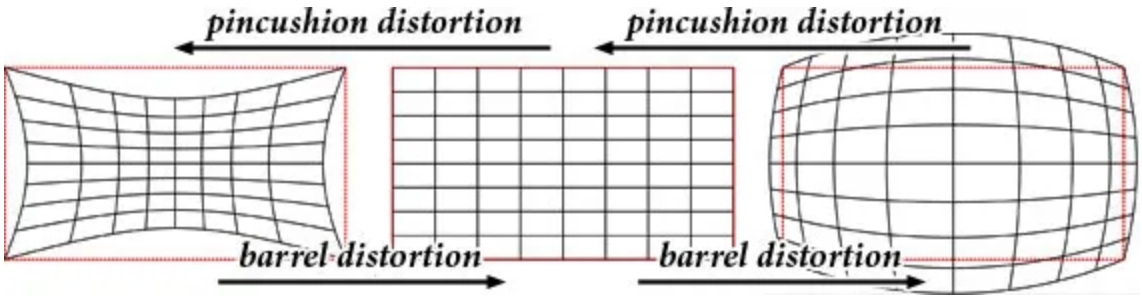
OpenCV Camera Calibration

Ali Yasin Eser · Follow
6 min read · Mar 1, 2020

Hello everyone! While I was working on my graduation project, I saw that there is not enough documentation for Computer Vision. For that reason, I've decided to document my project and share it with people who need it. Today we will cover the first part, the camera calibration. I won't dive into the Math behind it, but you can check the references or search a little bit. Let's start!

Technology is improving and getting cheaper each day. We can buy good quality cameras cheaper and use them for different purposes. But there is a downside with mass production cameras, they are not perfect after the build process. The precision is not enough and they need to be calibrated to extract meaningful data if we will use them for Vision purposes. After the calibration matrix(we will calculate it) is acquired, the fun part will start. Uncalibrated cameras have 2 kinds of distortion, barrel, and pincushion. Barrel distortion is looking like edges of the image are pushed. Pincushion distortion is looking like edges of the images are pulled.

