CameraCalibration

November 1, 2021

[1]:
```
import cv2
import numpy as np
import os
import glob
import matplotlib.pyplot as plt
```

[2]:
```
'''Function to plot a box based on the matrix given, as long as the first 8␣
↪values are the 8 corners'''
def plot(box, pointcolor = 'k', linecolor = 'k', degree1 = 30, degree2 = 45):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')


    ax.scatter3D(box[:, 0], box[: ,1], box[:, 2], c = pointcolor)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')


    verts = [[box[0],box[1],box[2],box[3]], [box[4],box[5],box[6],box[7]],␣
↪[box[0],box[1],box[5],box[4]], [box[3],box[2],box[6],box[7]]]
            #[box[1],box[2],box[6],box[5]], [box[1],box[2],box[5],box[7]]]

    # plot sides
    ax.add_collection3d(Poly3DCollection(verts, facecolors='w', linewidths=1,␣
↪edgecolors=linecolor, alpha=.25))
    ax.view_init(degree1, degree2)


#Rotation
'''Rotation function to return Omega based on what degree was given'''
def rotation(x):
    return np.array([[np.cos(x * np.pi/180), 0, -np.sin(x * np.pi/180)],
                     [0, 1, 0],
                     [np.sin(x * np.pi/180), 0 , np.cos(x * np.pi/180)]])
```

```
def homogenous(X):
    # print(X.shape[1])
    return np.vstack((X, np.ones([1,X.shape[1]])))

'''Pinhole camera funciton to find the projected values'''
def pinhole(W, Lambda, Omega, Tau):

    OmegaTau = np.hstack([Omega, Tau.T])

    X = Lambda@ OmegaTau @ W

    #divide by to the z values in order to make it 1
    X = X/X[2]

    X = X.T

    return X

'''Calculate the normalized coordinates'''
def findX_(Lambda, Omega, Tau, W):
    # print(Lambda.shape)
    # print(Omega.shape)
    # print(Tau.shape)
    # print(W.shape)
    OmegaTau = np.hstack([Omega,Tau.T])

    X =  Lambda @ OmegaTau @ W

    X = X/X[2]

    #equation 14.27
    X_ = np.linalg.inv(Lambda) @ X

    X_ = X_/X_[2]

    return X_

'''Consturct the systems of equations from equation 14.30'''
def constructA(W, X_):
    #make empty array to be able to populate
    #print(W.shape)
    A = np.zeros((W.shape[0]*2, 12))
    # print(A.shape)
    i = 0
    for w, x_ in zip(W, X_):
        # print("\nw: ",w)
        # print("x: ",x_)
```

```
        A[i] = np.array([w[0], w[1], w[2], 1, 0 ,0 ,0 , -w[0]*x_[0],␣
↪-w[1]*x_[0], -w[2]*x_[0], -x_[0]])
        A[i+1] =  np.array([0, 0, 0 ,0 ,w[0], w[1], w[2], 1 , -w[0]*x_[1],␣
↪-w[1]*x_[1], -w[2]*x_[1], -x_[1]])
        # print(A[i])
        # print(A[i+1])
        i += 2

    return A

''' Find the estimates of Omega and Tau'''
def findEstimate(A):

    U,L,V = np.linalg.svd(A)

    #set b hat equal to last column
    b_ = V.T[:,-1]

    Omega = np.array([[b_[0],b_[1],b_[2]],
                      [b_[4],b_[5],b_[6]],
                      [b_[8],b_[9],b_[10]]])

    Tau = np.array([b_[3], b_[7], b_[11]])

    #every fourth point is tau
    U_, L_, V_ = np.linalg.svd(Omega)

    Omega_hat = -(U_@V_)


    Tau_hat = np.array([np.sum(Omega_hat)/np.sum(Omega)*Tau])

    # Tau_hat = Tau_hat *-1

    # Tau_hat = Tau_hat*10

    # return np.hstack([Omega_hat, Tau_hat.T])
    return Omega_hat, Tau_hat
```

