CSE-578 Computer Vision

Assignment 1
Niharika Vadlamudi
2018122008

Febraury 10 2020

1 Direct Linear Transform

The Direct Linear Transform (DLT) algorithm is used to estimate the camera matrix (11 dof) from a set of at least 6 2D-3D point correspondences. The algorithm is based on solving the system of linear equations x = PX, which is then simply rearranged to form a system **Mp=0**, where p is a 12x1 vector and M is a 2Nx12 matrix and N is the number of correspondences. The solution p is obtained using SVD which is then rearranged to form back the matrix P. P is then decomposed into K,R,C using RQ decomposition.

1.1 DLT : Python Code

```
def DLT(Lc,Lw,verbose=False):
    M=[]
    #Check if its less than 6 points.
    if(len(Lc)<6 or (len(Lw)<6)):
        print("DLT cannot not be computed ! Need more than 6 points")
        return;
    I=np.asarray(Lc).shape[0]
    L=list(zip(Lc,Lw))
    #Creating point correspondance vector each row-col.
    for i in range(0,len(L)):
        M.append(computeRow(L[i][0],L[i][1]))
    M=np.reshape(np.asarray(M),(2*I,12))
    # Computing the SVD part .
    U, D, V_T =np.linalg.svd(M)
    #Extrating the last column of V
    P=np.reshape(V_T[-1,:],(3,4))

    # Calculating the Mean error now , as P is known to us .
    # Take up all world co-ordinates ,mul by P , we know the corresponding image co-ordinates .
    # MSE between the vectors.
    mse=0;
    for i in range(0,1,len(Lw)) :
        xcPred=np.dot(P,np.reshape(np.asarray(Lw[i]),(4,1)));
        xcPred=xcPred[0:2]/xcPred[-1]
        mse=mse+np.linalg.norm((np.reshape(np.asarray(Lc[i]),(2,1))-xcPred))

    # Proceeding to find K,R sepeartingly.
    H_inf=P[0:3,0:3]
    H_inf_inv=np.linalg.pinv(H_inf)
    t_original=-1*np.dot(H_inf_inv,P[:,-1])
    #Q R Decomposition.
    q, r = np.linalg.qr(H_inf_inv)
    K=np.linalg.pinv(r)
    # The last corner elemt must be 1 .
    K=K*(1/K[2,2])
    R=np.dot(np.linalg.pinv(K),H_inf_inv)

    if(verbose):
        print("Reconstructed Camera Matrix (P):")
        print(P)
        print("Camera Calibration Matrix : ")
```

```
        print(K)
        print("Rotation Matrix :")
        print(R)
        print("Translation Vector (t) :")
        print(t_original)
        print("Mean Square Error is",mse)

    return(P,K,R,t_original,mse)
```

1.2 Results : Checkerboard (Calib-Image)

Using 6 points on Chessboard Image , we have got Reprojection Error of 479.71 , later while using RANSAC method ,the error drastically reduces .
Note : The world co-ordinates here are given as (0,0,0),(1,0,0),etc and are not scaled to the chessboard block size of 28 mm x 28 mm .

```
Reconstructed Camera Matrix (P):
[[-1.72212123e-02 -2.35893816e-03 -7.36048323e-02  6.48086577e-01]
 [-5.90613670e-02  4.38088527e-02 -7.16087769e-02  7.50818585e-01]
 [-2.85608391e-05  1.66120829e-06 -3.54102053e-05  3.94971959e-04]]
Camera Calibration Matrix :
[[ 7.17729058e+02 -1.24441656e+02  1.49312386e+03]
 [-3.49175798e-13  8.87577487e+02  2.07265701e+03]
 [-9.17325104e-16  4.41345146e-15  1.00000000e+00]]
Rotation Matrix :
[[ 4.77168965e+01  3.97867043e+00 -3.70479942e+04]
 [ 4.48056071e+01  3.75740289e+00 -3.47862407e+04]
 [-1.91868118e+01 -1.59705808e+00  1.48715726e+04]]
Translation Vector (t) :
[4.28515203 1.3227859  7.7599619 ]
Mean Square Error is : 479.71819708601436
```

2 DLT with RANSAC

Random sample consensus (RANSAC) is a robust estimation algorithm. The idea is to de- termine a set of inliers from the set of correspondences, which are then used to estimate the calibration parameters. This is done by repeatedly sampling minimal sets (6 correspondences) and estimating the model, and then choosing the model which is supported by the least reprojection error.

