POUET Documentation

Release 0.2

Vivien Bonvin, Thibault Kuntzer

Feb 28, 2018

Contents:

POUET - Programing Observations Usefully at the Euler Telescope

This software helps you planning your observation nights at ANY telescope in the ESO-La Silla Observatory. But POULSO was a crappy software name, and both authors liked the Swiss Leonhard Euler Telescope a lot.

Why using this software and not one of the dozens of alternatives already existing out there? The strongest case is certainly that POUET provides a real-time analysis of the weather situation, including automatic clouds detection from the Danish telescope AllSky Camera.

Designed by observers for observers. With lots of love and hopefully bug-free.

CHAPTER 1

Quick links

1.1 pouet

1.1.1 pouet package

Subpackages

pouet.obsprogram package

Submodules

pouet.obsprogram.prog703 module

obsprogram.prog703.**get_exptime**(*attributes*, *obs_time*)

obsprogram.prog703.**observability**(*attributes*, *obs_time*)

pouet.obsprogram.prog714 module

obsprogram.prog714.**get_exptime**(*attributes*, *obs_time*)

obsprogram.prog714.**observability**(*attributes*, *obs_time*)

pouet.obsprogram.progbebop module

obsprogram.progbebop.**get_exptime**(*obj*, *obs_time*)

obsprogram.progbebop.**observability**(*attributes*, *obs_time*)

pouet.obsprogram.proglens module

obsprogram.proglens.**get_exptime**(*attributes*, *obs_time*)

obsprogram.proglens.**observability**(*attributes*, *obs_time*)

Module contents

pouet.LaSilla package

Submodules

pouet.config.LaSilla module

class config.LaSilla.**AllSky**

Station-specific class that handles the all sky image and its transformation of the sky.

Class constructor that saves some important all sky parameters as a class attribute.

get_image_coordinates(*az*, *elev*)

Converts the azimuth and elevation of a target in pixel coordinates

Parameters

- **az** – azimuth (in rad)
- **elev** – elevation (in rad)

Returns x and y position

get_mask(*ar*)

Returns the mask to apply on the AllSky hide unwanted features in the image. In the LaSilla case, to remove the danish and the text in the corners.

Parameters **ar** – original image (or at least an array with the same size). Used to get the image size.

get_radius(*elev*)

Method that computes the radius of a given elevation on the sky, in pixel.

Parameters **elev** – elevation (in radians)

Returns Radius, in px

class config.LaSilla.**WeatherReport**(*name='LaSilla'*)

This class is dedicated to recovering the weather report at the La Silla site and feeding the wind direction, wind speed, temperature and humidity back to pouet. It must contain at least a *get* method that returns the above variable.

Class constructor. Loads the LaSilla.cfg configuration file and saves it as attribute.

Parameters **name** – name of the cfg file, only included for completeness.

get(*debugmode*, *FLAG=-9999*)

Get method that reads the weather reports off the web. In the LaSilla case, it download a *meteo.last* and interprets the data.

Parameters

- **debugmode** – whether or not POUET is in debugmode. If true, it ought to return some static and dummy data

- **FLAG** – what to return in case the weather report cannot be downloaded or treated. Currently, POUET expect -9999 as a placeholder.

Returns Wind direction, speed, temperature and humidity

> **Warning:** Such a method *must* return the following variables in that precise order: wind direction, wind speed, temperature and humidity

Module contents

Submodules

pouet.clouds module

class clouds.**Clouds**(*name*, *fimage=None*, *debugmode=False*)

This class loads and analyses an all sky image of the Sky and returns an observability map that can be used by another method to advise the observer on the sky quality.

Note This module can also be used to explore older images, see the example code in *__main__()*.

Initialises the class

Parameters

- **fimage** – (default is None) filename of the all sky image to analyse
- **name** – (default is "LaSilla") name of the location to load the right config file
- **debugmode** – whether or not POUET is in debugmode. If true, it ought to return some static and dummy data

Type string

detect_stars(*sigma_blur=1.2*, *threshold=8*, *neighborhood_size=20*, *fwhm_threshold=5*, *meas_star=True*, *return_all=False*)

Analyses the images to find the stars.

> **Todo:** describe algo in more details

Parameters

- **sigma_blur** – Sigma of the Gaussian kernel (in px)
- **threshold** – threshold of detection
- **neighborhood_size** – footprint of the maximum and minimum filters
- **fwhm_threshold** – select objects smaller than this fwhm
- **meas_star** – if not interested in computing fwhm for all objects bypass and return positions of objects
- **return_all** – if *True*, returns the positions of stars + detected objects otherwise only stars

get_observability_map(*x*, *y*, *threshold=50*, *filter_sigma=2*)

Returns an observability map (an image of the sky)

Parameters

- **x** – x coordinates of detected stars
- **y** – y coordinates of detected stars
- **threshold** – distance threshold in px of stars such that we have observations
- **filter_sigma** – sigma of the Gaussian kernel, default=2

Returns an observability map with the same dimension as the input image.

retrieve_image()
Downloads the current all sky from the server and saves it to disk. The url of the image is retrived from the corresponding configuration file.

update(*donotdownloadtime=1.5*)
Downloads the image, detects the stars and returns the observability map

Parameters **donotdownloadtime** – minimum elapsed time before re-downloading an image, default: 1.5 min.

