Planar Camera Calibration: Report

Abstract

In this project, I have implemented camera calibration by saving multiple pictures of co-planar calibration target (chess board). Each of these saved images are used to extract 3D to 2D mapping of 49 chess board corners. All these mappings are saved in a yaml file. This yaml file is used to find the internal and external parameters of the camera. I have also used the 3D-2D ideal mapping data provided for this project and determined the internal and external parameters of the camera. For the last step, I have used noisy data 1 and 2 and prepared yaml file. These ymal files are fed to the RANSAC program one by one, where we estimate the best Homography matrix(H) for the first image and then proceed with normal calibration.

1. Problem Statement

The goal of this application is to perform camera calibration either using a non-planar calibration method or planar calibration method. We are required to gather calibration target, extract feature points and save it into a file, and then perform calibration using these mappings. In order to test the accuracy of our implementation we are supposed to use the 3D-2D mapping file provided on the web. Finally robust estimation is implemented to find best M/H for the noisy data and then find the best noisy records.

2. Proposed Solution

I have performed co-planar calibration and the entire process can be broken down into Following steps:

Step1: Image Capture

For this step I have written a python script to capture images of chess board using web-cam. This program saves the current video frame on key press ‘s’ and closes on ‘q’.
I executed this script and captured 8 images from different angle and these images are by default, saved under data directory unless a different directory is specified explicitly.

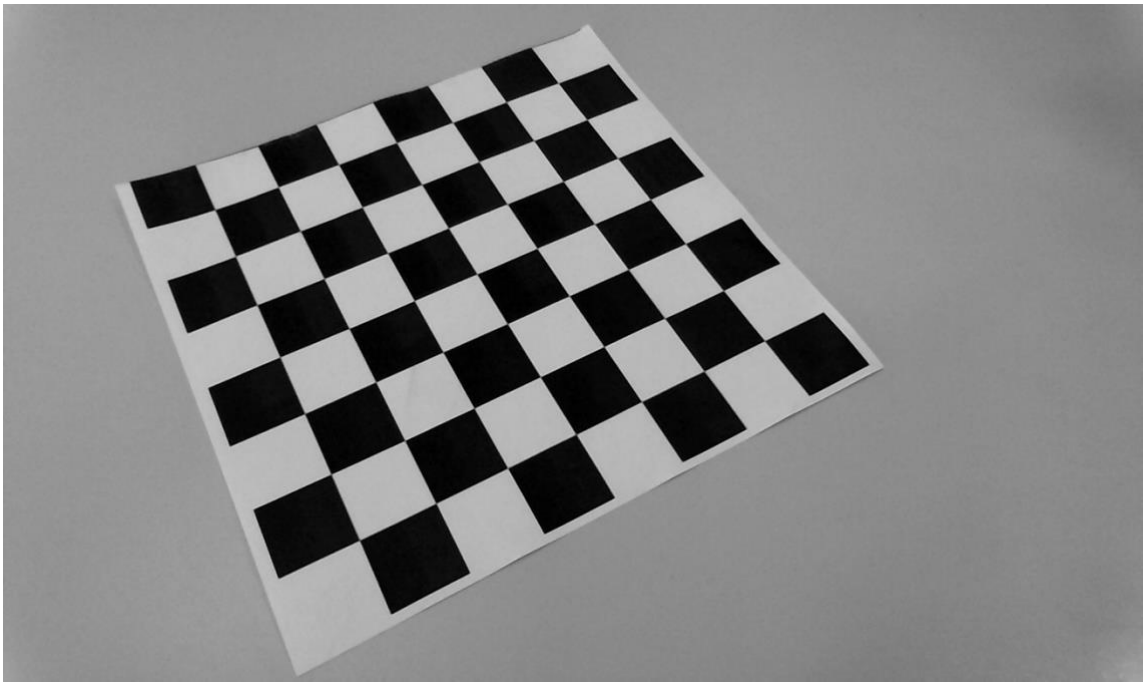
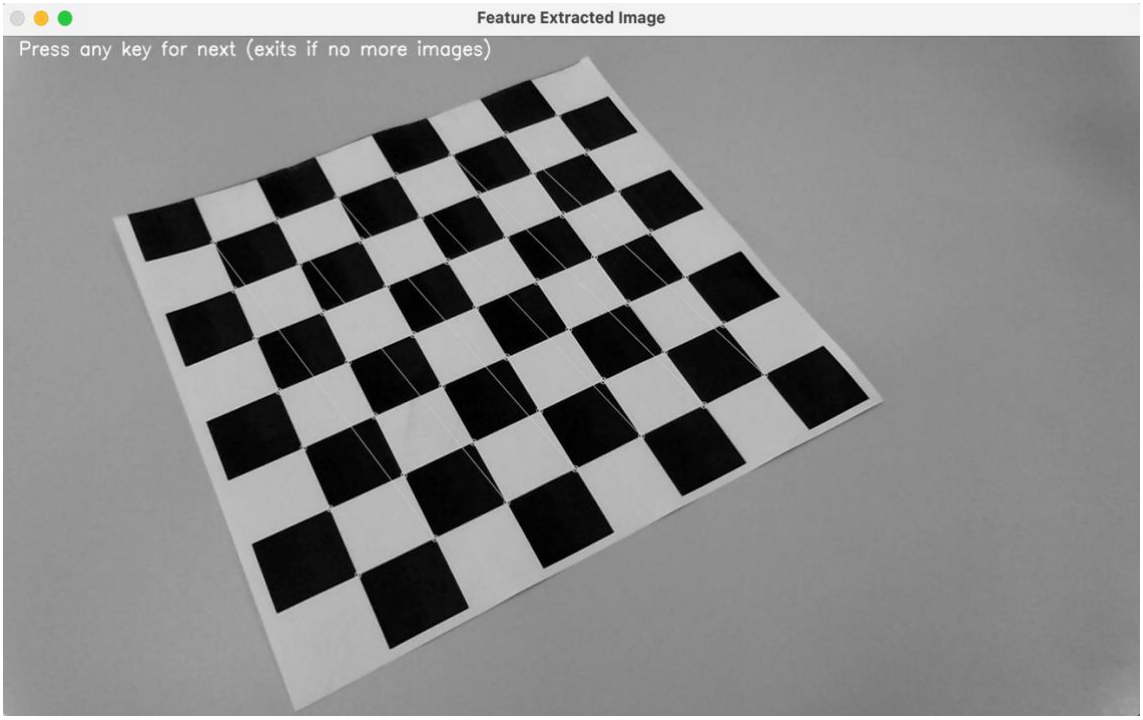


