pyarchi

Release 1

Oct 01, 2022

User Guide

This project is part of the work developped during my MSc's thesis, titled "An expansion to the CHEOPS mission official pipeline".

In this project we proposed to expand the functionality of the CHEOPS mission official data reduction pipeline (DRP), in a project named "An expansion foR the CHeops mission pipelIne - ARCHI ", to maximize the scientific gains from its operation.

CHAPTER 1

Installation

CHAPTER 2

Configuration

In order to configure the behavior of the pipeline, one has to edit the configuration.yaml file, shown in here .

Even though everything can be configured throughout this file, it's also possible to change any parameter from within python, as described in *Photometry Controller*. Before the routines start, the passed parameters are subjected to an analysis, to insure that all of the previded data has the correct data types.

2.1 General configurations

- **base_folder :**
  - Path to the data folder, eg: "/home/Kamuish/data_files/CHEOPSim_job6201/"
- **optimized_factors :**
  - path to save and load the optimized "increase factors"
- **official_curve: Uses one of the DRP's light curves to compare the results against our owns; The comparison is not made, if**
  - OPTIMAL
  - DEFAULT
  - RSUP
  - RINF
- **data_type: either choose between assuming the file structure from the simulated data or from the "real" data**
  - real
  - simulated
- **method: Format of the region in which the image will be analysed (for each star).**
  - "circle" -> uses a circular opening around each star;

  - "shape" -> uses the contour of the star to create a mask

- **initial_detect: How the centers are tracked**
  - fits : uses information from the Star Catalogue file, provided by the DRP
  - dynam : uses image processing;
- **detect_mode: How the stars are tracked**
  - "static" - Extracts the inicial position from the FITS file and, for each frame, they are rotated by the difference of the rotation angle of the satellite;
  - "dynam" - in each image the contours are detected and the center of each is extrapolated using the contour's moments;
  - "offsets" - uses the "static" process and afterwards calculates the corrections for the center star position and applies it to all the points.
- **uncertainties**
  - Enable the calculation of the uncertainties
- **grid_bg:**
  - Size of the grid used for improved resolution. If it's set to zero then no background grid is used. Otherwise, it needs to be a multiple of the image's sizes (200 px)
- **repeat_removal:** Number of times to remove the brightest mask in the image
- **optimize:**
  - If it's 1 then the data files will be pre-processed to find the optimal radius, i.e. , the radii that minimizes the dispersion for each star; If it's 0 the radii used will be the ones from the optimized_radius.json file
- **optimization_extensions:**
  - number of times that the optimization process is expanded
- **optim_processes:**
  - Number of cores to use for the optimization process. if it's running on a laptop, it's recommended to use 2 or 3
- **fine_tune_circle:**
  - fine search for the mask best size. The step keyword has no effect over this process
- **val_range:**
  - values that the mask size can take; used in the optimization process
- **step:**
  - Step between mask sizes for the optimization process. Recommended to be 1
- **headless:**
  - Is the code running on a headless server
- **low_memory:**
  - Activate low memory mode. Recommended to be active when working with the background grids or larger data sets.
- **CDPP_type: Which noise metric to use;**
  - K2

  - Can be a function that implements a custom noise metric. should accept the flux as first input and time as the second. Only returns the noise metric

- **debug:**
  - If it's 1 then we will have a comparison between the data obtained using this method and the one from the official pipeline. THis comparison is only done for the center star, since the official pipeline does not work for the outer stars.
- **plot_realtime:**
  - If it's 1 we will see the images with the region under study marked with a circle. When it's set to 1 the program shows a performance hit since the plot uses up some computational power. If the optimize parameter is set to 1 this one is set to 0 during the optimization process.
- **save_gif:**
  - If set to one archi generates a gif with the stars and the masks used for each one of them
- **show_results:**
  - Shows the photometric curve of each star
- **export_text:**
  - export the lightcurve of the central star to a text file, including the MJD_TIME and ROLL_ANGLE
- **export_fit :**
  - export, to FITS file, the photometric curves of all stars, calibration information and the relevant configuration values.

CHAPTER 3

A Quick Overview

In this package, as we have stated, we implemented aperture photometry to all of the background stars, that are not analysed during the official data reduction pipeline, for the CHEOPS mission.

The aperture photometry method, is schematized in the following figure, using some of the terms that will be presented in the near future.

The presented diagram assumes that the optimization routine has already happened, which may not be true. Further details on the optimziation steps are provided in Section *The Optimization routine*

The Photo_Controller is what controls the flow of the method, providing an easy interface for the user of the library. This "Controller" is also schematized in the following image, and the functions are explained in the Section *Photometry Controller*

If you do not wish to read the entire documentation, it's possible to jump right into the Section *How to extract Light Curves*, where a simple use case is given, and a small discussion is made on how we can access the data either from memory and from disk, after saving it.

