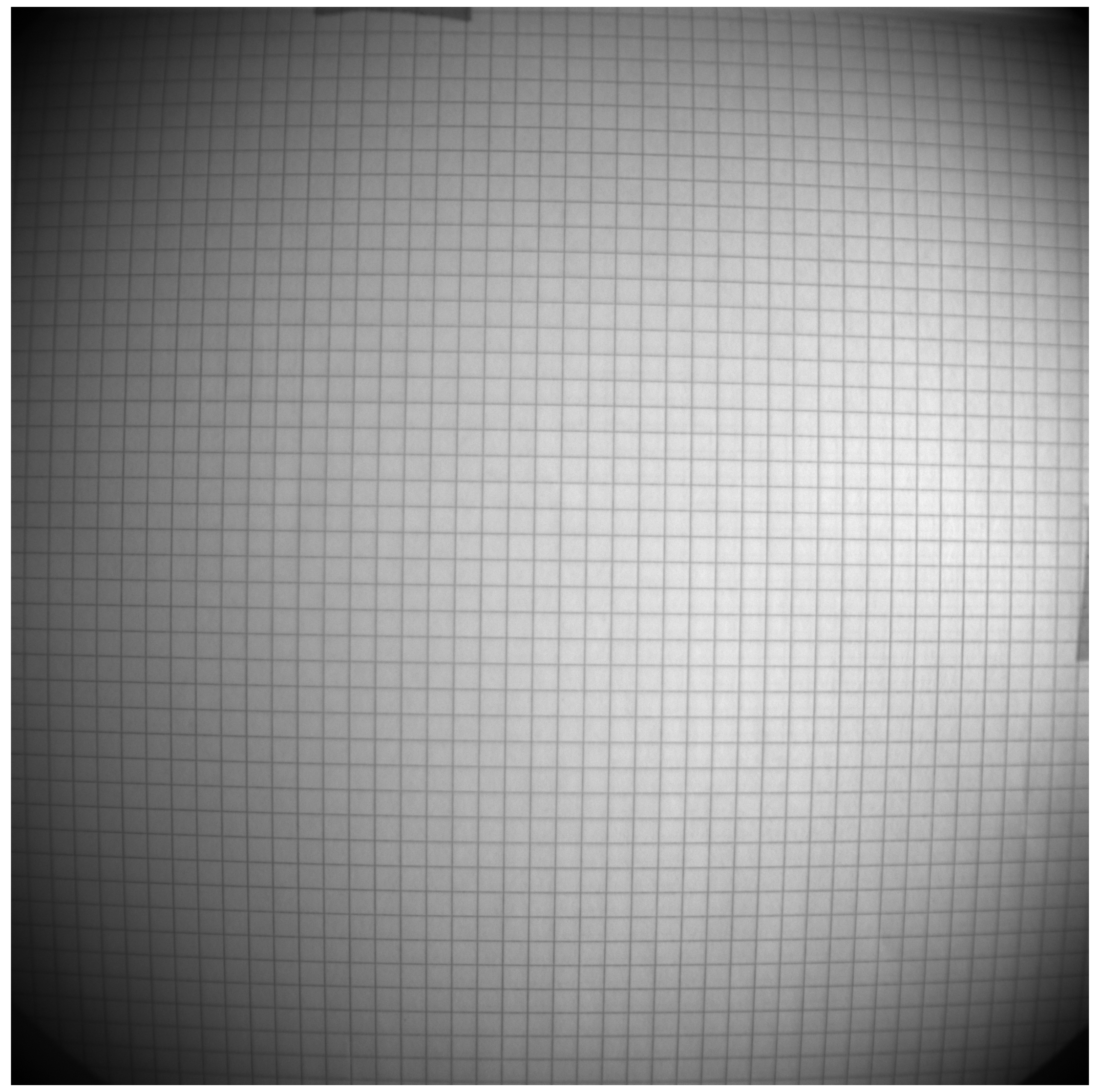
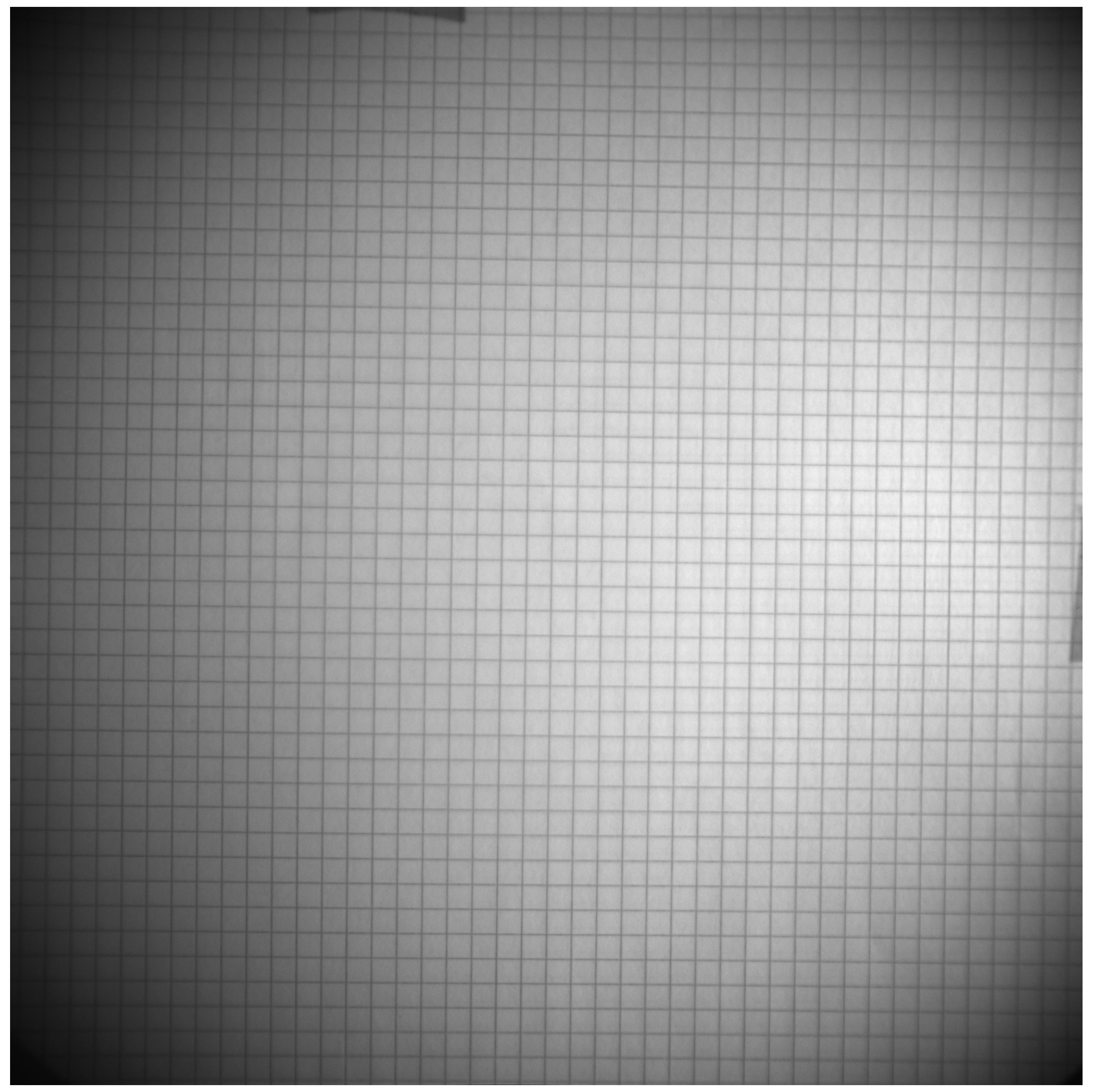
Correction of the optical distortion