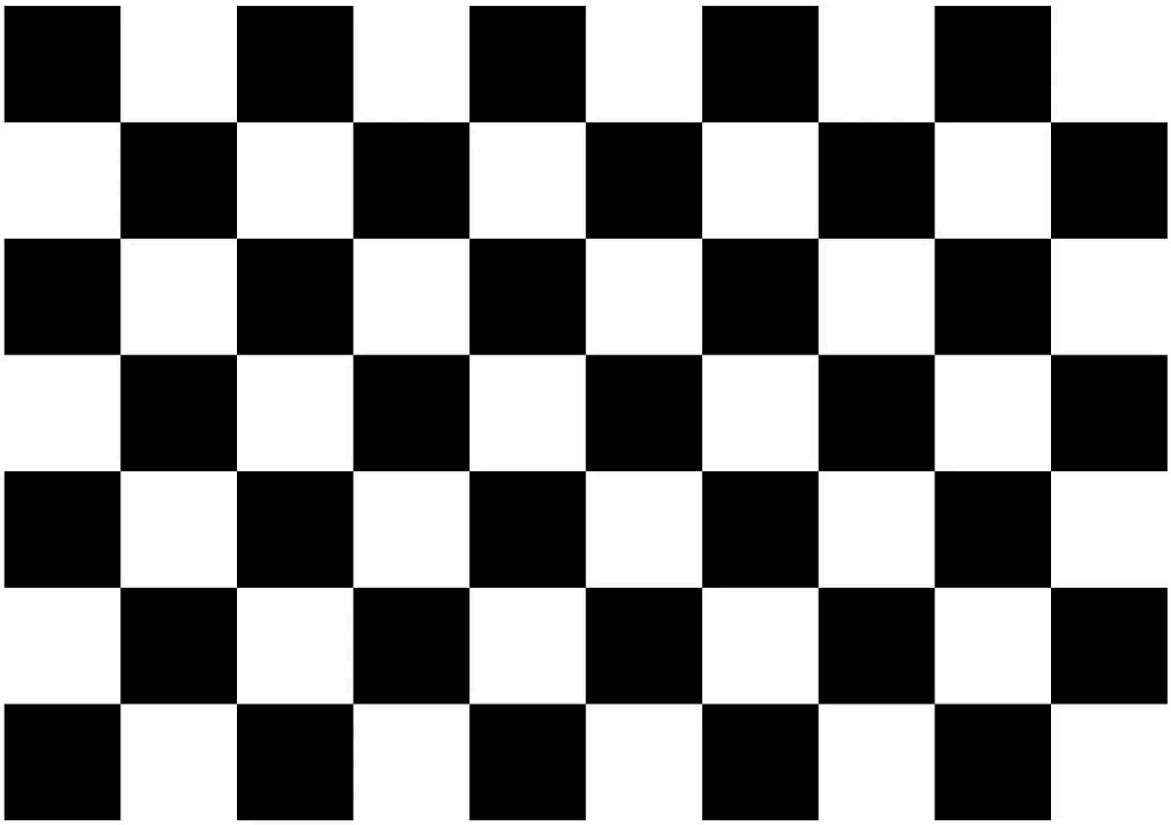


Barrel and Pincushion distortion.

(Left to Right) Distorted image and undistortion applied version.

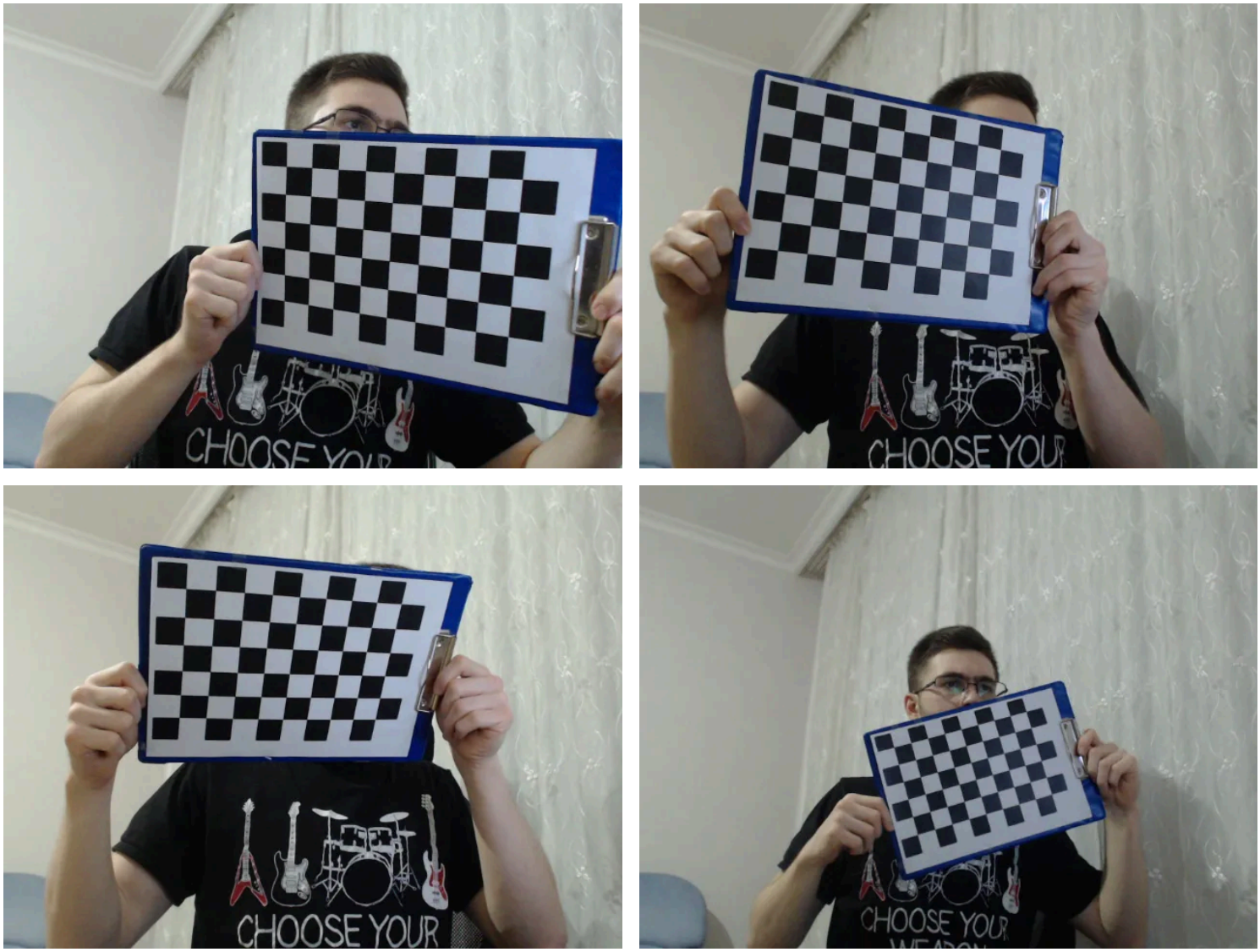
OpenCV library gives us some functions for camera calibration. Before starting, we need a chessboard for calibration. It should be well printed for quality. Please don't fit it to the page, otherwise, the ratio can be wrong. The key is that we will know each square size and we will assume each square is equal! There are different boards for calibration but chessboard is the most used one. Let's start:

1. Please download the chessboard(you can also search for a calibration board and download some other source). Measure the size of one square, for example, it can be 1.5 cm or so. This measurement is really important because we need to understand real-world distances. Chessboard:



The calibration chessboard. Print to A4 paper, no resize or fit(%100).

