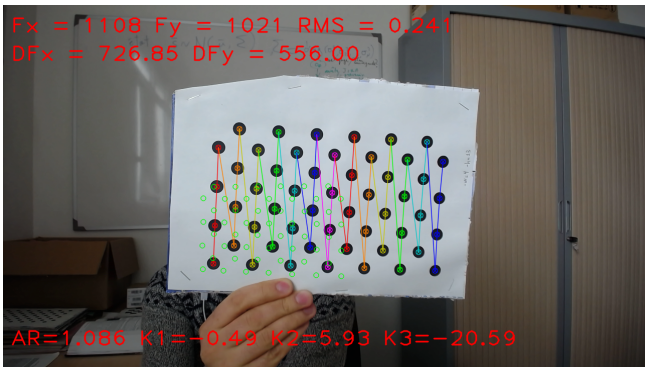
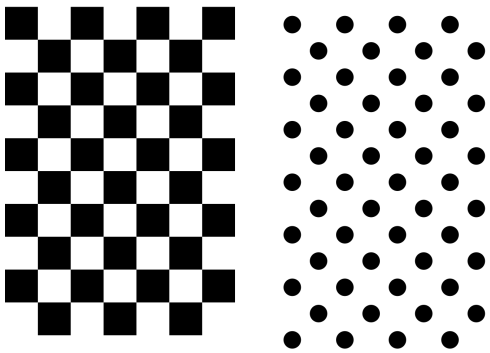
UE - Computer Vision
TP-4, **Single-view Geometry**

Panagiotis PAPADAKIS

The objective of this TP is to understand the process of perspective camera projection and the role of the various parameters involved.





(a) (b)

Figure 1: Alternate calibration patterns that you will use to calibrate your camera: (a) *chessboard*, (b) *circles*

1 Exercises

1.1 Calibrate your camera

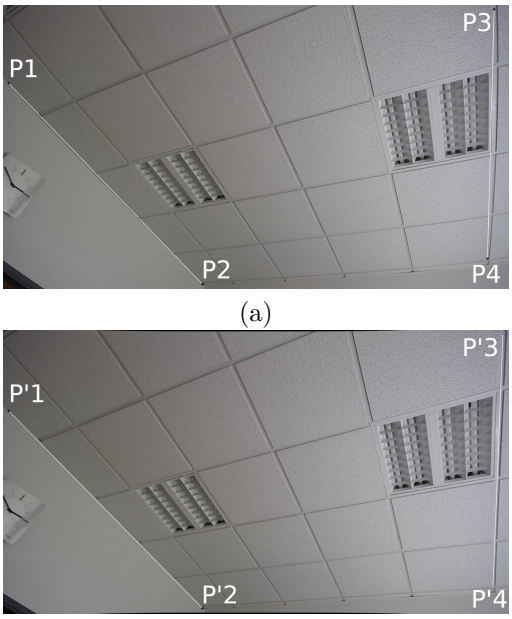
Plug the provided usb-camera to your computer. From an linux terminal, launch the already installed `OpenCV` software (`opencv_interactive-calibration`) for calibrating your camera, namely, for obtaining the camera intrinsic parameters. Depending on the calibration pattern that you are using, the arguments of the software should be adapted (check (`opencv_interactive-calibration --help`). Calibrate your camera first using the *chessboard* pattern and then the *circles* pattern. Keep the parameters that provided the least RMS error.

1.2 Undistort captured images

From this point forward, you will be using `python3` and the latest `OpenCV` version 4.2.0. To be able to use this version, for every new terminal session type:

```
source /opt/campux/virtualenv/computervision/bin/activate
```

Within an image taken by your camera, identify pairs of points that are part of straight lines in reality but they form curved lines in the projected images due to lens distortion (see Figure 2 (a) for an example).



(a)

(b)

Figure 2: Lens distortion effect: (a) Original image with lens distortion, (b) corrected (rectified) image.

Then, use `OpenCV`'s API functionality (`cv2.undistort`)[1] to *undistort* the captured image and verify that the same pairs of points do form straight lines (see Figure 2 (b)). To undistort your image, you will need the projection matrix and the radial distortion coefficients k_1, k_2, k_3 obtained previously from calibration. Allow your program to mouse-click directly on the distorted image to obtain the desired points and to draw the corresponding line segments (`cv2.setMouseCallback`)[2]

1.3 Projection of artificial object using camera's projection matrix

Show the projection of an artificial 3D rectangle provided to you as a set of 3D points[3], using the projection matrix that you obtained from your camera calibration (without accounting for lens distortion). You should obtain the image as shown below.

