Preliminary Design

Fabian Grimm, Sumeet Kumar

Apr 27, 2022

CONTENTS:

CHAPTER

ONE

INPUT

Missions and helicopter configurations are provided as `.yaml`-files in '*python/data*'. Examples are given below.

1.1 Missions

```yaml
Name: name

Mission:

    # Constant within segment
    Duration: [0.1, 0.1] # h
    Payload: [0, 100] # kg
    Crew mass: [0, 0] # kg
    Flight speed: [10, 10] # m/s
    Temperature offset: [0, 0] # °C (deviation from ISA)
    Gravity: [9.76, 9.80] # m/s^2 (optional, default 9.81)

    # Linear with defined points between segments
    Height: [0, 0, 0] # m
```

1.2 Configurations

```yaml
Name: name

Main rotor:

    Number of blades: 4
    Kappa: 1.15 # (real/ideal induced power)
    Zero-lift drag coeff.: 0.011 # (avg., for profile power)
    Tip velocity: 213 # m/s
    Installation height: 2.5 # m (for in-ground effect)

Tail rotor:

    # Other rotor attributes are possible, but not used.
    Power fraction: 0.05
```

(continues on next page)

(continued from previous page)

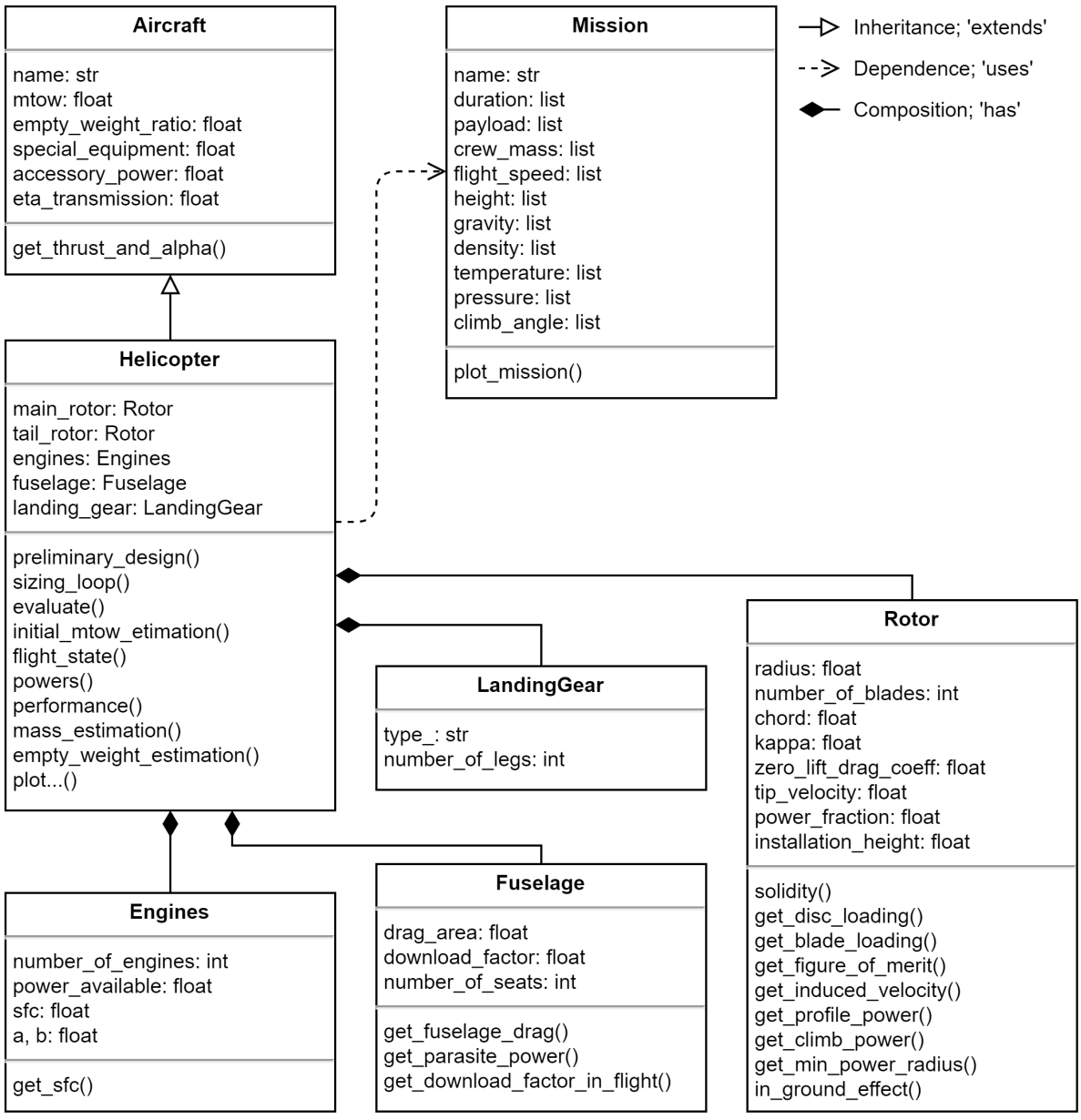
```yaml
Engines:

    Number of engines: 1
    Power available: 500_000 # W (per engine)
    SFC: 0.00035 # kg/Wh
    A: 40 # kg/h (A, B are optional, for power-dependent SFC)
    B: 0.00025 # kg/Wh

Fuselage:

    Download factor: 0.05 # (thrust increase due to fuselage download)
    Drag area: 1.5 # m^2
    Number of seats: 3 # (used in empty weight estimation)

Landing gear:

    Type: Rigid wheels # {Skids, Retractable wheels, Rigid wheels}
    Number of legs: 2 # (used in empty weight estimation)

Misc:

    Empty weight ratio: 0.5 # (EW / MTOW)
    Accessory power: 48_000 # W [p. 271]
    Transmission efficiency: 0.98 # (default 1.0)
    MTOW: 1500 # kg (not needed for sizing)
```

