cs512 Assignment 4: Report

Amit Nikam
Department of Computer Science
Illinois Institute of Technology

November 28, 2020.

Abstract

In this programming assignment 4, I have implemented a program that calibrates the camera by saving multiple pictures of the co-planar calibration target and then processing them. The saved images are feature extracted to capture points of interest in the image. These image points map to real world points and all together these mappings are saved in a file. This data from the file is used by the calibration program to find the intrinsic and extrinsic parameters.

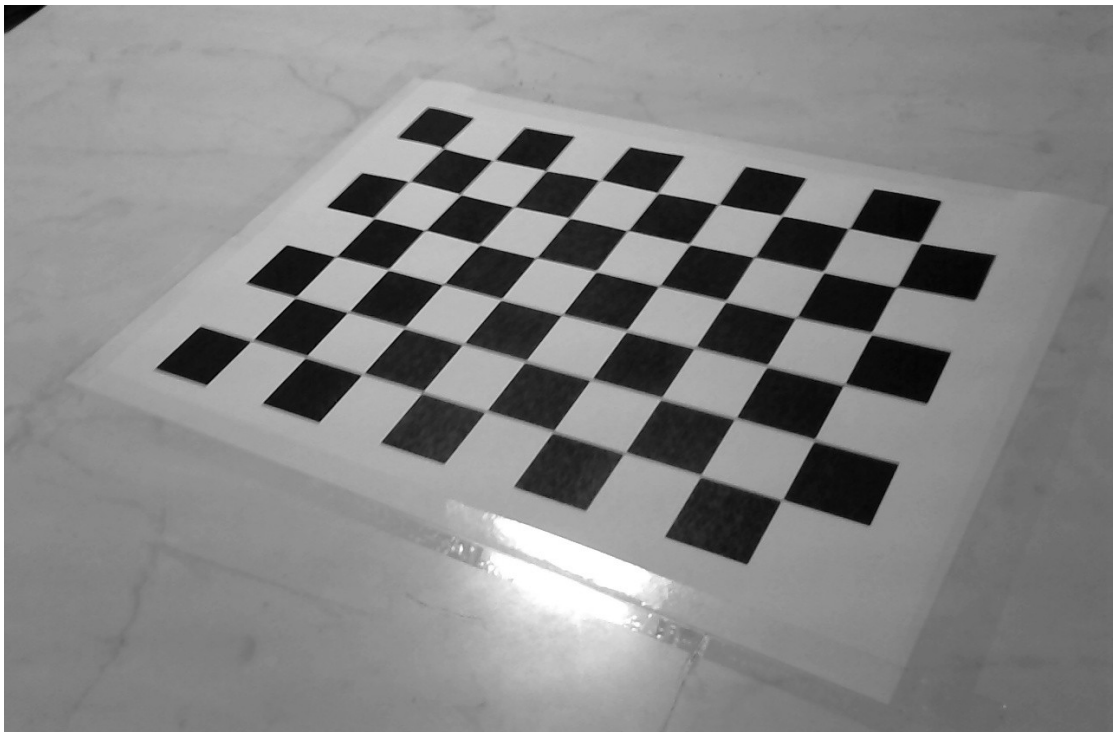
1 Problem Statement - Calibration

The goal of this application was to perform camera calibration process either using a non-coplanar calibration or planar calibration method. The calibration algorithm for either of the cases was to be implemented from scratch. The input for the algorithm would be a point correspondence file that would have the 3D point to 2D point mapping. After the calibration was over, the results were to be tested by projecting the points onto the image and then checking the difference between projected and actual point using a mean square error check. Finally a robust estimation was to be implemented to find the best fits by removing outliers.