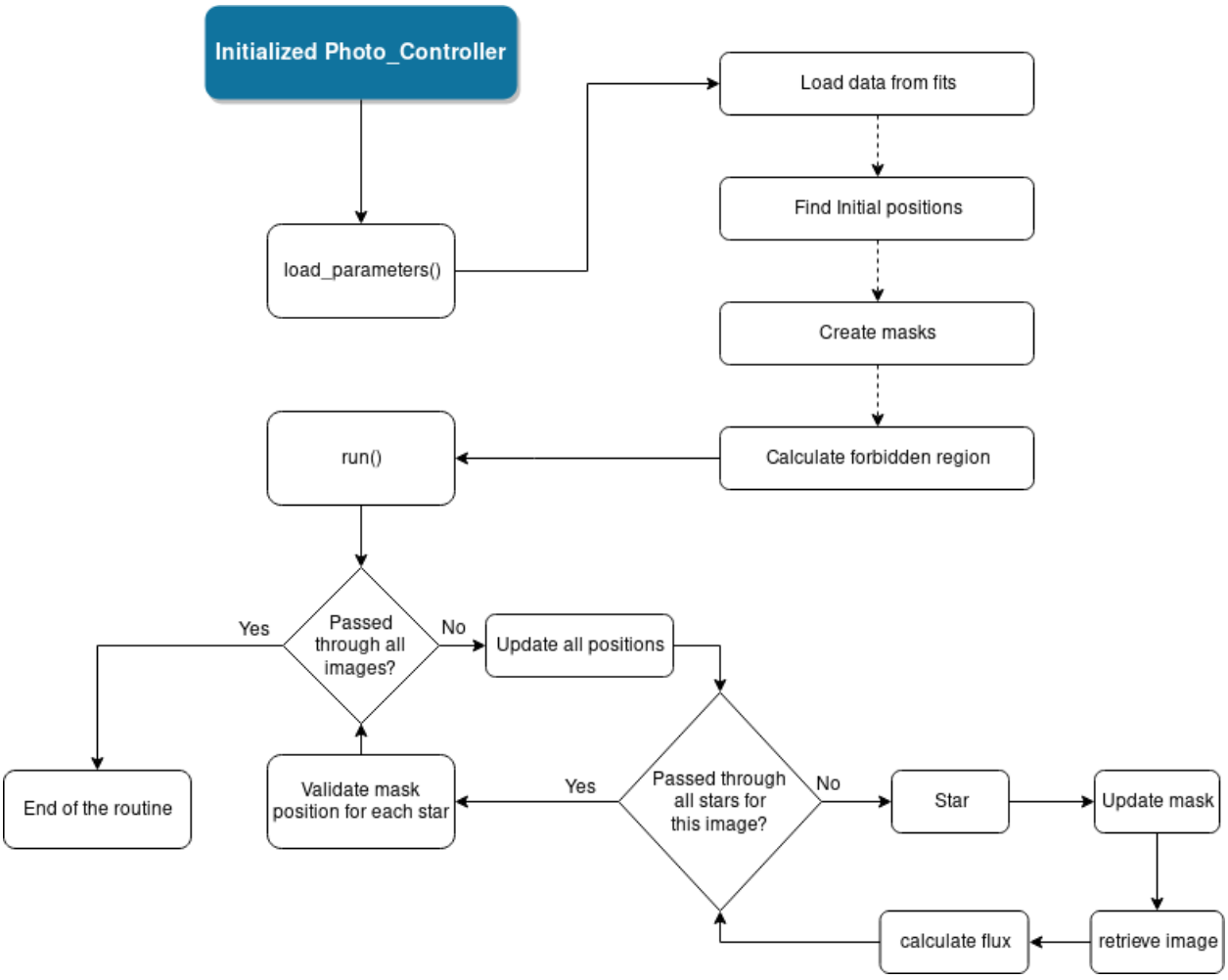

Fig. 1: Diagram of the photometry process.

