there are two turboshaft engines, sized so that no engines may fail if adequate power is required in all segments of the sizing mission (excluding margins - specified later in defaults.yaml, Engine parameters).

Table 3: Keywords for sizing a powerplant

Keyword	Meaning	Value in example
type	Type of powerplant, character string	'turboshaft'
num	Number of engines/units	2
redundancy	No. of unit failures allowed	0

3.1.1.4 Transmission

The **transmission** input block is shown below. There is only one transmission in this conventional helicopter, named 'Trans1'. The name of the transmission group

(Trans1) is not a keyword, but the attributes of this transmission group are defined using keywords described in Table 4.

```
Transmission:
   Trans1:
      type:      'mechanical'
      eta:       0.98
```

Table 4: Keywords for sizing a transmission

Keyword	Meaning	Value in example
type	Type of transmission, character string	'mechanical'
eta	Transmission efficiency (power out/power in)	0.98

3.1.1.5 High-fidelity model switches

The keyword **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 keyword **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.

3.1.2 Aircraft Configuration

The **Configuration** dictionary specifies the associations between rotors, wings (if present), fuselage, transmissions and powerplants (defined in the **Sizing** dictionary). This dictionary provides the crucial mapping that specifies where rotors are

mounted, and the transmissions connecting various powerplants to the rotor groups. The group names (SMR, Trans1 and TSEngine) are used to specify these connections and size power sources using rotor power required and transmission efficiencies.

```
Configuration:
   Fuselage: 'SMR'              # which rotors are mounted on fuselage
   Rotors:
      SMR: 'Trans1'             # transmission used to power the rotors
   Transmission:                 # powerplant connected to transmissions
      Trans1:                   # transmission group name
         Powerplant: 'TSEngine' # powerplant group name
         PowerFraction: 1.0     # rated power fraction supplied
```

The various keywords used to associate vehicle components are explained in Table 5. The associations between rotors, fuselages and wings are specified as key-value pairs. The present example is interpreted by `HYDRA` as

1. The **Fuselage** supports the rotor group called '`SMR`'
2. The **Rotors** dictionary specifies that the rotor group `SMR` draws power through the transmission group '`Trans1`'
3. The **Transmission** dictionary specifies that the transmission group `Trans1` draws 100% of its required power from the powerplant group `'TSEngine'`.

Table 5: Keywords for defining vehicle configuration

Keyword	Meaning	Value in example
Fuselage	Specifies rotor mounted on fuselage	'SMR'
Rotors	Specifies transmissions powering each rotor group	SMR: 'Trans1'
Transmission	Sub-dictionaries specify transmission details	'Trans1'
Powerplant	Following name specifies associated powerplant	'TSEngine'
PowerFraction	Fraction of rated power from 'TSEngine' above	1.0

3.1.3 Mission profile

The sizing mission profile is specified through several keywords; the meaning of these keywords is explained below, and summarized in Table 6.

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.

```
Mission:
   nsegments:        5
   flight_mode:      [ 'hover', 'cruise', 'cruise', 'cruise', 'cruise']
   segment_type:     [ 'all',   'all',   'all',   'all',   'all']
   time_seg:         [     5,    2.42,    8.23,    84.9,       5] # minutes
   start_altitude:   [ 1219.2,  1219.2,  1828.8,  1828.8,     0.0] # m
   end_altitude:     [ 1219.2,  1828.8,  1828.8,  1828.8,     0.0] # m
   delta_temp_isa:   [ 27.92,   11.88,        0,       0,       0] # deg C
   rate_of_climb:    [     0,   251.86,       0,       0,       0] # m/min
   cruise_speed:     [     0,    85.33,  157.16,  159.03,   70.74] # knots
   add_payload:      [     0,       0,       0,       0,       0] #
   distance:         [     0,     0.0,   39.93,   416.7,     0.0] # km
   fixed_GTOW:       6454.0             # derive payload for fixed weight
```

Table 6: Keywords for defining mission

Keyword	Meaning	Value in example
nsegments	Number of segments in sizing mission	6
flight_mode	Specifies type of mission segment	'hover', 'cruise'
time_seg	Segment duration in minutes	–
start_altitude	Altitude at start of segment (m)	–
end_altitude	Altitude at end of segment (m)	–
delta_temp_isa	Temperature offset at sea level (C)	–
rate_of_climb	Vehicle rate of climb in the segment (m/min)	–
cruise_speed	Horizontal speed parallel to ground (knots)	–
add_payload	Payload added at end of segment (kg)	–
distance	Horizontal distance covered (km)	–
fixed_GTOW	Total weight to hold constant (kg)	–

3.1.4 Aircraft details

The aircraft specification dictionary is show below, and the relevant keywords are summarized in Table 7. 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.

```
Aircraft:
   aircraftID: 1
# payload and crew (kg)
   mass_payload:    1134.0
   common_equipment: 1412.0  # fixed values from ndarc output
   pax_count:       0
   common_per_pax:  0.0
   avionics:        0
```

Table 7: Keywords for defining miscellaneous details

Keyword	Meaning	Value in example
aircraftID	Flag: helicopters (1) or eVTOL (other)	1
mass_payload	Payload mass (kg)	1134
common_equipment	Mission equipment mass (kg)	1412
pax_count	Number of passengers with bags	0
common_per_pax	Mission equipment per passenger (kg)	0
avionics	Avionics group mass (kg)	0

Input file 2: defaults.yaml

This input file contains the sizing constraints, motor efficiencies, powerplant details, calibration factors for empirical/reduced-order weight and peformance mod-

els 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.

3.1.5 Sizing constraints

```
Sizing:

   Constraints:

      max_rotor_radius:  30.0

      max_gtow: 20000.0

      max_ct_sigma: 0.14
```

Table 8: Keywords for defining sizing constraints

Keyword	Meaning	Value in example
max_rotor_radius	Upper limit on hover d-value (meters)	30
max_gtow	Upper limit on take-off mass (kg)	20000
max_ct_sigma	Upper limit on hover blade loading	0.14

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".

3.1.6 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.

3.1.6.1 Engines

For fuel-burning engines, the **Engines** dictionary is used to set relevant high-level properties of turboshaft engines, summarized in Table 9.

```
Engines:
    powerHoverAccs : 0.01 # percent of total power for accessories
    effHoverPower : 1.00 # hover power efficiencies
    loss_filter   : 0.00
    frac_install  : 1.05 # installation losses
# power lapse rates with density, temperature and installed fractions
    KD_mrp: 1.1132
    KD_irp: 1.1379
    KD_mcp: 1.1000
    KT_mrp: 2.1445
    KT_irp: 2.2248
    KT_mcp: 2.2407
```

```
fracPowerIdle: 0.2

fracPowerIRP : 0.932

fracPowerMCP : 1.0
```

Table 9: Keywords for defining turboshaft engine properties

Keyword	Meaning	Value in example
powerHoverAccs	Power for accessories (fraction of rotor power)	0.01
effHoverPower	Additional rotor efficiency knockdowns	1
loss_filter	Engine power loss fraction from intake filters	0.00
frac_install	Engine installation power loss (fraction)	0.00
KD_MRP	Max. rated power lapse rate w/ density	1.1132
KD_IRP	Intermediate power lapse rate w/ density	1.1379
KD_MCP	Max continuous power lapse rate w/ density	1.1000
KT_MRP	Max rated power lapse rate w/ temperature	2.1445
KT_IRP	IRP lapse rate w/ temperature	2.2248
KT_MCP	MCP lapse rate w/ temperature	2.2407
fracPowerIdle	Idle power to installed power	0.2
fracPowerIRP	Intermediate power to installed power	0.932
fracPowerMCP	MCP/installed power	1.0

3.1.6.2 Aerodynamics

The **aerodynamics** dictionary is used to specify parameters for calculating rotor hover efficiency, interference losses and body drag. A sample dictionary is shown below, and the keywords are summarized in Table 10.

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** accounts for non-ideal induced 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.

The empirical parameters used to model wing performance are the Oswald efficiency factor **oswald** (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.

```
Aerodynamics:
   Rotors:
      hover_dwld_factor:  0.045
      cd0:                0.011
      induced_power_factor: 1.15
      kint:               1.0
      FM:                 0.75
   Body:
      flat_plate_factor: 4.0
```

Table 10: Keywords for defining rotor performance and body drag

Keyword	Meaning	Value in example
hover_dwld_factor	Vertical down-load/rotor lift in hover	0.045
cd0	Blade airfoil profile drag coefficient	0.011
induced_power_factor	Total induced power/ideal power	1.15
kint	Rotor interference loss factor	1.0
FM	Rotor hover figure of merit	0.75
flat_plate_factor	Coefficient in drag-weight trends	4.0

3.1.6.3 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. If these factors are not entered, default values of 1 are automatically assigned. The relevant keywords are summarized in Table 11.

Table 11: Keywords for defining technology factors

Keyword	Group weight modified	Value in example
landing_gear	Alighting gear	0.3523
emergency_sys	Parachute for eVTOLs	0
rotor	Rotor blades, hubs, collective actuator	(assumed 1)
empennage	Horizontal/vertical stabilizers	(assumed 1)
fuselage	Fuselage	(assumed 1)
fuel_system	Fuel handling: tanks and pumps	(assumed 1)
drive_system	Transmissions	(assumed 1)
flight_control	Flight controls and hydraulics	(assumed 1)
powerplant	Engines and batteries	(assumed 1)
fuel	Fuel	(assumed 1)
battery	Batteries	(assumed 1)

The sample input block is shown below:

```
Tech_factors:
```

```
Weight_scaling:

   landing_gear:    0.3523     #

   emergency_sys:   0.0        # group absent
```

3.2 All-electric tandem tilt-wing

An example all-electric tandem tilt-wing is shown in Fig. 2. This aircraft incorporates two lifting wings, with four rotors mounted on each wing. Tilt actuators allow the wings (and therefore the rotors mounted on these wings) to tilt from a vertical orientation in hover to a horizontal orientation in cruise. The example shown in this document is a modified simplification of Vahana, i.e., it is not identical to the aircraft shown in Fig. 2. In this the example, both wings are assumed to be identical. All 8 rotors are also assumed to be identical.

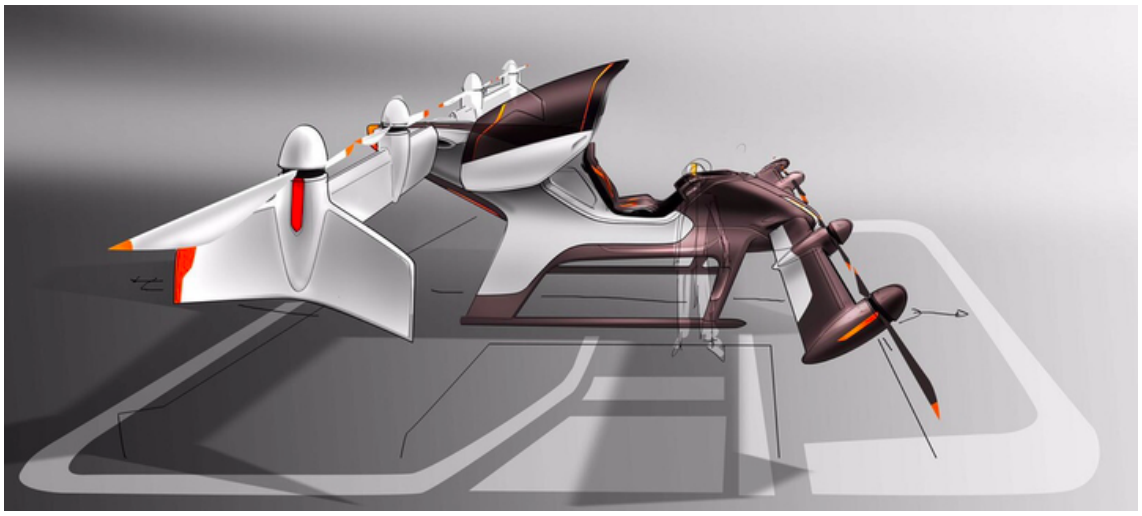


Figure 2: Vahana all-electric tandem tilt-wing with 8 rotors

For the all-electric configuration, the input files are slightly different from the

corresponding files for the conventional helicopter. An additional **Wings** dictionary is specified, and the propulsion architecture definition is also modified to reflect the new system.