Returns an observability map with the same dimension as the input image. Then, use all sky get image coordinate method to retrieve observability for a given target.

clouds.**fwhm**(*data*, *xc*, *yc*, *stampsize*, *show=False*, *verbose=False*)
Fits a 2D Gaussian profile and returns the FWHM in px

Parameters

- **data** – the stamp containing the image
- **xc** – centroid x position
- **yc** – centroid y position
- **stampsize** – size of nominal square stampe
- **show** – shows a diagnostic plot of the fit
- **verbose** – should I speak?

Returns fwhm in px

clouds.**gaussian**(*params*, *stamp*, *stampsize*)
Returns a 2D gaussian profile

Parameters

- **params** – a list of the Gaussian profile: centroid (*x* and *y*), sigma (*std*), intensity (*i0*) and constant background (*sky*)
- **stamp** – the original imagette
- **stampsize** – the size of the stamp to draw the Gaussian in.

Returns residues of the gaussian - *stamp*

clouds.**loadallsky**(*fnimg*, *station*, *return_complete=False*)
Loads the all sky image

Parameters **return_complete** – returns the masked image and the unmasked image

Returns Masked image or masked image and original image. Note that if cannot download, returns *None* or *None, None*.

clouds.**rgb2gray**(*arr*)
Converts from RGB to gray.

Note: The all sky in LaSilla is not RGB, but JPG is is 3D...

pouet.main module

pouet.meteo module

Define the METEO class and related functions

Meteo is an object containing all the external weather condition (wind speed and direction, temperature, moon position, clouds pattern,...) It is the only object that interact outside POUET, i.e. communicate with website to get the meteo,...

Observables interact only with Meteo to get their constaints (position to the moon, angle to wind, ...)

Warning: Do *NOT* call this *site.py* otherwise it clashes with some weird Python system package.

class meteo.**Meteo**(*name='uknsite'*, *time=None*, *moonaltitude=None*, *moonazimuth=None*, *sunaltitude=None*, *sunazimuth=None*, *winddirection=-1*, *windspeed=-1*, *cloudscheck=True*, *fimage=None*, *debugmode=False*)
Class to hold the meteorological conditions of the current night and the location of the site

Typically, a Site object is created when POUET starts, and then update itself every XX minutes

get_AzAlt(*alpha*, *delta*, *obs_time=None*, *ref_dir=0*)
idea from http://aa.usno.navy.mil/faq/docs/Alt_Az.php

Compute the azimuth and altitude of a source at a given time (by default current time of execution), given its alpha and delta coordinates.

get_moon(*obs_time=<Time object: scale='utc' format='datetime' value=2018-02-28 13:39:53.130723>*)

get_nighthours(*obs_night*, *twilight='nautical'*)
return a list of astropy Time objects, corresponding to the different hours of the obs_night

get_sun(*obs_time=<Time object: scale='utc' format='datetime' value=2018-02-28 13:39:53.130849>*)

get_telescope_params()

get_twilights(*obs_night*, *twilight='nautical'*)
return a list of astropy Time objects: twilight in, twilight out

Warning: The twilight times in PyEphem don't take into account the altitude ! See *https://github.com/brandon-rhodes/pyephem/issues/102*

update(*obs_time=<Time object: scale='utc' format='datetime' value=2018-02-28 13:39:53.130448>*, *minimal=False*)
minimal=True update only the moon and sun position. Useful for predictions (as you can't predict the clouds or winds, no need to refresh them)

updateclouds()
Excecutes the clouds code, if map not available, saves None to cloudmap

updatemoonpos(*obs_time=<Time object: scale='utc' format='datetime' value=2018-02-28 13:39:53.129664>*)

updatesunpos(*obs_time=<Time object: scale='utc' format='datetime' value=2018-02-28 13:39:53.129998>*)

updateweather()

pouet.obs module

Define the Observable class, the standard object of pouet, and related functions

class obs.**Observable**(*name='emptyobservable'*, *obsprogram=None*, *attributes=None*, *alpha=None*, *delta=None*, *minangletomoon=None*, *maxairmass=None*, *exptime=None*)

Class to hold a specific target from any observational progamm

Unvariable parameters are defined at initialisation

Variable parameters (distance to moon, azimuth, observability,...) are undefined until associated methods are called

compute_airmass(*meteo*)

Computes altitude and azimuth of the observable.

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

Returns None

compute_altaz(*meteo*)

Computes altitude and azimuth of the observable.

