opensoar Documentation

Release 0.1.0

Matthijs Beekman

May 20, 2023

Contents

Contents:

CHAPTER 1

opensoar package

1.1 Subpackages

1.1.1 opensoar.competition package

Submodules

opensoar.competition.competition_day module

class opensoar.competition.competition_day.**CompetitionDay**(*name: str, date: datetime.date, plane_class: str, competitors: List[opensoar.competition.competitor.Competito task: opensoar.task.task.Task*)

Bases: object

This class contains the competition day information, equal fo all competitors.

analyse_flights(*classification_method: str*, *analysis_progress=None*, *skip_failed_analyses: bool = False*)

Parameters

- **classification_method** – method for detecting thermals. See FlightPhases for more info.
- **analysis_progress** – optional function to log the analysis progress. Should have the following signature: func(number_of_analyses, total_number_of_flights)
- **skip_failed_analyses** – if True, exceptions are caught during a failed analysis. a list is return with the competition ids of all failed analyses.

Returns

opensoar.competition.competitor module

class opensoar.competition.competitor.**Competitor**(*trace: List[T], competition_id: str = None, plane_model: str = None, ranking: Union[int, str] = None, pilot_name: str = None*)

Bases: object

All the information of one entry in a CompetitionDay. This encompasses information about the pilot, the plane and the gps trace.

analyse(*task*, *classification_method: str*)

phases

pilot_name

plane_model

trip

opensoar.competition.daily_results_page module

class opensoar.competition.daily_results_page.**DailyResultsPage**(*url*)

Bases: abc.ABC

Abstract Base Class for daily result pages. Specific implementation example: soaringspot.

download_flight(*igc_url: str*, *competition_id: str*) → str

Download flight and return file_path

Parameters

- **igc_url** –
- **competition_id** –

Returns

generate_competition_day(*target_directory: str*, *download_progress=None*, *start_time_buffer: int = 0*, *include_hc_competitors: bool = True*) → opensoar.competition.competition_day.CompetitionDay

Construct a CompetitionDay. Information is pulled from the overview table and from the igc files, which are automatically downloaded.

Parameters

- **include_hc_competitors** – optional argument for including contestants which fly 'Hors Concours', which means that they don't officially participate in the competition.
- **target_directory** – directory in which the igc files are saved
- **download_progress** – optional progress function. Should have the following signature: func(downloads, total_number_of_flights)
- **start_time_buffer** – optional relaxation on the start time in seconds. E.g. start_time_buffer = 10 means that a contestant can cross the start line 10 seconds before the official opening time

Returns

igc_directory

igc_file_name(*competition_id: str*) → str
Create igc file name from competition_id

Parameters **competition_id** –

Returns

igc_file_path(*competition_id: str*) → str
Construct file_path from competition_id

Parameters **competition_id** –

Returns

set_igc_directory(*target_directory*, *competition_name*, *plane_class*, *date*)

opensoar.competition.soaringspot module

Helper functions for SoaringSpot competitions. The files from SoaringSpot always contain task information, which can be used for competition analysis.

class opensoar.competition.soaringspot.**SoaringSpotDaily**(*url: str*)
Bases: *opensoar.competition.daily_results_page.DailyResultsPage*

Helper class for dealing with daily result pages which are published on the SoaringSpot platform.

generate_competition_day(*target_directory: str*, *download_progress=None*, *start_time_buffer: int = 0*, *include_hc_competitors: bool = True*) → opensoar.competition.competition_day.CompetitionDay
Construct a CompetitionDay. Information is pulled from the overview table and from the igc files, which are automatically downloaded.

Parameters

- **include_hc_competitors** – optional argument for including contestants which fly 'Hors Concours', which means that they don't officially participate in the competition.
- **target_directory** – directory in which the igc files are saved
- **download_progress** – optional progress function. Should have the following signature: func(downloads, total_number_of_flights)
- **start_time_buffer** – optional relaxation on the start time in seconds. E.g. start_time_buffer = 10 means that a contestant can cross the start line 10 seconds before the official opening time

Returns

opensoar.competition.soaringspot.**get_comment_lines_from_parsed_file**(*parsed_igc_file: dict*) → List[str]
In the parsed file, lines are split into source and comment. This function stiches them back together

opensoar.competition.soaringspot.**get_distance_correction**(*lseeyou_line: str*) → Optional[str]

opensoar.competition.soaringspot.**get_fixed_orientation_angle**(*lseeyou_line: str*) → float

opensoar.competition.soaringspot.**get_info_from_comment_lines**(*parsed_igc_file: dict*, *start_time_buffer: int = 0*) → Tuple[opensoar.task.task.Task, dict, dict]