Figure 1: Sample Image

Step 2: Feature Extraction

For this step I wrote a python script to extract feature points from the captured images and map these feature points to the real-world points. Since for my calibration I have used chess board as the target, it is easy to get these locations using the OpenCV function and map them to real world points. This data is saved in “extracts.yaml” file from which it can be loaded for calibration.



Step 3: Calibration

The calibration program takes the name of ".yaml" file to load the mapping data. If no name is specified, the program takes the default "extracts.yaml" file from previous step and calibrate 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 have 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. The calibration algorithms create 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 SVD gives us the unknown parameter value.

These values from each view still don't any information about the intrinsic parameters. To find those, we need to find S which happens to be product of $K^{*-1}.K^{-1}$. For this we Form a matrix V which is a matrix from multiple different combination of already found Views. 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 SVD, 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.

Once we found the intrinsic parameters, we can easily find the extrinsic parameters.

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 K* matrix. Since we have all the parameters required to project points, we do the same by performing projection of each point in each view: K*[r1,r2,T*] PiT 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.

For robust estimation using RANSAC the very first step was to prepare data. In order to prepare data in ".yaml" format I took the noise1 data and treated it as view 1 and other two ideal views are treated as view2 and view3. The name of the file with noise 1 is "3D_to_2D_mapping_Noise1.yaml". Similarly I prepared data with noise 2 as first view and saved it in "3D_to_2D_mapping_Noise2.yaml". Now we use RANSAC to estimate the best Homography matrix(H) for the first view(Noise 1/ Noise2). Once we have the best H for the first view, we proceed with normal cameral calibration method as

described above. We performed RANSAC on data with Noise 1 and Noise 2 and found that for RANSAC is not able to handle data with Noise 1 because the value of v0 is coming as negative. This possible only when the mapping of at least one of the points is incorrect.

3. Implementation Details

The following files are included in src folder:
ImageCapture.py
FeatureExtraction.py
Calibration.py
algorithm_impl.py
RANSAC.py
RANSAC.config
extracts.yaml
calibration.yaml
3D_to_2D_mapping_ideal.yaml
3D_to_2D_mapping_Noise1.yaml
3D_to_2D_mapping_Noise2.yaml
ransac_output.yaml

The main algorithms are in algorithm_impl.py file and are imported where-ever required.

ImageCapture.py is used to capture images for the calibration target. Press 's' To save image and 'q' to quit program. To specify location and name to save files with, pass -n </save-loc/name> as argument during initialization.

FeatureExtraction.py extracts features from the images in a folder. To specify 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.

Calibration.py takes the file of extracted data points as input through passed argument as follows, -f <file name>. By default it takes the extracts.yaml from FeatureExtraction.py output.

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

3D_to_2D_mapping_ideal.yaml is the mapping file provided to us that can be Used as input to Calibration.py.