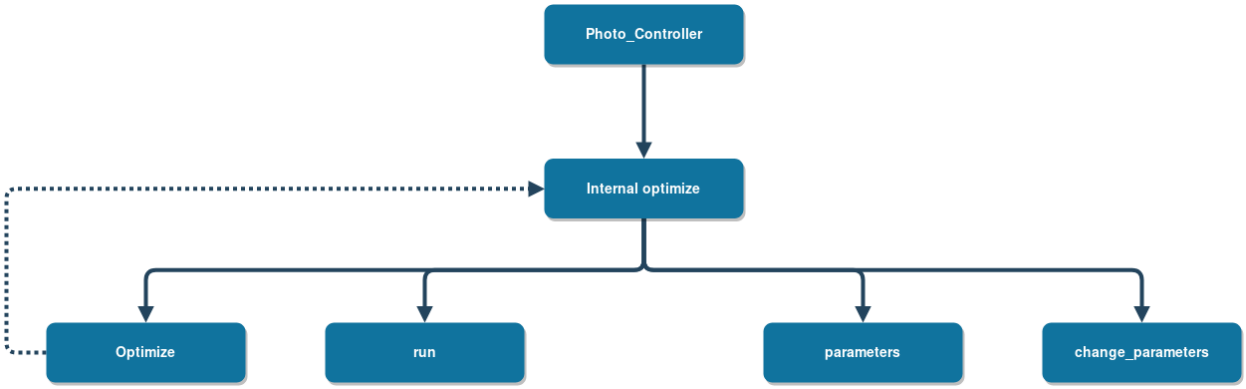

Fig. 2: Diagram of the photometry controller.

CHAPTER 4

User guide

4.1 Photometry Controller

This controller is the main point of access for any user, i.e. one does not need more to be able to extract Light Curves from the background stars in the CHEOPS mission;

class Photo_controller(*job_number*, *config_path='configuration_files/config.yaml'*, *no_optim=False*)

Controller class that allows an easy interface with all the important functions in the module. If the no_optim parameter is set to False, then it automatically starts the optimization routine, at instantiation time of the class.

change_parameters(*params_dict*)

Updates the kwargs with new values. A dictionary is passed in, with keys corresponding to parameters and each value it's the new configuration for that specific parameter. At this stage no validation is made over the passed values. Such validation occurs before the routines, to make sure that the correct parameters were added.

Parameters **params_dict** – Dictionary with the values that we wish to change

optimize()

Run the optimization routine without instantiating a new object.

run(*DataFits=None*, *factor=None*, ***kwargs*)

Calls the main method to run the star analysis pipeline

Parameters

- **factor** – Used for the optimization process. During normal functioning process then so value should be passed
- **kwargs** – Config values, retrieved from the file on the class initialization

Returns **data_fits**

Return type instance of *Data*, with the relevant information.

4.2 Data Object

class Data (*filename*)

Data structure to hold the information of all the stars in the field.

Parameters

- **base_folder** – location of the folder in which all the fits files are located.
- **roll_ang** – rotation angle for all the images.
- **imgs** – all the images in the fits file.
- **stars** – List in which all elements are a "Star" object that holds information of one star.
- **mask_type** – type of mask. either circle or shape
- **detect_mode** – detection mode used
- **init_detection_mode** – initial detection mode
- **mjd_time** – mjd time of the observations

abort_process

If the error_flag is set then the process should be aborted

all_curves

Light curve of all stars, in the star numbering order (from closest to center to farthest)

all_uncertainties

Light curve of all stars, in the star numbering order (from closest to center to farthest)

calculate_uncertainties (*index*)

Trigger the calculation of the uncertainties

Parameters **index** (*int*) – Image number

disable_star (*star_number*)

Disables a star to avoid computing redundant information i.e. after the best value has been found during the optimization process for one star, but others are still not yet optimized. :param star_number:

get_image (*number*)

Ask for a given image. If the background grid is in use, returns the increased image

Parameters **number** (*int*) – Image number

Returns **image** – Desired image

Return type numpy array

static get_rot_mat (*angle*, *clockwise*)

Returns a rotation matrix (around the origin) for a given angle :param angle: rotation angle, in degrees :param clockwise: True for a clockwise rotation and False for a counter clockwise rotation. :return: rotation matrix

is_empty

See if the object has been used in a run, i.e. if it has data

Returns True if it is empty or an error has been raised, False otherwise

Return type boolean

load_parameters (*factor=None*, ***kwargs*)

Loads all the necessary information from the .fits files. Launches the routines to find the initial position of the stars and create the corresponding masks.

Before running the internal parameters, as well as the star's ones are removed. This was made in order to minimize loading data from disk. However, keep in mind that the reset routine does not re-enable stars. So, if a star was disabled in this object, then it will stay that way unless a new object is instantiated. :param factor: :param kwargs:

Returns

- *-1* – Errors were found during runtime
- *0* – Everything went without problems

reload_images(***kwargs*)
Reload images from disk, in the case that the desired image has been deleted :param kwargs:

stars
Returns the list with all of the *Star* objects. If an error has been raised, returns an empty list

4.3 Outputs

store_data(*data_fits*, *job_number*, *singular=None*, ***kwargs*)
Stores the data extracted from the entire pipeline.