$\longrightarrow$



Stefano Piacentini

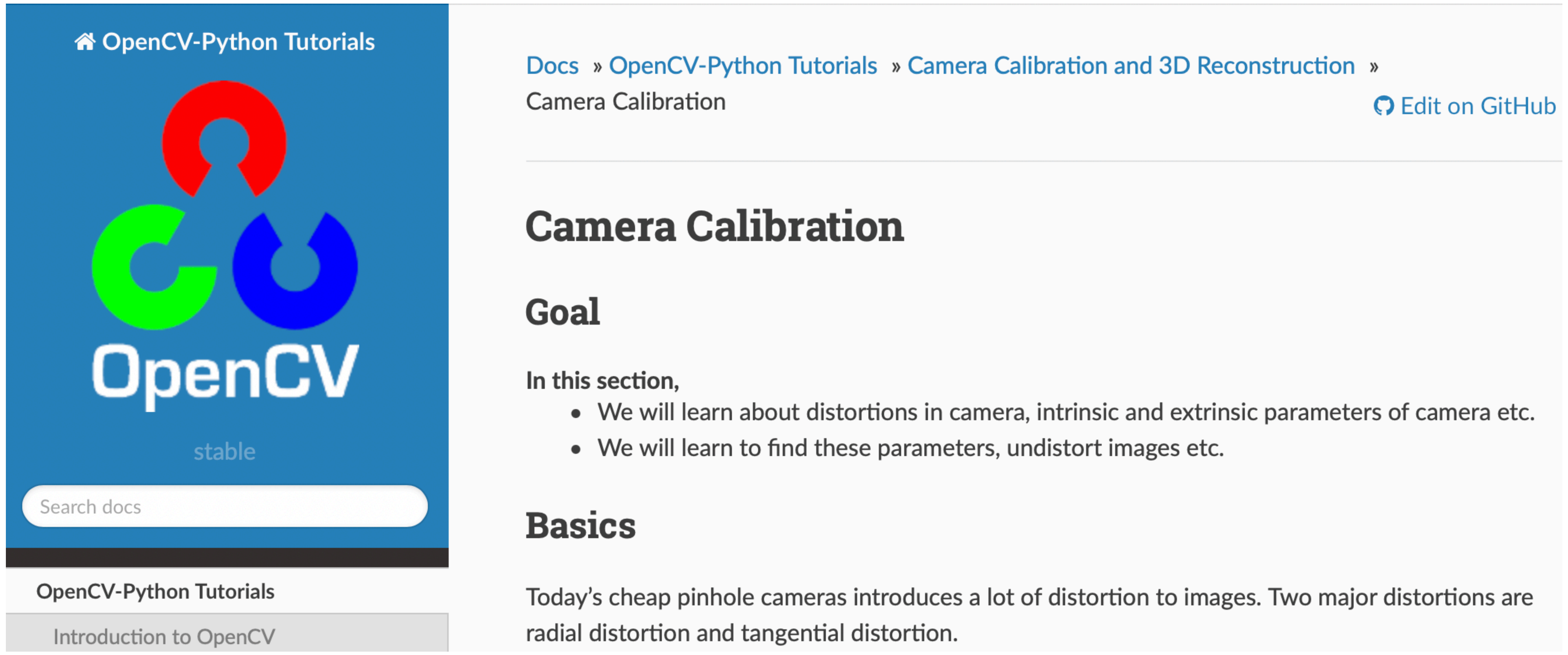
22/09/2022

Introduction

- Currently our pictures are affected by a **distortion effect**
- This effect is related to the optical properties of the lens and the camera and **acts differently in different position of the picture**
- This effect could be an **issue** when we extract the **directional** information:
  - e.g. the direction of the same ellipsoidal spot may look different in different xy positions
  - e.g. the diffusion may look different in the x and y directions in different xy positions
- We think this effect may have an impact on the LIME ^{55}Fe campaign.

OpenCV

- In principle this effect **can be corrected** with a set of 2D geometrical transformations.
- **OpenCV**, a Python package, can do it for us.



Camera Calibration

Goal

In this section,

- We will learn about distortions in camera, intrinsic and extrinsic parameters of camera etc.
- We will learn to find these parameters, undistort images etc.