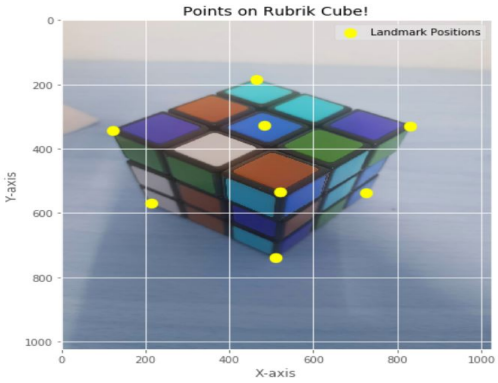
2.1 Code Snippet

```
def dltwithRANSAC(Lcn,Lwn,verbose=False):
    # Randomly pick n points out of 6 .
    n=len(Lcn)
    msemax=10000;
    # Takes subset of 6 items in given n list.
    for subset in itertools.combinations(range(0,n),6):
        Lci=[Lcn[i] for i in subset]
        Lwi=[Lwn[i] for i in subset]
        # Calling DLT function .
        P,K,R,T,E=DLT(Lci,Lwi,False)
        if(E<msemax):
            Pf,Kf,Rf,Tf,mse=DLT(Lci,Lwi,False)
            msemax=mse
    return(Pf,Kf,Rf,Tf,msemax)
```

3 Own Setup

For testing the code on my setup, I used my phone camera after fixing its focal length. I used a Rubik's cube as my calibration object since it is of known dimensions and the points are easy to measure.

(a) Original Image



(b) Marked Landmarks

Figure 1: Own Setup

3.1 DLT Results

For this particular, the DLT results turned out to be very accurate , with reprojection error of 2.5 only !

```
Reconstructed Camera Matrix (P):
[[ 1.52787842e-01 -2.67107205e-01 -9.35412309e-02 -6.44748464e-01]
 [ 3.90406411e-02  2.61165058e-02 -2.34284679e-01 -6.50970010e-01]
 [-1.13858879e-04 -1.78297823e-04 -2.06350419e-04 -1.21530004e-03]]
Camera Calibration Matrix :
[[ 9.28569248e+02 -2.30288983e+01  5.67137607e+02]
 [-5.86318075e-13  6.72196857e+02  4.49342733e+02]
 [-1.80577360e-14  1.60422334e-14  1.00000000e+00]]
Rotation Matrix :
[[-1.52198135e-01  2.24782751e+00  5.57055170e+02]
 [-1.68657441e-01  2.39623833e+00  5.95314786e+02]
 [ 2.47506121e-01 -3.57883987e+00 -8.95005096e+02]]
Translation Vector (t) :
[-1.46439351 -2.11056761 -3.25783796]
Mean Square Error is 2.452636678720338
```

4 Distortion Parameters

The radial and tangential distortion parameters were estimated using Matlab's estimateCamera- Parameters function. The checkerboard images were used for this purpose. Using these pa- rameters, the image of the 3D object was undistorted using OpenCV's undistort function. I assumed the 3D object, and the checkerboard were both captured using the same camera. The equations are as follows :

$$x_{\text{distortion}} = x + (2p_1xy + p_2(r^2 + 2x^2))$$
$$y_{\text{distortion}} = x + (2p_2xy + p_1(r^2 + 2y^2))$$

$$x_{\text{distortion}} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$
$$y_{\text{distortion}} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

The three radial distortion values correspond to k1; k2; k3, and tangential to p1; p2. The distortion parameters are usually estimated by adding the straight line in the image, and then doing a search over the values of p1,p2,k1,k2,k3 that minimizes the distance between the midpoint of these lines and the average of their endpoints. So,I tried to undistort the image (Fig1.jpg) and obtained the following result .