Like the controllers, it can work with a *Data* object that was used for both parts or only for one of them.

If one wishes, it can also only create the folders and extract the data only for the chosen star.

Parameters

- **data_fits** – *Data* object.
- **job_number** – Job number, from the supernova server. Will be used to save the parent folder inside the kwargs["results_folder"] to create the entire directory structure.
- **singular** – If it's not None, then only that star's light curve is saved. Only works for the "show_results".
- **kwargs** – Dictionary with the configuration values. Can be obtained with the *Photo_controller* parameters property

CHAPTER 5

Structure of the outputs

There are two different ways to get access to the data extracted from the images: from the *Data* object or from the files that are written to the disk, with the *store_data()* function.

5.1 In memory

In order to access the data in memory, one can refer to the *Data* documentation and `Data`, thus being able to find all that is necessary.

5.2 In Disk

There are two possible ways of storing data on disk: we either store data as a text file, whose structure does not allow for proper organization of the data, or a fits file. We shall now look into the structure of each one of them.

5.2.1 Text file

export_txt (*Data_fits*, *path*, ***kwargs*)

Stores the light curves on a text file with the following format: MJD_TIME;ROLL_ANGLE FLux <Star i> FLUX_ERR <Star i>

Parameters

- **Data_fits** – *Data* object.
- **path** – Path in which the file shall be stored
- **kwargs** – Configuration values

5.2.2 Fits file

create_fits(*master_folder*, *data_fits*, ***kwargs*)

Export the results to a fits file, containing the light curves and a correspondence of star to Cv and respective radius factor.

Parameters

- **master_folder** – Path in which the data shall be stored
- **data_fits** – *Data* object.
- **kwargs** –

Notes

Data stored in the header unit of the file :

Keyword data

method type of mask used detect tracking method initial initial detection method grid size of the background grid CDPP_TYPE CDPP algorithm in use

In the data unit of the file, we have each star, with the corresponding time, rotation angle, flux values and uncertainties

CHAPTER 6

The Optimization routine

The optimization process is one of the most important factors for achieving low noise on the extracted light curves, since it allows us to tweak the mask size. The process is quite simple, since it consists in running the algorithm with different masks sizes, and searching the one that minimizes the CDPP.

To speed up this process, it was implemented in a concurrent way, allowing to have multiple factors being tested at once. If the CDPP is deemed invalid, i.e. if the mask went outside of the image region, then we attribute an arbitrarily high noise level, such as 2e7, so it's certain that any valid values will have a far lower noise and thus a valid mask is chosen.

If any star has a mask size that lies within a tolerance range of the maximum value, the search for the optimal size shall continue, now with a lower limit of the previous maximum value and an upper limit of two times the previous maximum. This repetition has a user-defined maximum number, but we found that limiting it to 5 times is enough to find the best sizes for all the stars, when considering background grids smaller or equal to 1800 points.

The design of the shape mask does not allow for partial increases in the size. However, that's not the case for the circle mask, whose radius can be increased in fractions. So, for this mask, after finding the optimal value, a second optimization step is launched, now searching the values within 1 unit from the optimal one, in steps of 0.1 units. We have found that gains from using smaller steps were not enough to justify the increase in the computational cost.

CHAPTER 7

What is new in archi

7.1 v1.2.1 (Current)

- **Bug Fixes:**
    1. No images were stored when a gif was set to be created
    2. Missing imageio in the requirements.txt

7.2 v1.2.0 (Current)

- **New Features:**
    1. Allow the user to provide a custom metric for the lightcurve noise
- **Bug Fixes:**
    1. removed critical bug when evaluating old keyword

7.3 v1.1.0

- **New Features:**
    1. Added the ability to store a gif of the star tracking routine
    2. Improved the logging of cases where archi fails
    3. If the *dynam* star tracking routine fails it uses the *static* routine to estimate the next point
    4. If the roll angle from the DRP fails (Nans instead of numbers) the *dynam* routine estimates the roll angle based on passed time between images. This is done allow the predictions to still be calculated
- **Bug Fixes:**

- Removed bug where the multiple processes would stay open when the workers would raise an Exception

7.4 v1.0.0

Original Version

CHAPTER 8

Tutorials

8.1 How to extract Light Curves

Assuming that the user has already applied the official Data Reduction pipeline over the data set that is to be studied and that the desired configurations have been already chosen, as specified in Section *Configuration* , one only has to run the following code, to get light curves from all of the stars.

```
from pyarchi import Photo_controller, store_data

controller = Photo_controller(job_number = 1,
                              config_path="<path_to_file>/config.yaml"
                              )
controller.optimize()  # not needed if the optimization process has already been done
data_fits = controller.run()

store_data(data_fits, job_number = 1, **controller.parameters)
```

