Camera Calibration & Birdseye View

—Homework4 for computer vision

Name: Wang Jun

Student ID：3170100186

Date: 2021-12-12

Course: Computer Vision

Instructor: Mingli Song

Chapter 1: The purpose and requirements of the experiment

1. Use checkerboard images to calibrate a camera.
- Example: Learning OpenCV 18-1
- Write the intrinsic parameters to intrinsics.xml or another file specified
- Refer to the checkboard images at Learning
- OpenCV/LearningOpenCV_Code/LearningOpenCV_Code/calibratio as examples

2. Do a perspective warp on images calibrated using the intrinsic parameters provided by the previous step
- Example: Learning OpenCV 19-1
- Refer to the test images in Learning
- OpenCV/LearningOpenCV_Code/LearningOpenCV_Code/birdseye as examples

Notes:
1. Use the example code unchanged if you prefer, and analyze principles of it in your report if you do
2. Specify usage of your program and expected results
3. Submit a experiment report, the source code and a packaged executable

Chapter 2: Experimental content and principle

OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

Calibration

A camera is an integral part of several domains like robotics, space exploration, etc camera is playing a major role. It helps to capture each and every moment and helpful for many analyses. In order to use the camera as a visual sensor, we should know the parameters of the camera. Camera Calibration is nothing but estimating the parameters of a camera, parameters about the camera are required to determine an accurate relationship between a 3D point in the real world and its corresponding 2D projection (pixel) in the image captured by that calibrated camera.

We need to consider both internal parameters like focal length, optical center, and radial distortion coefficients of the lens etc., and external parameters like rotation and translation of the camera with respect to some real world coordinate system.

Camera Calibration can be done in a step-by-step approach:

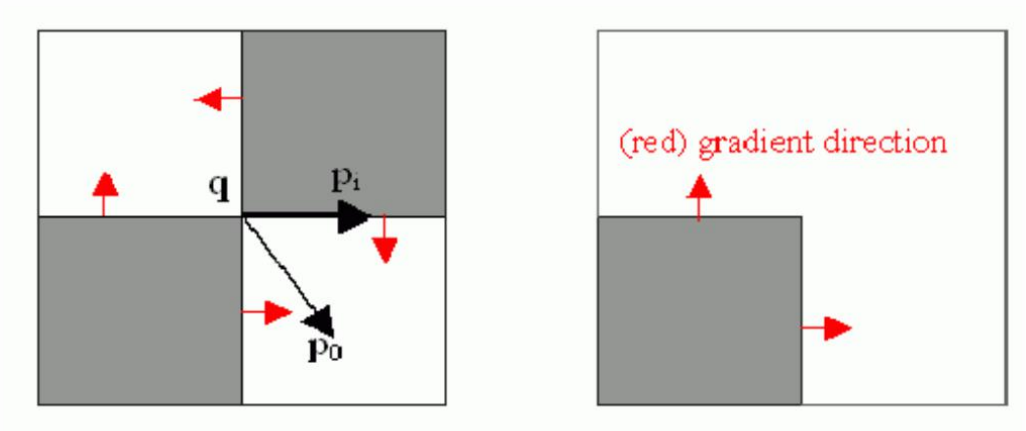
- Step 1: First define real world coordinates of 3D points using known size of checkerboard pattern.
- Step 2: Different viewpoints of check-board image is captured.
- Step 3: findChessboardCorners() is a method in OpenCV and used to find pixel coordinates (u, v) for each 3D point in different images
- Step 4: Then calibrateCamera() method is used to find camera parameters.
- It will take our calculated (threedpoints, twodpoints, grayColor.shape[::-1], None, None) as parameters and returns list having elements as Camera matrix, Distortion coefficient, Rotation Vectors, and Translation Vectors.
- Camera Matrix helps to transform 3D objects points to 2D image points and the Distortion Coefficient returns the position of the camera in the world, with the values of Rotation and Translation vectors

The function attempts to determine whether the input image is a view of the chessboard pattern and locate the internal chessboard corners. The function returns a non-zero value if all of the corners are found and they are placed in a certain order (row by row, left to right in every row). Otherwise, if the function fails to find all the corners or reorder them, it returns 0.

For example, a regular chessboard has 8 x 8 squares and 7 x 7 internal corners, that is, points where the black squares touch each other. The detected coordinates are approximate, and to determine their positions more accurately, the function calls **cornerSubPix** . You also may use the function **cornerSubPix** with different parameters if returned coordinates are not accurate enough.