CHAPTER

TWO

STRUCTURE

2.1 UML


Aircraft
name: str
mtow: float
empty_weight_ratio: float
special_equipment: float
accessory_power: float
eta_transmission: float
get_thrust_and_alpha()
Mission
name: str
duration: list
payload: list
crew_mass: list
flight_speed: list
height: list
gravity: list
density: list
temperature: list
pressure: list
climb_angle: list
plot_mission()
Inheritance; 'extends'
Dependence; 'uses'
Composition; 'has'
Helicopter
main_rotor: Rotor
tail_rotor: Rotor
engines: Engines
fuselage: Fuselage
landing_gear: LandingGear
preliminary_design()
sizing_loop()
evaluate()
initial_mtow_etimation()
flight_state()
powers()
performance()
mass_estimation()
empty_weight_estimation()
plot...()
LandingGear
type_: str
number_of_legs: int
Rotor
radius: float
number_of_blades: int
chord: float
kappa: float
zero_lift_drag_coeff: float
tip_velocity: float
power_fraction: float
installation_height: float
solidity()
get_disc_loading()
get_blade_loading()
get_figure_of_merit()
get_induced_velocity()
get_profile_power()
get_climb_power()
get_min_power_radius()
in_ground_effect()
Engines
number_of_engines: int
power_available: float
sfc: float
a, b: float
get_sfc()
Fuselage
drag_area: float
download_factor: float
number_of_seats: int
get_fuselage_drag()
get_parasite_power()
get_download_factor_in_flight()

CHAPTER

THREE

MODULES

3.1 Aircraft

3.1.1 Thrust and angle of attack

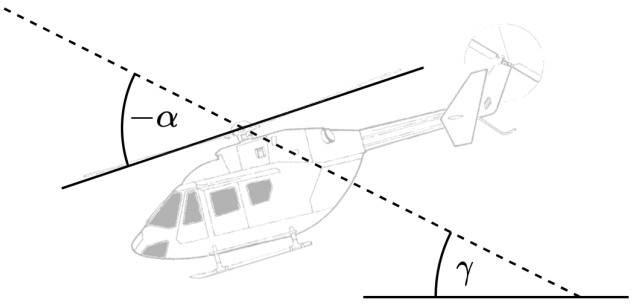


Fig. 1: Definition of the climb angle and angle of attack.