Input file 1: input.yaml

3.2.1 Sizing

This dictionary specifies the values of high-level design variables to use for sizing the vehicle. For this vehicle type, **Rotors**, **Wings**, **Fuselage**, **Powerplant** and **Transmission** are all defined and linked with the **Configuration** dictionary.

3.2.1.1 Rotors

The **Rotor** sub-dictionary is shown below. Here, the number of inputs are far fewer than those required to size a helicopter rotor, for several reasons. The blade first flap natural frequency is not specified for eVTOLs, because rotor dynamics is no longer a key driver for blade sizing - sizing for strength automatically results in a very stiff design at the scales of interest. Additionally, neither disk loading nor rotor radius are specified, and so the wing-mounted rotors are automatically sized to ensure that the maximum possible disk area is achieved. Finally, the tip speed is markedly lower than that of a conventional helicopter's rotor, to reduce noise. This input value automatically guarantees that the advancing blade tip mach number in cruise is sufficiently low as to ensure almost zero compressibility drag. By not specifying the cruise rotor speed, a default value of 50% hover tip speed is assigned.

```
Rotors:
   All_rotors:
   All_rotors:
      Nb:        [3]                  # per rotor
      Vtip:      [170]                # hover tip speed, m/s
      ctsigma:   [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 per rotor **Nb** = 3
2. Tip speed **Vtip** = 170 m/s
3. Hover blade loading **ctsigma** = 0.139

For the variable-RPM rotors used in eVTOLs, additional thrust can be achieved by changing the rotor speeds at the same blade loading. Therefore, the strict upper limits of C_T/σ - usually observed in conventional helicopters with constant-speed rotors - can be relaxed for eVTOLs.

Rotor sizing rules

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.

3.2.1.2 Wings

The **Wings** dictionary consists of sub-dictionaries, with each sub-dictionary corresponding to a fixed wing group. 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 shown below, and the relevant keywords are summarized in Table 12.

```
Wings:

   Main_wing:

      nwing:          [2]

      aspectratio:    [3,4,5,6,7,8]

      cl:             [0.3,0.4,0.5,0.6,0.7]

      liftfraction:   [1]

      nrotors:        [4]
```

Table 12: Keywords for defining wing parameters

Keyword	Meaning	Value in example
nwing	Number of fixed wings in this set	2
aspectratio	Wing geometric aspect ratio	[3, 4, ..]
cl	Wing lift coefficient in cruise	[0.3, 0.4, ..]
liftfraction	Weight fraction in cruise carried by this wing set	1
nrotors	Number of rotors mounted on each wing	4

This input specifies one fixed wing group called "Main_wing", of which there are **nwing**=2 identical units. The name "Main_wing" is not a keyword - it is a character string that is used to refer to the set of wings defined using **keywords**, and build associations with rotor groups through the **Configuration** dictionary. 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 the **nwing** units in this wing group during cruise. Finally, the parameter **nrotors** specifies the number of rotors mounted on a wing of this group. Here, 4 rotors are mounted on each wing in this group. Each input parameter can be specified as a single value or list of values to use for sizing.

3.2.1.3 Fuselage

The fuselage input dictionary for the tilt-wing is shown below. The dictionary specifies that the fuselage does not directly support any rotors, and that it is

responsible for carrying no cruise lift.

```
Fuselage:
    nrotors:     [0]
    liftfraction: [0.0]
```

3.2.1.4 Powerplant

The source of power for the all-electric aircraft is a Lithium-ion battery, defined as shown below. The name `'BatteryPack'` is not a keyword - it is only the name of the powerplant group that supplies energy to the system. The keyword **type** is defined as 'battery' to indicate the use of a Lithium-ion pack. The details for the battery cells and pack assembly are provided in **defaults.yaml**.

```
Powerplant:
    BatteryPack:
        type: 'battery'
        num:  1
        redundancy: 0
```

3.2.1.5 Transmission

In this aircraft, there is a single electric transmission, indicated by the keyword **type** defined as `'electric'` below. The transmission efficiency is 1, i.e., 100% of the energy is transmitted without losses (usually 0.98, here shown as 1 for illustrative purposes). Additionally, motor efficiencies in hover and cruise are defined through the keywords **hover_efficiency** and **cruise_efficiency**. These keywords will be

ignored if a higher-fidelity motor model is integrated into the sizing loop (future expansions).

```
Transmission:
   ElTrans:
      type:     'electric'
      eta:          1.00
      redundancy:   1.0
      Motors:
         hover_efficiency: 0.90
         cruise_efficiency: 0.85
```

3.2.1.6 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.

3.2.2 Aircraft Configuration

The vehicle configuration details the association of rotors, wings, transmissions and powerplants. The **Wings** dictionary shows that the wing group called 'Main_wing' supports rotors belonging to the group 'All_rotors'. The **Rotors** dictionary shows that the rotor group 'All_rotors' draws power through the transmis-

sion group 'ElTrans'. The **Transmission** dictionary indicates that the transmission called 'ElTrans' draws all of its power (**PowerFraction**=1.0) from 'BatteryPack'. Each of these building blocks have been defined in the **Sizing** dictionary; the **Configuration** dictionary is used to create associations between these components.

For electric transmissions, the analysis automatically sizes the drive motors based on all rotor groups that draw power through a given transmission.

```
Configuration:
   Wings:
      Main_wing:    'All_rotors'  # whichrotor group to mount on this wing
   Rotors:
      All_rotors:   'ElTrans'     # transmission used to power the rotors
   Transmission:
      ElTrans:                      # transmission group name
         Powerplant: 'BatteryPack' # connected powerplant group name
         PowerFraction: 1.0        # rated power fraction supplied
```

Configuration mapping rules

At present, multiple rotors from the same group may be mounted on wings or fuselages. One powerplant may supply energy to multiple transmissions, but each rotor group may draw power from only one transmission group. However, one transmission group may supply power to multiple rotor groups. Work is ongoing to release advanced features with hybrid powerplants with heterogeneous power sources.

3.2.3 Mission profile

The mission profile from **input.yaml** is show below. This input does not require any modification for all-electric vehicles, because it does not depend on the configuration.

```
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

   rate_of_climb:    [     0,      0,       0,      0   ] # m/min

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

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

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

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

   sizing_order:     [     1,      2,       0,      0   ] # order of sizing
```

3.2.4 Aircraft

The aircraft specification dictionary is shown below. This input block also has identical keywords compared to the single main rotor helicopter, because it does not depend on the configuration (only the mission).

```
Aircraft:
```

```
Aircraft:
   aircraftID: 2
   mass_payload:    345.0      # 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         # number of passengers
```

Input file 2: defaults.yaml

This input file contains the sizing constraints, powerplant details, calibration factors for empirical/reduced-order weight and performance 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 chunks, and each dictionary in this input file is detailed below.

3.2.5 Sizing dictionary

```
Sizing:

   Constraints:

      max_rotor_radius:  14.01 # 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".

3.2.6 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, and the keywords are summarized in Table 13.

3.2.6.1 Battery

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).

```
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 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.

5. Finally, the battery sizing option **Force_sizing** is a string 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.

Table 13: Keywords for defining battery parameters

Keyword	Meaning	Value in example
Cell	Indented entries pertain to cell properties	–
sp_energy	Specific energy of a cell (Watt-hr/kg)	–
Tmax	Maximum cell temperature (deg C)	70.0
energy_vol	Cell energy per unit volume (Watt-hr/liter)	632.0
volume	Volume of cell unit (liters)	0.01708
Pack	Indented entries pertain to pack properties	
SOH	Average cell state of health (0 to 1)	0.8
DOD_min	Minimum depth of discharge allowed (0 to 1)	0.075
integ_fac	Mass of cells / mass of pack	0.75
vol_fac	Volume of cells / volume of pack	0.3
Force_sizing	String input to choose type of sizing	'energy'

3.2.6.2 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 sub-dictionaries for Rotors, Propellers and Body have been described in the previous section and are not repeated here. 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**, summarized in Table 14.

Table 14: Keywords for defining wing performance parameters

Keyword	Meaning	Value in example
oswald	Oswald efficiency factor	0.8
cd0	Wing airfoil section profile drag coefficient	0.014

Table 15: Keywords for defining geometric parameters

Keyword	Meaning	Value in example
fuselage_width	Fuselage equivalent diameter (m)	1.0
fuselage_length	Fuselage length (m)	7.0
clearance	Rotor tip clearance/radius	7.0

3.2.6.3 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. The geometry parameters are summarized in Table 15.

3.2.6.4 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. In the present case, the wing mass is scaled to 90% of the predicted value, landing gear is scaled to 40% of the model prediction and anti-icing group is neglected.

```
Tech_factors:

   Weight_scaling:

      wing:           0.9         # wings

      landing_gear:   0.4         #

      anti_icing:     0.0         # ignored
```

3.2.7 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.

3.2.7.1 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. The keywords for defining acquisition cost parameters are shown in Table 16.

Table 16: Keywords for defining acquisition cost parameters

Keyword	Meaning	Value in example
sense_avoid	System purchase cost	189817.0
avionics	System purchase cost	145807.0
interiors	System purchase cost	45152.0
testing	System purchase cost	6400.0

3.2.7.2 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, and the relevant keywords are summarized in Table 17.

```
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
```

3.2.7.3 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

Table 17: Keywords for defining cost elements that scale with vehicle properties

Keyword	Meaning	Value in example
final_assem_line	Cost per kg of take-off mass	90.07
BRS	Cost per kg of emergency parachute mass	10.885
fuselage	Cost per kg of fuselage mass	2807.0
landing_gear	Cost per kg of landing gear mass	2807.0
motors	Cost per kg of electric motor mass	2669.0
power_dist	Cost per kg of power distribution system mass	31.0
rotor_blade	Cost per kg of rotor blade mass	77605.0
rotor_hub	Cost per kg of rotor hub mass	14133.0
wires	Cost per kg of wire mass	20.3
tilt_actuator	Cost per kg of tilt actuator mass	2868.0
wing_flap	Cost per kg of wing flaps mass	2619.0

the component is less expensive when manufactured in bulk or performed at large scales. The keywords to define these parameters are summarized in Table 18.

```
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
```

Table 18: Keywords for defining cost scaling factors

Keyword	Meaning	Value in example
sense_avoid	Cost multiplier for sense and avoid system	0.5
avionics	Cost multiplier for avionics	0.75
interiors	Cost multiplier for interiors	1.0
testing	Cost multiplier for final testing	1.0
final_assem_line	Cost multiplier for final assembly line	0.5
BRS	Cost multiplier for emergency parachute	0.5
fuselage	Cost multiplier for fuselage structure	0.5
landing_gear	Cost multiplier for landing gear structure	0.5
wing_structure	Cost multiplier for wing structure	0.5
motors	Cost multiplier for motors	0.5
power_dist	Cost multiplier for power distribution systems	0.5
rotor_blade	Cost multiplier for rotor blades	0.2
rotor_hub	Cost multiplier for rotor hubs	0.2
wires	Costmultiplier for wires	1.0
tilt_actuator	Cost multiplier for tilt actuators	0.75
wing_flap	Cost multiplier for wing flaps	0.75

3.2.8 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.

3.2.8.1 Annual costs

The sample **Annual** operating cost inputs are shown below, and the relevant keywords are summarized in Table 19.

```
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 vehi-

cle 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**).