For every image in the specified folder, we find the checkerboard corners and refine them using **cornersSubPix**

Refines the corner locations:
The function iterates to find the sub-pixel accurate location of corners or radial saddle points as described, and as shown on the figure below.



$$\epsilon_i = DI_{p_i}{}^T \cdot (q - p_i)$$

We also found a very helpful blog at explain how the corner refinement actually work

A good blog:

Takeaway: The corner of a square is a point where all others' points gradient are orthogonal to the vector between the corner and the point in question.
Then we used **calibrateCamera** to calculate the camera intrinsic parameters and distortion coefficients used to describe the projection of that camera.We don't use the strange flags in our implementation because they seem to make the rectification process unacceptable

Then we use FileStorage to write the parameters and coefficients to the specified file.
If the user wants so, we can also use undistort to undistort a image using the parameters calculated earlier.
Or, if we prefer a more manual approach, we can use **initUndistortRectifyMap** and **remap** to undistort the image

Birdseye View

Similar as before, we use **FileStorage** to load the saved parameters and coefficients from the file provided by the user.
Then we use **findChessboardCorners** and **cornerSubPix** to find subpixel locations

of the checkerboard corners for the image. Using the parameters, we undistort the image first.

And we use **getPerspectiveTransform** to compute perspective transform from one set of 4 points to another. Later for the images we use **warpPerspectiveTransform** to apply the transform obtained earlier.

In **Perspective Transformation**, ,we can change the perspective of a given image or video for getting better insights about the required information. In Perspective Transformation, we need provide the points on the image from which want to gather information by changing the perspective. We also need to provide the points inside which we want to display our image. Then, we get the perspective transform from the two given set of points and wrap it with the original image.

Note that:
dst(x, y) are defined by src(homo(matmul(M, (x, y, 1))))
so the transformation actually acts on the coordinates of the dstination image
thus OpenCV would firstly apply a inversion by default

So we'd have to set the flag **WARP_INVERSE_MAP** if we want to manipulate the transform matrix by hand.When the transformation matrix is computed, we'd actually like to manipulate it by hand because the user may want to set the zooming scale or the Birdseye View height. We utilized the fact that transform as a matrix can be layered together by matrix multiplication.

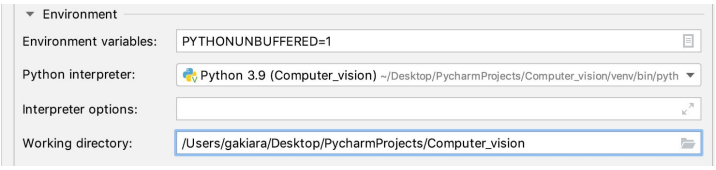
Chapter 3: Experimental procedure and analysis

Install opencv-python

Install opencv-python for the next process

```
(venv) gakiaradeMacBook-Pro:Computer_vision gakiara$ pip install opencv-python
Collecting opencv-python
  Downloading opencv-python-4.5.4.60.tar.gz (89.8 MB)
     |███                             | 8.6 MB 47 kB/s eta 0:28:32
```

Set the working directory:



Read the Utilities

```
import cv2
import numpy as np
import os
import coloredlogs
import argparse
import matplotlib as mpl


class CustomFormatter(argparse.ArgumentDefaultsHelpFormatter, argparse.RawDescriptionHelpFormatter):
    pass

# initailization for plotting and logging
# Setting up font for matplotlib
mpl.rc("font", family=["Josefin Sans", "Trebuchet MS", "Inconsolata"], weight="medium")

coloredlogs.install(level='INFO')  # Change this to DEBUG to see more info.

import logging
# Setting up logger for the project
log = logging.getLogger(__name__)
```

Calibration

- **Initialization :**

```
# parse and process the parsed arguments
path = args.input_path
fpaths = [os.path.join(path, fname) for fname in os.listdir(path)]  # all image paths
board_w = args.columns
board_h = args.rows
board_n = board_w*board_h
board_sz = (board_w, board_h)
show_imgs = args.should_check
output = args.output_file

# initializating
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)  # criteria for subPix corners finder
img_pts = []  # image points in calibrate camera
obj_pts = []  # world points in calibrate camera
img_shp = ()  # size/shape of the image to be used for calibrattion
objp = np.zeros((board_n, 3), "float32")  # all object/world points are the same set
objp[:, :2] = np.mgrid[0:board_h, 0:board_w].T.reshape(-1, 2)
imgs = []  # so we don't have to read all images in again
files = []  # valid filenames
```