[3]:
```
#Functions used to rename and resize the images

# for count, filename in enumerate(os.listdir("images")):
#     dst ="image" + str(count) + ".jpg"
#     src ='images/'+ filename
#     dst ='images/'+ dst

#     # rename() function will
```

```
#     # rename all the files
#     os.rename(src, dst)

# images = glob.glob('./images/*.jpg')
# # print(images[0])

# for image in images:
#     img = cv2.imread(image)

#     print('Original Dimensions : ',img.shape)

#     scale_percent = 50 # percent of original size
#     width = int(img.shape[1] * scale_percent / 100)
#     height = int(img.shape[0] * scale_percent / 100)
#     dim = (width, height)

#     # resize image
#     resized = cv2.resize(img, dim)
#     cv2.imwrite(image,resized)

images = glob.glob('./images/*.jpg')

for fname in images:
    #read images
    img = cv2.imread(fname)
    print(img.shape)
```

```
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
```

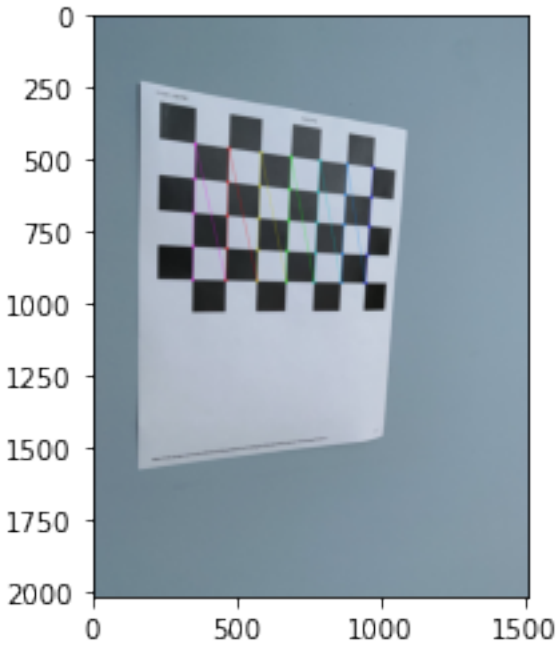
[4]:
```
#number of vertical and horizontal
checkerboard = (5,7)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objpoints = []

imgpoints = []
```

[5]:
```
objp = np.zeros((1, checkerboard[0] * checkerboard[1],3),np.float32)

objp[0,:,:2] = np.mgrid[0:checkerboard[0], 0:checkerboard[1]].T.reshape(-1, 2)

print(objp.shape)


prev_img_shape = None
```

```
(1, 35, 3)
```

[6]:
```
print(len(images))
for fname in images:
    #read images
    img = cv2.imread(fname)
    print(img.shape)
    #get grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


    #function to find corners
    ret, corners = cv2.findChessboardCorners(gray, checkerboard)
                                             # flags
                                             # cv2.CALIB_CB_ADAPTIVE_THRESH +
                                             # cv2.CALIB_CB_FAST_CHECK +
                                             # cv2.CALIB_CB_NORMALIZE_IMAGE)

    if ret == True:
        objpoints.append(objp)

        #get the best corners inside small neighborhood of the original location
        #critera = number of iterations
        corners2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,1), criteria)

        imgpoints.append(corners2)

        img = cv2.drawChessboardCorners(img, checkerboard, corners2, ret)



# type(img)
plt.imshow(img)
plt.show()

#     cv2.imshow('img', img)
#     cv2.waitKey(0)

# cv2.destroyAllWindows()
```

```
11
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
(2016, 1512, 3)
```



[7]:
```
h,w = img.shape[:2]
print(len(objpoints))
print(len(imgpoints))
print(h,w)
print(gray.shape[::-1])

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

print("Camera matrix :")
```

```
print(mtx)

# print("rvecs :")
# print(rvecs)
# # print(np.array([rvecs]).reshape(3, 3, 1))
# print("tvecs :")
# print(tvecs)
# print(np.array([tvecs]).reshape(3, 3, 1))
```