There is specific contest information stored in the comment lines of the IGC files. This function extracts this information

opensoar.competition.soaringspot.**get_lat_long**(*lcu_line: str*) → Tuple[float, float]

Parameters **lcu_line** – line in soaringspot igc file starting with 'LCU::C'

Returns latitude, longitude in degrees

opensoar.competition.soaringspot.**get_sector_dimensions**(*lseeyou_line: str*) → Tuple[int, int, int, int]

opensoar.competition.soaringspot.**get_sector_orientation**(*lseeyou_line: str*, *number_of_waypoints: int*) → str

Parameters

- **lseeyou_line** – e.g. 'LSEEYOU OZ=-1,Style=1,R1=500m,A1=180'
- **number_of_waypoints** –

Returns

opensoar.competition.soaringspot.**get_task_rules**(*lseeyou_tsk_line: str*) → Tuple[datetime.time, datetime.timedelta, bool]

opensoar.competition.soaringspot.**get_waypoint**(*lcu_line: str*, *lseeyou_line: str*, *number_of_waypoints: int*) → opensoar.task.waypoint.Waypoint

Parameters

- **number_of_waypoints** –
- **lcu_line** – line in soaringspot igc starting with 'LCU::C
- **lseeyou_line** – line in soaringspot igc starting with 'LSEEYOU OZ'

Returns Waypoint

opensoar.competition.soaringspot.**get_waypoints**(*lcu_lines: List[str]*, *lseeyou_lines: List[str]*) → List[opensoar.task.waypoint.Waypoint]

Parameters

- **lcu_lines** – lines in soaringspot igc file starting with 'LCU::C'
- **lseeyou_lines** – lines in soaringspot igc starting with 'LSEEYOU OZ'

Returns list of Waypoints

opensoar.competition.strepla module

Helper functions for Strepla competitions. The files from Strepla always contain task information, which can be used for competition analysis.

opensoar.competition.strepla.**get_info_from_comment_lines**(*parsed_igc_file: dict*, *start_time_buffer: int = 0*)

opensoar.competition.strepla.**get_task_and_competitor_info**(*lscsd_lines: List[str]*, *lscsr_lines: List[str]*, *lscsa_lines: List[str]*) → Tuple[dict, dict]

opensoar.competition.strepla.**get_waypoint**(*lscs_line_tp: str*, *task_info: dict*, *n: int*, *n_tp: int*) → opensoar.task.waypoint.Waypoint

opensoar.competition.strepla.**get_waypoint_name_lat_long**(*lscs_line_tp: str*) → Tuple[str, float, float]

Parse LSCSCT line (LSCSCT:074 Main Lohr-M:N4959700:E00934900)

opensoar.competition.strepla.**get_waypoints**(*lscsc_lines: List[str]*, *task_info: dict*) → List[opensoar.task.waypoint.Waypoint]

Module contents

This package provides functionality to combine multiple flights in a CompetitionDay. Besides manually combining Competitors, it also provides high level interfaces with competition websites (e.g. SoaringSpot), which make it very easy to download and analyse all flights within a published CompetitionDay.

1.1.2 opensoar.task package

Submodules

opensoar.task.aat module

class opensoar.task.aat.**AAT**(*waypoints*, *t_min: datetime.timedelta*, *timezone: int = None*, *start_opening: datetime.time = None*, *start_time_buffer: int = 0*, *multistart: bool = False*)

Bases: *opensoar.task.task.Task*

Assigned Area Task.

apply_rules(*trace*)

t_min

opensoar.task.race_task module

class opensoar.task.race_task.**RaceTask**(*waypoints*, *timezone=None*, *start_opening=None*, *start_time_buffer=0*, *multistart=False*)

Bases: *opensoar.task.task.Task*

Race task.

apply_rules(*trace*)

calculate_task_distances()

determine_outlanding_distance(*outlanding_leg*, *fix*)

determine_outlanding_fix(*trace*, *fixes*, *start_fixes*, *enl_fix*)

determine_trip_distances(*fixes*, *outlanding_fix*)

determine_trip_fixes(*trace*)

finished_leg(*leg*, *fix1*, *fix2*)
Determines whether leg is finished.

total_distance

opensoar.task.task module

class opensoar.task.task.**Task**(*waypoints: List[opensoar.task.waypoint.Waypoint], timezone: int, start_opening: datetime.time, start_time_buffer: int, multistart: bool*)
Bases: object

Base Class for specific task implementations.