RANSAC.py takes two arguments first being yaml file with noise as input through passed an argument as follows -f1 <file name>. By default it takes the 3D_to_2D_mapping_Noise1.yaml. Second argument is the configuration file which is passed as argument as follows -f2 <file name>. By default it takes the RANSAC.config as input.

3D_to_2D_mapping_Noise1.yaml happens to be the mapping file for Noise 1
3D_to_2D_mapping_Noise2.yaml happens to be the mapping file for Noise 2

ransac_output.yaml contains the calibration results with noisy data

1. Run the python script to capture images from command line using following command:

```
python ImageCapture.py
```

2. Run the python script to extract features from images captured in previous step and prepare mapping file.

   ```
   python FeatureExtraction.py
   ```

3. Run the python script to calibrate camera.

   ```
   python Calibration.py
   ```

   or

   ```
   python calibration.py -f 3D_to_2D_mapping_ideal.yaml
   ```

4. Run the python script to perform RANSAC and calibrate camera.

   ```
   python RANSAC.py
   ```

   or

   ```
   python RANSAC.py -f1 3D_to_2D_mapping_Noise2.yaml -f2 RANSAC.config
   ```

4. Results and Discussion

Calibration result with features extracted from custom images.

```
(py38) sauravkantkumar@Sauravs-MacBook-Pro src % python Calibration.py
Reading feature points from file: extracts.yaml

Known Parameters:
----------------
(u0,v0): (980.6533, 413.88513)
(alphaU,alphaV): (1785.0986, 1759.5015)
s: 0.0

Image 0
T*: [-0.45188668 -2.9653337  18.158533  ]
R*:
[[ 0.49048305 -0.8693347  -0.00920205]
 [ 0.6690019   0.364863    0.64160913]
 [-0.55844676 -0.3183229   0.7605457 ]]
----------------

Image 1
T*: [-1.3540541 -3.144129  18.551441 ]
R*:
[[ 0.7032728  -0.70748603  0.008625  ]
 [ 0.5444798   0.5407648   0.6315374 ]
 [-0.45710945 -0.43814978  0.76551706]]
----------------

Image 2
T*: [-3.6436913 -1.6787647 15.810306 ]
R*:
[[ 0.99931324  0.01674102 -0.03703171]
 [ 0.02159758  0.79951066  0.60006   ]
 [ 0.03011074 -0.59996796  0.7986    ]]
```

```
----------------

Image 7
T*: [ 3.5564291 -0.5660818 16.042099 ]
R*:
[[-0.4286981  -0.8948227  -0.11982538]
 [ 0.5611238  -0.37322986  0.7439508 ]
 [-0.7080665   0.257423    0.66210926]]
----------------

Mean Square Error: 1.3879467636620022
Pixel Error: 1.178111524288767

Calibration data saved at calibration.yaml
```

calibration.yaml

```yaml
K*:
  alphaU: 1785.0986328125
  alphaV: 1759.50146484375
  skew: 0.0
  u0: 980.6533203125
  v0: 413.8851318359375
MSE: 1.3879467636620022
PIXEL_ERR: 1.178111524288767
R*_all:
- - - 0.4904830455780029
- - - 0.703272819519043
- - - 0.999313235282898
- - - 0.910261750221252
- - - 0.978969693183898
- - - 0.998111605644226
- - - 0.978836119174957
- - - -0.428698092699050
T*_all:
- - -0.4518866837024688
- - -1.3540540933609009
- - -3.643691301345825
- - -2.898462295532226
- - -3.176405429840088
- - -3.457426309585571
- - -3.395884275436401
- - 3.556429147720337
```

Calibration result with data provided on the web:

```
(py38) sauravkantkumar@Sauravs-MacBook-Pro src % python Calibration.py -f 3D_to_2D_mapping_ideal.yaml
Reading feature points from file: 3D_to_2D_mapping_ideal.yaml

Known Parameters:
-----------------
(u0,v0): (377.75662, 71.486496)
(alphaU,alphaV): (1442.3278, 1432.8971)
s: 0.0

Image 0
T*: [-129.41226  -92.49279  956.92786]
R*:
[[-4.2857432e-01  9.2437536e-01 -1.0430813e-07]
 [ 6.0446179e-01  3.0223086e-01 -7.6464403e-01]
 [-6.7152816e-01 -3.3576423e-01 -6.8827796e-01]]
-----------------

Image 1
T*: [-78.71984 -64.92901 958.16016]
R*:
[[-5.7736415e-01  8.2795310e-01 -1.3411045e-07]
 [ 4.2551127e-01  3.1913343e-01 -8.7870318e-01]
 [-6.9684345e-01 -5.2263284e-01 -5.3655958e-01]]
-----------------

Image 2
T*: [-184.86021   -88.286896  979.2847  ]
R*:
[[-0.22368711  0.9837497  -0.02976259]
 [ 0.73784375  0.18050323 -0.6703513 ]
 [-0.6368286  -0.19612853 -0.7662298 ]]
-----------------

Mean Square Error: 3.9110015295171143
Pixel Error: 1.977625224737264

Calibration data saved at calibration.yaml
```