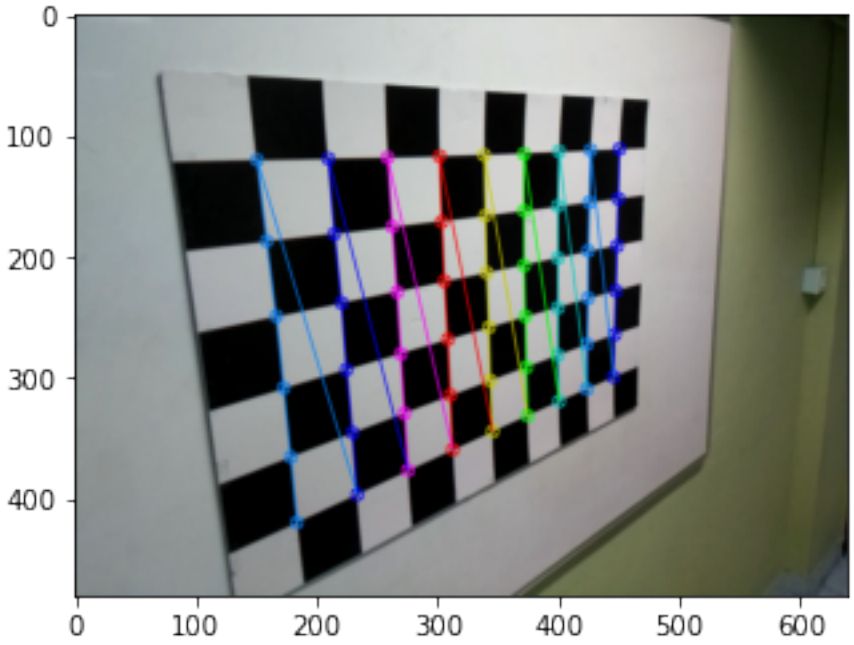
```
11
11
2016 1512
(1512, 2016)
Camera matrix :
[[1.58608401e+03 0.00000000e+00 7.72116125e+02]
 [0.00000000e+00 1.59247560e+03 1.01937428e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

0.0.1 Test run on test dataset below

[8]:
```
checkerboard = (6,9)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objpoints = []

imgpoints = []

objp = np.zeros((1, checkerboard[0] * checkerboard[1],3),np.float32)

objp[0,:,:2] = np.mgrid[0:checkerboard[0], 0:checkerboard[1]].T.reshape(-1, 2)

# print(objp.shape)

prev_img_shape = None

imagestest = glob.glob('./imagetest/*.jpg')

print(len(imagestest))
for fname in imagestest:
    #read images
    img = cv2.imread(fname)
    # print(img.shape)
    #get grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


    #function to find corners
    ret, corners = cv2.findChessboardCorners(gray, checkerboard)
```

```
                                                # flags
                                                # cv2.CALIB_CB_ADAPTIVE_THRESH +
                                                # cv2.CALIB_CB_FAST_CHECK +
                                                # cv2.CALIB_CB_NORMALIZE_IMAGE)

    if ret == True:
        objpoints.append(objp)

        #get the best corners inside small neighborhood of the original location
        #critera = number of iterations
        corners2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,1), criteria)

        imgpoints.append(corners2)

        img = cv2.drawChessboardCorners(img, checkerboard, corners2, ret)



# type(img)
plt.imshow(img)
plt.show()

h,w = img.shape[:2]
print(len(objpoints))
print(len(imgpoints))
print(h,w)
print(gray.shape[::-1])

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

print("Camera matrix :")
print(mtx)
```

12




```
11
11
480 640
(640, 480)
Camera matrix :
[[503.30682824   0.         313.22345756]
 [  0.         502.96055205 244.25404517]
 [  0.           0.           1.        ]]
```

[ ]: