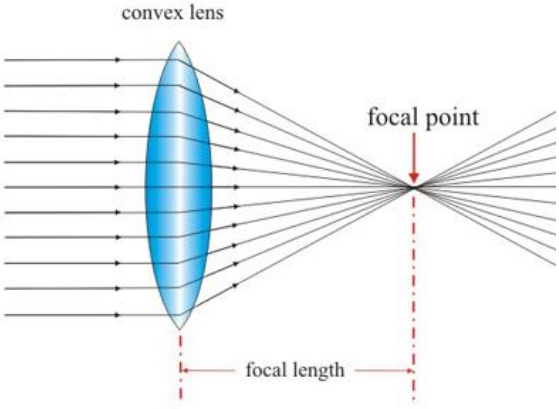
Camera Calibration Utilizing OpenCV and Python

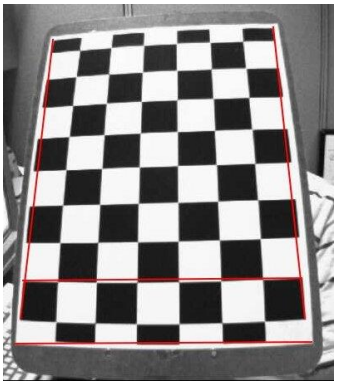
Created for the EE417 Course at Sabanci University by Kaan Özçelik No.17793

Camera calibration is used to reduce both radial and tengential distortion caused by both external factors and the lens located within regular CMOS cameras (After all the image plane projection method focuses rays of light on to a surface utilizing a lens that bends usually parallel rays of light on to a single point, thus creating the notion of FOV or field of view). Radial and Tangential distortion causes lines within a photo to appear skewed and non parallel. Due to this we can utilize basic geometry and image processing techniques to reduce this distortion to an acceptable level. (An extreme example of distortion can be 120-Degree Field of View cameras or fisheye lenses).

The image below demonstrates how light rays interact with a convex lens.

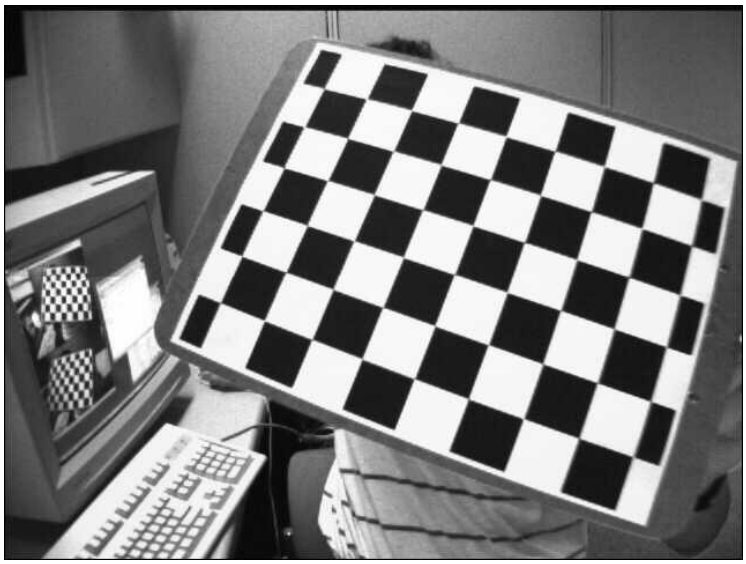


Below is an image demonstrating how distortion takes place in a usual photographic image. (Courtesy of the official OpenCV libraries.)



As you can see above the edges do not match with the red lines. A regular checkerboard pattern can be utilized to determine corner points within such an image and then reduce the distortion utilizing geometry and OpenCV's built in functions.

Lets attempt to discover the corners in the following distorted image.