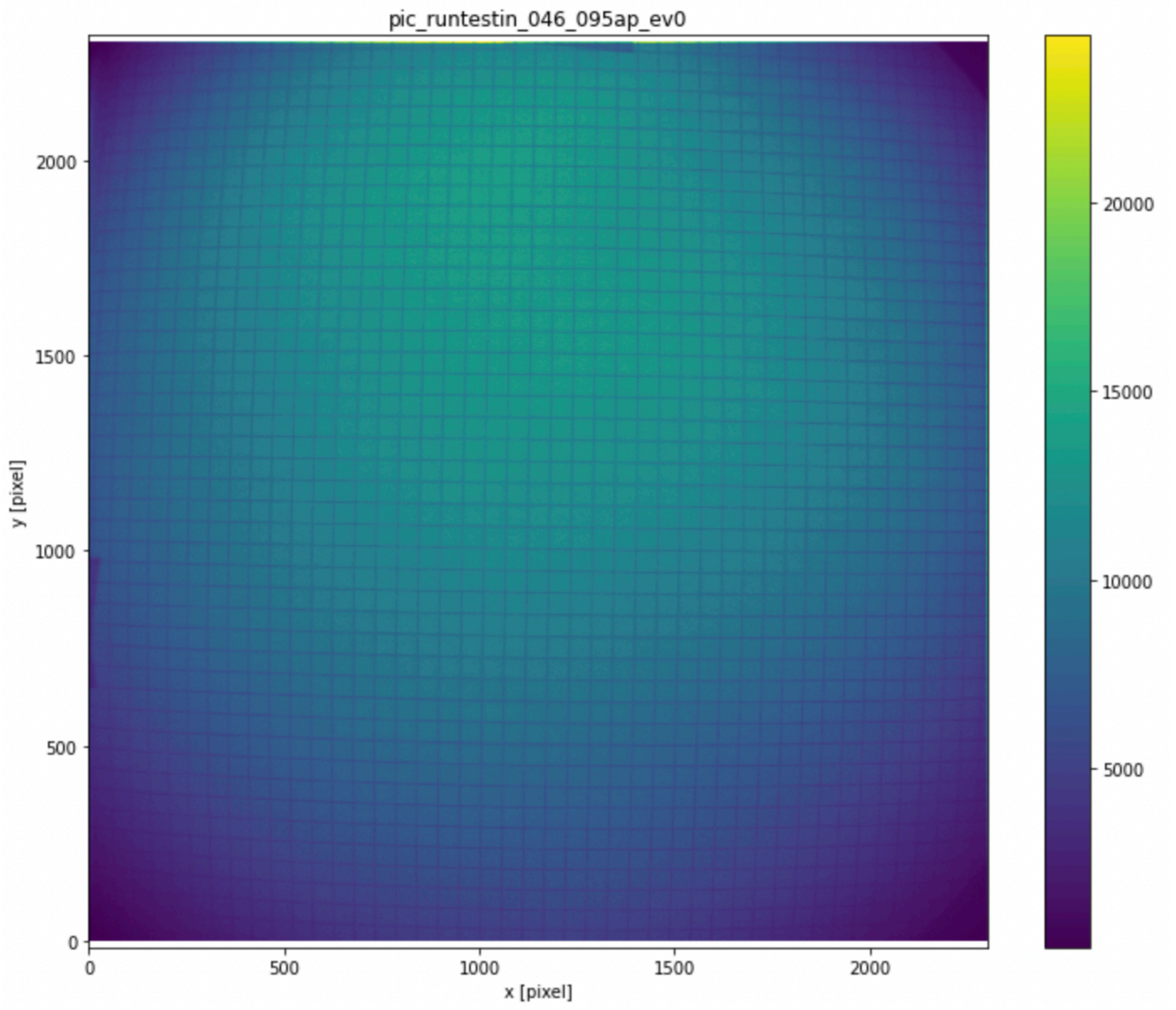
Basics

Today's cheap pinhole cameras introduces a lot of distortion to images. Two major distortions are radial distortion and tangential distortion.

An example with the MANGO optical setup

Phase 1 - **Calibration**:

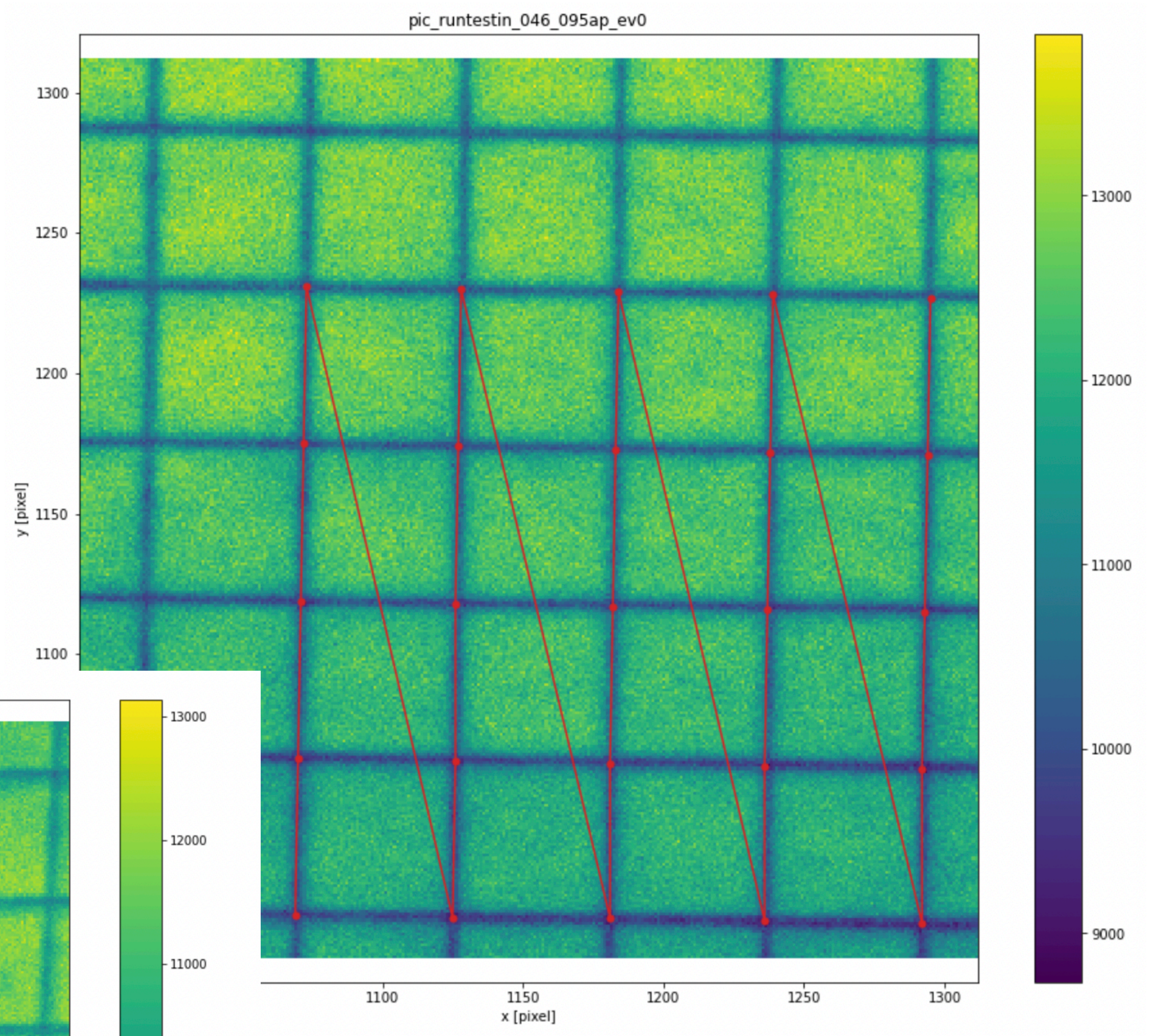
- we used a picture of a 4mm squared sheet collected with MANGO
- As "the object" we considered any 5x5 grid of vertices of the squares