Thrust vector:

$$\vec{T} = -\vec{W} - \vec{D} = -\begin{bmatrix} 0 \\ -mg \end{bmatrix} - \begin{bmatrix} D\cos\gamma \\ -D\sin\gamma \end{bmatrix}$$

Angle of attack:

$$\alpha = \frac{\pi}{2} - \theta$$

with

$$\theta = \arccos(\bar{\vec{D}} \cdot \bar{\vec{T}}), \quad \bar{\vec{D}} = \frac{\vec{D}}{||\vec{D}||}, \quad \bar{\vec{T}} = \frac{\vec{T}}{||\vec{T}||}$$

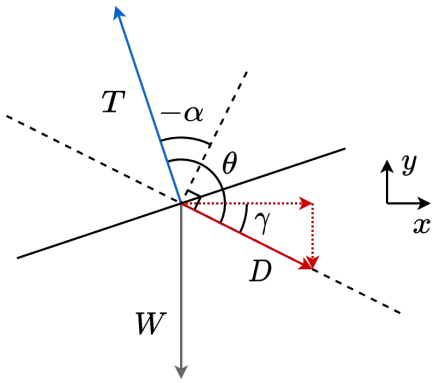


Fig. 2: Force balance between thrust, drag, and aircraft weight.

class aircraft.**Aircraft**(*filename: str*)

Bases: `object`

Vertical flight aircraft.

name

Name of the aircraft.

Type str

mtow

Maximum take-off weight; kg

Type float

empty_weight_ratio

Ratio of empty weight to maximum take-off weight.

Type float

special_equipment

Special equipment mass; kg

Type float

accessory_power

Accessory power; W

Type float

eta_transmission

Efficiency of the transmission.

Type float

get_thrust_and_alpha(*gross_weight*, *drag*, *gamma*, *gravity*)

Determine the thrust and angle of attack by means of a force balance with drag and weight vectors (isolated rotor).

Parameters

- **gross_weight** (*float*) – Aircraft mass; kg
- **drag** (*float*) – Drag force acting on the aircraft; N

- **gamma** (*float*) – Climb angle; rad
- **gravity** (*float*) – Gravitational acceleration; m/s^2

Returns

- *float* – Thrust; N
- *float* – Angle of attack; rad

3.2 Engines

class engines.**Engines**(*engine_data: dict*)

Bases: object

Turboshaft engines as aircraft components.

number_of_engines

Number of engines.

Type int

power_available

Power available per engine; W

Type float

sfc

Specific fuel consumption; kg/Wh

Type float

a, b

Engine parameters A and B, determining the fuel consumption.

Type float

get_sfc(*temperature_ratio*, *pressure_ratio*, *power*)

Calculate the power-dependent specific fuel consumption based on the engine parameters A and B. [p.310]

Parameters

- **temperature_ratio** (*float*) – Temperature ratio relative to mean sea level.
- **pressure_ratio** (*float*) – Pressure ratio relative to mean sea level.
- **power** (*float*) – Total power; W

Returns Specific fuel consumption; kg/Wh

Return type float

3.3 Fuselage

class fuselage.**Fuselage**(*fuselage_data: dict*)

Bases: object

Fuselage as an aircraft component.

drag_area
Drag area of the fuselage.

Type float

download_stationary
Relative increase in thrust due to obstructed downwash (stationary).

Type float

number_of_seats
Number of seats.

Type int

get_download_factor_in_flight(*advance_ratio*)
Calculate the download factor in forward flight, assuming a linear decline until advance ratio 0.5.

Parameters **advance_ratio** (*float*) – Advance ratio.

Returns Download factor in forward flight.

Return type float

get_fuselage_drag(*density*, *flight_speed*)
Calculate the fuselage drag based on a constant drag area.

Parameters

- **density** (*float*) – Density of the surrounding air; kg/m^3
- **flight_speed** (*float*) – Flight speed; m/s

Returns Fuselage drag; N

Return type float

get_parasite_power(*density*, *flight_speed*)
Calculate the parasite power due to fuselage drag in forward flight.

