Advanced-Lane-Line

October 21, 2019

0.1 Camera calibration

When camera's lens capture an image, they do not capture the true image, but a distortion of the original image. In photography, there are two types of distortions: optical and perspective. Both result in some kind of deformation of images – some lightly and others very noticeably. While optical distortion is caused by the optical design of lenses (and is therefore often called "lens distortion"), perspective distortion is caused by the position of the camera relative to the subject or by the position of the subject within the image frame. And it is certainly important to distinguish between these types of distortions and identify them, since you will see them all quite a bit in photography. The points at the center of image have lower distortions where as the points away from the center have higher distortion. This distortion can be due to difference in distance from the center of camera, differential bending of rays at different lens location or perspective distortion.

```
In [1]: # Load nessesery modules and set up
        import cv2
        import glob
        import pickle
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        from sklearn.metrics import mean_squared_error
        %matplotlib inline
```

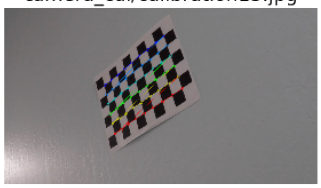
0.2 Extract object points and image points for camera calibration.

```
In [2]: x_cor = 9 #Number of corners to find
        y_cor = 6
        # Prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
        objp = np.zeros((y_cor*x_cor,3), np.float32)
        objp[:,:2] = np.mgrid[0:x_cor, 0:y_cor].T.reshape(-1,2)

        # Arrays to store object points and image points from all the images.
        objpoints = [] # 3d points in real world space
        imgpoints = [] # 2d points in image plane.
        images = glob.glob('camera_cal/calibration*.jpg') # Make a list of paths to calibration
        # Step through the list and search for chessboard corners
        corners_not_found = [] #Calibration images in which opencv failed to find corners
```

```
plt.figure(figsize=(12, 18)) #Figure for calibration images
plt.figtext(0.5,0.9,'Image with corners patterns drawn', fontsize=22, ha='center')
for idx, fname in enumerate(images):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Conver to grayscale
    ret, corners = cv2.findChessboardCorners(gray, (x_cor,y_cor), None) # Find the chess
    # If found, add object points, image points
    if ret == True:
        objpoints.append(objp)
        imgpoints.append(corners)
        plt.subplot(6, 3, len(imgpoints))
        cv2.drawChessboardCorners(img, (x_cor,y_cor), corners, ret)
        plt.imshow(img)
        plt.title(fname)
        plt.axis('off')
    else:
        corners_not_found.append(fname)
plt.show()

print ('Corners were found on', str(len(imgpoints)), 'out of', str(len(images)), 'it is'
# Draw pictures
plt.figure(figsize=(12, 4))
plt.figtext(.5,.8,'Images in which cv2 failed to find desired corners', fontsize=22, ha=
for i, p in enumerate(corners_not_found):
    plt.subplot(1, 3, i+1)
    plt.imshow(mpimg.imread(p)) #draw the first image of each class
    plt.title(p)
    plt.axis('off')
plt.show()
#plt.savefig("no_corners.jpg")
```

Image with corners patterns drawn

camera_cal/calibration13.jpg



camera_cal/calibration10.jpg

camera_cal/calibration11.jpg

camera_cal/calibration12.jpg

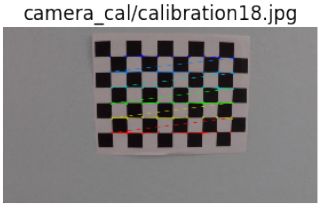
camera_cal/calibration14.jpg

camera_cal/calibration15.jpg

camera_cal/calibration16.jpg

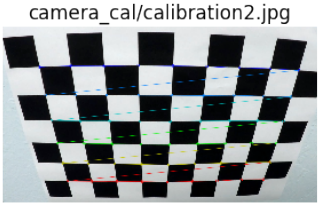
camera_cal/calibration17.jpg

camera_cal/calibration18.jpg

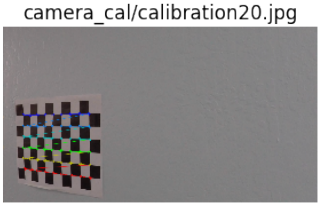


camera_cal/calibration19.jpg

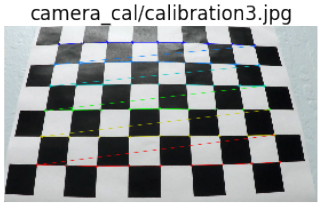
camera_cal/calibration2.jpg



camera_cal/calibration20.jpg



camera_cal/calibration3.jpg



camera_cal/calibration6.jpg

camera_cal/calibration7.jpg

camera_cal/calibration8.jpg

camera_cal/calibration9.jpg

```
Corners were found on 17 out of 20 it is 85.0 % of calibration images
```

As we can see, 85% of calibration images were succesfully recognized by `cv2.findChessboardCorners`. The rest 3 images were not processed as the algorithm failed to find required number of corners.

We can apply pinhole camera model using OpenCV easily, and some useful methods like cv2.calibrateCamera() could help us to apply the pinhole camera model on the images. Then we pass the processed images to cv2.undistort() function.

We can obvious found that the original image was slightly distorted in some places compared to undistorted images thanks to the regular white and black shape of the squares on the chessboard. But if we pass the real-world data into the model, the scene camera images have only changed a little.

```
In [3]: # Undistortion process
        # Test undistortion on an image
        img = cv2.imread('camera_cal/calibration1.jpg')
        img_size = (img.shape[1], img.shape[0])

        # Do camera calibration given object points and image points
        ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img_size,None,N

        def undistort(img):
            return cv2.undistort(img, mtx, dist, None, mtx)

        plt.figure(figsize=(12, 7))
        plt.subplot(2, 2, 1)
        plt.imshow(img)
        plt.title("Original Image")
        plt.subplot(2, 2, 2)
        plt.imshow(undistort(img))
        plt.title("Undistorted Image")
        img = cv2.imread('test_images/test1.jpg')
        plt.subplot(2, 2, 3)
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
plt.title("Original Image")
plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(undistort(img), cv2.COLOR_BGR2RGB))
plt.title("Undistorted Image")
plt.savefig("output_images/undist_img.jpg")
```