[H]

4.1 Results

Figure 2: Distortion Corrected Image

```
(After Distortion Process)
Reconstructed Camera Matrix (P):
[[ 4.25919083e-13  3.12629714e-01 -8.27338992e-13  6.99425671e-12]
 [-1.51443422e-13  9.49874946e-01 -2.28704146e-13  7.23689630e-12]
 [-1.82212955e-16  4.99408489e-04 -2.07870593e-16  4.55820531e-15]]
Camera Calibration Matrix :
[[ 1.70828202e+02  6.00122309e+13  6.26000001e+02]
 [-1.91226648e-06  1.82337481e+14  1.90200000e+03]
 [ 3.68694448e-03  9.58661833e+10  1.00000000e+00]]
Rotation Matrix :
[[ 3.51749577e+09 -1.15770338e+09 -5.24535840e+05]
 [-2.70123099e-06  8.89048480e-07  4.02812844e-10]
 [-2.81770996e-17  9.27384870e-18  4.20182415e-21]]
Translation Vector (t) :
[-2.77087840e+00 -7.00595812e-12  4.38008614e+00]
Mean Square Error is 1.520856267979129e-11
```

5 Zhang's Method for Camera Callibration

The only difference in the approach for Zhang's method and the DLT method , it here its done for planar objects and hence the Z row will eventually become zero.This method is applicable for planar point detection only .

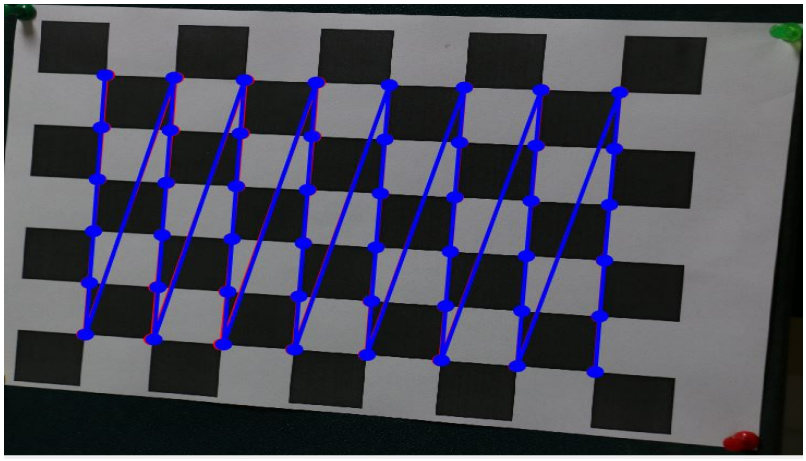
5.1 Code Snippet

```

clc
close all
for i =  1:15
    %imageFileName = sprintf('img%d.jpg',i);
    %imageFileNames{i} = fullfile('../data/phone-camera/zhang',imageFileName);

    imageFileName = sprintf('img%d.JPG',i);
    imageFileNames{i} = fullfile('../data/zhang',imageFileName);
end
[imagePoints,boardSize] = detectCheckerboardPoints(imageFileNames);
worldPoints = generateCheckerboardPoints(boardSize, 0.029);
[params,imagesUsed,estimataionErrors] = estimateCameraParameters(imagePoints,worldPoints, ...
'EstimateSkew',true,'NumRadialDistortionCoefficients',3, 'EstimateTangentialDistortion',true);
figure, imshow(imageFileNames{1}), hold on
plot(imagePoints(:,1,1),imagePoints(:,2,1),'ro-','LineWidth',2,'MarkerFaceColor','r');
plot(params.ReprojectedPoints(:,1,1),params.ReprojectedPoints(:,2,1),'bo-','LineWidth',2,'
    MarkerFaceColor','b');
disp('Average reprojection error:');
disp(params.MeanReprojectionError);
disp('Calibration matrix:');
disp(params.IntrinsicMatrix');
disp('Radial distortion:');
disp(params.RadialDistortion);
disp('Tangential distortion:');
disp(params.TangentialDistortion);
```