2. Glue the chessboard to a flat and solid object. It is also important that it should be flat, otherwise our perspective will be different. Open the camera(you can use OpenCV codes or just a standard camera app.) and take at least 20 images. They should be in different angles and distances because the calibration code needs various points with different perspectives. Some examples:



Example calibration images.

3. Move the images into a directory. We can work on the python code now. We need the OpenCV library for python now. I used Python 3.6.4 for this example, please keep that in mind. You can use the command below to install OpenCV for python:

```
1  pip install opencv-python opencv-contrib-python numpy
```

install_necessary_libs.sh hosted with ❤️ by GitHub view raw

OpenCV-python is the OpenCV library. Contrib will be used next blog, it is not necessary for now but definitely recommended. Numpy is a scientific computation package and OpenCV also uses it, that's why we need it.

Let's start to the calibration code:

```
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)

def calibrate(dirpath, prefix, image_format, square_size, width=9,
height=6):
    """ Apply camera calibration operation for images in the given
directory path. """
    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,
(8,6,0)
    objp = np.zeros((height*width, 3), np.float32)
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)
```

Parameters:

- dirpath: The directory that we moved our images.
- prefix: Images should have the same name. This prefix represents that name. (If the list is: image1.jpg, image2.jpg … it shows that the prefix is "image". Code is generalized but we need a prefix to iterate, otherwise, there can be any other file that we don't care about.)
- image_format: "jpg" or"png". These formats are supported by OpenCV.
- square_size: Edge size of one square.

- width: Number of intersection points of squares in the long side of the calibration board. It is 9 by default if you use the chessboard above.
- height: Number of intersection points of squares in the short side of the calibration board. It is 6by default if you use the chessboard above.

objp is our chessboard matrix. We will initialize it with coordinates and multiply with our measurement, square size. It will become our map for the chessboard and represents how the board should be.

```
objp = objp * square_size # if square_size is 1.5 centimeters, it
would be better to write it as 0.015 meters. Meter is a better metric
because most of the time we are working on meter level projects.
```

The chessboard is a 9x6 matrix so we set our width=9 and height=6. These numbers are the intersection points square corners met. “Criteria” is our computation criteria to iterate calibration function. You can check OpenCV documentation for the parameters.

```
# Arrays to store object points and image points from all the images.
objpoints = []  # 3d point in real world space
imgpoints = []  # 2d points in image plane.

# Some people will add "/" character to the end. It may brake the
code so I wrote a check.
if dirpath[-1:] == '/':
    dirpath = dirpath[:-1]

images = glob.glob(dirpath+'/' + prefix + '*.' + image_format) #
```

objpoints is the map we use for the chessboard. imgpoints is a matrix that holds chessboard corners in the 3D world. These coordinates are coming

from the pictures we have taken.

```
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (width, height),
None)

    # If found, add object points, image points (after refining them)
    if ret:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1,
-1), criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (width, height),
corners2, ret)

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

return [ret, mtx, dist, rvecs, tvecs]
```

We have a for loop to iterate over the images. imread gets the image and cvtColor changes it to grayscale. findChessboardCorners gets the points(so easy!) and we have the points already! Be careful that it will look for the number of corners, if you write them wrong it can't find the chessboard. You can check the ret value for that.

If the function returns successfully we can start to interpolate. Our goal is here to check if the function found the corners good enough. We show it to the user, thanks to the drawChessboardCorners function. If corners are not matching good enough, drop that image and get some new ones. Otherwise, it can affect the calibration process.