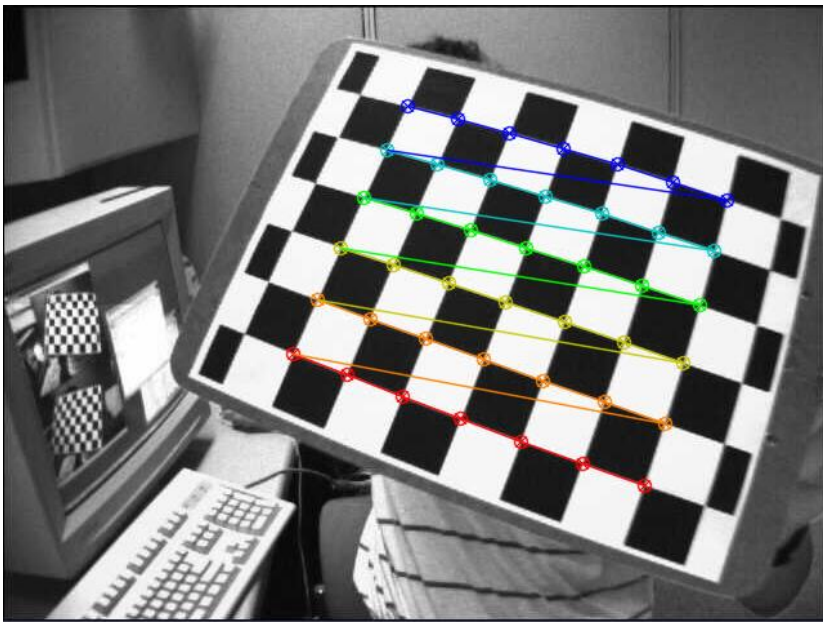


We utilize the python script below to extract the corners. (We assume 2D and 3D points for the image and then map them accordingly)

```
#-- Initial imports and required libraries --#
import numpy as np
import cv2
import glob

#-- Criteria to terminate the execution --#
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

#-- Create an array of possible object points (Intuitive) --#
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

#-- Main array to store possible 3-D and 2-D points --#
threeDPoints = [] #-- 3D points in real world space. --#
twoDPoints = [] #-- 2D points on the image plane. --#

imagePath = 'distortedImage.jpg'

#-- Read the image from the hard drive --#
inputImage = cv2.imread(imagePath)
#-- Convert the image to grayscale --#
grayScale = cv2.cvtColor(inputImage,cv2.COLOR_BGR2GRAY)

#-- Detect the corners on the checkerboard utilizing openCV's built in method --#
ret, corners = cv2.findChessboardCorners(grayScale, (7,6),None)

#-- If the points are discovered then map them to 3-D points --#
if ret == True:
    threeDPoints.append(objp)

    corners2 = cv2.cornerSubPix(grayScale,corners,(11,11),(-1,-1),criteria)
    twoDPoints.append(corners2)

    #-- Render the corners and display them in a window named 'Output' --#
    inputImage = cv2.drawChessboardCorners(inputImage, (7,6), corners2,ret)
    cv2.imshow('Output',inputImage)

#-- Wait for 10 seconds or for an input --#
cv2.waitKey(10000)
cv2.destroyAllWindows()
```

,

Now we get the following result.



Now we have to obtain the camera calibration matrix to undistort the image. Thankfully OpenCV has a built in function for this purpose. But if it did not we would have to utilize the mathematical equations below for both tangential and radial distortion to obtain the matrix.

Radial Distortion:

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

Tangential Distortion:

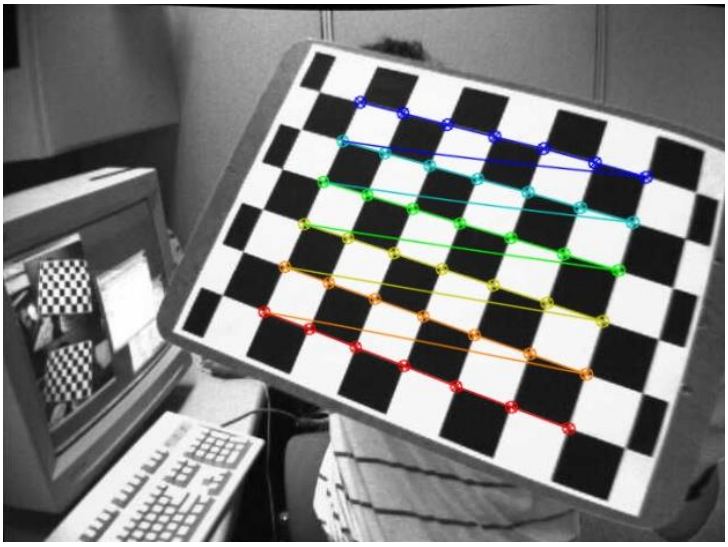
$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

See the next page for results and source code.

Below is the rest of the code of the script for camera calibration.

```
#-- Calibration Section --#
#-- We retrieve the return values from he cv2.calibrateCamera function -- #
#-- These return values are variables such as the distortion coefficients, rotation and translation vectors --#
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(threeDPoints, twoDPoints, grayScale.shape[::-1],None,None)

#-- Get the height and width values of the input image --#
h,  w = inputImage.shape[:2]

#-- We have to ensure the last two elements of the distortion matrix are zero! --#
dist[0][3] = 0.0
dist[0][4] = 0.0

#-- Generate a new optimal camera matrix --#
newCamMatrix, roi = cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))

#-- Apply the openCV undistort function --#
dst = cv2.undistort(inputImage, mtx, dist, None, newCamMatrix)

#-- Crop any excess from the image --#
x,y,w,h = roi
dst = dst[y:y + h, x:x + w]

#-- Display and write the undistorted image to the hard drive --#
cv2.imshow('Calibrated Result',dst)
cv2.imwrite('CalibratedResult.jpg', dst)
cv2.waitKey(10000)

cv2.destroyAllWindows()
```

Finally the image should appear to be something like the result below.



As can be observed there is a slight skew on the top edge of the image to compensate for the distortion translation in the image. In essence after correction the checkerboard appears straight yet the top of the clipboard appears to be slightly distorted.

Image Resources:

http://i.stack.imgur.com/bwiBG.jpg

OpenCV library sample images.

Textual Resources:

http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

http://www.ipol.im/pub/art/2011/ags-alde/

http://cameratico.com/articles/lens-distortion-correction-on-post-processing/