Table 19: Keywords for defining cost scaling factors

Keyword	Meaning	Value in example
Flight_hours	Flight hours operated per year	1500
Liability	Cost for liability insurance per year	22000
Inspection	Cost for annual inspection	7700
Insurance_percent	Hull insurance as % of acquisition cost	4.5
Depreciation_percent	Depreciation as % of acquisition cost	10
Pilot	Pilot annual salary and benefits	280500
Training	Pilot annual training update cost	9900

3.2.8.2 Hourly costs

This dictionary specifies the maintenance and inspection costs associated with operating an air vehicle. A sample input chunk for **Hourly** costs is shown below, and the keywords are summarized in Table 21. 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
```

Table 20: Keywords for defining maintenance costs incurred per flight hour

Keyword	Meaning	Value in example
Frame_maintenance	Frame upkeep cost per flight hour	37.35
Rotor_inspection	Rotor inspection cost per flight hour	1.0
Motor_inspection	Motor inspection cost per flight hour	0.625

3.2.8.3 Battery costs

This dictionary specifies the number of charge/discharge **Cycles** that a battery can withstand before its usable energy reduces to the threshold 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). The keywords for battery costs are summarized in Table **??**. In the

present implementation, a charge/discharge cycle is assumed to correspond to one flight. Therefore, in the example input provided, the battery is replaced after 900 flights, regardless of the actual charge/discharge levels achieved.

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

Table 21: Keywords for defining battery costs incurred

Keyword	Meaning	Value in example
Cycles	Number of charge/discharge cycles	900
Cost_per_kwh	Battery purchase cost/unit energy (kWh) capacity	180.0
Electricity	Electricity cost/stored energy (kWh) for recharging	0.2

3.2.8.4 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
```

Table 22: Keywords for defining vertiport-related operation parameters

Keyword	Meaning	Value in example
Landing_fees	Cost per landing	20.0
Padding_time	Door → door time minus air time (min)	20.0
Ground_distance	Door → door trip minus VTOL range (km)	2.0

Table 23: Keywords for defining taxi-related operation parameters

Keyword	Meaning	Value in example
Distance_rate	Tariff per ground distance covered (km)	0.55
Time_rate	Tariff per minute spent in taxi	0.36
Padding_time	Door → door time minus time in taxi (min)	15.0

3.2.8.5 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). The keywords for the **Taxi** are summarized in Table 23.

```
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
```

3.2.9 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. The keywords for system redundancies are summarized in Table 24.

```
Redundancies:

   wing_flap:          1.0

   tilt_actuator:      2.0

   wires:              1.0

   avionics:           1.0
```

Table 24: Keywords for defining aircraft redundancy parameters

Keyword	Meaning	Value in example
wing_flap	Number of actuators per wing flap	1.0
tilt_actuator	Number of actuators per wing/rotor tilt hinge	2.0
wires	Number of redundant power cables, signal wires	1.0
avionics	Number of redundant systems for avionics	1.0

4 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.

4.1 log<XYZ>.yaml

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

4.1.1 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.

4.1.2 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
```

4.1.3 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)
         ...
```

4.1.4 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.

4.1.4.1 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]
```

4.1.4.2 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]
```

4.1.4.3 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*.

4.1.5 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]
```

4.1.6 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.

4.1.7 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] # [%]
```

4.1.8 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]
```

4.1.9 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
```

4.2 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.

4.3 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

...

5 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.

6 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

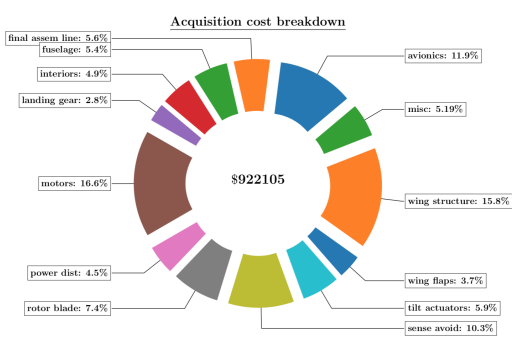
   python3 Postprocessing/piethon.py <XYZ>

   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. 3

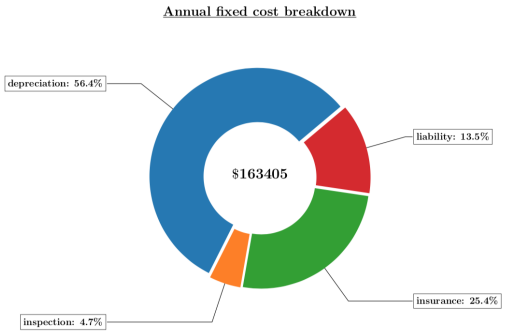
2. **Battery charge profile**: `HYDRA` features the ability to visualize the battery state of charge as a function of time along the mission profile. This postprocessor can be invoked with the command

   python3 Postprocessing/battery_draw.py 10

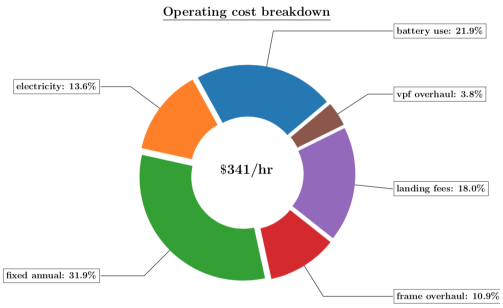
   Here, 10 is the unique design ID for the sized vehicle that needs to be postprocessed. The resulting plot is stored in a PDF called **battery_design_10.pdf**, and shown in Fig. 4. The first subplot shows the estimated vehicle range for



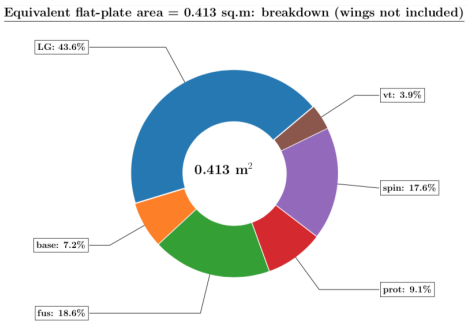
(a) Acquisition cost



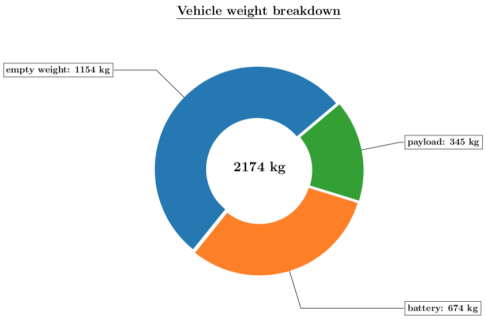
(b) Annual cost



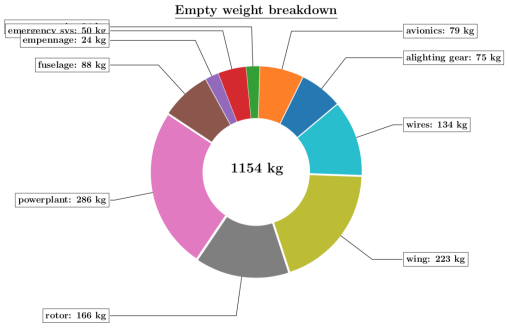
(c) Operating cost



(d) Parasitic drag breakdown



(e) Vehicle weight



(f) Empty weight

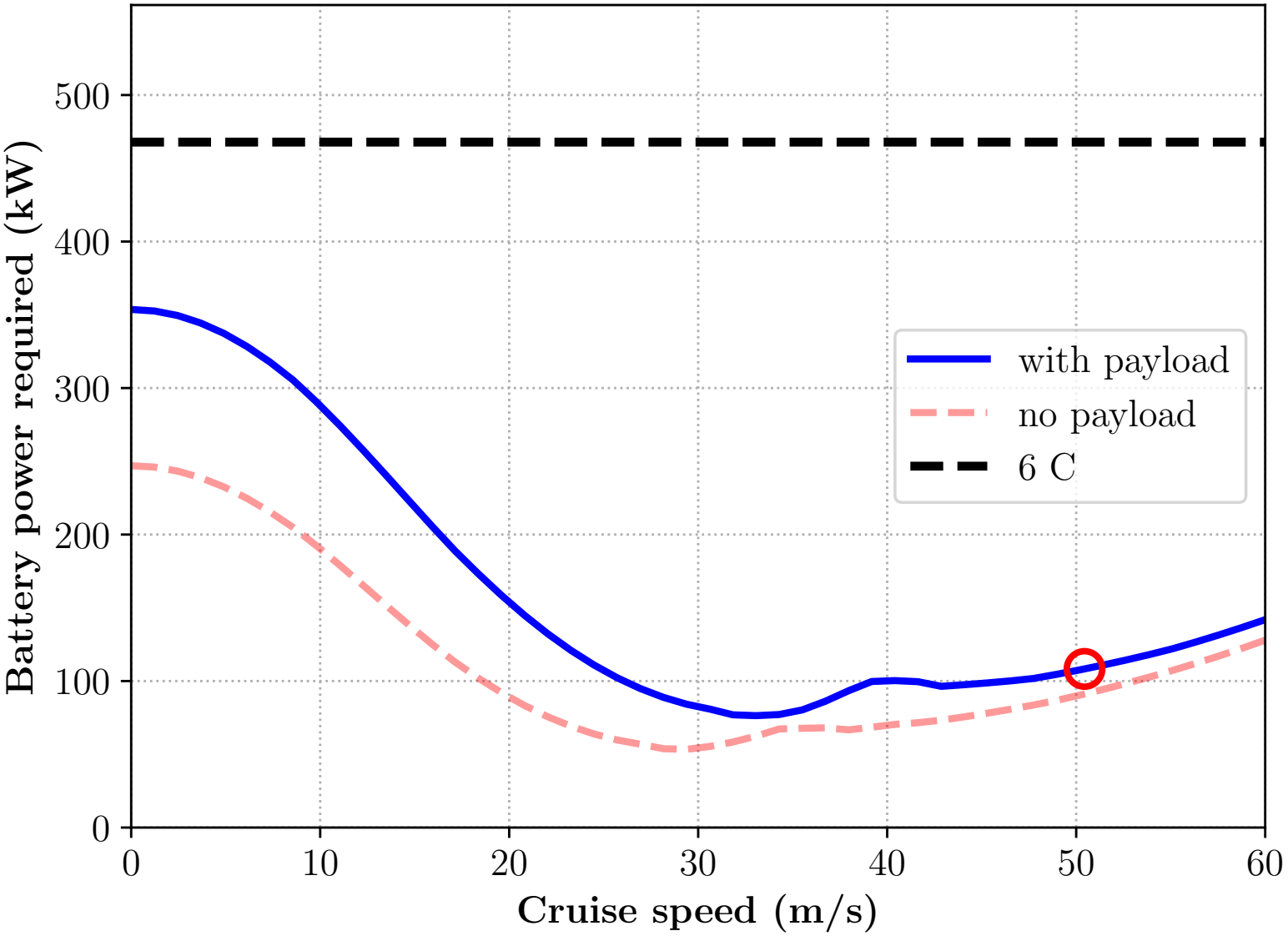
Figure 3: Pie charts generated by piethon.py postprocessor for a single design

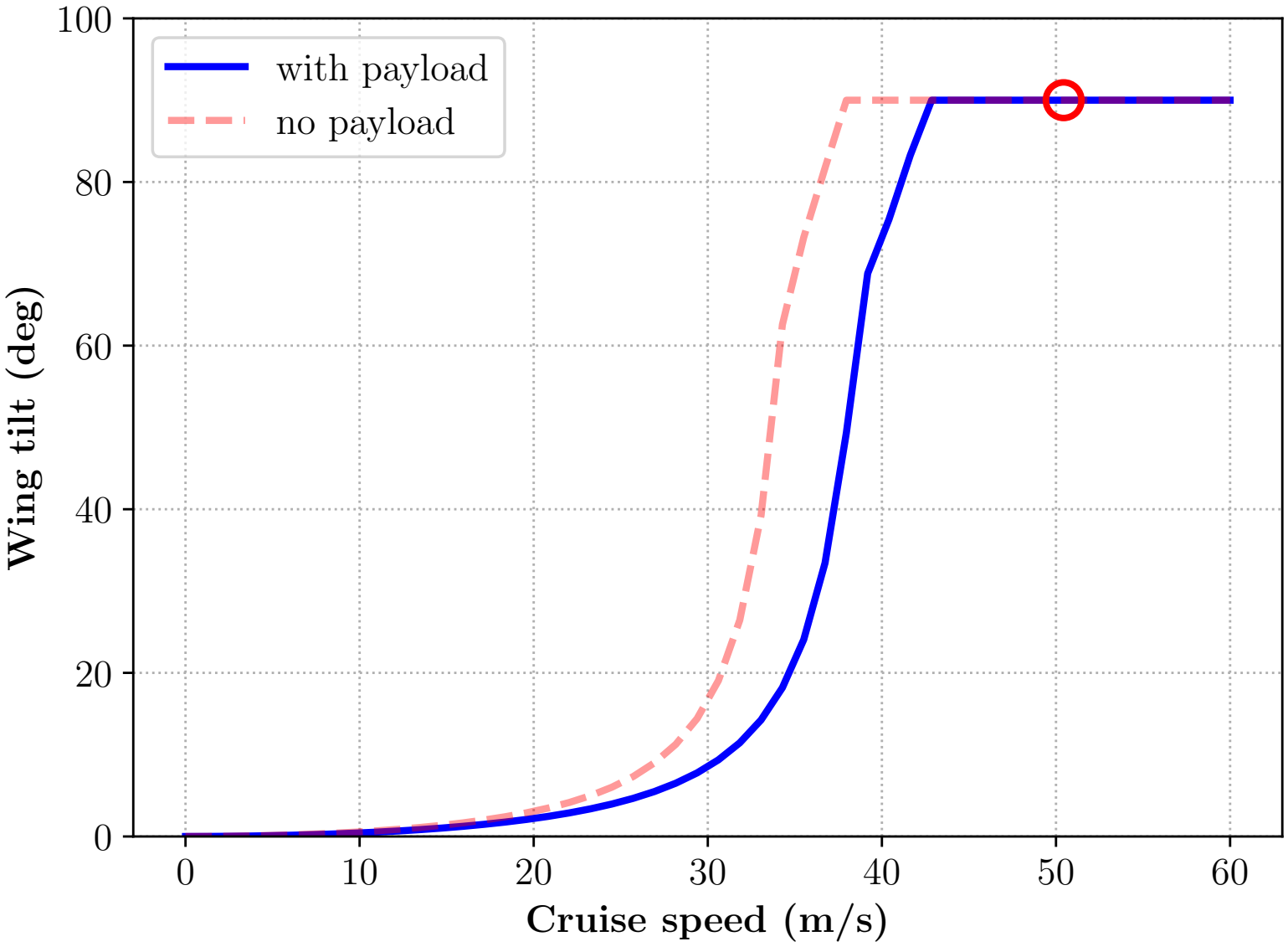
a "perfect battery" starting the mission at its theoretical maximum state of charge. The second subplot shows the same "ideal battery" charge, along with the estimated battery state of charge used in `HYDRA` for sizing the vehicle.