The last step, use calibrateCamera function and read the parameters. We feed our map and all the points we detected from the images we have and magic happens! You can return it, write to a file or print out. The whole code is below for taking images, load and save the camera matrix and do the calibration:

```python
import numpy as np
import cv2
import glob
import argparse

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)


def calibrate(dirpath, prefix, image_format, square_size, width=9, height=6):
    """ Apply camera calibration operation for images in the given directory path. """
    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(8,6,0)
    objp = np.zeros((height*width, 3), np.float32)
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)

    objp = objp * square_size

    # Arrays to store object points and image points from all the images.
    objpoints = []  # 3d point in real world space
    imgpoints = []  # 2d points in image plane.

    if dirpath[-1:] == '/':
        dirpath = dirpath[:-1]

    images = glob.glob(dirpath+'/' + prefix + '*.' + image_format)

    for fname in images:
        img = cv2.imread(fname)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Find the chess board corners
        ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

        # If found, add object points, image points (after refining them)
        if ret:
            objpoints.append(objp)

            corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
            imgpoints.append(corners2)

            # Draw and display the corners
            img = cv2.drawChessboardCorners(img, (width, height), corners2, ret)

    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[
```

```
    return [ret, mtx, dist, rvecs, tvecs]
def save_coefficients(mtx, dist, path):
    """ Save the camera matrix and the distortion coefficients to given path/file. """
    cv_file = cv2.FileStorage(path, cv2.FILE_STORAGE_WRITE)
    cv_file.write("K", mtx)
    cv_file.write("D", dist)
    # note you *release* you don't close() a FileStorage object
    cv_file.release()
def load_coefficients(path):
    """ Loads camera matrix and distortion coefficients. """
    # FILE_STORAGE_READ
    cv_file = cv2.FileStorage(path, cv2.FILE_STORAGE_READ)

    # note we also have to specify the type to retrieve other wise we only get a
    # FileNode object back instead of a matrix
    camera_matrix = cv_file.getNode("K").mat()
    dist_matrix = cv_file.getNode("D").mat()

    cv_file.release()
    return [camera_matrix, dist_matrix]
```

python_opencv_calibration.py hosted with ❤️ by GitHub view raw

Example Usage:

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Camera calibration')
    parser.add_argument('--image_dir', type=str, required=True, help='image directory pa
    parser.add_argument('--image_format', type=str, required=True,  help='image format,
    parser.add_argument('--prefix', type=str, required=True, help='image prefix')
    parser.add_argument('--square_size', type=float, required=False, help='chessboard sq
    parser.add_argument('--width', type=int, required=False, help='chessboard width size
    parser.add_argument('--height', type=int, required=False, help='chessboard height si
    parser.add_argument('--save_file', type=str, required=True, help='YML file to save c

    args = parser.parse_args()
    ret, mtx, dist, rvecs, tvecs = calibrate(args.image_dir, args.prefix, args.image_for
    save_coefficients(mtx, dist, args.save_file)
    print("Calibration is finished. RMS: ", ret)
```

usage_python_opencv_calibration.py hosted with ❤️ by GitHub view raw

argparse library is not required but I used it because it makes our code more readable. Arguments are the same as we feed into the functions, except “save_file”. This argument asks for a filename that we will store our calibration matrix. An example: “camera.yml”.

I tried to explain as easily as possible. I hope it helps people who need calibration. Calibration is a fatal step to start, before implementing any Computer Vision task. Without a good calibration, all things can fail. So please make sure that you calibrated the camera well. Thanks for reading!

My next Blog: ArUco markers and Tracking

Resources:

1. Images: https://www.google.com.tr/search?q=camera+distortion+example&client=chrome-omni&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjv9sDFoMrbAhWKhqYKHZsHDk8Q_AUICigB&biw=1920&bih=929#imgrc=BbnVAnjEndc0qM: https://www.google.com.tr/search?q=barrel+distortion&source=lnms&tbm=isch&sa=X&ved=0ahUKEwj54qXSn8rbAhXBlCwKHTraA_QQ_AUICigB&biw=1920&bih=929#imgrc=FD8BNL4aL3iFaM: https://www.google.com.tr/search?q=opencv+chessboard&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjPnt3TocrbAhXH2SwKHaM1DscQ_AUICigB&biw=1920&bih=929#imgrc=3Y_uhSD2kFeCqM:
2. OpenCV calibration documentation. They also explain the math side of it: https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/cam

era_calibration.html
https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

3. Basis of the code. It is an ArUco tracking code but calibration included: https://github.com/njanirudh/Aruco_Tracker

Opencv | Opencv Python | Camera Calibration | Calibration | Cameras

Written by Ali Yasin Eser

216 Followers · 51 Following

iOS Developer with Computer Vision and Embedded Systems background. Solo musician with 3 albums.

Follow

Responses (7)

What are your thoughts?

Respond

Neil Young
Mar 5, 2022

Until today I was also focused to get the proper square_size. But have you also already noted, that it doesn't play any role, what square_size you configure?