Furthermore, if we want to change the configuration parameters of archi without having to edit the configuration file, we can do the following:

```
from pyarchi import Photo_controller, store_data

controller = Photo_controller(job_number = 1,
                              config_path="<path_to_file>/config.yaml"
                              )

configs_override = {'base_folder': "<new_path>",
                    "grid_bg": 0,
                    "initial_detect": 'dynam',
                    "method": "shape",
                    "detect_mode": "dynam",
                    "optim_processes": n_tasks,
                    "val_range": [1, 10],
                    "low_memory":0,
```

(continues on next page)

(continued from previous page)

```
                    "fine_tune_circle":1,
                    "optimize":0,
                    'uncertainties': 1,
                    'CDPP_type': "K2",
                    "debug": 1,
                    "plot_realtime": 0,
                    'repeat_removal': 0}

# we can override all parameters from the file. Typically, one only needs to change␣
↪these

controller.change_parameters(configs_override)
data_fits = controller.run()

store_data(data_fits, job_number = 1, **controller.parameters)
```

If we had a data set with bright and faint stars, we would have to change the "repeat_removal" parameter, to remove the brighter regions in the image. It is recommended to avoid more than 2 iterations, as using larger values may start recognizing regions of the background as stars.

In order to understand the organization of data inside the "data_fits" objects and the data that is stored with the "store_data" function, please refer back to Section *Structure of the outputs*.

It is highly recommended that the user builds its own routine to estimate the noise in the lightcurve. To do so, a given metric mu

This metric will be minimized during the mask selection routine. The current version only gives the user the flux and observation time.

```
from pyarchi import Photo_controller, store_data

controller = Photo_controller(job_number = 1,
                              config_path="<path_to_file>/config.yaml"
                              )
def noise_metric(flux, time):
    metric = np.std(flux)
    return metric
configs_override = {'CDPP_type': noise_metric}
```

8.2 How to plot the outputs

By default archi creates some plots, that are stored in the created folders, containing information of the light curves from each star and by comparing the target one against the DRP.

However, if we want to do so manually, we can do it this way:

```
from pyarchi import Photo_controller, store_data
import matplotlib.pyplot as plt

controller = Photo_controller(job_number = 1,
                              config_path="<path_to_file>/config.yaml"
                              )
controller.optimize()  # not needed if the optimization process has already been done
data_fits = controller.run()

for star in data_fits.stars: # iterate through all of the stars
    plt.plot(data_fits.mjd_time, star.photom)
```

(continues on next page)

(continued from previous page)

```
    plt.title(f"{star.name}")
    plt.xlabel("MJD time [days]")
    plt.ylabel("Flux [ADU]")
    plt.show()
```

In order to better understand what information one can get from the controller, stars and data_fits, please refer back to relevant documentation.

CHAPTER 9

Stars

9.1 Star information

class Star(*cdpp_type: str*, *pos=None*, *dist=None*)

Star class that holds the star information. We have a class attribute, number, that will be used to name the star.
Each star increments this number by one.

Parameters

- **cdpp_type** –

  CDPP algorithm; Can be a string or a function. If it is a string, it should be "K2". Otherwise, the function sh

  def foo(flux, time): return metric
- **pos=None** – Initial position
- **= None** (*dist*) – Distance to center

Notes

name: Designation of the star

masks: Holds the mask used in each image.

GP_data: Holds the GPs results

positions: position, in pixels, of the centroid. Always on coordinates of the [200,200] grid

init_pos: Initial position, in pixels, of the centroid. Always on coordinates of the [200,200] grid

out_bound: An overlap between the mask and the empty region was found

_active: Do we want to calculate the flux for this star

photom: holds the light curve information

debug: If 1 we compare the data obtained from our analysis with the one produced by the official pipeline

add_initial_mask(*mask*, *factor*, *scaling_factor*, *low_memory=0*)
Initializes the Masks class

Parameters

- **mask** – Initial mask for the star
- **factor** – Increase factor for the mask
- **size_grid_change** – Size of the background grid
- **low_memory** – Mode in which only the necessary information is kept during the process

Returns

- *0* – If no error is found
- *-1* – If the factor is a negative number

add_mask(*mask*)
Adds a mask to the Masks object

Parameters **mask** – mask to add

calculate_cdpp(*time=None*)
Calculates the CDPP of the light curve, in order to quantify the results

Returns **CDPP, CDPP_def**

Return type noise for my light curve and for the official one, respectively

change_init_pos(*pos*)
Change the star's initial position. Used when more than one initial center determination method is active and some information needs to be overwritten :param pos: New position for the centroid of the star (for the 1st image)

disable()
Sets the flag to disable the calculation of the flux of this star.