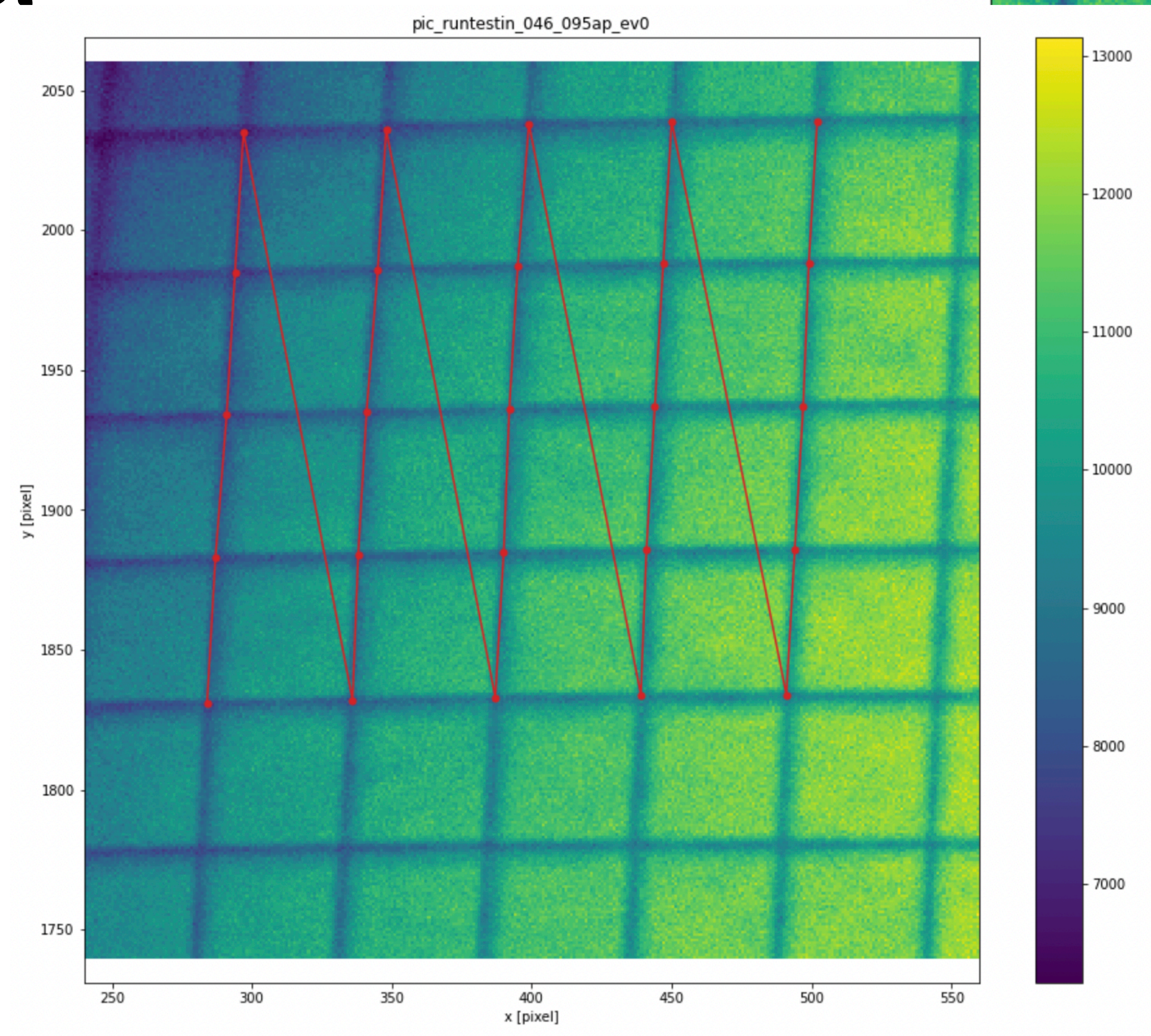


An example with the MANGO optical setup

Phase 1 - **Calibration**:

- As “the object” we considered any 5x5 grid of vertices of the squares
- We identified 11 of these grids. and we found the pixels correspondent to the vertices of the squares.