Parameters

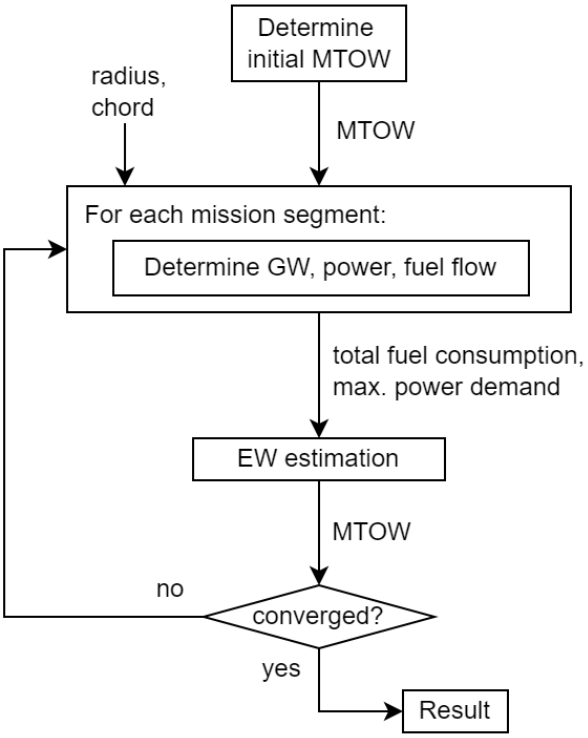
- **density** (*float*) – Density of the surrounding air; kg/m^3
- **flight_speed** (*float*) – Flight speed; m/s

Returns Parasite power; W

Return type float

3.4 Helicopter

3.4.1 Sizing loop



class helicopter.**Helicopter**(*filename: str*)

Bases: *aircraft.Aircraft*

Aircraft sub-class for conventional helicopter configurations containing methods for preliminary design and analysis.

Note: Requires *Main rotor*, *Tail rotor*, *Engines*, *Fuselage*, and *Landing Gear* section in the configuration file.

---

main_rotor
Main rotor.

Type *Rotor*

tail_rotor
Tail rotor.

Type *Rotor*

engines
Turboshaft engines.

Type *Engines*

fuselage
Fuselage.

Type *Fuselage*

landing_gear
Landing Gear.

Type *LandingGear*

empty_weight_estimation(*fuel_mass*, *power*)
Estimate the empty weight components of medium helicopters. [pp.408-416]

Parameters

- **fuel_mass** (*float*) – Fuel mass; kg, used for fuel tank sizing.
- **power** (*float*) – Total power; W, used for propulsion system and instrument sizing.

Returns Dictionary containing the empty weight components main rotor, tail rotor, fuselage and tail, landing gear, transmission, engines, fuel tanks, flight control systems, hydraulic systems, avionics, instruments, furnishing, air conditioning and anti-ice, loading and handling, and electrical systems. The sum represents the total empty weight.

Return type dict

evaluate(*mission:* mission.Mission)
Evaluate the flight state, powers, and performance throughout the mission for a fixed configuration. For each segment, the Gross Weight at the beginning of it is assumed.

Parameters **mission** (Mission) – Mission profile.

Returns Dictionary containing lists of the flight state, power composition, performance, fuel mass, and gross weight throughout the mission.

Return type dict

flight_state(*gross_weight*, *density*, *velocity*, *gamma*, *gravity*)
Determine flight state variables.

Parameters

- **gross_weight** (*float*) – Aircraft mass; kg
- **density** (*float*) – Density of the surrounding air; kg/m^3
- **velocity** (*float*) – Flight speed; m/s
- **gamma** (*float*) – Climb angle; rad
- **gravity** (*float*) – Gravitational acceleration; m/s^2

Returns Dictionary containing the density, velocity, climb angle, advance ratio, drag, angle of attack, thrust, induced velocity, download factor, blade loading, gravity, and gross weight.

Return type dict

initial_mtow_estimation(*duration*, *payload*, *crew_mass*)
Estimate the initial maximum take-off weight based on the installed power and total duration. Specific fuel consumption and empty weight ratio are assumed constant. [pp.90-91]

Parameters

- **duration** (*float*) – Mission duration; h
- **payload** (*float*) – Payload; kg
- **crew_mass** (*float*) – Crew mass; kg

mass_estimation(*power*, *fuel_mass*, *crew_mass*, *payload*, *use_ew_models*)
Determine the maximum take-off weight composition.

Parameters

- **power** (*float*) – Power; W
- **fuel_mass** (*float*) – Fuel mass; kg
- **crew_mass** (*float*) – Crew mass; kg
- **payload** (*float*) – Payload; kg
- **use_ew_models** (*bool*) – Use empirical empty weight models, otherwise the ratio to the maximum take-off weight is assumed constant.

