Guide to obtain the attenuation light coefficient

SUMMARY

This document contains a brief description of how to use the **python module** called **get_attenuation.py**. In the module, you can find a bunch of different functions which you need in order to process the data obtained from the CCD cameras we have in the lab. The functions are useful for calibration of the images and for calculating physical parameters, in our case, **attenuation light coefficient**. The code should work in Linux and Windows.

Required packages and files

To use the codes, there are some required Python files and modules we need to add and install. First of all, the codes are based on the Python model called Astropy. We based the analysis on reading folders or images from the path of these objects in the computer, for that, we use the Python module called Pathlib. For fitting and statistical analysis we use Iminuit. Finally, there is a file.py that needs to be added to the folder where you will be placing **get_attenuation.py** file, this file is named **convenience_functions.py** and must be required to the authors of this document in case it is not found in the computer of the lab. This file is used mostly for the visualization of the images.

Brief introduction to CCD cameras

Charge-Coupled Device (CCD) cameras are digital cameras that use an array of photosensitive elements, called pixels, to capture images. CCD cameras are commonly used in applications where high-quality images are required, such as in scientific research, astronomy, and photography. CCD cameras work by converting light into electrical signals. When light falls on a pixel, it generates an electrical charge that is proportional to the amount of light that was received. The charges are then transferred to a row of pixels in a process called ”charge transfer,” which is facilitated by a series of electrodes called the ”shift register.” The charge is then read out by the camera’s analog-to-digital converter and converted into a digital signal that can be stored or processed by a computer.

Discussion on the step-by-step process

For getting the final product, which in this case is the attenuation light coefficient, there are several steps we must follow. Here we do mention not in much detail how we proceed to calculate the parameter. We can basically divide the process into 16 steps:

1. Take calibration images: bias, dark and flat frames.
2. Take science images: images from the sky, a laser, or whatever you want to measure.
3. Combine bias frames to create a **master bias**.
4. Subtract the master bias to all the dark frames.
5. Combine the calibrated dark frames to create a **master dark**.
6. Subtract the master bias and master dark to all flat frames.
7. Combine the calibrated flat frames to create a **master flat** (You can also normalize the master flat by dividing the image by the mean value).

8. Use a calibrated longer exposure time dark frame to calculate a mask for hot pixels.
9. Use one or two calibrated different exposure time flat frames to calculate non-sensitive bad pixels.
10. Combine these two masks to create a **combined mask**.
11. Use calibrated flat frames to calculate the gain of the CCD camera(s).
12. Use calibrated flat frames to do a linearity test of the camera(s).
13. Use bias frames to calculate the readout noise error matrix for statistical error of the image(s).
14. Use dark frames to calculate the dark current statistical error of the image(s).
15. Use all the previous objects to calibrate the science images.
16. Use two science images to calculate the attenuation coefficient.

Code Tutorial

If you have the chance to look into the file **get_attenuation.py**, you will see that the functions inside are organized such that they are in the order in which we need to use them. As a spoiler, remember to change from **False** to **True** the arguments **compare** and **distribution** if you want to have the images displayed but it could make the run take longer.

3. Combine bias

For creating the master bias. The path is the absolute path of the folder containing the bias frames. It has different functionalities which can be read out in more detail in the code itself. An example of how to use it is shown below.

```
path ='\folder\to\the\bias\frames'
get_master_bias(path, folder =False, compare =False, bins =500, distribution =False,
                title ='Master Bias' ,warning ='ignore')
```

The output will be a new image called "masterbias" placed in the folder "path".

4. Calibration of dark frames

First, we need to subtract the bias level from all dark frames. For this, we just do an arithmetic subtraction. It would look as shown below.

```
path_dark ='\path\folder\to\the\dark\frames'
path_master_bias ='\path\to\the\master\bias'
get_calibrated_dark(path_dark, path_master_bias, folder =False, compare =False,
                    distribution =False, cmap ='gray' , warning ='ignore')
```

The output will be a new folder containing all calibrated dark frames placed inside "path_dark".

5. Combine darks

For creating the master dark. The path is the absolute path of the folder containing the calibrated dark frames. It has different functionalities which can be read out in more detail in the code itself. An example of how to use it is shown below.

```
path= '\path\folder\to\the\calibrated\dark\frames'
get_master_dark(path, folder =False, compare =False, distribution =False,
                title ='Master Dark' , same_time =True ,warning ='ignore')
```

The output will be a new image called "masterdark", placed in "path".

6. Flat frames calibration

Arithmetic subtraction of the master dark and master bias from all flat frames.

```
path_flat ='\path\to\folder\with\all\flat\frames'
path_master_bias ='\path\to\master\bias'
path_master_dark ='\path\to\master\dark'
get_calibrated_flat(path_flat, path_master_dark, path_master_bias =None, same_time =True,
                    folder =False, compare =False, cmap ='gray' , warning ='ignore')
```

The output will be a folder containing all calibrated flat frames placed inside "path_flat".

7. Combine flats

For calculating the master flat.

```
path= '\path\to\folder\with\all\calibrated\flat\frames'
get_master_flat(path, folder =False, same_time =True ,compare =False, distribution =False,
                cmap ='gray' ,warning ='ignore')
```

The output will be an image placed in "path".

8. Hot pixel mask

The next function is used to calculate a mask for the hot pixels. We make use of a calibrated dark frame. See the example below.

```
path_im ='\path\to\calibrated\dark\frame\image'
path_fol ='\path\to\where\you\want\to\save\the\mask'
get_hot_pixels(path_im, path_fol, sigma =5, cenfunc ='median', stdfunc ='mad_std',
                compare =False, hot_pixels =False ,cmap ='gray' ,warning ='ignore')
```

The output will be a hot_pixel_mask.fits image placed in "path_fol".