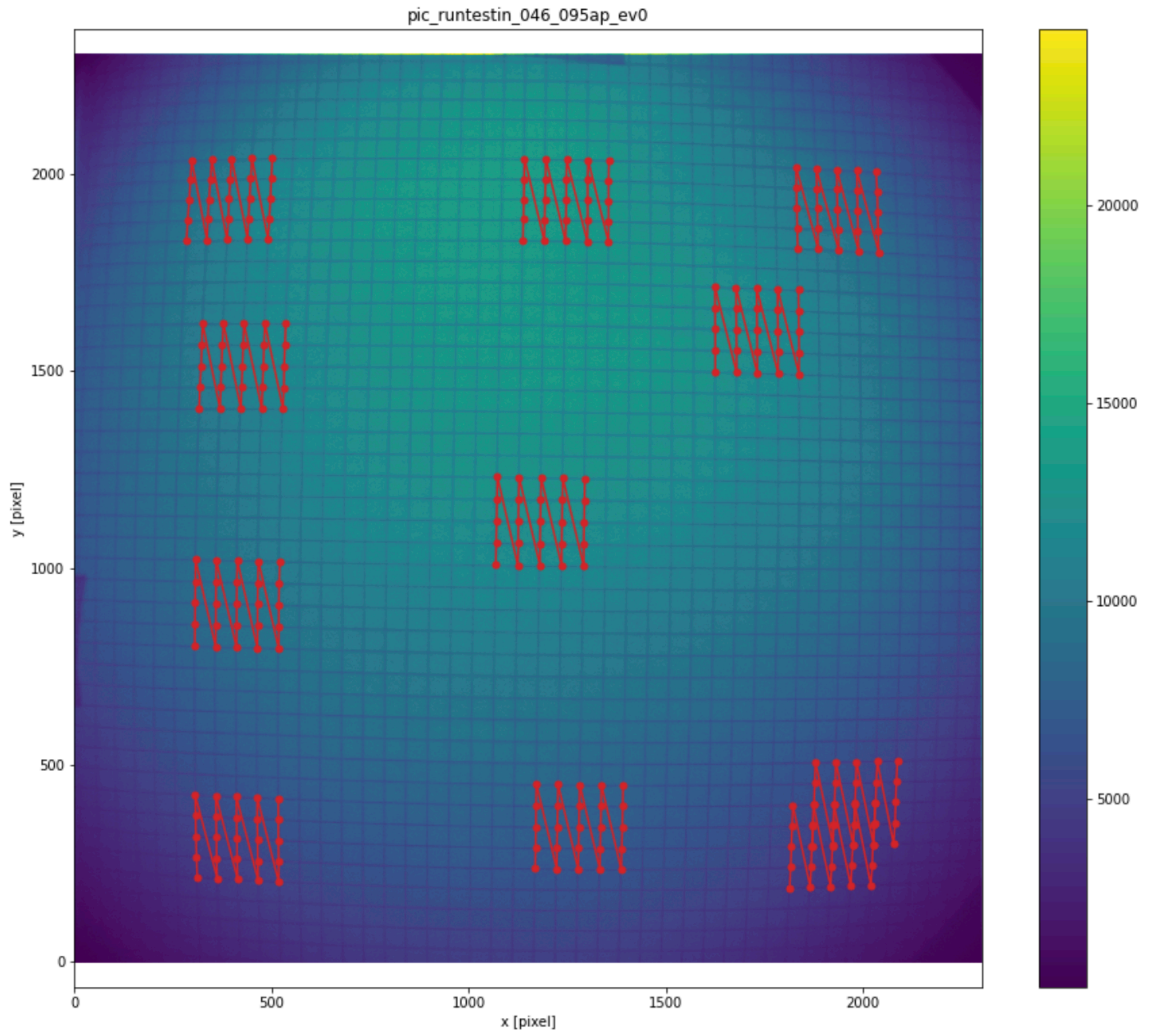




An example with the MANGO optical setup

Phase 1 - **Calibration**:

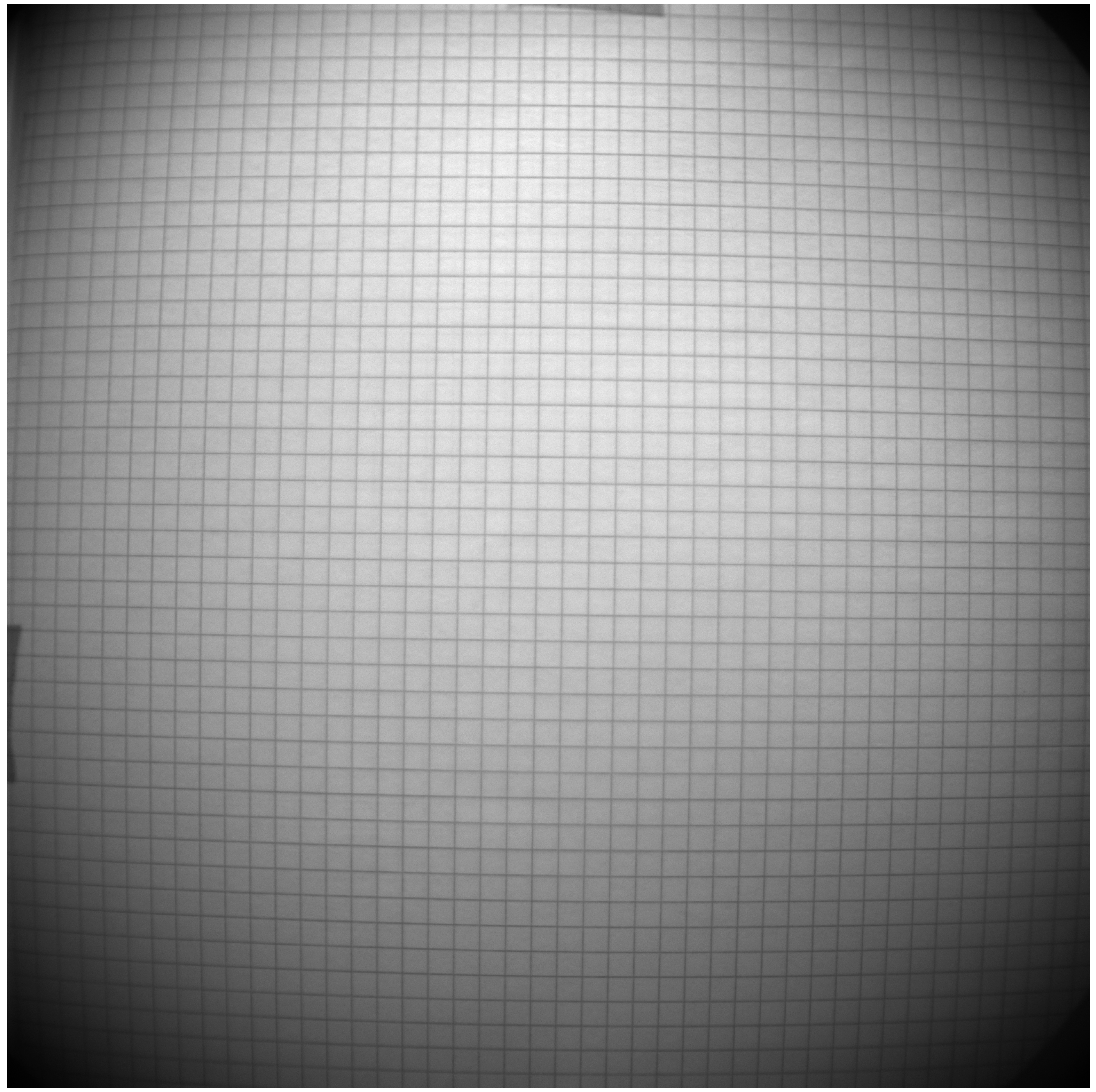
- with these calibration points we compute the distortion parameters...