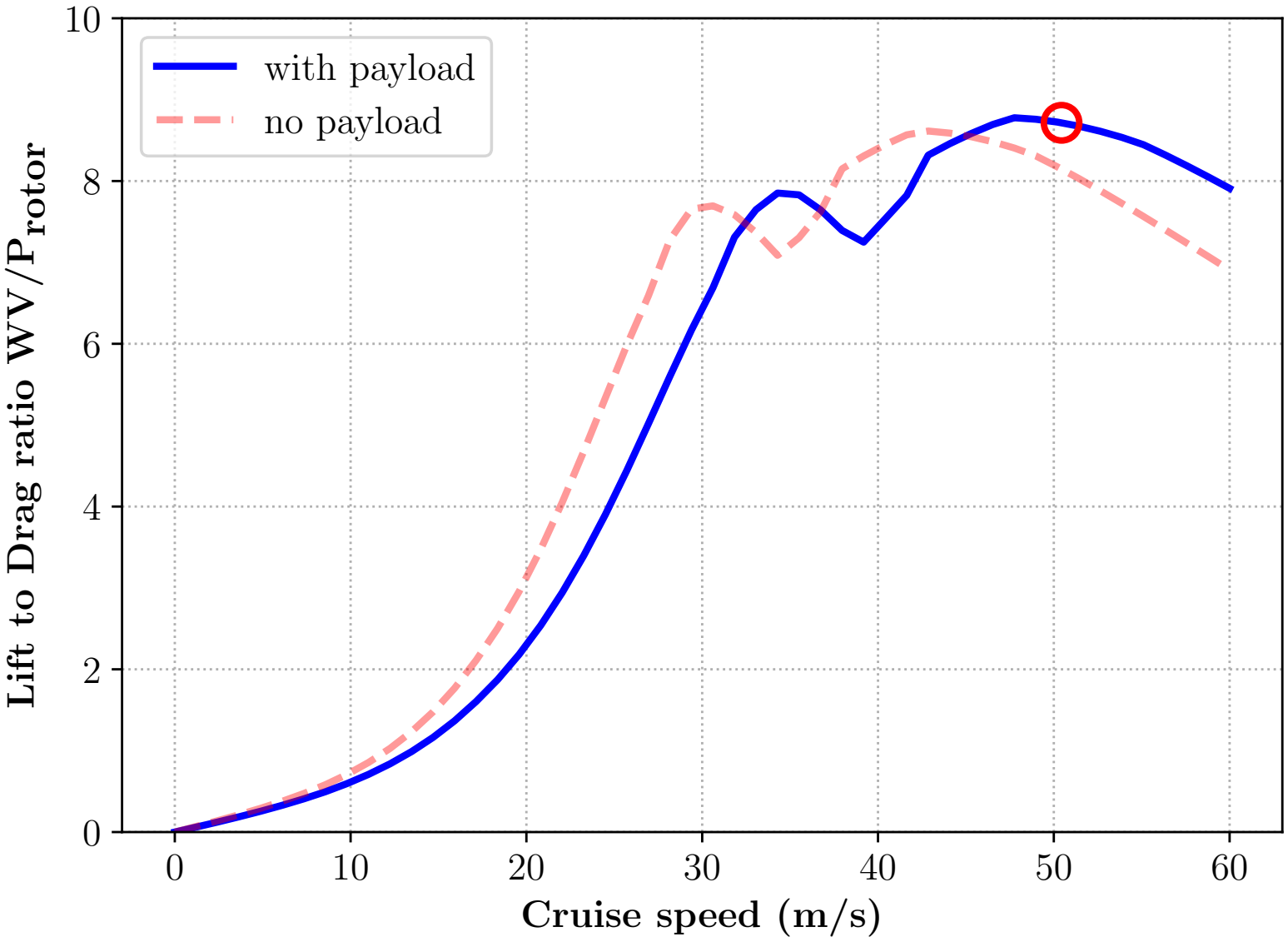
3. **Vehicle performance**: To automatically generate the (approximate) power curve, payload-range and payload-endurance trade offs, `HYDRA` features a performance postprocessor that can be invoked as follows:

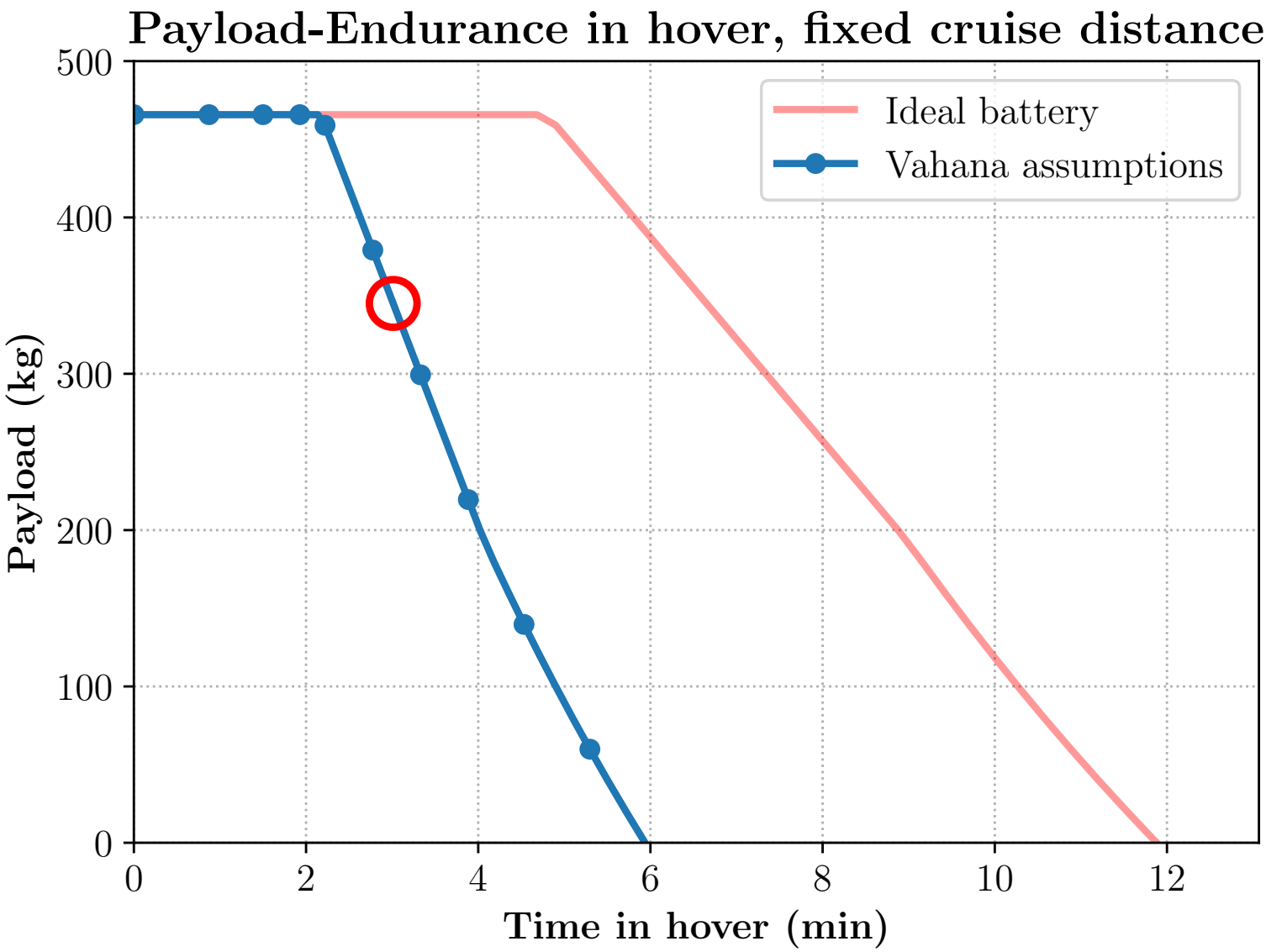
python3 Postprocessing/performance.py 28

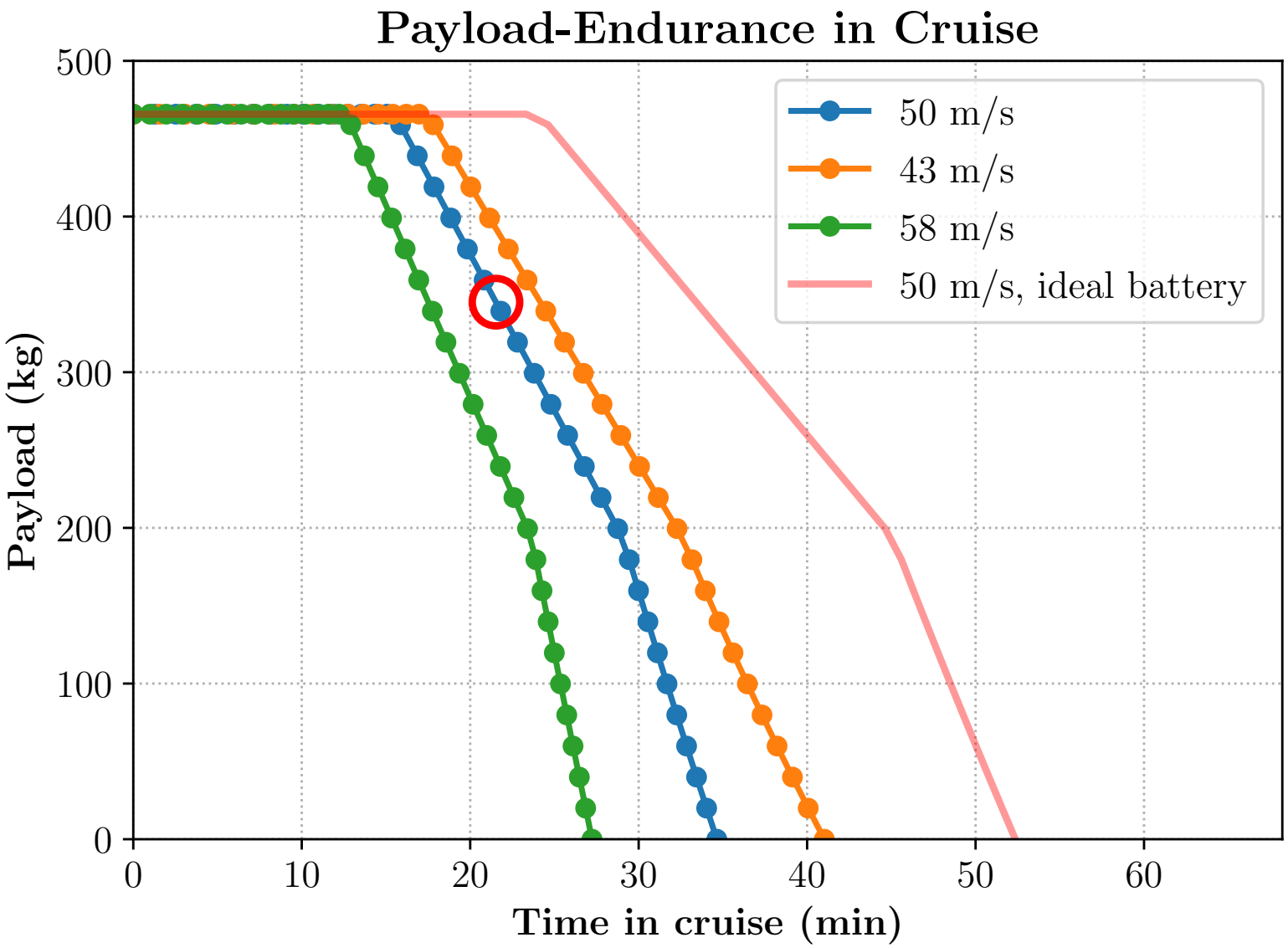
Here, 28 is the unique design identifier that will be analyzed at various airspeeds and longitudinal trim will be performed. The outputs from the postprocessor are shown in the following pages.