ENL_TIME_THRESHOLD = 30

ENL_VALUE_THRESHOLD = 500

determine_refined_start(*trace*, *fixes*)

static distance_moved_turnpoint(*distance*, *begin*, *end*, *moved_point*, *move_direction='reduce'*)

static distance_shortened_leg(*distance*, *current*, *currentP1*, *shortened_point*)

enl_time_exceeded(*enl_time*)

enl_value_exceeded(*fix*) → bool
Check whether ENL value is exceeded. :param fix: :return: returns False when not exceeded or when ENL information is not present in fix

finish

finished(*fix1*, *fix2*)

no_legs

no_tps

static set_orientation_angles(*waypoints*)

start

started(*fix1*, *fix2*)

waypoints

opensoar.task.trip module

class opensoar.task.trip.**Trip**(*task*, *trace*)
Bases: object

Realised

completed_legs()

fix_after_leg(*fix*, *leg*)

fix_before_leg(*fix*, *leg*)

fix_on_leg(*fix*, *leg*)
Return whether fix takes place within certain leg, excluding the boundaries :param fix: :param leg: :return:

outlanded()

outlanding_leg()

started_legs()

opensoar.task.waypoint module

class opensoar.task.waypoint.**Waypoint**(*name: str*, *latitude: float*, *longitude: float*, *r_min: float*, *angle_min: float*, *r_max: float*, *angle_max: float*, *is_line: bool*, *sector_orientation: str*, *distance_correction=None*, *orientation_angle=None*)

Bases: object

SEEYOU_SECTOR_MARGIN = 12

crossed_line(*fix1*, *fix2*)

fix

inside_sector(*fix*)

outside_sector(*fix*)

set_orientation_angle(*angle_start=None*, *angle_previous=None*, *angle_next=None*)

Module contents

This package provides the necessary classes for creating tasks and evaluating the performance of a competitor.

1.1.3 opensoar.thermals package

Submodules

opensoar.thermals.flight_phases module

class opensoar.thermals.flight_phases.**FlightPhases**(*classification_method: str*, *trace: list*, *trip=None*)

Bases: object

Container to combine the different flight phases (thermal and cruise) with helper methods for easy access.

all_phases(*leg: Union[int, str] = None*) → List[opensoar.thermals.flight_phases.Phase]
Obtain all phases (cruise and thermal).

Parameters **leg** – obtain only phases within specified leg (using int for leg), or obtain only phases within trip (using leg='all')

Returns

cruises(*leg: Union[int, str] = None*) → List[opensoar.thermals.flight_phases.Phase]
Obtain only cruise phases.

:param leg:can be 0, 1, ... or 'all'. Obtain only cruises within specified leg or all legs. :return:

thermals(*leg: Union[int, str] = None*) → List[opensoar.thermals.flight_phases.Phase]
Obtain only thermal phases.

Parameters **leg** – can be 0, 1, 2 or 'all'. Obtain only thermals within specified leg or all legs.

Returns

class opensoar.thermals.flight_phases.**Phase**(*is_cruise*, *fixes*)
Bases: tuple

fixes
Alias for field number 1

is_cruise
Alias for field number 0

opensoar.thermals.pysoar_thermal_detector module

class opensoar.thermals.pysoar_thermal_detector.**PySoarThermalDetector**
Bases: object

Detector taken from the PySoar project.

CRUISE_THRESHOLD_BEARINGRATE = 4

CRUISE_THRESHOLD_BEARINGTOT = 225

MINIMUM_BEARING_CHANGE_RATE = 0.01

THERMAL_THRESHOLD_BEARINGRATE = 4

THERMAL_THRESHOLD_BEARINGRATE_AVG = 2

THERMAL_THRESHOLD_DISTANCE = 1000

analyse(*trace*)

Module contents

This package contains the algorithms for thermal detection and a container class to combine the thermal- and cruise phases with helper methods for easy access.

1.1.4 opensoar.utilities package

Submodules

opensoar.utilities.helper_functions module

opensoar.utilities.helper_functions.**add_seconds**(*time: datetime.time*, *seconds: int*) → datetime.time
Add seconds to datetime.time object and return resulting datetime.time object.

Parameters

- **time** –
- **seconds** – not limited to 0-59.