Returns Dictionary containing the empty weight, fuel, payload, and crew mass. The sum represents the maximum take-off weight.

Return type dict

performance(*powers: dict*, *temperature*, *pressure*)
Determine performance parameters aside from the individual power components.

Parameters

- **powers** (*dict*) – Output of *powers()*.
- **temperature** (*float*) – Temperature of the surrounding air; K
- **pressure** (*float*) – Pressure of the surrounding air; Pa

Returns Dictionary containing the total power, the power requirement at mean sea level, specific fuel consumption, and the fuel mass flow.

Return type dict

plot_blade_loading(*blade_loading*, *advance_ratio*)
Plot the blade loading over the advance ratio, in order to evaluate the stall margin. [p.142]

Parameters

- **blade_loading** (*float*) – Blade loading.
- **advance_ratio** (*float*) – Advance ratio.

plot_empty_weight_pie(*empty_weight_parts: dict*)
Plot the empty weight composition as a pie chart.

Parameters **empty_weight_parts** (*dict*) – Output of *empty_weight_estimation()*.

plot_fuel_curve(*gross_weight*, *density*, *temperature*, *pressure*, *max_velocity=None*)
Plot the fuel mass flow and total power over a range of horizontal flight speeds. Annotations show the velocities for best range and endurance.

Parameters

- **gross_weight** (*float*) – Aircraft mass; kg
- **density** (*float*) – Density of the surrounding air; kg/m^3
- **temperature** (*float*) – Temperature of the surrounding air; K
- **pressure** (*float*) – Pressure of the surrounding air; Pa
- **max_velocity** (*float, optional*) – Upper limit for the forward flight speed to be considered in the plot. If not provided, the limit is defined by the installed power.

plot_gross_weight_over_time(*mission: mission.Mission*, *fuel_mass_per_segment: list*)
Plot the Gross Weight and fuel flow over the mission duration.

Parameters

- **mission** (*Mission*) – Mission profile.
- **fuel_mass_per_segment** (*list[float]*) – Fuel demand for each segment; kg

plot_masses_pie(*masses: dict*)
Plot the maximum take-off weight composition as a pie chart.

Parameters **masses** (*dict*) – Output of *mass_estimation()*.

plot_mtow_convergence(*mtow_list: list*)
Plot the maximum take-off weight over the iterations.

Parameters **mtow_list** (*list[float]*) – List of the maximum take-off weight; kg

plot_power_curves(*gross_weight*, *density*, *max_velocity=None*)
Plot the individual power components over a range of horizontal flight speeds.

Parameters

- **gross_weight** (*float*) – Aircraft mass; kg
- **density** (*float*) – Density of the surrounding air; kg/m^3
- **max_velocity** (*float, optional*) – Upper limit for the forward flight speed to be considered in the plot. If not provided, the limit is defined by the installed power.

plot_power_sweep_drag_area(*drag_area_range: tuple*, *gross_weight*, *density*, *max_velocity=75*)
Plot power curves for a range of drag areas.

Parameters

- **drag_area_range** (*tuple(min, max, N)*) – Drag area range; kg
- **gross_weight** (*float*) – Aircraft mass; kg
- **density** (*float*) – Density of the surrounding air; kg/m^3
- **max_velocity** (*float, optional*) – Maximum velocity to be considered in the plot; m/s

plot_power_sweep_gw(*gw_range: tuple*, *density*, *max_velocity=75*)
Plot power curves for a range of Gross Weights.

Parameters

- **gw_range** (*tuple(min, max, N)*) – Gross Weight range; kg
- **density** (*float*) – Density of the surrounding air; kg/m^3
- **max_velocity** (*float, optional*) – Maximum velocity to be considered in the plot; m/s

plot_power_sweep_height(*height_range: tuple*, *gross_weight*, *max_velocity=75*)
Plot power curves for a range of heights.

Parameters

- **height_range** (*tuple(min, max, N)*) – Height range; kg
- **gross_weight** (*float*) – Aircraft mass; kg
- **max_velocity** (*float, optional*) – Maximum velocity to be considered in the plot; m/s

plot_powers_pie(*powers: dict*)
Plot the power composition as a pie chart.