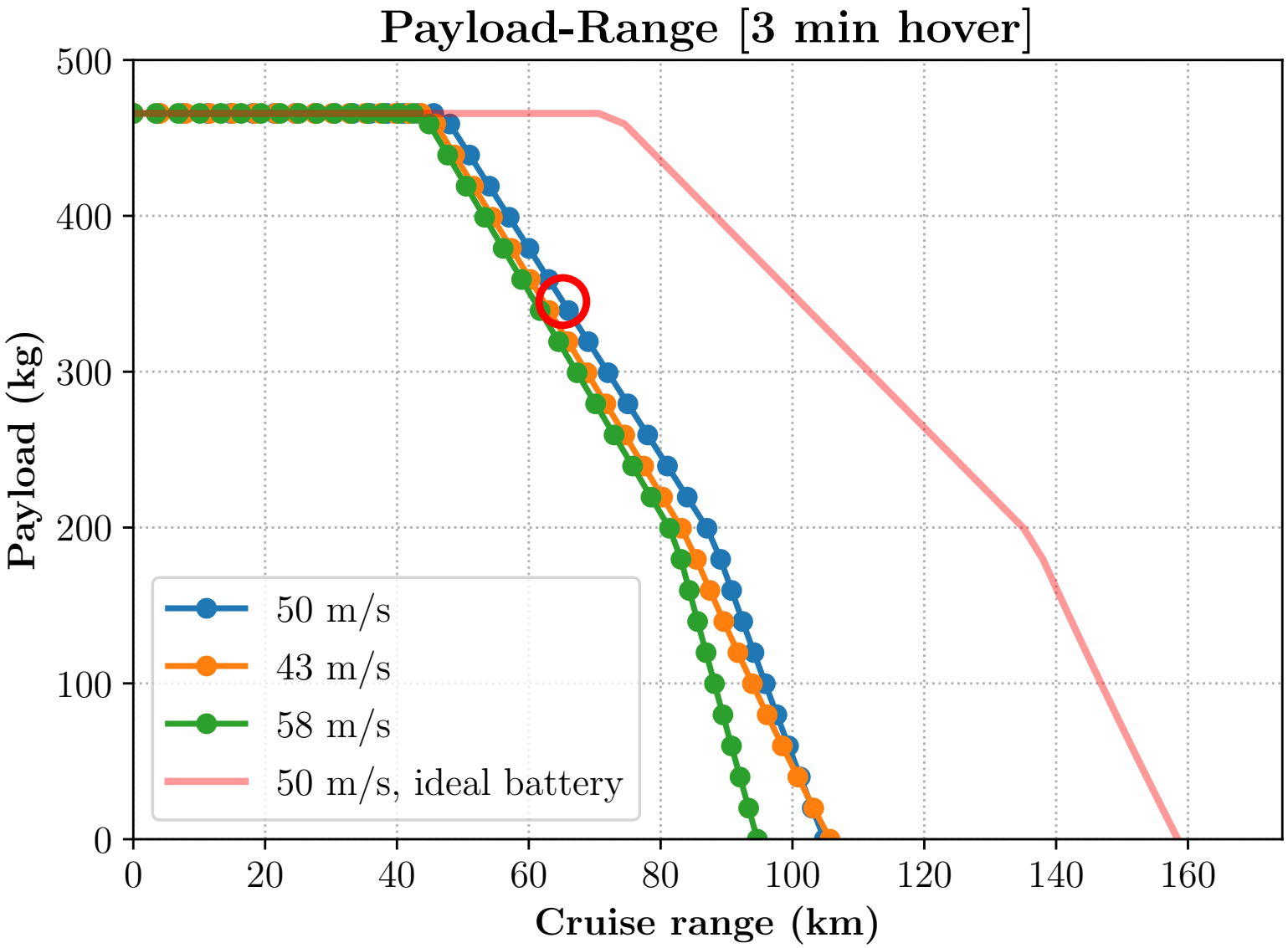

Battery power required (kW)
0
100
200
300
400
500
0
10
20
30
40
50
60
Cruise speed (m/s)
with payload
no payload
6 C


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


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


Payload-Endurance in hover, fixed cruise distance
Ideal battery
Vahana assumptions
Payload (kg)
Time in hover (min)
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)
0
100
200
300
400
500
0
20
40
60
80
100
120
140
160
50 m/s
43 m/s
58 m/s
50 m/s, ideal battery

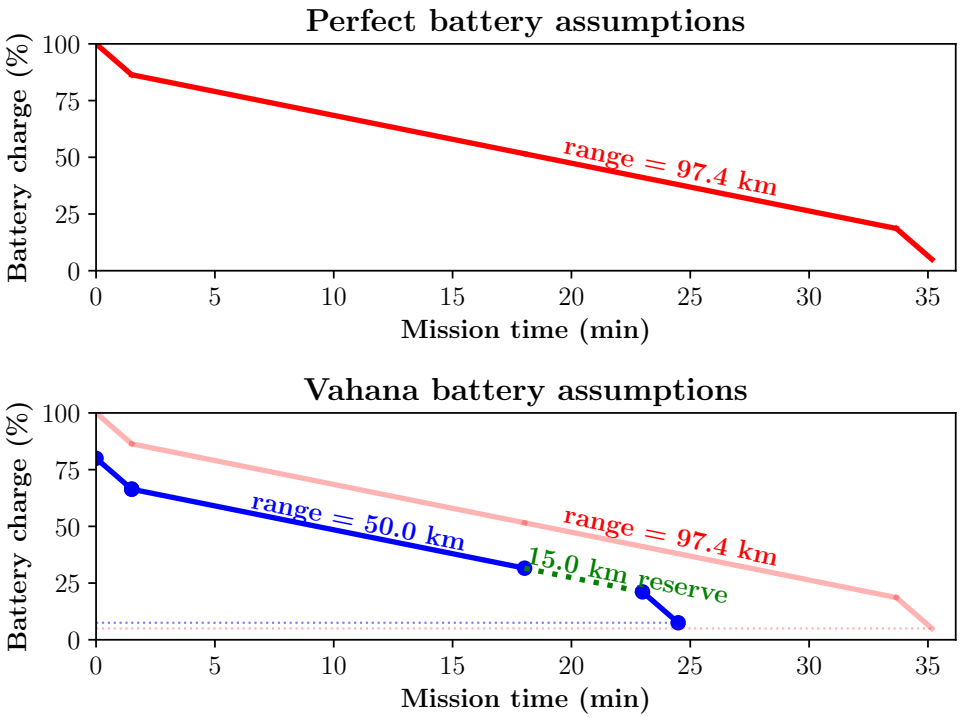
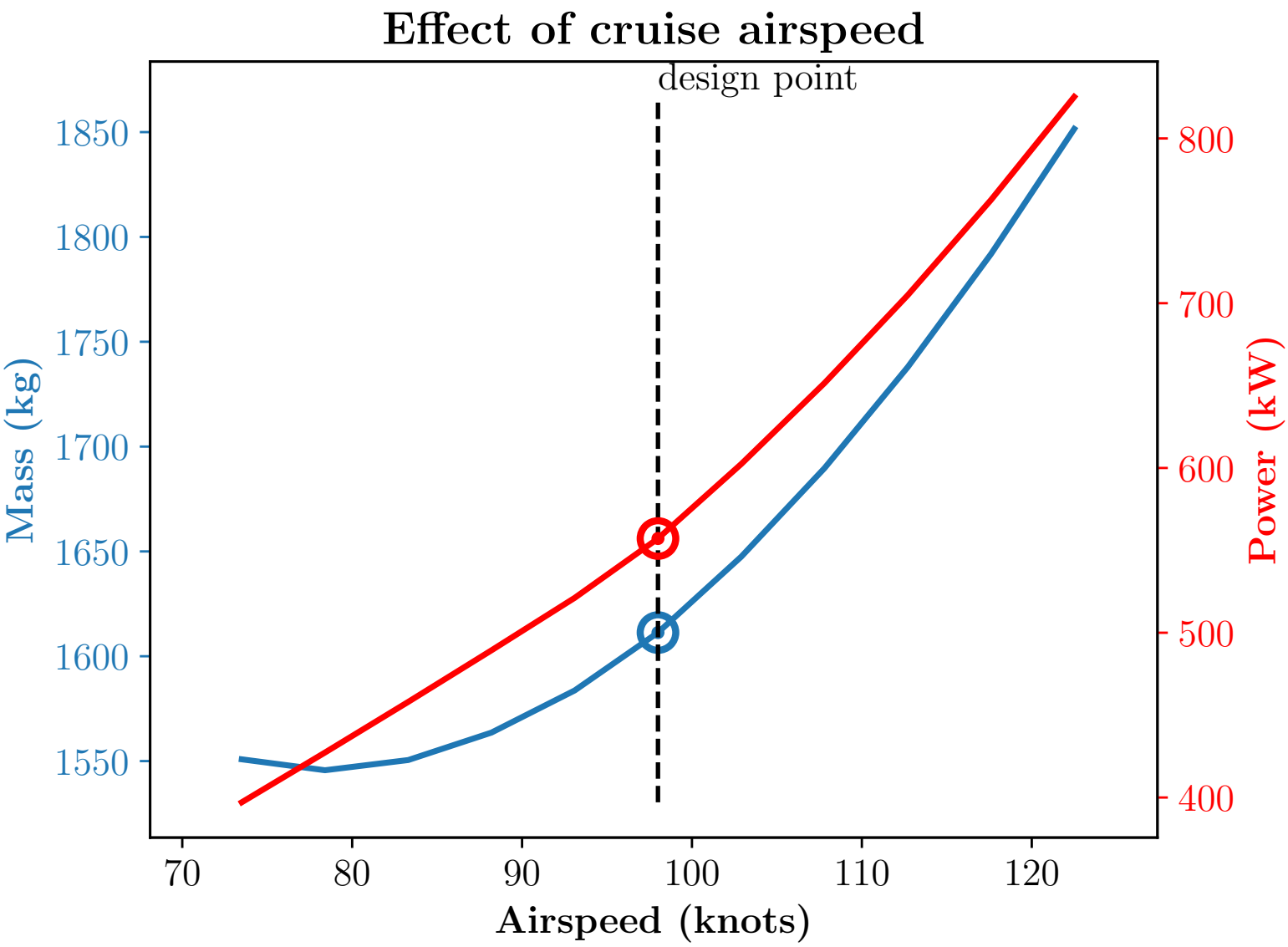