enable_debug(***kwargs*)
Extracts the default data from the FITS files :param base_folder: path to folder in which the FITS files are located

import_photom(*photom*)
Used when loading data from disk :param photom:

latest_mask
Returns the last mask added to the star.

out_bounds(*index*)
Take into account that star was out of bonds in a given frame

remove_data()
Empty all information stored on this star. Used during the optimization process

update_mask(*scaling_factor*, *index*)
Updates the mask with the information from the current image :param scaling_factor: Size of the background grid. :param index: Index of the current image

9.2 Initial Detection

centers_from_fits(*primary: str, secondary: str, stars: List[T], initial_angle: float, initial_offset: List[T], **kwargs*)

Using information stored on the fits files, we determine the centers positions. The centers are determined using relations between the differences in RA and DEC of all stars in relation to the known point : the central star.

After determining the center, we use the primary and secondary arguments to see if this function should change the initial position of the star.

Parameters

- **primary** (*str*) – Methodology to apply to the central star. If it's fits then the initial position of that star is changed to be the one determined here.
- **secondary** (*str*) –

  Methodology to apply to the outer stars. If it's fits then the initial position of those stars are changed to be the ones determined here.
- **stars** (*list*) – List with all the stars found with the dynam method
- **initial_angle** (*float*) – Rotation angle of the satellite for the first image
- **initial_offset** (*list*) – DRP's estimation of the central star location
- **kwargs** – kwargs

Returns Updated list of stars, with the positions determined by the fits method

Return type List

initial_dynam_centers(*img*, *bg_grid*, ***kwargs*)

Uses the same method as the one in dynamical_centers to detect the original position of each star. Afterwards, the positions are ordered by closeness to the center which allows us to keep the same order as in the "fits" initial position method. :param im: First image in the data set :param bg_grid: Size of the background grid

9.3 Star tracking

9.3.1 Handler

star_tracking_handler(*Data_fits*, *index*, ***kwargs*)

Handles the detection mode for the target star and the ones around it. Allows to have two different modes of detection active at the same time.

Parameters

- **Data_fits** – `pyarchi.main.initial_loads.Data` object.
- **index** – Image number
- **kwargs** –

Returns Error code of the different star tracking methods. 0 if everything as expected 1 otherwise

Return type int

9.3.2 Methods

static_method(*Data_fits*, *img_number*, *primary*, *secondary*)

Rotates the points in the last know position by the corresponding rotation angle (difference between current angle and the one from the last image).

Parameters

- **Data_fits** – `pyarchi.main.initial_loads.Data` object.
- **img_number** – Number of the current image
- **primary** – Methodology to apply to the central star. If it's static then the central star is tracked using this method
- **secondary** – Methodology to apply to the outer stars. If it's static then they are tracked using this method

create_predictions(*Data_fits*, *img_number*)

Rotates the points in the last know position by the corresponding rotation angle (difference between current angle and the one from the last image), predicting the next position of the star

Parameters

- **Data_fits** – `pyarchi.main.initial_loads.Data` object.
- **img_number** – Number of the current image

dynam_method(*Data_fits*, *index*, *primary: str*, *secondary: str*, *repeat_removal: int*)

This function is used to calculate the position of the center of each contour. In order to do that we calculate the moments of the image, which allows us to derive it's "center of mass". All contours with less than 7 points are discarded and, to associate center to star we use the rotate_points function to predict the center's expected position. BY comparing the expected positions with the outputs of the algorithm we can associate a center to each star.

If the image processing routine is not able to detect any star, the it uses the predictions to shift the masks.

Parameters

- **Data_fits** – `pyarchi.main.initial_loads.Data` object.
- **index** – image's number
- **primary** – Methodology to apply to the central star. If it's dynam then the central star is tracked using this method
- **secondary** – Methodology to apply to the outer stars. If it's dynam then they are tracked using this method

offsets_method(*Data_fits*, *index*, *primary*, *secondary*)

This function expands the functionality of rotate_points. After rotating the points we calculate the offset experienced by the central star, by calculating the deviation between the center location from the last two images. The offset center for the central star is simply obtained from the fits files, without rotating the previous point.

Parameters

- **Data_fits** – `pyarchi.main.initial_loads.Data` object.
- **index** – index of the image

- **primary** – Methodology to apply to the central star. If it's offsets then the central star is tracked using this method
- **secondary** – Methodology to apply to the outer stars. If it's offsets then they are tracked using this method

CHAPTER 10

Masks

10.1 How to store them

class Masks(*mask_factor*, *grid_increase*, *initial_mask*, *low_memory=0*)

Class used to hold the mask for each iteration, as well as some of some important methods

add_mask(*mask*)

Stores a new mask. If the low memory mode is active, only two masks are stored in memory: the initial one and the latest.