Parameters **powers** (*dict*) – Output of *powers()*.

plot_results(*df: pandas.core.frame.DataFrame*, *param: str*)
Plot one of the preliminary design parameters as a 3D surface over rotor radius and chord length.

Parameters

- **df** (*pd.DataFrame*) – Output of *preliminary_design()*.
- **param** (*str*) – z-axis parameter, key of *sizing_loop()* output.

powers(*flight_state: dict*)
Determine the power composition in the current flight state.

Parameters **flight_state** (*dict*) – Output of *flight_state()*.

Returns Dictionary containing the induced, profile, climb, parasite, tail rotor, and accessory power as well as transmission losses. The sum represents the total power.

Return type dict

preliminary_design(*mission:* mission.Mission, *radius_range: tuple*, *chord_range: tuple*, *conv_tol=0.0001*, *use_ew_models=False*, *status=True*)
Generate a dataframe for given ranges of the rotor radius and chord length, containing maximum take-off weight, fuel consumption, and other performance parameters.

Parameters

- **mission** (Mission) – Mission profile.
- **radius_range** (*tuple(min, max, N)*) – Range of the rotor radius variation; m
- **chord_range** (*tuple(min, max, M)*) – Range of the chord length variation; m
- **conv_tol** (*float, optional*) – Convergence tolerance of the sizing loop.
- **use_ew_models** (*bool, optional*) – Use empirical empty weight models, otherwise the ratio to the maximum take-off weight is assumed constant.
- **status** (*bool, optional*) – Print status updates, default True.

Returns Dataframe containing the results of *sizing_loop()* as rows.

Return type pd.DataFrame

sizing_loop(*mission:* mission.Mission, *conv_tol*, *use_ew_models*)
Determine the maximum take-off weight and performance iteratively for a given mission.

Parameters

- **mission** (Mission) – Mission profile.
- **conv_tol** (*float*) – Convergence tolerance.
- **use_ew_models** (*bool*) – Use empirical empty weight models, otherwise the ratio to the maximum take-off weight is assumed constant.

Returns Dictionary containing the radius, chord length, maximum take-off weight, fuel mass, empty weight ratio, solidity, disc loading, and blade loading.

Return type dict

3.5 Landing Gear

class landing_gear.**LandingGear**(*landing_gear_data: dict*)
Bases: object

Landing gear as an aircraft component.

type_
Type of landing gear.

Type str

number_of_legs
Number of legs.

Type int

3.6 Mission

class mission.**Mission**(*filename: str*)
Bases: object

Mission profile.

name
Name of the mission.

Type str

duration
Duration of each segment; h

Type list[float]

payload
Payload; kg

Type list[float]

crew_mass
Crew mass; kg

Type list[float]

flight_speed
Flight speed; m/s

Type list[float]

height
Height above ground; m, length n + 1 as defined between segments. In case of climb/descent, the half-way height is used to calculate density, temperature, and pressure.

Type list[float]

gravity
Gravitational acceleration; m/s^2

Type list[float]

density
Density of the surrounding air; kg/m^3

Type list[float]

temperature
Temperature of the surrounding air; K

Type list[float]

pressure
Pressure of the surrounding air; Pa

Type list[float]

climb_angle
Climb angle; rad

Type list[float]

plot_mission()
Plot the height and payload over the duration of the mission.

mission.**atmosphere**(*height*, *temp_offset*)
Calculate the density, temperature, and pressure at a given height based on the international standard atmosphere (ISA). Deviation from the ISA is considered via a temperature offset. [p. 278]

Parameters

- **height** (*float*) – Height; m
- **temp_offset** (*float*) – Temperature offset compared to the ISA; K or °C

Returns

- *float* – Density; kg/m^3
- *float* – Temperature; K
- *float* – Pressure; Pa

mission.**get_climb_angle**(*flight_speed*, *climb*, *duration*)
Calculate the climb angle.

Parameters

- **flight_speed** (*float*) – Flight speed; m/s
- **climb** (*float*) – Vertical distance; m
- **duration** (*float*) – Duration; h

Returns Climb angle; rad

Return type float

3.7 Rotor

class rotor.**Rotor**(*rotor_data: dict*)
Bases: object

Rotors as aircraft components.

radius
Rotor radius; m

Type float

`number_of_blades`
Number of blades.