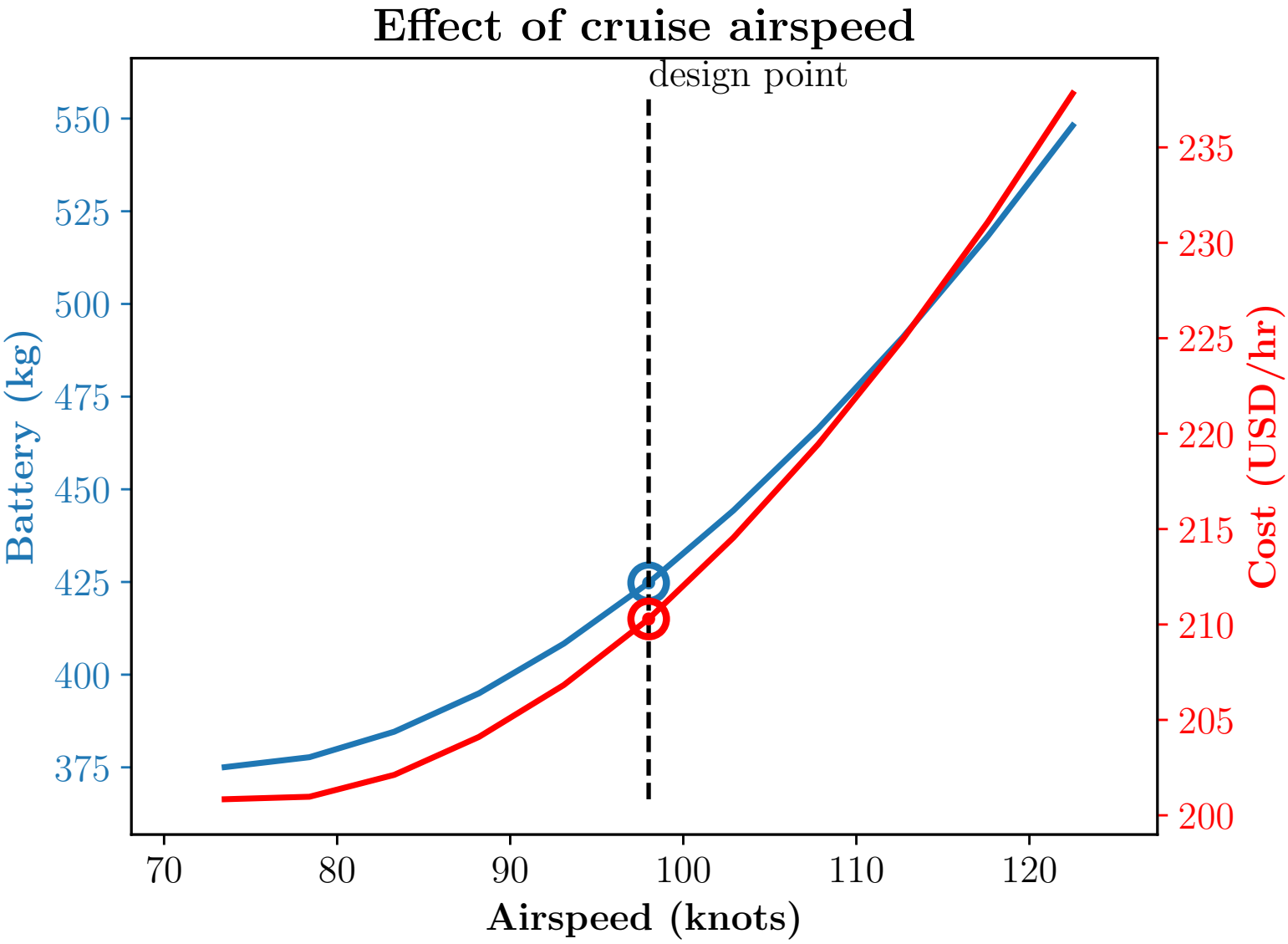
Figure 4: Battery state of charge along mission profile

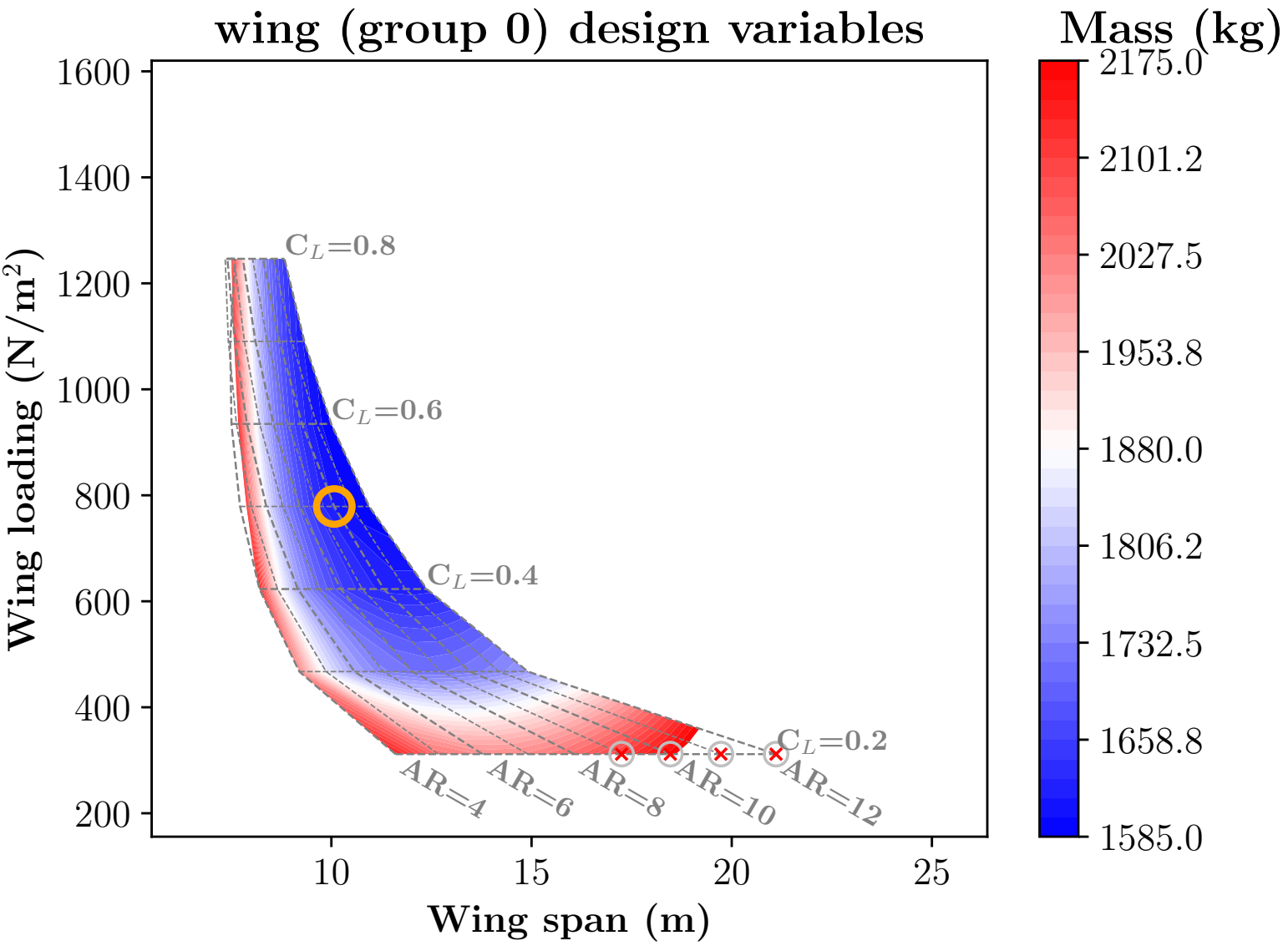
4. **Sensitivity analysis**: single and two-perturbation sensitivity studies can be launched using the following script:

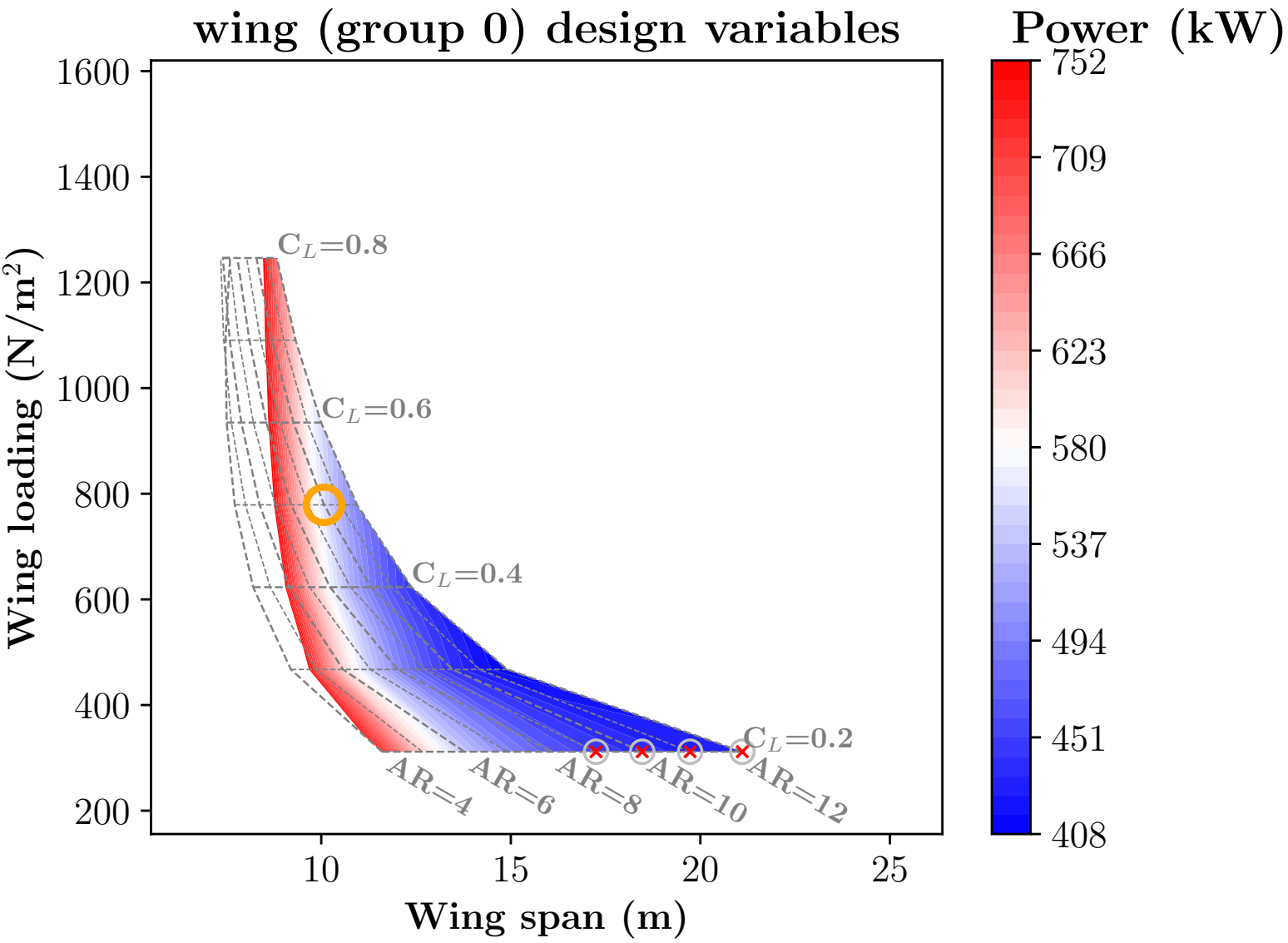
python3 Postprocessing/sensitivity_wrapper.py 32