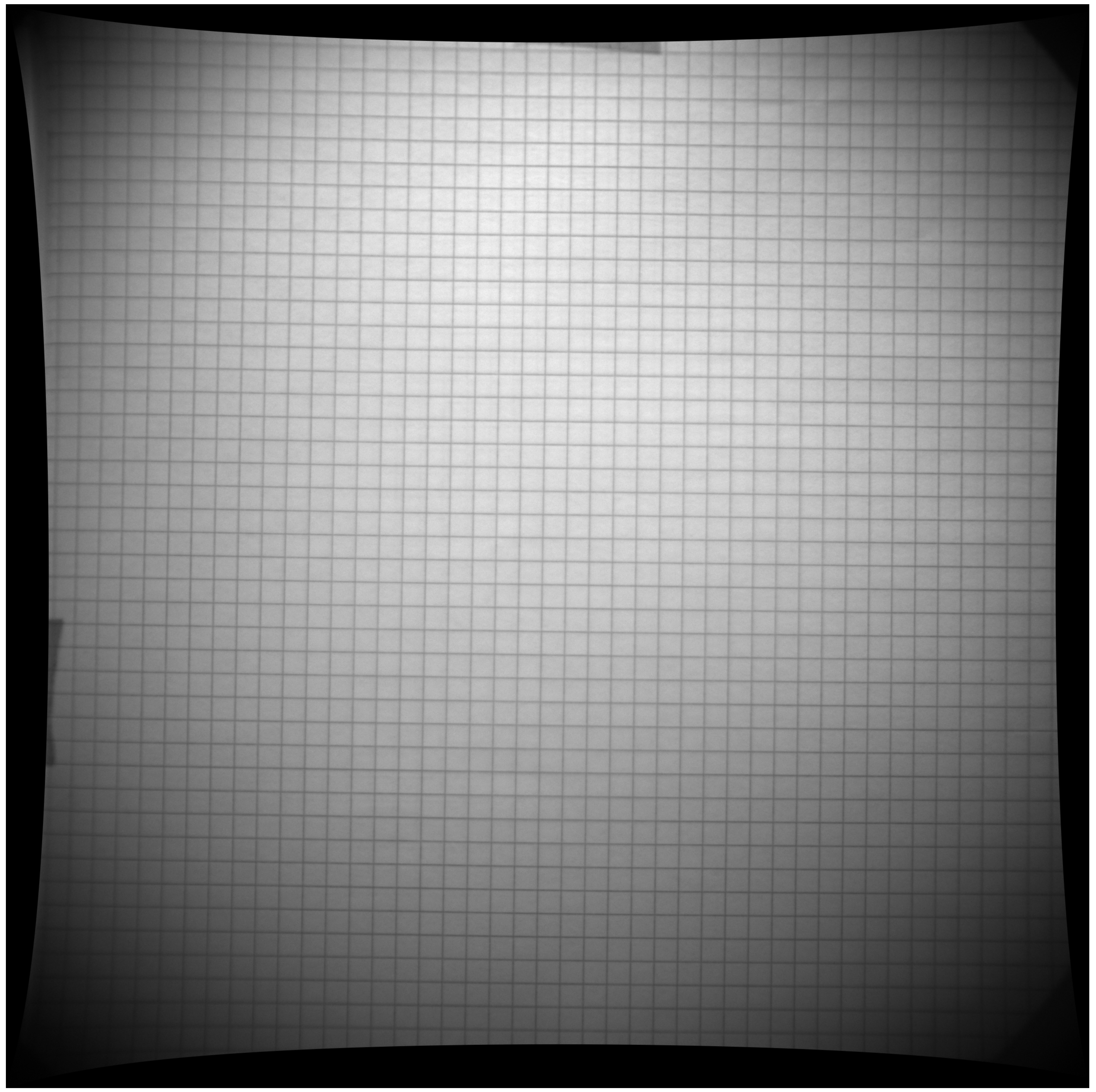
How does it work?

Phase 2 - **Correction**:

Original



Corrected



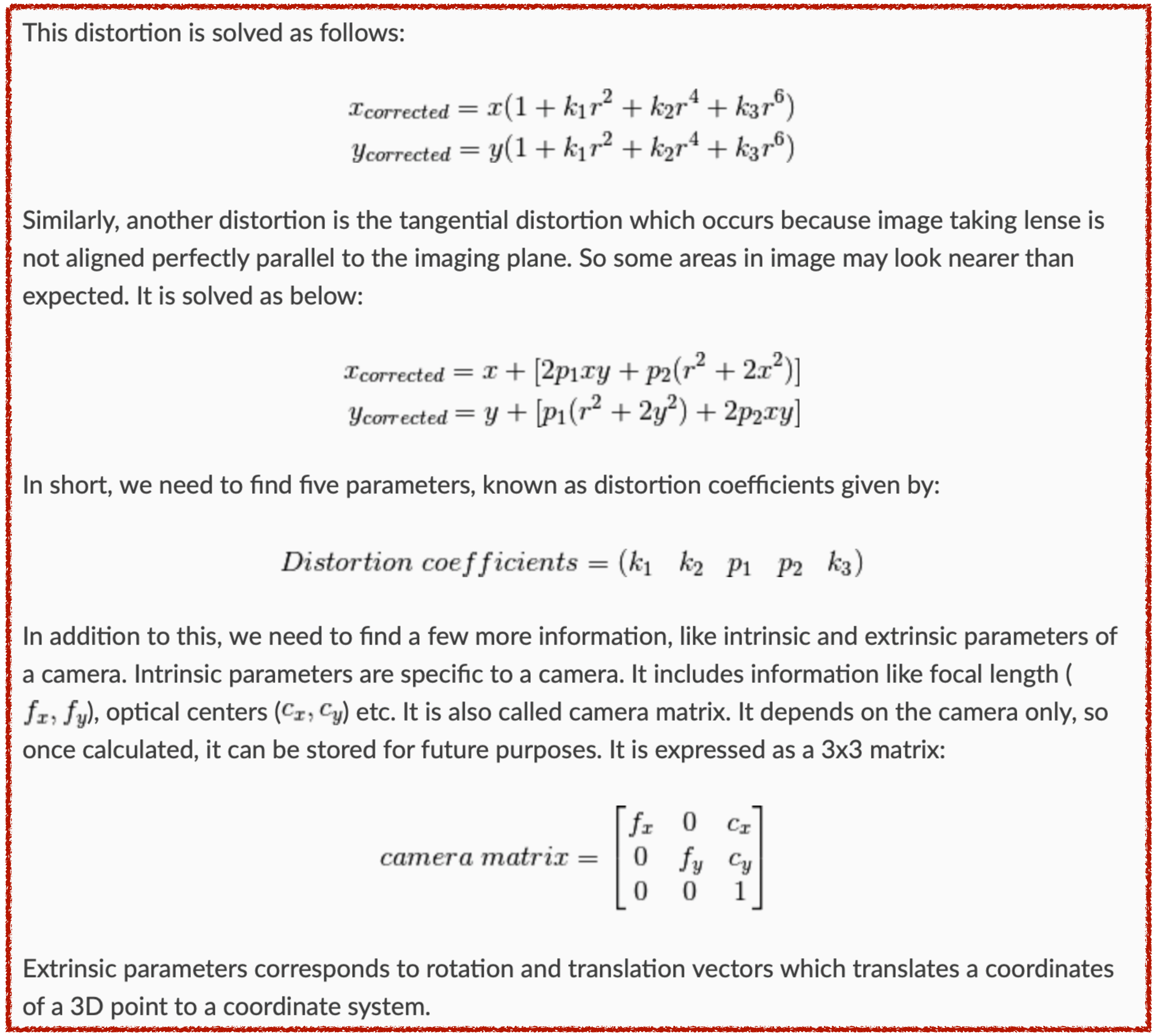
Conclusions

- Currently our pictures are affected by a **distortion effect.** We can apply a correction, and we found a package to do it that gives promising results.
- Open points:
  - ◉ test the effect of the correction on the LIME ^{55}Fe data.
  - ◉ To do that, we need at least one calibration picture (it can be the picture of a squared sheet as the one used for MANGO or a picture of the GEMs where the holes are clearly visible)

Backup

How does it work?