Type int

`chord`
Chord length; m

Type float

`kappa`
Factor between ideal and real induced power.

Type float

`zero_lift_drag_coeff`
Zero-lift drag coefficient of the rotor blade, average.

Type float

`tip_velocity`
Tip velocity; m/s

Type float

`power_fraction`
Power fraction relative to the main rotor, if applicable.

Type float

`installation_height`
Distance between landing gear and rotor; m

Type float

`get_blade_loading`(*density*, *thrust*)
Calculate the blade loading CT / sigma. [p.133]

Parameters

- **density** (*float*) – Density of the surrounding air; kg/m^3
- **thrust** (*float*) – Thrust; N

Returns Blade loading.

Return type float

`get_climb_power`(*flight_speed*, *gamma*, *weight*)
Calculate the climb power due to the change in potential energy.

Parameters

- **flight_speed** (*float*) – Flight speed; m/s
- **gamma** (*float*) – Climb angle; rad
- **weight** (*float*) – Aircraft weight; N

Returns Climb power; W

Return type float

`get_disc_loading`(*thrust*)
Calculate the disc loading.

Parameters **thrust** (*float*) – Thrust; N

Returns Disc loading; N/m^2

Return type float

`get_figure_of_merit`(*ideal_induced_power*, *profile_power*)
Calculate the figure of merit.

Parameters

- **ideal_induced_power** (*float*) – Ideal induced power; W
- **profile_power** (*float*) – Profile power; W

Returns Figure of merit

Return type float

`get_induced_velocity`(*density*, *v_inf*, *alpha*, *thrust*)
Calculate the induced velocity iteratively in forward flight (not valid for low sink rate in axial flight). [p.253]

Parameters

- **density** (*float*) – Density of the surrounding air; kg/m^3
- **v_inf** (*float*) – Inflow velocity; m/s
- **alpha** (*float*) – Angle of attack; rad
- **thrust** (*float*) – Thrust; N

Returns Induced velocity; m/s

Return type float

`get_min_power_radius`(*density*, *thrust*)
Calculate the optimal rotor radius with respect to induced and profile power in hover. [p.103]

Parameters

- **density** (*float*) – Density of the surrounding air; kg/m^3
- **thrust** (*float*) – Thrust; N

Returns Optimal radius; m

Return type float

`get_profile_power`(*density*, *advance_ratio*)
Calculate the profile power. [p.258]

Parameters

- **density** (*float*) – Density of the surrounding air; kg/m^3
- **advance_ratio** (*float*) – Advance ratio.

Returns Profile power; W

Return type float

`in_ground_effect`(*induced_power*, *height*)
Calculate the induced power in ground effect (IGE) according to Hayden. [p.288]

Parameters

- **induced_power** (*float*) – Induced power, W
- **height** (*float*) – Height of the landing gear above ground; m

Returns Induced power in ground effect; W

Return type float

`property solidity`

Rotor solidity (rectangular approximation).

Note: The solidity is implemented as a property, which means it can be used as an attribute, calculated on call. This way it will always be in sync with the rotor radius and chord length.

Returns Solidity.

Return type float

CHAPTER

FOUR

RESSOURCES

This site was created using Sphinx, a tool that can generate the documentation for Python modules automatically based on the provided doc-strings. The following page describes how to add information or create a similar documentation from scratch.

4.1 Setup

```
pip install sphinx
pip install sphinx_rtd_theme
```

In the *docs* directory:

1. Initialize Sphinx. (Use default values by repeatedly pressing `Enter`)

   ```
   sphinx-quickstart
   ```

2. Edit **conf.py** to set up directories, extentions, themes, etc.

3. Create the reST files, one for each module and a combined page **modules.rst**

   ```
   sphinx-apidoc -o <output_folder> <py_folder>
   ```

4. Edit **index.rst** to include the desired pages

4.2 Maintain

1. Edit index, modules, or add new pages (in the reStructured-Text format)

2. Build HTML and PDF files

   In the *docs* directory:

   ```
   make html
   make latexpdf
   ```

4.3 Guides

- Getting started with Sphinx
- reStructured-Text

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX

N

P

R

S

T

Z