calibration.yaml

```
K*:
  alphaU: 1442.3277587890625
  alphaV: 1432.8970947265625
  skew: 0.0
  u0: 377.7566223144531
  v0: 71.48649597167969
MSE: 3.9110015295171143
PIXEL_ERR: 1.977625224737264
R*_all:
- - - -0.4285743236541748...
- - - -0.5773641467094421...
- - - -0.2236871123313903...
T*_all:
- - -129.41226196289062...
- - -78.71984100341797...
- - -184.86021423339844...
```

Calibration with RANSAC result with Noise 1 data provided on the web:

```
(py38) sauravkantkumar@Sauravs-MacBook-Pro src % python RANSAC.py
Known Parameters:
----------------
(u0,v0): (373.6133, -781.9635)
(alphaU,alphaV): (1735.5571, 1010.9763)
s: 0.0

Image 0
T*: [-126.719444  813.96375  1146.8911  ]
R*:
[[-0.42720336  0.9220366   0.08245444]
 [ 0.2706552   0.21133664 -0.95306337]
 [-0.8626952  -0.36897373 -0.33983773]]
----------------

Image 1
T*: [ -74.5248  846.1923 1131.0629]
R*:
[[-5.6836402e-01  8.1075799e-01 -2.2538006e-07]
 [ 1.7506192e-02  1.3129376e-02 -1.0175706e+00]
 [-8.2259089e-01 -6.1694342e-01 -2.1655548e-02]]
----------------

Image 2
T*: [-180.9656  839.1789 1171.376 ]
R*:
[[-0.22417682  0.97734565 -0.06035585]
 [ 0.6078557   0.10797191 -0.7970805 ]
 [-0.76174545 -0.23460004 -0.6182899 ]]
----------------

Mean Square Error: 6.339200518708508
Pixel Error: 2.5177769001062242

Calibration data saved at ransac_output.yaml
```

ransac_output.yaml

```yaml
K*:
  alphaU: 1735.55712890625
  alphaV: 1010.976318359375
  skew: 0.0
  u0: 373.6133117675781
  v0: -781.9635009765625
MSE: 6.339200518708508
PIXEL_ERR: 2.5177769001062242
R*_all:
- - - -0.42720335721969604…
- - - -0.5683640241622925…
- - - -0.224176824092865…
T*_all:
- - -126.71944427490234…
- - -74.5248031616211…
- - -180.96560668945312…
```

Calibration with RANSAC result with Noise 1 data provided on the web:

```
(py38) sauravkantkumar@Sauravs-MacBook-Pro src % python RANSAC.py -f1 3D_to_2D_mapping_Noise2.yaml -f2 RANSAC.config
Known Parameters:
----------------
(u0,v0): (377.76154, 71.57108)
(alphaU,alphaV): (1442.2925, 1432.9093)
s: 0.0

Image 0
T*: [-129.41557   -92.546394  956.90497 ]
R*:
[[-4.2857251e-01  9.2437732e-01 -1.2516975e-06]
 [ 6.0448235e-01  3.0224085e-01 -7.6462549e-01]
 [-6.7151070e-01 -3.3575705e-01 -6.8830186e-01]]
----------------

Image 1
T*: [-78.72323 -64.98352 958.1383 ]
R*:
[[-5.7736266e-01  8.2795626e-01 -1.0430813e-07]
 [ 4.2553908e-01  3.1915429e-01 -8.7868452e-01]
 [-6.9682753e-01 -5.2262086e-01 -5.3659552e-01]]
----------------

Image 2
T*: [-184.86365  -88.34184  979.26135]
R*:
[[-0.22368512  0.98375106 -0.02976091]
 [ 0.73785746  0.18050896 -0.6703358 ]
 [-0.6368134  -0.19612384 -0.7662452 ]]
----------------

Mean Square Error: 31.058008522285512
Pixel Error: 5.572971247215036

Calibration data saved at ransac_output.yaml
```

ransac_output.yaml

```
K*:
  alphaU: 1442.29248046875
  alphaV: 1432.9093017578125
  skew: 0.0
  u0: 377.76153564453125
  v0: 71.57108306884766
MSE: 31.058008522285512
PIXEL_ERR: 5.572971247215036
R*_all:
- - - -0.42857250571250916
- - - -0.5773626565933228
- - - -0.2236851155757904
T*_all:
- - -129.4155731201172
- - -78.72322845458984
- - -184.8636474609375
```

Conclusion: RANSAC algorithm is not able to handle the Noise 1 data as the value of v0 is negative. It happens because any/some of the points have wrong correspondence.