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

Returns None

compute_angletomoon(*meteo*)

Computes the distance to the moon

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

Returns None

compute_angletosun(*meteo*)

Computes distance to the Sun

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

Returns None

compute_angletowind(*meteo*)

Computes the angle to wind

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

Returns None

compute_observability(*meteo*, *cwvalidity=30*, *displayall=True*, *cloudscheck=True*, *verbose=True*, *future=False*)

Update the status using *update()*. Compute the observability, a value between 0 and 1 that tells if the target can be observed at a given time. Also define flags for each parameter (moon, wind, etc...)

The closer to 1 the better 0 is impossible to observe

Parameters

- **meteo** – a Meteo object, whose time attribute has been actualized beforehand

- **displayall** – boolean

copy()

Returns Observable: a python deep copy of the current observable

is_cloudfree(*meteo*)

Computes whether the pointing direction is cloudy according to the altaz coordinates in memory

Parameters **meteo** – a Meteo object, whose cloudmap attribute has been actualized beforehand

todo: instead of taking altaz coordinates in memory, shouldn't we use meteo.time to recompute altaz on the fly?

Returns None

Note is_cloudfree is actualized with 0: cloudy or 1: no clouds. If unavailable, returns 2: connection error, if error during computation of observability from map: 3

update(*meteo*)

Update the observable parameters according to the meteo object passed: altitude, azimuth, angle to wind, airmass, angle to moon and angle to sun.

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

Returns None

obs.**rdbexport**(*observables*, *filepath*, *namecol=1*, *alphacol=2*, *deltacol=3*, *obsprogramcol=4*, *verbose=False*)

Export a list of observables as an rdb catalogue, to be read again later

obs.**rdbimport**(*filepath*, *namecol=1*, *alphacol=2*, *deltacol=3*, *obsprogramcol=4*, *startline=1*, *obsprogram=None*, *verbose=False*)

Import an rdb catalog into a list of observables

obs.**showstatus**(*observables*, *meteo*, *obs_time=None*, *displayall=True*, *cloudscheck=True*)

Using a list of observables, print their observability at the given obs_time. The moon position and all observables are updated according to the given obs_time. The wind is always taken at the current time.

displayall = True allows all the targets to be displayed, even if they cannot be observed

pouet.run module

High-level functions around meteo and obs Running the script should provide a minimal text output

run.**hide_observables**(*observables*, *criteria*)

run.**refresh_status**(*meteo*, *observables=None*, *minimal=False*, *obs_time=None*)

Refresh the status

Parameters

- **observables** –
- **obs_time** –

Returns

run.**retrieve_obsprogramlist**()

run.**startup**(*name='LaSilla'*, *cloudscheck=True*, *debugmode=False*)

Initialize meteo

Returns

pouet.util module

Useful functions and definitions

util.**check_value**(*var*, *flag*)

util.**elev2airmass**(*el*, *alt*, *threshold=10.0*)
Converts the elevation to airmass. :param elevation_deg: elevation [radians] :param alt: altitude of station [m] :return: air mass This is the code used for the Euler EDP at La Silla.

util.**excelimport**(*filename*, *obsprogram=None*)

util.**grid_points**(*res_x=400*, *res_y=200*)
Generates grid points on the sky

util.**hilite**(*string*, *status*, *bold*)
Graphism: colors and bold in the terminal

util.**load_station**(*name*)

util.**readconfig**(*configpath*)
Reads in a config file

util.**readpickle**(*filepath*, *verbose=True*)
I read a pickle file and return whatever object it contains. If the filepath ends with .gz, I'll unzip the pickle file.

util.**takeclosest**(*dico*, *key*, *value*)
Assumes dict[key] is sorted. Returns the dict value which dict[key] is closest to value. If two dict[key] are equally close to value, return the highest (i.e. latest). This is much faster than a simple min loop, although a bit more tedious to use.

util.**time2hhmm**(*obstime*)

util.**writepickle**(*obj*, *filepath*, *verbose=True*, *protocol=-1*)
I write your python object obj into a pickle file at filepath. If filepath ends with .gz, I'll use gzip to compress the pickle. Leave protocol = -1 : I'll use the latest binary protocol of pickle.

pouet.plots module

plots.**plot_airmass_on_sky**(*target*, *meteo*, *ax=None*)
Plots the airmass evolution on the sky of a given target at a given time.

Parameters

- **target** – a *pouet.obs.Observable* class instance
- **meteo** – a *pouet.meteo.Meteo* class instance
- **ax** – the matplotlib axis to plot on. If None, then plot on a new figure

plots.**plot_target_on_sky**(*target*, *figure=None*, *northisup=True*, *eastisright=False*, *boxsize=None*, *survey='DSS'*, *cmap='Greys'*)
Uses astroquery (hopefully soon accessible from *astropy.vo*) to plot an image of the target

plots.**shownightobs**(*observable*, *meteo*, *obs_night=None*, *savefig=False*, *dirpath=None*, *verbose=False*)
Plot the observability of one observable along the night

#todo: add the option to be returned in an Axes object instead of plotting

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

m

o

p

r

u

Index

R

S

T

U

W