Parameters **mask** – Mask to be added

all

Return a list with all of the stored masks

factor

Returns the mask's size (i.e. circle radius or layers of pixels added to the mask)

first

Returns the first mask

latest

Returns the latest mask stored in the class

normalized_points

Returns the number of points, normalized to the "normal" grid, with 200 by 200 px

number_masks

Return the number of masks stored in the obejct

size

Returns the number of pixels in the mask

update_mask(*x_change*, *y_change*, *image_number*)

Changes the mask to the correct position in the new image. If the grid_bg is not None, then the calculations are made with the bigger grid.

Parameters

- **x_change** – change in the x direction
- **y_change** – change in the y direction
- **image_number** – Current image number.

10.2 Shape Based mask

create_shape_mask(*im*, *stars*, *increase_factor*, *scaling_factor*, *primary*, *secondary*, *bg_grid*, *repeat_removal=0*)

Finds the contours of the image, with openCv default functions

Parameters

- **im** – copy of the image used for the shape detection
- **stars** – list of all the `pyarchi.star_track.Star_class.Star` objects
- **increase_factor** – Number of pixels added to the outside of the shape. For example, if factor = 1 then we add a layer of pixels around the entire shape
- **size_grid_change** – SIze of the background grid in use
- **primary** – Methodology to apply to the central star. If it's dynam then the initial position of that star is changed to be the one determined here.
- **secondary** – Methodology to apply to the outer stars. If it's dynam then the initial position of those stars are changed to be the ones determined here.
- **repeat_removal** – Number of times that we wish to remove the brightest mask from the image, to search for fainter stars

Returns Dictionary where the keys are the number of the star and the values the corresponding mask

Return type masks_dict

10.3 Circular mask

create_circular_mask(*img*, *stars*, *radius*, *primary*, *secondary*)

Defines a circular mask for all the stars, with a pre determined radius. If a size_grid_change is different than zero it converts the mask to that grid size

Parameters

- **stars** – list of all the `pyarchi.star_track.Star_class.Star` objects
- **img** – First image
- **radius** – radius of the circles. Can be a general radius(same for all the stars) or a dict with different radii,
- **which the keys are the indexes and the values are the radius** (*in*) –
- **size_grid_change** – size of the bigger grid, to which we can convert the mask. If it's zero we don't convert. Otherwise it is converted
- **primary** – Methodology to apply to the central star. If it's fits then the initial position of that star is changed to be the one determined here.

- **secondary** – Methodology to apply to the outer stars. If it's fits then the initial position of those stars are changed to be the ones determined here.

Returns Dictionary where the keys are the number of the star and the values the corresponding mask

Return type masks_dict

CHAPTER 11

Utility scripts

11.1 data_export

photo_SaveInfo (*path*, *data_fits*)
Stores information related to the photometric run.

In specific:

- which file was used
- mask type
- detect mode
- background grid
- all star related info

Parameters

- **path** – path in which the file will be stored
- **data_fits** – *Data* object.

For details on the functions that write to disk refer to Section *Structure of the outputs*.

11.2 factors_handler

11.3 noise_metrics

CDPP (*flux_vals*, *times*, *sized=41*, *winlen=10*, *win=30*, *outl=True*)
Ported version of the K2 CDPP algorithm, implemented by Pedro Silva

Parameters

- **flux_vals** – flux values
- **times** – time array
- **sized** – Since we are calculating the CDPP for 1 hour. Window for the Savgol filter
- **winlen** – Convolution window
- **win** – Time over which we want to calculate the CDPP
- **outl** – Remove the outliers.

Interpolate(*time*, *mask*, *y*)

Masks certain elements in the array *y* and linearly interpolates over them, returning an array *y'* of the same length.

Parameters

- **time** (*array_like*) – The time array
- **mask** (*array_like*) – The indices to be interpolated over
- **y** (*array_like*) – The dependent array

SavGol(*y*, *win=49*)

Subtracts a second order Savitsky-Golay filter with window size *win* and returns the result. This acts as a high pass filter.

Scatter(*y*, *win=13*, *remove_outliers=False*)

Return the scatter in ppm based on the median running standard deviation for a window size of `win` = 13 cadences (for K2, this is ~6.5 hours, as in VJ14).

Parameters

- **y** (*ndarray*) – The array whose CDPP is to be computed
- **win** (*int*) – The window size in cadences. Default *13*
- **remove_outliers** (*bool*) – Clip outliers at 5 sigma before computing the CDPP? Default *False*

Smooth(*x*, *window_len=100*, *window='hanning'*)

Smooth data by convolving on a given timescale.