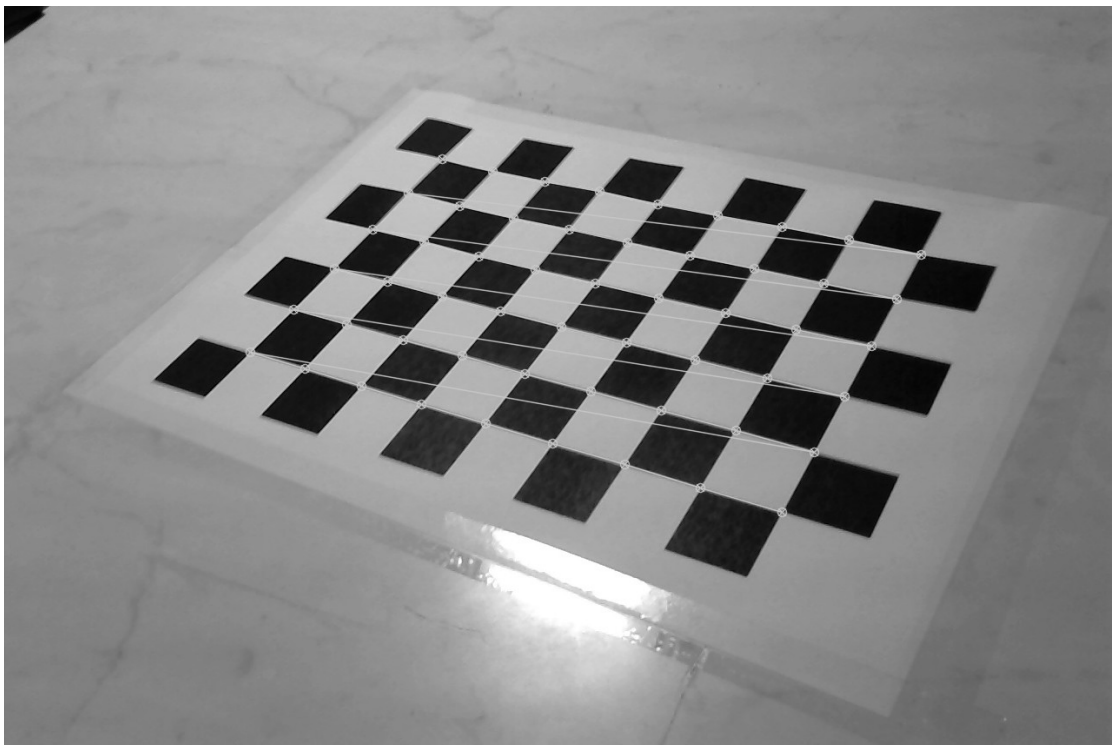
2 Solution

To understand the solution better, let's breakdown the problem first. As my solution I have implemented a co-planar camera calibration. So I had to make sure I have multiple views available.

So to capture multiple views, I implemented a simple program that saves the current video frame on key press 's' and closes on 'q'. These images are saved in the location specified during initialization.



Next the challenge was to extract features from the captured images and map these feature points to the real world points. Since for my calibration process I implemented a chess board pattern, it was easy to get these locations using the openCV function and map them to real world points. This data is saved in a readable ‘extracts.yaml’ file from which it can be loaded later for calibration.



After the points were extracted successfully from the images, the next step was to calibrate using that feature mapped data. The calibration program takes the

name of the '.yaml' file to load extracted data from during initialization. If no name is specified, the program takes the default 'extracts.yaml' file from previous step and calibrates it. This program is divided into two parts, (1) Calibration (2) Mean Square Error.

In calibration, I have implemented an algorithm that takes two inputs – List of image points and List of object points. Each of these lists has a sub-list for each view. This algorithm returns intrinsic parameter matrix K*, list of extrinsic R* rotation matrix (one for each view) and a list of extrinsic T* translation matrix as output. This calibration algorithms creates a point matrix A for each view of size (2*number of views X 9), such that it represents AX = 0 form and also is a 2D homography matrix(i.e. Z=0). This matrix when decomposed using right null space (SVD), gives us the unknown parameters value.