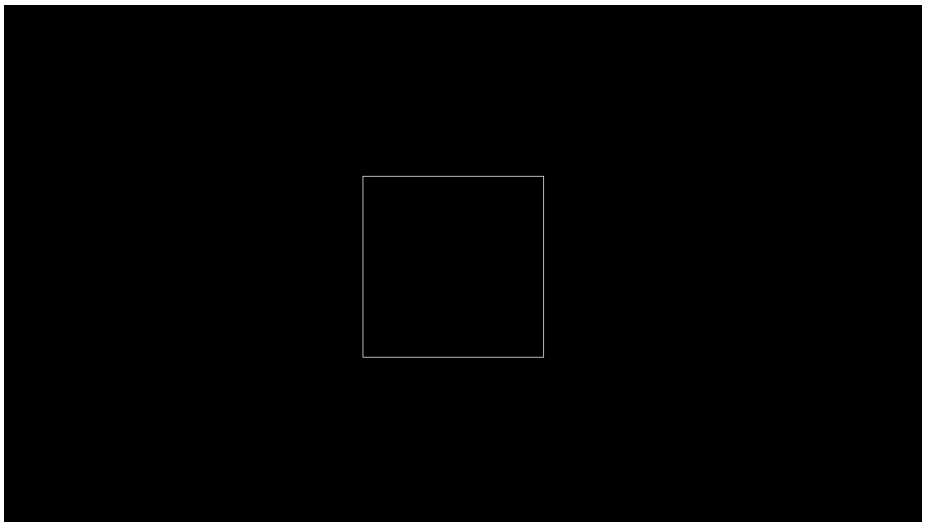


Figure 3: 3D rectangle projection using the camera's projection matrix

1.4 Alternate the focal length

Project the same 3D rectangle for a projection matrix with (i) double and (ii) half the original focal length. What do you expect with respect to the change

[1] See `https://docs.opencv.org/4.2.0/d9/d0c/group__calib3d.html#ga69f2545a8b62a6b0fc2ee060dc30559d`

[2] See `https://docs.opencv.org/4.2.0/d7/dfc/group__highgui.html#ga89e7806b0a616f6f1d502bd8c183ad3e`

[3] `rectangle` variable in file `https://drive.google.com/file/d/1n_pwyxI3Bu1NfwNOLnxROPolmNjUjwaX/view?usp=sharing`

of size of the projected rectangle? Does it match your expectation? You should obtain the images as shown below.

