Camera Calibration Report

Xinyu Gu

May 2019

1 Introduction

Cameras introduces a lot of distortions to images. Due to radial distortion, straight lines may appear curved. In addition, tangential distortion may cause object to look closer than the real distance. In order to find to parameters defining the target camera model, camera calibration will be performed on checkerboard pattern images taken using target camera.

2 Procedure

In order to assess the correct model of the target camera, calibrations were direct at both pinhole lens and fisheye lens (will be referenced as pinhole calibration and fisheye calibration in later context). pinhole calibration and fisheye calibration were performed on pictures of size 1280 X 1024, result examples will be attached below. Because the resolution of the sinus surgery videos are 320 x 240, pictures were also resized to 320 X 240 attempting to get the camera parameters under this setting.
A total of 30 frames were extracted from the video of checkerboard pattern taken using the endoscope, which were used in both calibration models. (Source code can be found under Appendix)

3 Result Analysis

3.0.1 Pinhole 1280 X 1024

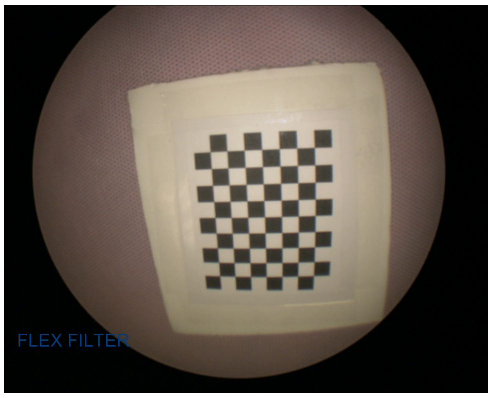


Figure 1: Source.

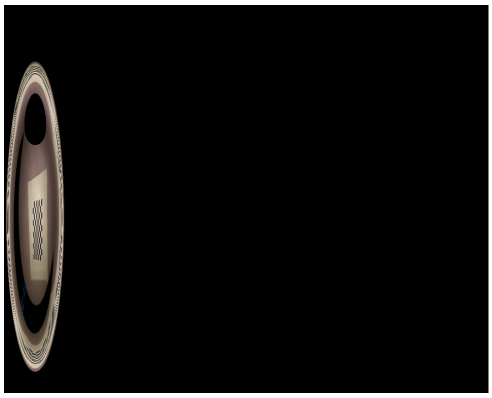


Figure 2: Undistorted.

From the sample result above, it is easy to see that pinhole calibration is not the appropriate model for the target camera. A checkerboard pattern cannot even be identified in most result pictures.

3.0.2 Pinhole 320 X 240

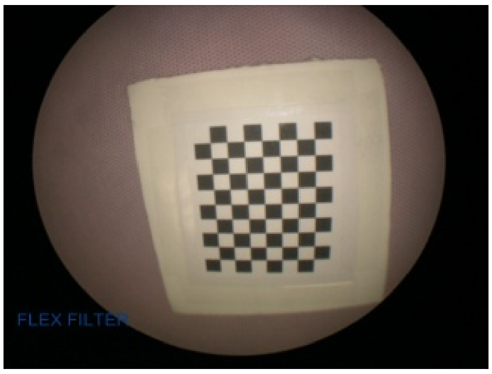


Figure 3: Source.

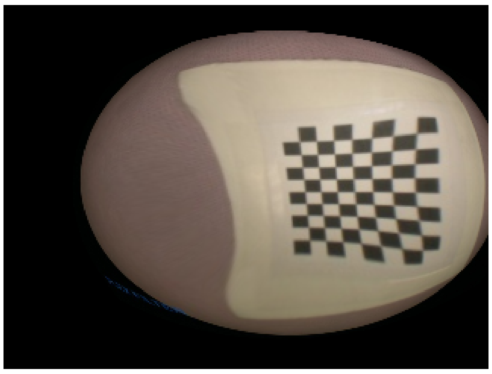


Figure 4: Undistorted.

A checkerboard pattern can be identified in most of the result pictures. However, the result pictures possessed even more significant distortion.

