Camera Calibration

Tommy Le

Northern Illinois University, Mechanical Engineering Department

1. INTRODUCTION

Computer vision is a growing field and plays an important role in many robotics applications. A camera is essentially a sensor that takes the 3D world and turns it into a 2D image. Unfortunately, images from a camera are prone to distortion due to the curvature of their lenses, and/or the configuration of their models, making camera calibration a crucial step in computer vision applications, especially ones where distance information about the frame is needed. Camera calibration is the process of determining the intrinsic and extrinsic parameters of the camera and using those to mitigate the distortion of the frame. This document will explain a basic process for camera calibration.

2. ASSUMPTIONS

This manual assumes that you have general knowledge of Python and OpenCV programming as well as the following packages downloaded and installed

- A programming IDE (VS Code)
- Python libraries
  - OpenCV-Python
  - NumPy
  - Glob

3. METHODS

The process for calibrating your camera using the attached files is as follows:

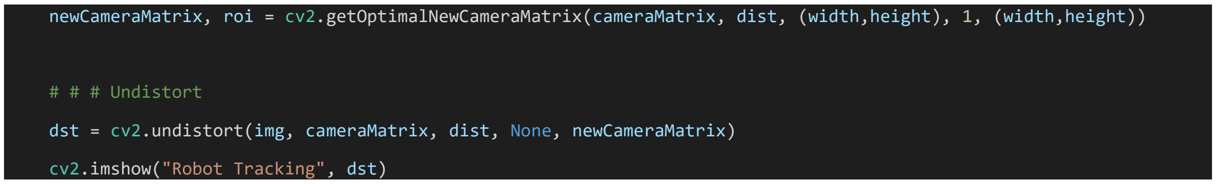
1) Print out the checkerboard pattern, “pattern-8x8.pdf”, and tape it to a flat, rigid surface.
2) Using “Recording.py”, record a video of the checkerboard pattern translated and rotated at different positions, focusing on collecting images with a high variance in poses.
3) Find the video you just recorded in your working directory, go through it and screenshot images from the video of the checkerboard in different poses and save them all into one folder. Aim for 20 or more images, the more images the more accurate your calibration will be.
4) Go into the folder you just created (“cd <directory>” in the terminal) and open “Cam_Calib.py”. Included in this folder is the folder “calib_pics” which are images I took to use for calibration which you can use as an example.
5) Go to line 31 in the code, and change the entry in glob.glob() to whatever image extension you are using (i.e ‘*.png’ or ‘*.jpg’)
6) Run the program. You should see your RMS error, camera matrix, and distortion coefficients returned in the terminal.

6a) OPTIONAL. Uncomment the portion under “UNDISTORTION” on line 69 in the program and enter the name of an image you would like to undistort in line 71. This will save two undistorted images in your working directory called “caliResult1.png” and “caliResult2.png”. The former being a cropped image and the latter being a remapped image.

6b) OPTIONAL. Uncomment the section under “Reprojection Error” on line 99 and run the program. You will then be returned the reprojection error of your calibrated camera

7) Save the outputted RMS error, camera matrix, and distortion coefficients as variables in your desired program. I saved it back into “Recording.py” so that I could save a calibrated video to apply post processing techniques onto later. The rest of this tutorial will be referencing “Recording.py” for the implementation

8) After saving your camera information into their respective variables, go to your video capturing loop and add the following lines of code to undistort your frame, and view it in real time.



```
newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist, (width,height), 1, (width,height))

# # # Undistort
dst = cv2.undistort(img, cameraMatrix, dist, None, newCameraMatrix)
cv2.imshow("Robot Tracking", dst)
```

9) This should supply you with an undistorted image which can be used as long as you are using the same camera. For more information look at the appendix, or the attached files.

4. APPENDIX

Recording.py

```
# imports
import cv2
import numpy as np

#Found cam calib constants
ret = 0.3316338761548716
cameraMatrix = np.array([[715.03132426, 0.0, 462.25626965],
 [0.0, 715.67733789, 265.2425816],
 [0.0,        0,               1]])
dist = np.array([[0.05515529], [-0.20330285], [-0.00108968], [-0.00468666],
[0.25046973]])

cap = cv2.VideoCapture(1, cv2.CAP_DSHOW) # change this value until you get the
correct cam
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

# print(height)
# print(width)

fourcc = cv2.VideoWriter_fourcc(*'MJPG') #*'mp4v' *'MJPG'
out = cv2.VideoWriter('250PWM.avi', fourcc, 30.0, (width, height), isColor=True)

while True:
    _, img = cap.read()
    # resizing for faster processing
    img = cv2.resize(img, (width,height), interpolation = cv2.INTER_LINEAR)

    newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist,
(width,height), 1, (width,height))

    # # # Undistort
    dst = cv2.undistort(img, cameraMatrix, dist, None, newCameraMatrix)

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

    out.write(dst)
    cv2.imshow("Robot Tracking", img)
    # 'Esc' key to exit video streaming
    key = cv2.waitKey(1)
    if key ==27:
        break

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

Cam_Calib.py

```
import numpy as np
import cv2 as cv
import glob



################ FIND CHESSBOARD CORNERS - OBJECT POINTS AND IMAGE POINTS
#############################

chessboardSize = (7,7)
frameSize = (1440,1080)



# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.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((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 19
objp = objp * size_of_chessboard_squares_mm


# 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('*.png')

for image in images:

    #cv.namedWindow("output", cv.WINDOW_NORMAL)
    img = cv.imread(image)
    img = cv.resize(img, (960, 540))
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)

    # If found, add object points, image points (after refining them)
```

```python
    if ret == True:

        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)

        # Draw and display the corners
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(1000)


cv.destroyAllWindows()




############## CALIBRATION
#######################################################

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
frameSize, None, None)
print("ret: " + str(ret))
print("cam matrix: " + str(cameraMatrix))
print("dist: " + str(dist))


############## UNDISTORTION #####################################################

# img = cv.imread('Screenshot (40).png')
# h,  w = img.shape[:2]
# newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,h),
1, (w,h))



# # Undistort
# dst = cv.undistort(img, cameraMatrix, dist, None, newCameraMatrix)

# # crop the image
# x, y, w, h = roi
# dst = dst[y:y+h, x:x+w]
# cv.imwrite('caliResult1.png', dst)
# # Undistort with Remapping
# mapx, mapy = cv.initUndistortRectifyMap(cameraMatrix, dist, None,
newCameraMatrix, (w,h), 5)
# dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)

# # crop the image
# x, y, w, h = roi
# dst = dst[y:y+h, x:x+w]
# cv.imwrite('caliResult2.png', dst)




#Reprojection Error
# mean_error = 0

# for i in range(len(objpoints)):
#     imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i],
cameraMatrix, dist)
#     error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
#     mean_error += error

# print( "total error: {}".format(mean_error/len(objpoints)) )
```