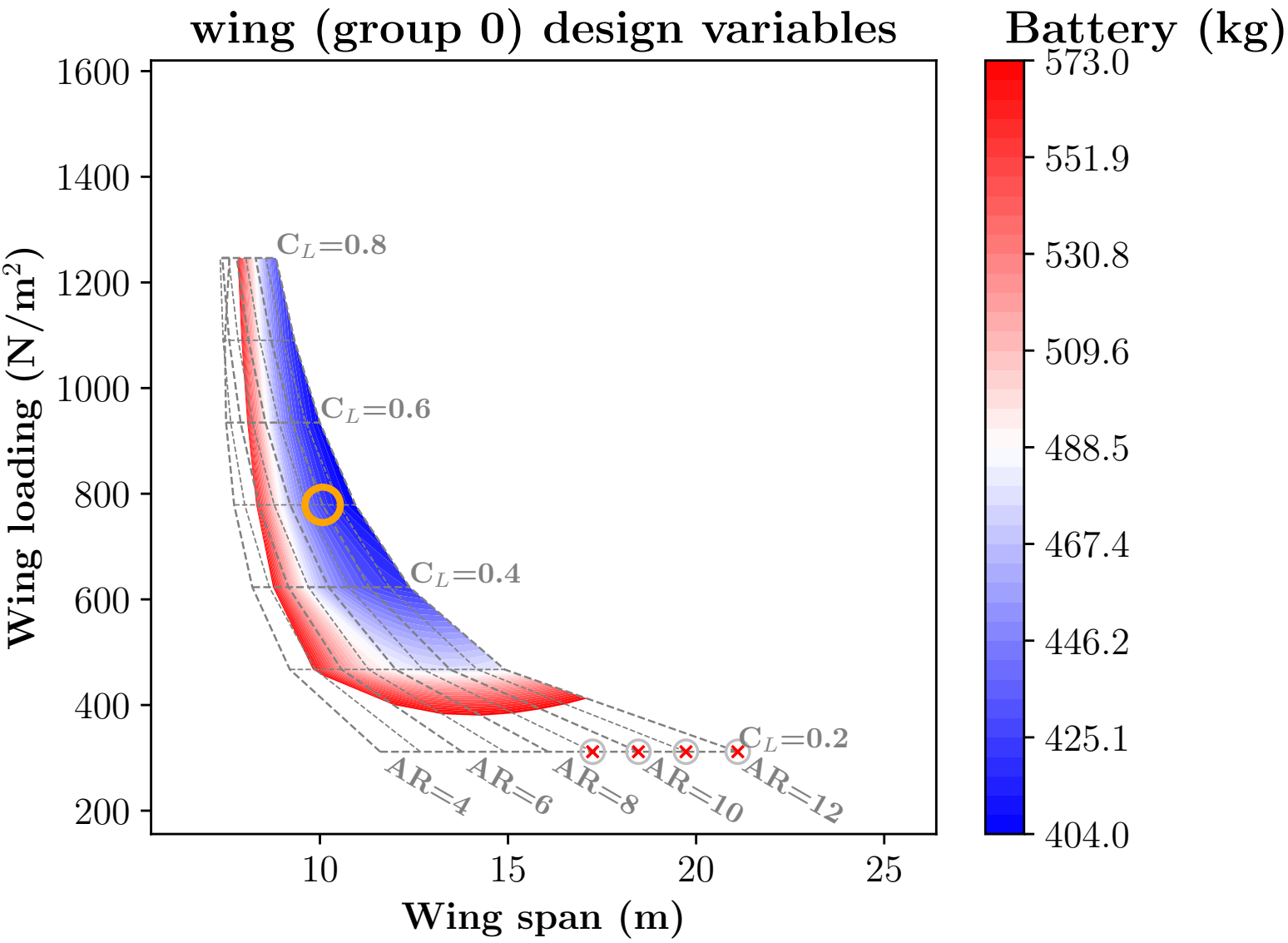
Here, 32 is the unique identifier for a converged vehicle design. The resulting plots from the sensitivity of the converged design to perturbations in target airspeed (for the same mission range) are shown in the first two plots. The next four plots show the effect of changing the cruise wing loading 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 ($\times$) inside a gray circle $\circ$. 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.

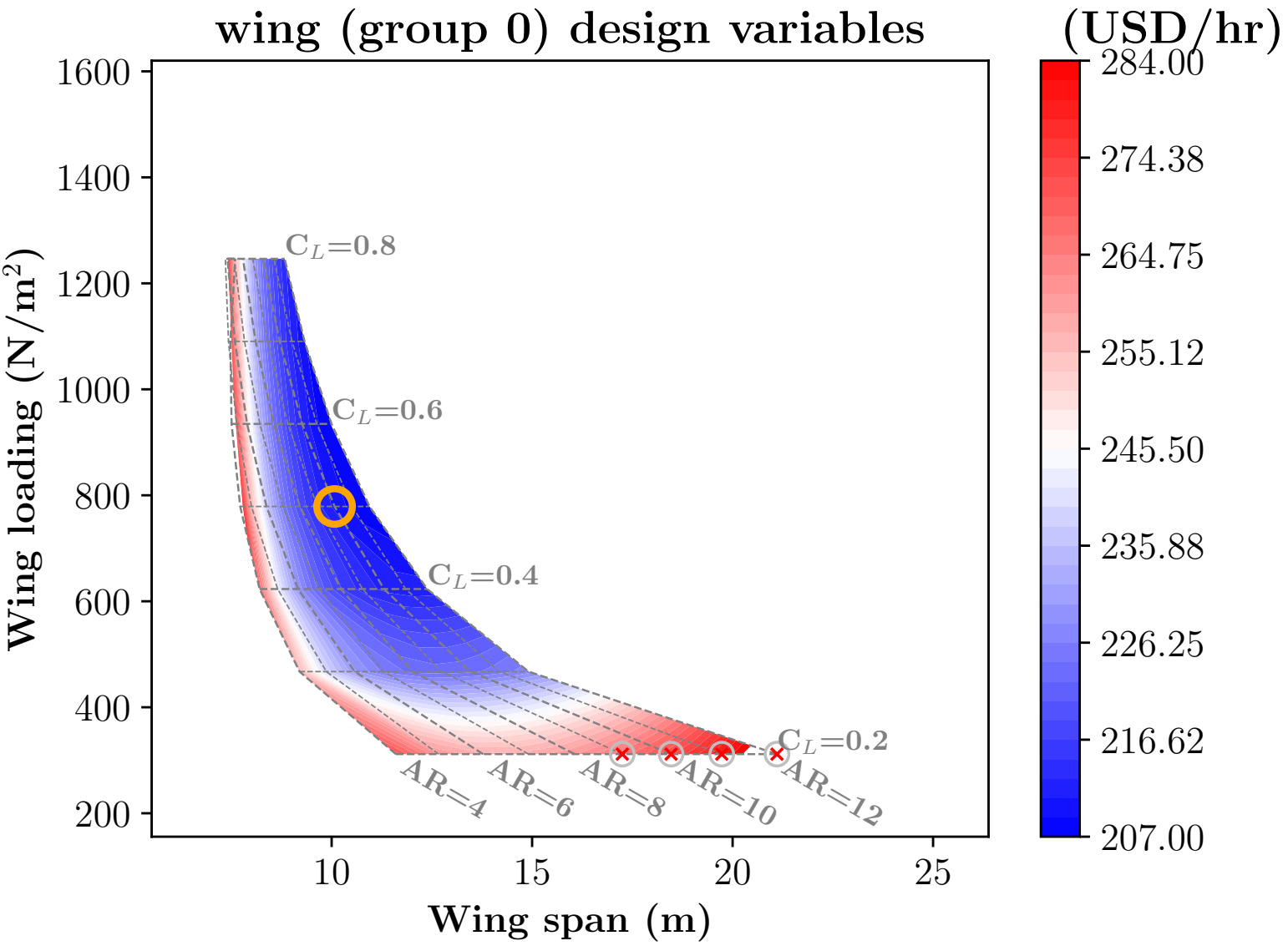

Effect of cruise airspeed
design point
1850
1800
1750
1700
1650
1600
1550
Mass (kg)
800
700
600
500
400
Power (kW)
70
80
90
100
110
120
Airspeed (knots)

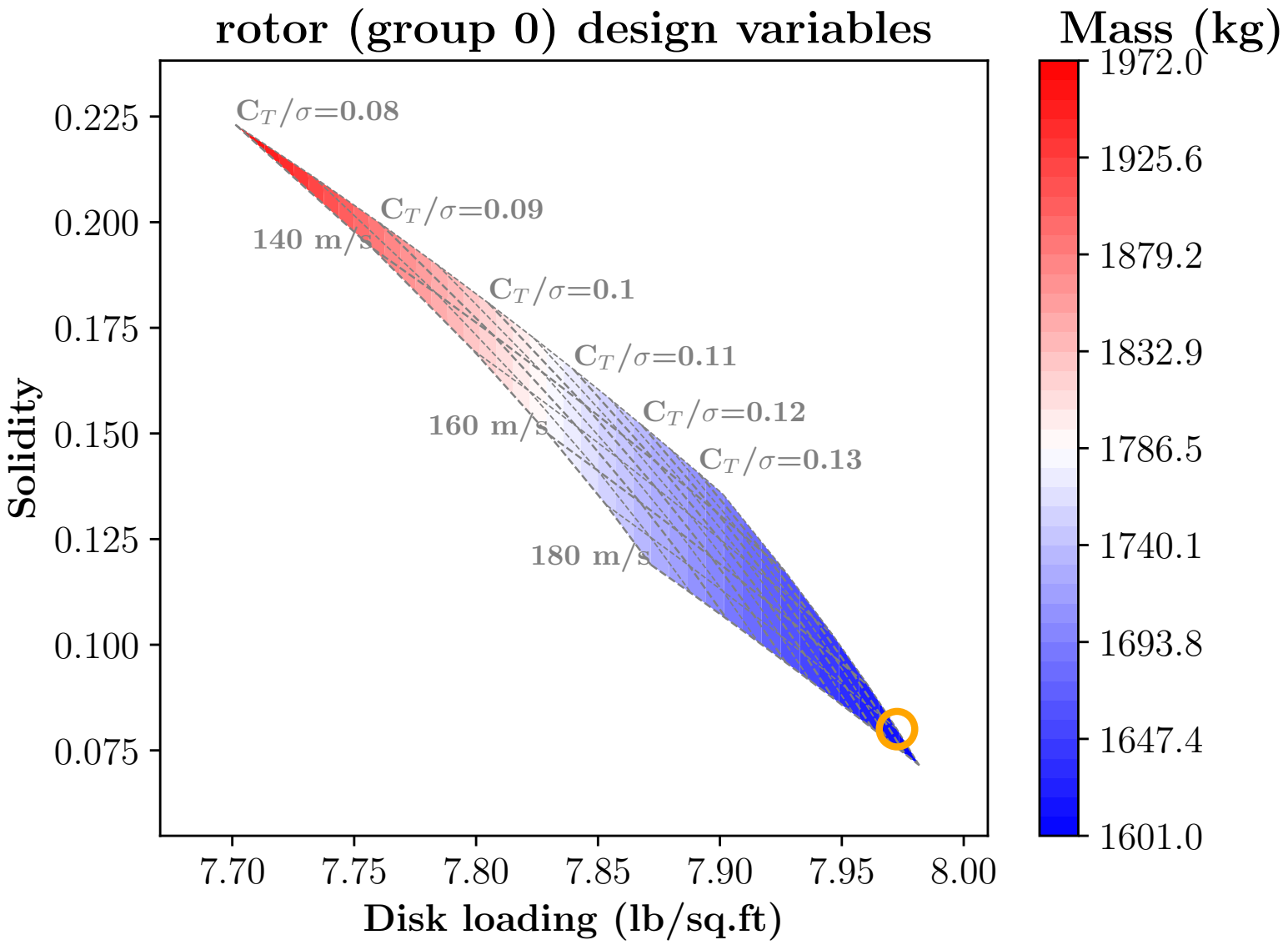

Effect of cruise airspeed
design point
Battery (kg)
Cost (USD/hr)
Airspeed (knots)
550
525
500
475
450
425
400
375
235
230
225
220
215
210
205
200
70
80
90
100
110
120