Returns

opensoar.utilities.helper_functions.**add_times**(*start_time: datetime.time*, *delta_time: datetime.timedelta*)
Helper to circumvent problem that normal datetime.time instances can not be added. :param start_time: :param delta_time: :return:

opensoar.utilities.helper_functions.**altitude_gain_and_loss**(*fixes: List[dict], gps_altitude=True*)

opensoar.utilities.helper_functions.**both_none_or_same_float**(*var1*, *var2*)
Determine wheter both vars are the same. Either None or float

opensoar.utilities.helper_functions.**both_none_or_same_str**(*var1*, *var2*)
Determine wheter both vars are the same. Either None or float

opensoar.utilities.helper_functions.**calculate_average_bearing**(*bearing1*, *bearing2*)
Calculate the average bearing :param bearing1: bearing in degrees :param bearing2: bearing in degrees :return: average bearing in degrees

opensoar.utilities.helper_functions.**calculate_bearing_change**(*fix_minus2*, *fix_minus1*, *fix*)
Calculate bearing change between three fixes. :param fix_minus2: b-record from IGC file (dict with keys 'lat' and 'lon') :param fix_minus1: b-record from IGC file (dict with keys 'lat' and 'lon') :param fix: fix1: b-record from IGC file (dict with keys 'lat' and 'lon') :return: bearing change in degrees between -180 and +180 degrees. Return 0 when two of the of the fixes are the same.

opensoar.utilities.helper_functions.**calculate_bearing_difference**(*bearing1*, *bearing2*)
Calculate smallest difference from bearing 1 -> bearing2. :param bearing1: start bearing in degrees (0-360) :param bearing2: end bearing in degrees (0-360) :return: angle between -180 and +180 degrees.

opensoar.utilities.helper_functions.**calculate_destination**(*start_fix*, *distance*, *bearing*)

opensoar.utilities.helper_functions.**calculate_distance_bearing**(*fix1*, *fix2*, *final_bearing=False*)
Calculate bearing between fix1 and fix. By default the bearing is taking tangent to the great circle at fix1. :param final_bearing: switch to True results in taking the tangent at fix2. :param fix1: b-record from IGC file (dict with keys 'lat' and 'lon') :param fix2: b-record from IGC file (dict with keys 'lat' and 'lon') :return: distance in meters, bearing in degrees

opensoar.utilities.helper_functions.**calculate_time_differences**(*time1*, *time2*, *interval*)

opensoar.utilities.helper_functions.**dm2dd**(*degrees*, *minutes*, *cardinal*)
convert coordinate format with degrees and minutes to degrees

opensoar.utilities.helper_functions.**dms2dd**(*degrees*, *minutes*, *seconds*, *cardinal*)
convert coordinate format with degrees, minutes and second to degrees

opensoar.utilities.helper_functions.**double_iterator**(*lst*)
Create iterator with two values. E.g.: current, plus1 in a for loop

opensoar.utilities.helper_functions.**height_difference_fixes**(*fix1*, *fix2*, *gps_altitude=True*)

opensoar.utilities.helper_functions.**interpolate_fixes**(*fix1*, *fix2*, *interval=1*)
Create list of fixes between fix1 and fix2. Split is defined at time interval. Only time, latitude and longitude are interpolated. :param fix1: b-record from IGC file (dict with keys 'lat' and 'lon') :param fix2: b-record from IGC file (dict with keys 'lat' and 'lon') :param interval: interval between fixes in seconds :return: list of fixes between fix1 and fix2 with given interval

opensoar.utilities.helper_functions.**range_with_bounds**(*start: int*, *stop: int*, *interval: int*) → List[int]
Return list

opensoar.utilities.helper_functions.**seconds_time_difference**(*time1: datetime.time*, *time2: datetime.time*)
Determines the time difference between to datetime.time instances, mocking the operation time2 - time1 It is assumed that both take place at the same day. :param time1: :param time2: :return: time difference in seconds

opensoar.utilities.helper_functions.**seconds_time_difference_fixes**(*fix1*, *fix2*)

opensoar.utilities.helper_functions.**subtract_times**(*start_time: datetime.time*, *delta_time: datetime.time*)

opensoar.utilities.helper_functions.**total_distance_travelled**(*fixes: List[dict]*)
Calculates the total distance, summing over the inter fix distances

opensoar.utilities.helper_functions.**triple_iterator**(*lst*)
Create iterator with three values. E.g.: current, plus1, plus2 in a for loop

Module contents

This package contains helper functions, which support the other packages

1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

E

F

G

H

I

M

N

O

P

R

S

T

W