3.0.3 Fisheye 1280 X 1024

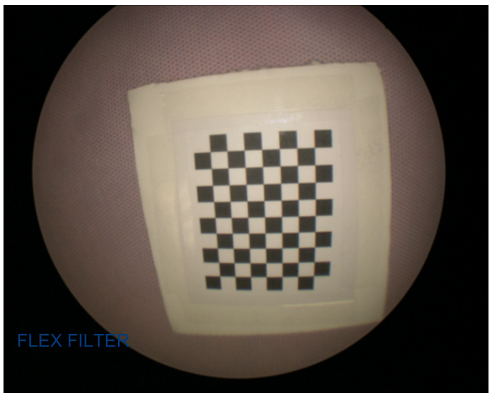


Figure 5: Source.

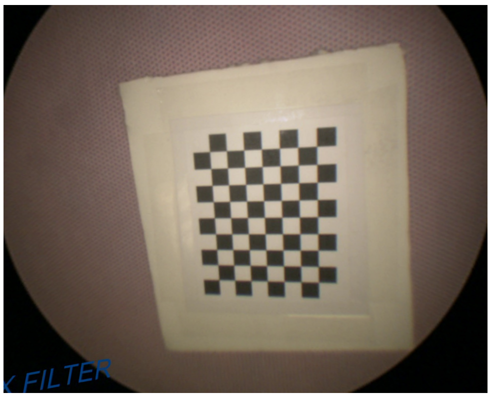


Figure 6: Undistorted.

Fisheye Calibration presented much more consistent results. Corrections can be easily identified in all the result pictures. All result pictures presented clear edges.

4 Problems Checklist

- The very first step to a successful camera calibration is a good checkerboard pattern, the following conditions should be satisfied.
  - print on thick-stock paper that won't stretch too much
  - choose flat backing
  - wait for glue-paper stress to reach long term steady state
- Temperature and humidity change can affect paper quality and in turn affect calibration result.
- The auto-focus feature should be off when taking pictures using target camera.
- Minimize vibrations and associated motion blur when taking photos.
- Take lots of measurements and pictures.
- Use optimal lighting

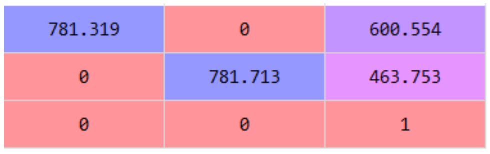
- Draw the detected corner positions on top of the images and remove out-liners.
- Use the simplest model that satisfies the measurements

5 Problems Assessment

- Attempt to run fisheye calibration on the resized pictures (320 X 240) failed as it resulted in empty image points and object points.
- Error calculated was either too large or too small. The calculated error does not match the results.

6 Conclusion

Fisheye is a better model for the target camera. The intrinsic camera matrix from fisheye calibration is



781.319	0	600.554
0	781.713	463.753
0	0	1

Figure 7: Intrinsic Matrix.

7 Appendices

7.1 Video Capture

```
import cv2
#import matplotlib.pyplot as plt

cap = cv2.VideoCapture(’ch1_video_02.mpg’)
print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

i = 0
j = 0
while(cap.isOpened()):
    ret, frame = cap.read()

    #for i in range(10):
    #    ret, frame = cap.read()
    if ret == True:

        if (i == 100):
            cv2.imwrite(’opencv’+str(j)+’.png ’, frame)
            j = j + 1
            i = 0

        i = i + 1
    else:
        break

cap.release()
cv2.destroyAllWindows()
```

7.2 Pinhole Calibration

```python
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((7*9,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:9].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

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

tot_error = 0
for fname in images:
    img = cv2.imread(fname)

    # resize
#    width = 320
#    height = 240
#    dim = (width, height)
#    img = cv2.resize(cv2.imread(fname), dim, interpolation = cv2.INTER_AREA)
#    cv2.imwrite(fname + '_resized.jpg', img)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (7,9),None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        # deep copy image
        #img2 = cv2.drawChessboardCorners(img, (7,9), corners2, ret)

        #cv2.imwrite('border' + fname + '.jpg ', img2)
for fname in images:
    img = cv2.imread(fname)
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
        gray.shape[::-1],None,None)

    h,  w = img.shape[:2]
    newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))


    mapx,mapy = cv2.initUndistortRectifyMap(mtx,dist,None,newcameramtx,(w,h),5)
    dst = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)

    x,y,w,h = roi
        #dst = dst[y:y+h, x:x+w]
        # undistort
        #dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

# crop the image
#       x,y,w,h = roi
#       dst = dst[y:y+h, x:x+w]

    cv2.imwrite('ud_' + fname + '.png ',dst)
        #i = i + 1

#       cv2.imwrite(fname, img)
#       cv2.waitKey(500)

#cv2.destroyAllWindows()

#ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
     gray.shape[::-1],None,None)
#mean_error = 0

for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
    tot_error += error

print('total error:', tot_error/len(objpoints))
```