- The distortion determined by a **set of parameters**
- This parameters include:
  - ◉ **distortion coefficients**, due to lens and other geometrical effects
  - ◉ **camera parameters**, like focal length, optical center, etc.
  - ◉ **extrinsic parameter**, related to the position and inclination of an object in the 3D space



This distortion is solved as follows:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Similarly, another distortion is the tangential distortion which occurs because image taking lense is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected. It is solved as below:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

In short, we need to find five parameters, known as distortion coefficients given by:

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

In addition to this, we need to find a few more information, like intrinsic and extrinsic parameters of a camera. Intrinsic parameters are specific to a camera. It includes information like focal length (f_x, f_y), optical centers (c_x, c_y) etc. It is also called camera matrix. It depends on the camera only, so once calculated, it can be stored for future purposes. It is expressed as a 3x3 matrix:

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Extrinsic parameters corresponds to rotation and translation vectors which translates a coordinates of a 3D point to a coordinate system.

How does it work?

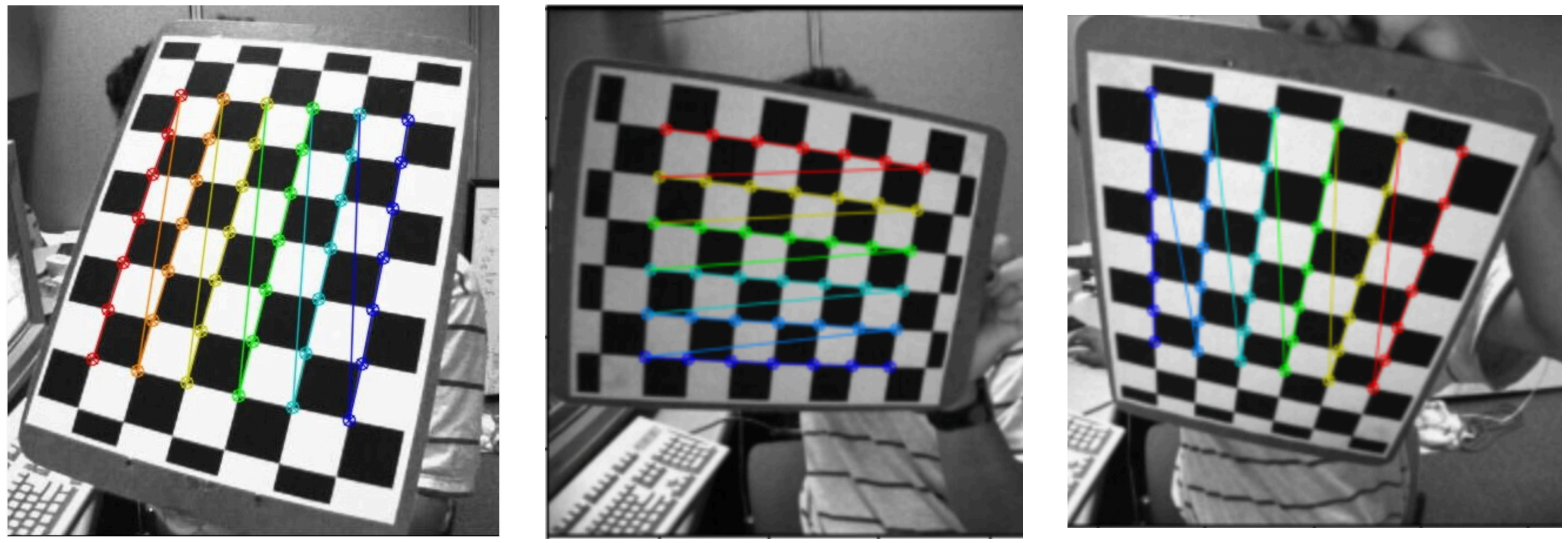
Phase 1 - **Calibration**:

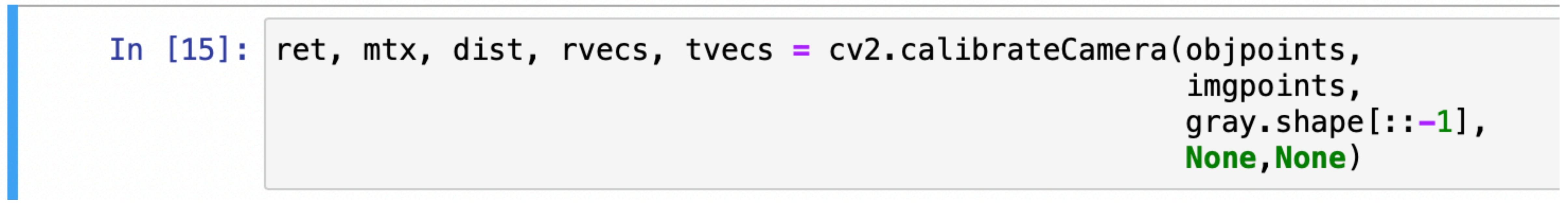
- a set of images (> 10) of the same object are collected in order to identify the best-fit values of all the parameters

- a pattern of points of the object (5x5, 7x7, 8x6, ...)

- a map to associate the pixel to the coordinate of the object in each picture

- with all these elements, we run **cv2.calibrateCamera()** and we get the parameters