These values from each view still don't give any information about the intrinsic parameters. To find those, we need to find S which happens to be a product of ($K^{*-T}.k^{*-1}$). For this we form a matrix V which is a matrix of multiple different combination of already found values. For each view, we get a V12 and V11-V22 row in this matrix. Thus this matrix is of size 2m*6.

Next we solve this using singular value decomposition, since it is in the AX = 0 form. We get 6 specific values from this step. These six values help us to find the intrinsic parameters. The following image shows the calculation performed to find the intrinsic parameters from the found S matrix values.

```
# Find Intrinsic Parameters
c1 = (s[1]*s[3]) - (s[0]*s[4])
c2 = (s[0]*s[2]) - (s[1]**2)
v0 = c1/c2
lamb = s[5] - ((s[3]**2) + (v0*c1))/s[0]
alp_u = math.sqrt(lamb/s[0])
alp_v = math.sqrt((lamb*s[0])/c2)
skew = -(s[1]*(alp_u**2)*alp_v)/lamb
u0 = ((skew*v0)/alp_u) - ((s[3]*alp_u**2)/lamb)
```

Once we have found the intrinsic parameters, we can easily find the extrinsic parameters using these values as follow.

```
# Homography for view
h_hat = H[view,]
h1 = h_hat[:,0]
h2 = h_hat[:,1]
h3 = h_hat[:,2]

# Find Alpha
mod_alpha = 1/np.linalg.norm(np.dot(K_inv,h1))
sign = np.dot(K_inv,h3)
sign = np.sign(sign[2])
alpha = sign*mod_alpha

# Find R*
r1 = alpha*(np.dot(K_inv,h1))
r2 = alpha*(np.dot(K_inv,h2))
r3 = np.cross(r1,r2)
r1 = np.reshape(r1, (3,1))
r2 = np.reshape(r2, (3,1))
r3 = np.reshape(r3, (3,1))
R = np.concatenate((r1,r2,r3),axis=1)
R_sequence.append(R)

# Find T*
T = alpha*(np.dot(K_inv,h3))
T_sequence.append(T)
```

Thus we get intrinsic parameters K* and for each view extrinsic parameters R* and T*. This was all about finding the parameters, once the parameters are found we can find the mean square error.

To find the mean square error, we take the list of actual image points, list of object points, list of R*, list of T* and the found K* matrix. Since we have all the parameters required to project points, we do the same by performing following projection for each point in each view: K*[r1,r2,T*] $\underline{Pi}^T$ where the input point is world point with Z = 0. For each of the points, we find the predicted values of X and Y from this equation.

Next for error calculation, we take the sum of L2 of the error over each point for each view and then divide it by total points we predicted. This gives us the mean square error of the derived calibration parameters for our camera. The aim was to use this error to find the best fit after robust estimation using RANSAC. Although this did not happen, as I faced difficulty implementing RANSAC and modifying my input matrix.

Finally all the best found calibration values are saved in a ‘calibration.yaml’ file from where it can be imported for later use.

3 Implementation Details

The following files are included in src folder:
- 0_capture.py
- 1_extraction.py
- 2_calibration.py
- algorithm.py
- extracts.yaml
- calibration.yaml

The main algorithms are in the algorithm.py program file and are imported where-ever required.

The program with 0 serial is to capture images of the calibration target. Press ‘s’ to save image and ‘q’ to quit program. To specify specific location and name to save files with, pass -n </save-loc/name> as argument during initialization.

Serial 1 file extracts features from the images in a folder. To specifiy the folder location pass argument -f <folder location> to the program. The program takes all the pictures in the folder and processes them and saves their found feature points.

“extracts.yaml” happens to be the saved data points file.