9. Bad pixel mask

The next function is used to calculate a mask for the bad pixels. We make use of a calibrated flat frame. See the example below.

```
path1 ='\path\to\calibrated\flat\frame'
path_fol ='\path\to\where\you\want\to\save\the\mask'
path2 ='\path\to\a\second\flat\frame'
get_bad_pixels(path1, path_fol , path2 =None , compare =False, cmap ='gray',
               bad_pixels =False, warning ='ignore')
```

In order to know more about when to use one or two flat frames, we recommend getting into the code where it is explained. We also recommend saving the mas together with the hot pixel mask. The output will be a bad_pixel_mask.fits image placed in "path_fol".

10. Combine mask

For combining the mask obtained previously. This is done to have a single mask with the information about all pixels not working properly.

```
darkmask ='\path\to\hot\pixels\mask'
flatmask ='\path\to\bad\pixels\mask'
path_fol ='\path\to\where\you\want\to\save\the\combine\mask'
get_combined_mask(darkmask, flatmask, path_fol, useless_pixels =False,
                  show_mask =False, cmap ='viridis')
```

Setting **useless_pixels** and **show_mask** to **True** may be useful for getting more information and visualizing. The output is a mask.fit file place in "paht_fol".

11. Calculate the gain

The gain is the conversion factor between electrons and ADU units. It is possible that when using different cameras they have different gain which must be corrected if you want to compare measurements. To correct using the right value, we need to estimate its value using flat frames. The method is not explained here but again the reader should go to the file since the gain calculation could be more confusing. We use flat frames of different exposure times, then they must be calibrated using dark frames with the same exposure time. To do this process faster, we have created a function called **get_flats_for_test** which identifies images with the same exposure times.

```
path ='\path\to\folder\containing\flat\frames\for\gain\and\masterbias\masterdarks'
get_flats_for_test(path, compare =False, folder =False ,cmap ='gray' , warning ='ignore')
```

The output will be a folder with all calibrated flat frames even though they have different exposure times. The folder is placed in "path". Now for the gain, we use these images to calculate the gain. See below.

```
path ='\path\to\folder\with\calibrated\flat\created\step\above'
get_gain_factor(path, position, size, gain, title ='Variance diagram CCD: slope yields gain',
                compare =False , save =False ,cmap ='gray' , warning ='ignore')
```

The output will be a diagram from where the gain is calculated and also the gain value obtained is print out.

12. Linearity test

We need the camera(s) to work linearly between the exposure time and light collection. This test is done to find such a linear range at which we are able to use the camera(s). Again for this, the method is not explained here and the reader must check what kind of flats are needed in the file with the code. We start by calibrating the flats taken specifically for the test.

```
path ='\path\to\folder\containing\flat\frames\for\test\and\masterbias\masterdarks'
get_flats_for_test(path, compare =False, folder =False ,cmap ='gray' , warning ='ignore')
```

The output will be a folder with all calibrated flat frames even though they have different exposure times. The folder is placed in "path". Now for the linearity test, we use these images. See below.

```
path ='\path\to\folder\with\calibrated\flat\created\step\above'
mask\_path ='\path\to\combined\mask\not\mandatory\but\recommended'
adu_max ='set max number of ADU allowed'
adu_min ='set min number of ADU allowed'
linearity_test_cdd(path, mask_path =None ,adu_max =None, adu_min =None,
                   save =False ,warning ='ignore')
```

The output will be a plot showing how linear the camera is. The linearity value R is also printed out together with a linear error measuring how linear the data is.

13. Readout noise

We calculate a statistical error matrix associated with readout noise. We use bias frames for this.

```
path_bias_folder ='\path\folder\bias\frames\for\readout\noise'
readout_noise(path_bias_folder, gain =None, warning ='ignore')
```

The output will be a matrix with the same dimensions as the images. Will be used later for calibration of science images.

14. Dark current error

We calculate a statistical error matrix associated with the dark current. We use dark frames for this.

```
path_dark_frames ='\path\folder\dark\frames\for\dark\current'
path_masterbias ='\path\to\master\bias'
dark_current_err(path_dark_frames, path_masterbias, gain =None, warning ='ignore')
```

The output will be a matrix with the same dimensions as the images. Will be used later for calibration of science images.

15. Calibrating science images

We use all the results obtained from all the previous steps to calibrate the science images.

```
path_science ='\path\to\science\images'
path_master_dark ='\path\to\master\dark'
path_master_flat ='\path\to\master\flat'
matrix_error ='list with error matrices like readout noise and dark current'
path_mask =''\path\to\combined\mask
path_master_bias ='\path\to\master\bias'
gain ='gain if you need to rescale the images by a given gain'
get_calibrated_ccd_image(path_science, path_master_dark, path_master_flat, matrix_error,
                         path_mask, path_master_bias ,gain =None , compare =False,
                         readnoise =None, cmap ='hot', warning ='ignore')
```

The output will be a folder containing all calibrated science images, placed in "path_science".

16. Obtaining attenuation light

Since the science images are calibrated, we can calculate the attenuation light. We based our analysis on the fact that you have two images measuring light going two the same medium but different distances.

```
path_image1 ='\path\fist\science\image'
path_image2 ='\path\second\science\image'
att(path_image1, path_image2)
```

The output will be the attenuation light coefficient value in meters.

Authors

Felipe Villazon, JGU Mainz
Greta Blank, JGU Mainz