7.3 Fisheye Calibration

```
import cv2

import numpy as np
import os
import glob
CHECKERBOARD = (7,9)
subpix_criteria = (cv2.TERM_CRITERIA_EPS+cv2.TERM_CRITERIA_MAX_ITER, 30, 0.1)
calibration_flags =
     cv2.fisheye.CALIB_RECOMPUTE_EXTRINSIC+cv2.fisheye.CALIB_CHECK_COND+cv2.fisheye.CALIB_FIX_SKEW
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)
_img_shape = None
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('*.png')
#i = 0;
for fname in images:
    img = cv2.imread(fname)
    # unable to resize, resulting in empty imgpoints
    #img = cv2.resize(cv2.imread(fname), (320, 240), interpolation = cv2.INTER_AREA)
    if _img_shape == None:
        _img_shape = img.shape[:2]
    else:
        assert _img_shape == img.shape[:2], "All images must share the same size."
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD,
        cv2.CALIB_CB_ADAPTIVE_THRESH+cv2.CALIB_CB_FAST_CHECK+cv2.CALIB_CB_NORMALIZE_IMAGE)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        cv2.cornerSubPix(gray,corners,(3,3),(-1,-1),subpix_criteria)
        imgpoints.append(corners)



        # Draw and display the corners
        # deep copy image
        img2 = cv2.drawChessboardCorners(img, (7,9), corners, ret)

        cv2.imwrite('border' + fname + '.jpg ', img2)
    N_OK = len(objpoints)
    K = np.zeros((3, 3))
    D = np.zeros((4, 1))
    rvecs = [np.zeros((1, 1, 3), dtype=np.float64) for i in range(N_OK)]
    tvecs = [np.zeros((1, 1, 3), dtype=np.float64) for i in range(N_OK)]
    rms, _, _, _, _ = \
        cv2.fisheye.calibrate(
            objpoints,
            imgpoints,
            gray.shape[::-1],
            K,
            D,
            rvecs,
            tvecs,
            calibration_flags,
            (cv2.TERM_CRITERIA_EPS+cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-6)
        )
#    print("Found " + str(N_OK) + " valid images for calibration")
#    print("DIM=" + str(_img_shape[::-1]))
#    print("K=np.array(" + str(K.tolist()) + ")")
#    print("D=np.array(" + str(D.tolist()) + ")")
    DIM=_img_shape[::-1]
#    K
#    D=np.array(ZZZ)
def undistort(img_path):
    img = cv2.imread(img_path)
    h,w = img.shape[:2]
    map1, map2 = cv2.fisheye.initUndistortRectifyMap(K, D, np.eye(3), K, DIM, cv2.CV_16SC2)

    undistorted_img = cv2.remap(img, map1, map2, interpolation=cv2.INTER_LINEAR,
         borderMode=cv2.BORDER_CONSTANT)
    #cv2.imshow("undistorted", undistorted_img)
    cv2.imwrite('ud' + fname + '_ud.jpg ',undistorted_img)

    cv2.waitKey(0)
        #cv2.destroyAllWindows()
#    if __name__ == '__main__':
#        for p in sys.argv[1:]:
#            undistort(p)
for fname in images:
    undistort(fname)
    #i = i + 1

tot_error = 0
for i in range(len(objpoints)):
     imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
     error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
     tot_error += error

print('total error:', tot_error/len(objpoints))
```

7.3.1 Resize

```
import numpy as np
import cv2
import glob

images = glob.glob('*.png')
```

```python
for fname in images:
    width = 320
    height = 240
    dim = (width, height)
    img = cv2.resize(cv2.imread(fname), dim, interpolation = cv2.INTER_AREA)

    cv2.imwrite('resized_' + fname + '.jpg', img);
```