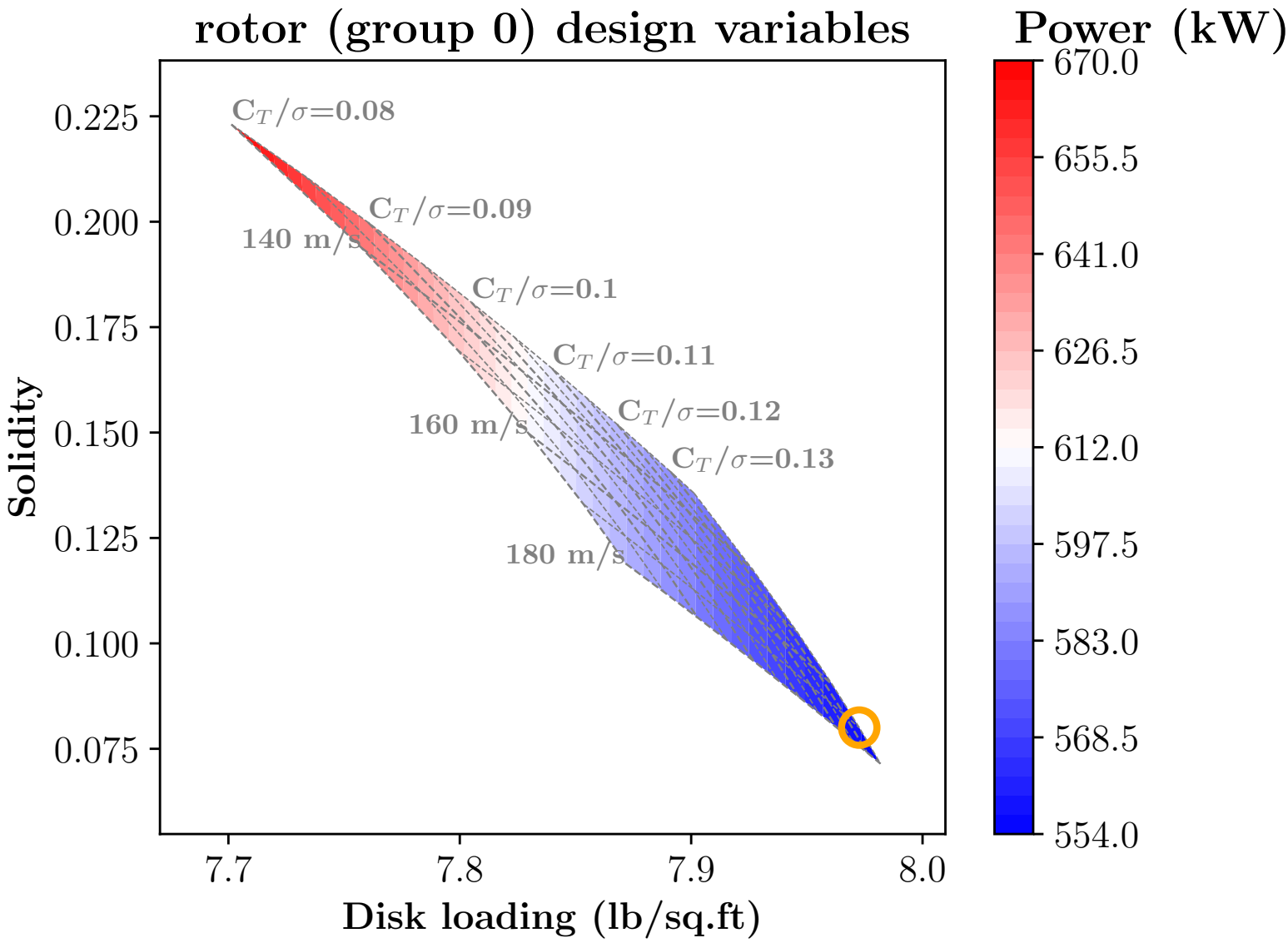

wing (group 0) design variables
Mass (kg)
2175.0
2101.2
2027.5
1953.8
1880.0
1806.2
1732.5
1658.8
1585.0
Wing loading (N/m^2)
1600
1400
1200
1000
800
600
400
200
10
15
20
25
Wing span (m)
C_L=0.8
C_L=0.6
C_L=0.4
C_L=0.2
AR=4
AR=6
AR=8
AR=10
AR=12

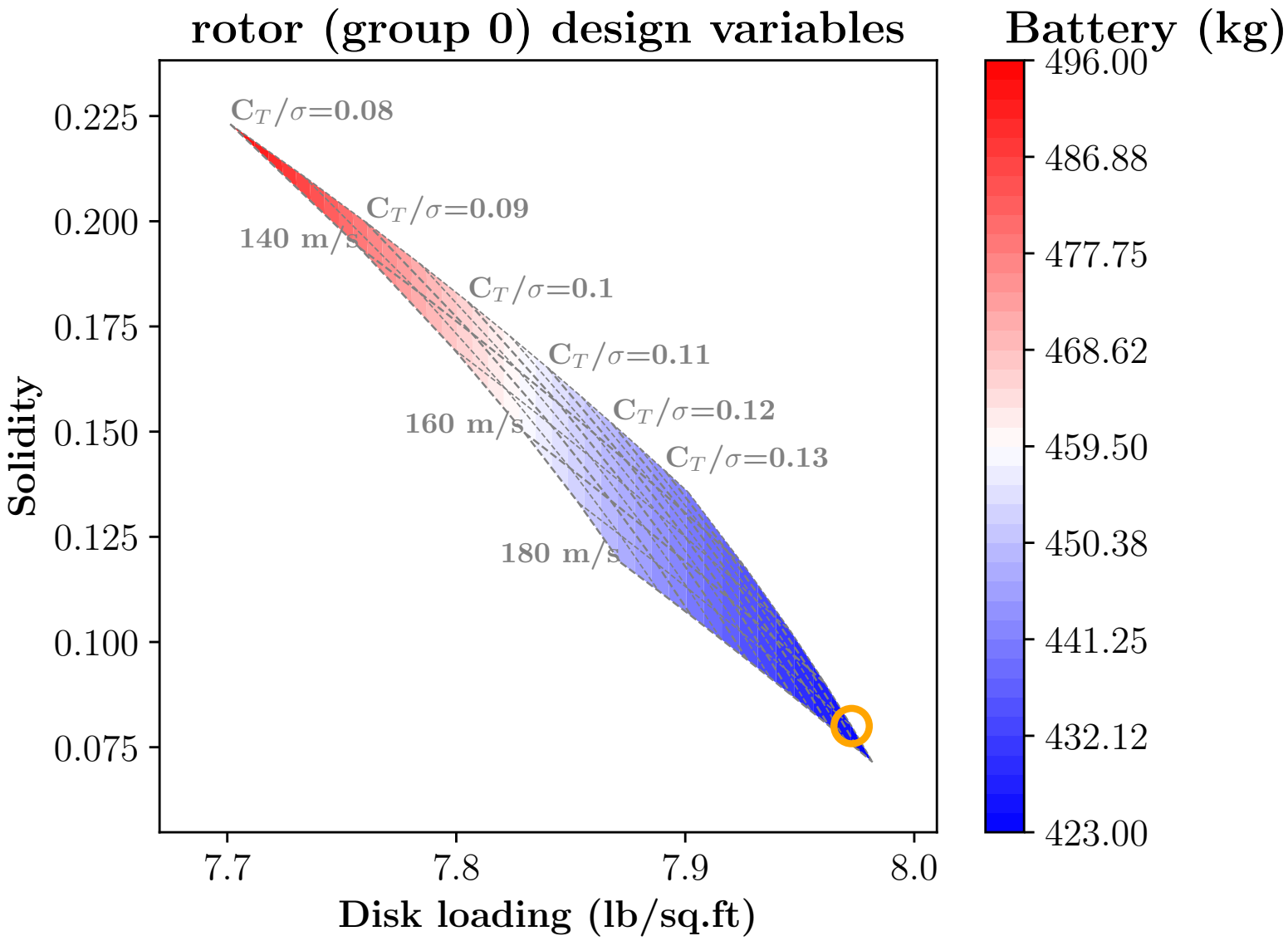

wing (group 0) design variables
Power (kW)
Wing loading (N/m^2)
Wing span (m)
1600
1400
1200
1000
800
600
400
200
10
15
20
25
752
709
666
623
580
537
494
451
408
C_L=0.8
C_L=0.6
C_L=0.4
C_L=0.2
AR=4
AR=6
AR=8
AR=10
AR=12

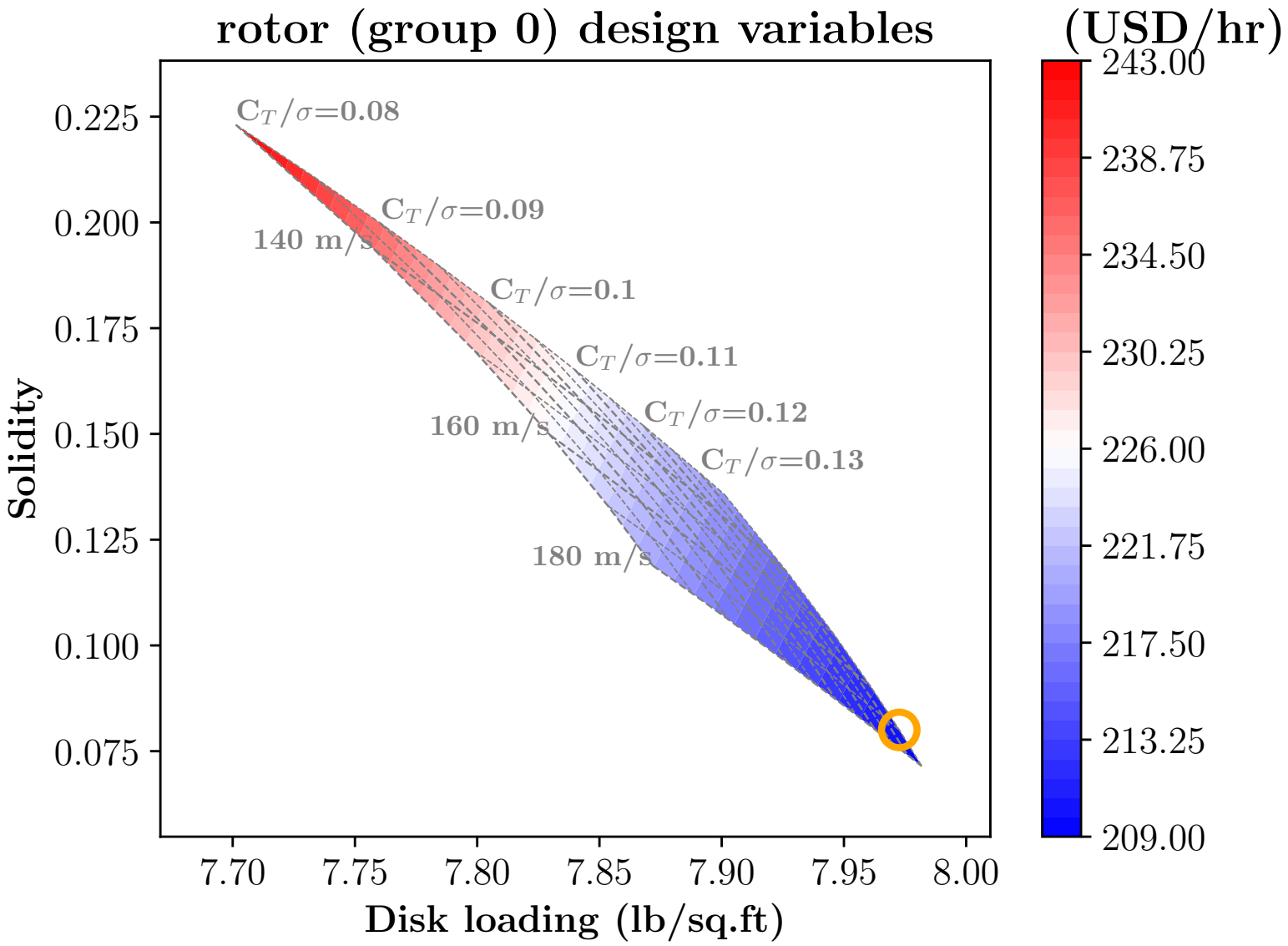

wing (group 0) design variables
Battery (kg)
Wing loading (N/m^2)
Wing span (m)
1600
1400
1200
1000
800
600
400
200
10
15
20
25
573.0
551.9
530.8
509.6
488.5
467.4
446.2
425.1
404.0
C_L=0.8
C_L=0.6
C_L=0.4
C_L=0.2
AR=4
AR=6
AR=8
AR=10
AR=12


wing (group 0) design variables
(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
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
7.70
7.75
7.80
7.85
7.90
7.95
8.00
Disk loading (lb/sq.ft)


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
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
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)
243.00
238.75
234.50
230.25
226.00
221.75
217.50
213.25
209.00
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
7.70
7.75
7.80
7.85
7.90
7.95
8.00
Disk loading (lb/sq.ft)

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**.

7 What Next?

This section details features of HYDRA that have not yet been documented, but may already exist in the code. It is a to-do list and reminder to document features.

7.1 Rotor attributes

1. Keyword recognition: tilting rotors
2. Keyword recognition: ducts
3. Keyword recognition: symmetric/asymmetric booms
4. Auto-percolation of tilting attribute from wings (where applicable)
5. Dissimilar rotor groups on same wing, fuselage

7.2 Wing and fuselage attributes

1. Keyword recognition: tilt-wings vs. fixed
2. Parametrically build FEM beam lattice representation around cabins

7.3 Powerplant attributes

1. Auto-addition of generator for turbine engine with electric transmission

7.4 Postprocessing

1. Generation of vehicle top view with rotor/wing attachment, show components