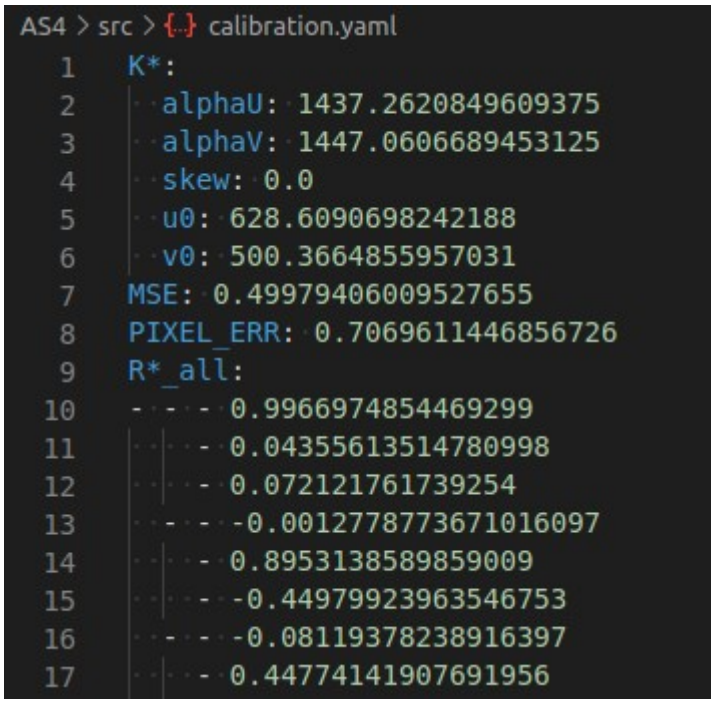
Serial 2 file takes file of extracted data points as input through a passed argument as follows, -f <file name>. By default it takes the extracts.yaml from serial 1’s output.

“calibration.yaml” happens to be the saved data for the calibration process.

4 Results

```
anikam@hp-notebook:~/Projects/computervision$ python3 2_
Reading feature points from file: extracts.yaml

Known Parameters:
----------------
(u0,v0): (628.6091, 500.3665)
(aplhaU,aplhaV): (1437.2621, 1447.0607)
s: 0.0

Image 0
T*: [-3.9397535 -2.5447266 18.406052 ]
R*:
[[ 0.9966975   0.04355614  0.07212176]
 [-0.00127788  0.89531386 -0.44979924]
 [-0.08119378  0.44774142  0.8924127 ]]
----------------

Image 1
T*: [-2.8172095 -4.3369894 25.150217 ]
R*:
[[ 0.9189268  -0.22109255  0.3307394 ]
 [ 0.10229436  0.9386975   0.32533646]
 [-0.38093227 -0.26238778  0.8852108 ]]
----------------

Image 2
T*: [-3.7740898 -2.8998067 19.923796 ]
R*:
[[ 0.9929624   0.02215367  0.11590774]
 [-0.00971276  0.9908146  -0.10886744]
 [-0.11803123  0.10700568  0.9840569 ]]
----------------

Image 3
T*: [ 1.4718543 -2.314539  25.340248 ]
R*:
[[ 8.7139374e-01 -2.6585853e-03 -4.9640679e-01]
 [ 3.1565144e-04  1.0118605e+00  9.1977743e-03]
 [ 4.9058440e-01 -1.2051997e-02  8.8172972e-01]]
----------------

Image 4
T*: [-3.4565194 -1.4103639 17.139368 ]
R*:
```

```
Image 12
T*: [-4.157828  -2.8358448 13.784113 ]
R*:
[[ 0.9888025   0.01104416  0.14865744]
 [-0.00434188  0.99826247 -0.05869313]
 [-0.1491671   0.05769171  0.9871324 ]]
----------------

Mean Square Error: 0.49979406009527655
Pixel Error: 0.7069611446856726

Calibration data saved at calibration.yaml
anikam@hp-notebook:~/Projects/computervision$
```



Conclusion: In the end the application to calibrate the camera from co-planar calibrator image was completed. This assignment was challenging and definitely put my knowleadge of the computer vision and python programming to use. Other than calibration which of course was challenging, basics of image processing were also put to use. I look forward to more such assignments.