5.2 Overlaying a wireframe over the Chessboard

After performing Zhang's calibration method we take up the calibration matrix - P and solve reproject it back

```
%% World Points
% worldPoints={[0,0,1],[0,0,0],[0,0,2],[0,0,1],[1,0,0],[2,0,0],[3,0,0],[0,1,0],[0,2,0],[0,0,3]};
% imgPoints={};

% for i=1:15
%     imgPointsPred{i}=dot(params.IntrinsicMatrix,worldPoints{i});
% end
figure, imshow(imageFileNames{1}), hold on
plot(imagePoints(:,1,1),imagePoints(:,2,1),'ro-','LineWidth',2,'MarkerFaceColor','r');
plot(imgPointsPred(:,1,1),imgPointsPred(:,2,1),'bo-','LineWidth',2,'MarkerFaceColor','b');
```



5.3 Results : Checkerboard

Zhang's method was tested on the provided checkerboard images using Matlab's implementation, available in the Computer Vision toolbox. Code is provided in Listing 6. The following are the estimated values:

```
Average reprojection error:
    3.8164
Calibration matrix:
   1.0e+04 *
    1.3472    0.0078    0.3096
         0    1.3510    0.1869
         0         0    0.0001

Radial distortion:
    0.2712    1.0233  -60.9349

Tangential distortion:
    0.0026    0.0164
```

The results are obviously much better than the ones obtained using my implementation of DLT. One can observe that the cx cy values almost exactly overlap with the center of the image. The reprojection error is also less than 4 pixels.

5.4 Own Setup : Chessboard

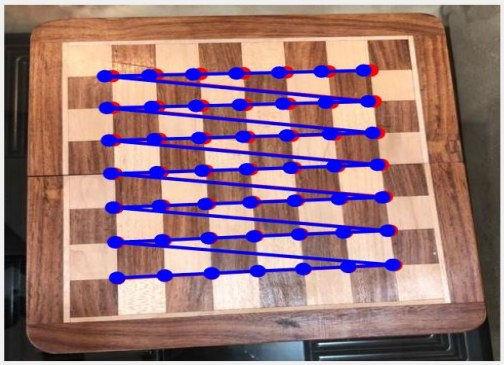
I implemented Zhang's Method of calibration using my smatphone , as the error is less than 3 pixels !

```
Average reprojection error:
    2.7545
Calibration matrix:
  553.5479   13.5128  367.5084
         0  569.8669  362.8300
         0         0    1.0000

```

```
Radial distortion:
    0.0670   -0.0188   -0.0277

Tangential distortion:
    0.0100    0.0092
```

The following are qualitative results , where the red dots are the original wireframe of points and blue lines are the predicted ones , the deviation of the lines is minimal .

(a) Original Image



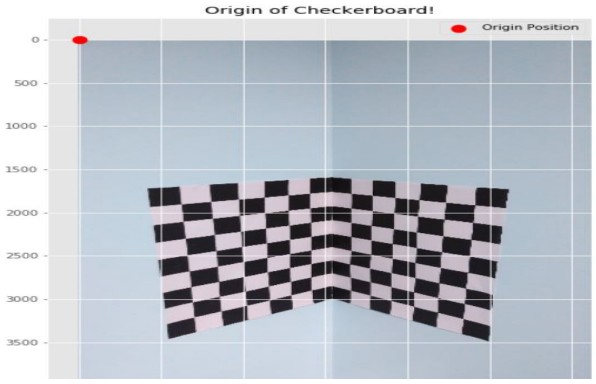
(b) Marked Landmarks

Figure 3: Wireframe overlayed over the checkerboard after calibration. Blue indicates the estimated points, red indicates the measured points.p

5.5 Understanding the centre of the camera

The world origin when multiplied by the camera matrix gets projected onto the corresponding location in the image as expected,it turns out at the left corner of the image .

```
imgOrigin=np.dot(P,np.asarray([0,0,0,1]));

imgarr=cv2.imread("C:\\Users\dell\\Desktop\\shmy_CV_1\\main.jpg")

fig=plt.figure(figsize=(10,10))
plt.title('Origin of Checkerboard!')
plt.style.use('ggplot')
plt.imshow(imgarr)
plt.scatter(imgOrigin[0],imgOrigin[1],marker='o',s=120,color='red',label='Origin Position')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
```



6 Conclusion

I got deeper perspective of camera parameters , also complete implementation of DLT made me understand the mathematical flair behind the process.A lot of new OpenCV libraries and functions were learnt in teh process.