Parameters

- **x** (*ndarray*) – The data array
- **window_len** (*int*) – The size of the smoothing window. Default *100*
- **window** (*str*) – The window type. Default *hanning*

11.4 optimization

optimizer(*value_range*, *max_process*, *func*, *data_f*, *file_path*, *to_disable=[]*, ***kwargs*)

Decides which factors will be used in each process that it spawns. After the processes are done, extracts the resulting data from the Queue and parses it, in order to find the noise values and mask factors. Writes to a .txt file the resulting values for each factor. Take note that the values are not guaranteed to be in order, since they are saved in the order "imposed" by the processes. :param value_range: Lower and upper limit of the values to be tested :param max_process: Maximum number of processes that can be launched during this routine :param func: function that launches the photometric process :param data_f: `pyarchi.main.initial_loads.Data` object with all the stars information inside :param file_path: Path in which run time information shall be stored :param kwargs:

Returns

- *min_cvs* – list with the minimum noise found
- *optimized_dict* – Dictionary in which the keys are the star numbers and the values are the optimal factors

run_function(*queue*, *func*, *factors*, *data_f*, *to_disable=[]*, ***kwargs*)
Run each interaction of the function and returns the results ordered on a dictionary

Parameters

- **queue** –
- **func** –
- **factors** –

circular_tuner(*best_values*, *max_process*, *func*, *data_f*, *file_path*, *to_disable=[]*, ***kwargs*)
Decides which factors will be used in each process that it spawns. After the processes are done, extracts the resulting data from the Queue and parses it, in order to find the noise values and mask factors. Writes to a .txt file the resulting values for each factor. Take note that the values are not guaranteed to be in order, since they are saved in the order "imposed" by the processes. :param value_range: Lower and upper limit of the values to be tested :param max_process: Maximum number of processes that can be launched during this routine :param func: function that launches the photometric process :param data_f: `pyarchi.main.initial_loads.Data` object with all the stars information inside :param file_path: Path in which run time information shall be stored :param kwargs:

Returns

- *min_cvs* – list with the minimum noise found
- *optimized_dict* – Dictionary in which the keys are the star numbers and the values are the optimal factors

run_function(*queue*, *func*, *factors*, *data_f*, *to_disable=[]*, ***kwargs*)
Run each interaction of the function and returns the results ordered on a dictionary

Parameters

- **queue** –
- **func** –
- **factors** –

general_optimizer(*func*, *data_f*, *job_number*, *max_process*, ***kwargs*)
Responsible for setting up the variables used during the optimization process.

If the factor determined to be the best one is near the upper limit, then the range of values is extended and a new search for minimum noise starts. If the determined factor is inside a "safe" distance away from the highest possible value then the star is disabled and no further studies are made on it.

Also responsible for creating .txt files with all the relevant information :param func: function that launches the photometric process :param data_f: `pyarchi.main.initial_loads.Data` object with all the stars information inside :param job_number: JOb number assigned by the SLURM workload manager :param max_process: Maximum number of processes that can be launched during this routine

Returns Dictionary in which the keys are the star numbers and the values are the optimal factors

Return type optimized_dict

11.5 misc

parameters_validator(***kwargs*)
Loads the configuration parameters from the .yaml file. After loading, it checks some of the parameters to see if they have valid values.

Parameters **parameters** –

path_finder(*mode*, *off_curve=None*, ***kwargs*)
Searches inside the base folder for the desired files

Parameters

- **mode** –
  - subarray: retrieves the subarray file path
  - default: default lightcurve
  - stars : path for the Star catalogue file
- **kwargs** –
  - config values

matrix_clockwise(*angle*)
Creates a rotation matrix for clockwise rotations. Expects an angle in degrees

Parameters **angle** – rotation angle, in degrees.

Returns counter clockwise rotation matrix

matrix_cnter_clock(*angle*)
Creates a rotation matrix for counter clockwise rotations. Expects an angle in degrees

Parameters **angle** – rotation angle, in degrees.

Returns counter clockwise rotation matrix

my_timer(*orig_func*)
Decorator to measure the time spent on the function :param orig_func: FUnction to be analysed

CHAPTER 12

Output creation

12.1 Internal interface

photom_plots(*data_fits*, *master_folder*, *singular=None*, ***kwargs*)

Processes the data collected from the images and outputs the desired graphs.

Parameters

- **data_fits** – *Data* object with all the stars information inside
- **singular** – If it's not None, then only that star's light curve is saved. Only works for the "show_results".
- **master_folder** – Root folder in which all the data shall be stored
- **kwargs** – Configuration values that are used to change the data processed and the output data:

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

N

O

P

R

S

U