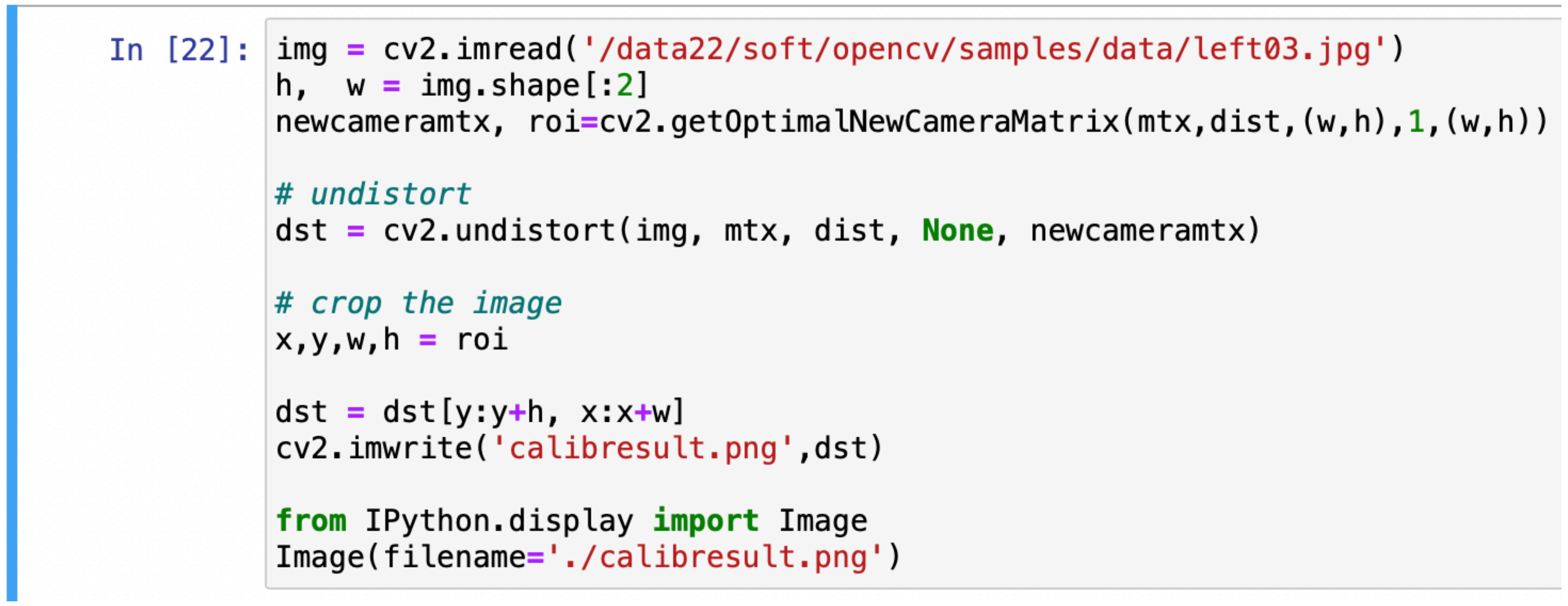


```
In [15]: ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
                                                             imgpoints,
                                                             gray.shape[::-1],
                                                             None,None)
```

How does it work?

Phase 2 - **Correction**:

- with the newly obtained parameters, we apply the transformations to the image



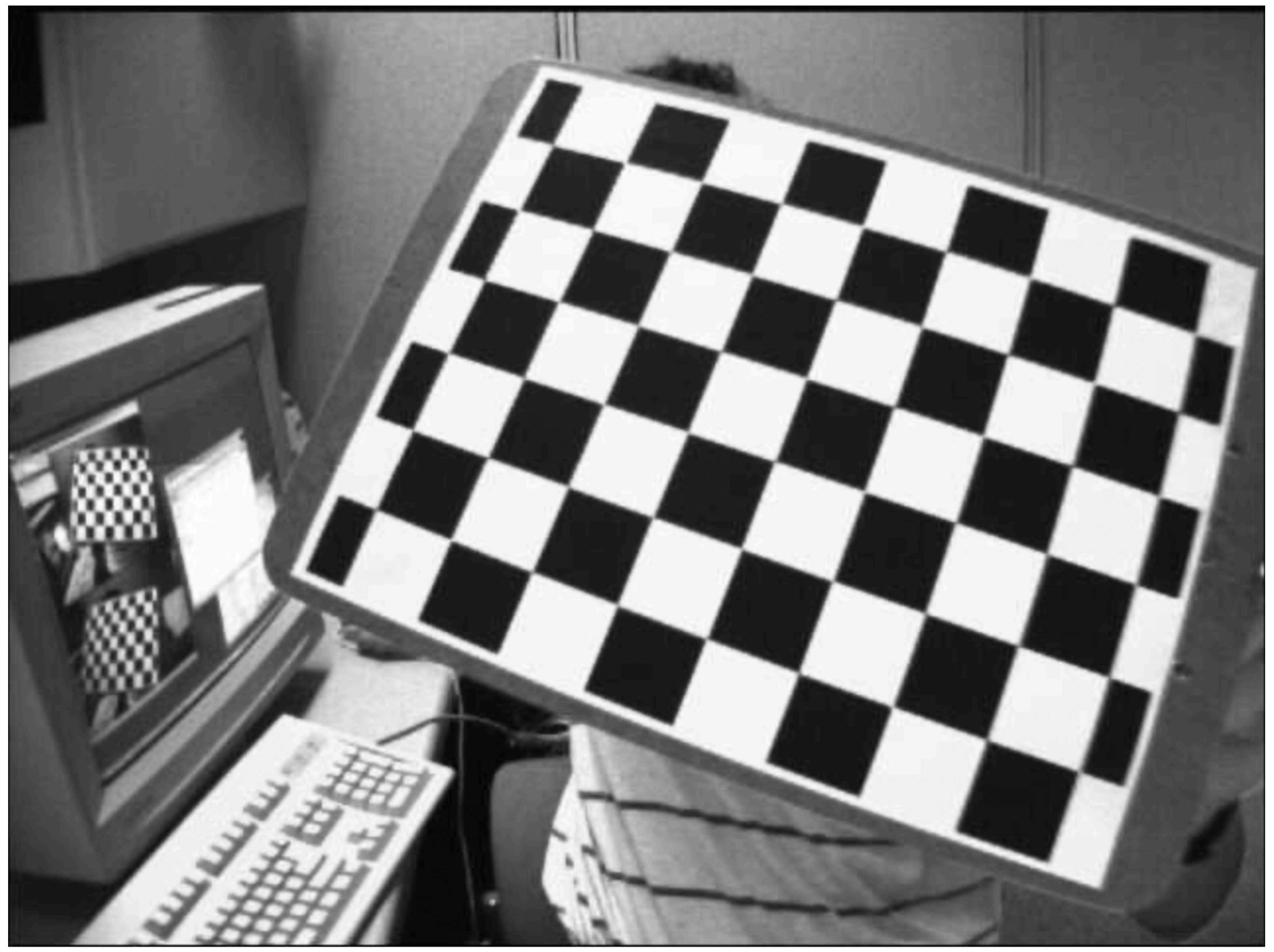
```
In [22]: img = cv2.imread('/data22/soft/opencv/samples/data/left03.jpg')
         h,  w = img.shape[:2]
         newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))

         # undistort
         dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

         # crop the image
         x,y,w,h = roi

         dst = dst[y:y+h, x:x+w]
         cv2.imwrite('calibresult.png',dst)

         from IPython.display import Image
         Image(filename='./calibresult.png')
```



$\longrightarrow$