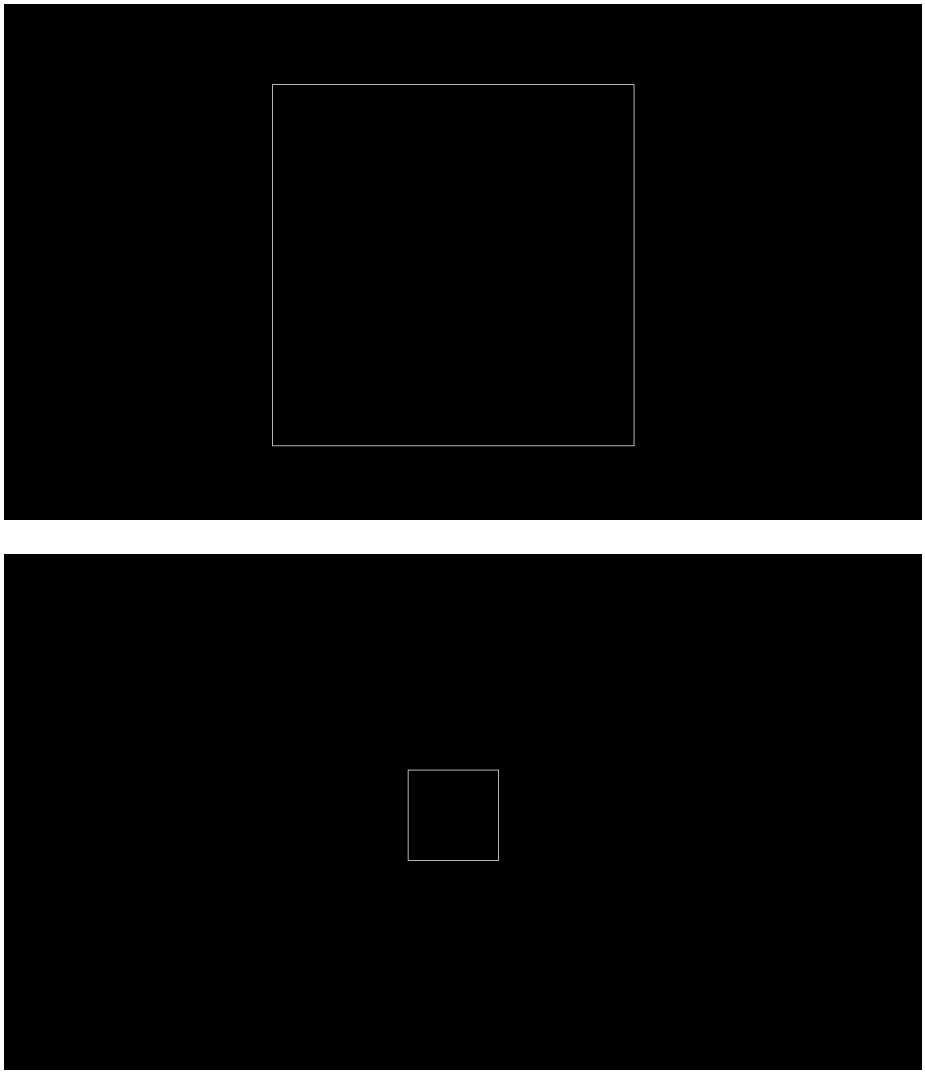


Figure 4: Projections of the same 3D rectangle using alternate focal lengths while preserving remaining intrinsic parameters

1.5 Projection after changing of camera coordinates

Show the projection of an artificial 3D cube provided to you as a set of 3D points[4], assuming that the camera coordinate frame is placed at ${}^{W}T_{C} = [\mathrm{R}|\mathbf{t}]$ with respect to the world, where:

$$ {}^{W}T_{C} = \begin{bmatrix} 0.92387953 & 0. & 0.38268343 & 0. \\ 0. & 1. & 0. & 0. \\ -0.38268343 & 0. & 0.92387953 & -0.2 \end{bmatrix} \tag{1} $$

Figure 5 depicts the image that you should obtain.

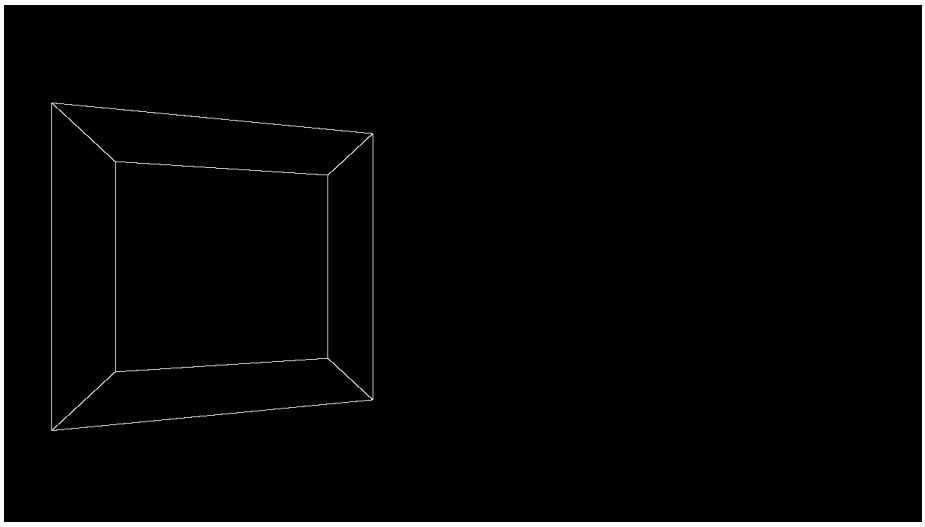


Figure 5: Projection of (wireframe) 3D cube from the camera assuming it is placed at a given pose $[\mathrm{R}|\mathbf{t}]$.

[4] `cube` variable in previously provided `tp4_data.py` file