- **Image Points Loop :**

find all checker board corners and add corresponding 3D space locations:
do a normalization beforehand
CALIB_CB_ADAPTIVE_THRESH Use adaptive thresholding to convert the image to black and white, rather than a fixed threshold level (computed from the average image brightness).
use adaptive threashold to BW image instead of a fixed one
CALIB_CB_NORMALIZE_IMAGE Normalize the image gamma with equalizeHist before applying fixed or adaptive thresholding.

```
for fpath in fpaths:
    log.info(f"Begin processing {fpath}")
    img = cv2.imread(fpath)
    if img is None:
        log.warning(f"Cannot read image {fpath}, is this really a image?")
        break
    img_shp = img.shape[:2][::-1]  # OpenCV wants (width, height)
    found, corners = cv2.findChessboardCorners(
        img,  # the BGR image to be used to find checker board corners on
        board_sz,  # (board_w, board_h)
        flags=cv2.CALIB_CB_ADAPTIVE_THRESH | cv2.CALIB_CB_NORMALIZE_IMAGE
```

- **do the calibration:**

```
err, intr, dist, rota, tran = cv2.calibrateCamera(
    obj_pts,  # object points in 3D
    img_pts,  # corner points in 2D
    img_shp,  # shape of the image to be calibrated
    None,  # setting to None to let the function return them
    None,  # setting to None to let the function return them
    # flags=cv2.CALIB_ZERO_TANGENT_DIST | cv2.CALIB_FIX_PRINCIPAL_POINT
    # CALIB_ZERO_TANGENT_DIST Tangential distortion coefficients (p1,p2) are set to zeros and stay zero
    # The principal point is not changed during the global optimization. It stays at the center
)
```

- **Save Results:**

```
log.info(f"Opening {output} for output")
# store camera intrinsics and distortion coefficients
fs = cv2.FileStorage(output, cv2.FILE_STORAGE_WRITE)
fs.write("image_width", img_shp[0])
fs.write("image_height", img_shp[1])
fs.write("camera_matrix", intr)
fs.write("distortion_coefficients", dist)
fs.release()
log.info(f"camera_matrix and distortion_coefficients stored to {output}")
```

Birdseye

- **Initialization :**

parse the arguments & compute things from the arguments:

```
args = parser.parse_args()
intrfile = args.input_file
path = args.directory
board_w = args.columns
board_h = args.rows

fpaths = [os.path.join(path, fname) for fname in os.listdir(path)]
board_n = board_w*board_h
board_sz = (board_w, board_h)
```

- **Load Coefficients And Parameters:**

read in camera calibration information:
camera intrinsics and distortion coefficients:

```
fs = cv2.FileStorage(intrfile, cv2.FILE_STORAGE_READ)
img_shp = tuple(map(int, (fs.getNode("image_width").real(), fs.getNode("image_height").real())))
intr = fs.getNode("camera_matrix").mat()
dist = fs.getNode("distortion_coefficients").mat()
```

- **Prepare For Transform:**

initializating stuff:

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

criteria in subPix corner refinemnet the iteration of cornerSubPix stops when criteria.maxIter is reached or the corner position moves less than epsilon

currently using the borad's default coordinates:

```
objPts[0][0] = 0
objPts[0][1] = 0
objPts[1][0] = board_w - 1
objPts[1][1] = 0
objPts[2][0] = 0
objPts[2][1] = board_h - 1
objPts[3][0] = board_w - 1
objPts[3][1] = board_h - 1
```

image points:

```
imgPts = np.zeros((4, 2), "float32")
```

- **Actual Transform And Interaction Loop:**

```
for fpath in fpaths:
    log.info(f"Begin processing {fpath}")
    img_dist = cv2.imread(fpath)
    if img_dist is None:
        log.warning(f"Cannot read image {fpath}, is this really a image?")
        break
    img = cv2.undistort(img_dist, intr, dist)  # do an undistortion using the parameters obtained from calibrate.py
    log.info(f"{fpath} undistorded")
    img_shp = img.shape[:2][::-1]  # OpenCV wants (width, height)
    found, corners = cv2.findChessboardCorners(img, board_sz, None, cv2.CALIB_CB_ADAPTIVE_THRESH | cv2.CALIB_CB_NORMALIZE_IMAGE)
    log.info(f"Found {corners.shape[0]} checkerboard corners of shape: {corners.shape}")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # OpenCV wants gray image for subPix detection
    corners = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)  # refine checker board location
    log.info(f"Refined {corners.shape[0]} checkerboard corners of shape: {corners.shape}")
    corners = np.squeeze(corners)
    log.info(f"Corners squeezed to {corners.shape}")
```

points on the image plane:

```
imgPts[0] = corners[0]
imgPts[1] = corners[board_w - 1]
imgPts[2] = corners[(board_h - 1) * board_w]
imgPts[3] = corners[(board_h - 1) * board_w + board_w - 1]
```

- draw the checker board to amuse the user:

```
cv2.drawChessboardCorners(img, board_sz, corners, found)
log.info(f"imgPts shape: {imgPts.shape}, objPts shape: {objPts.shape}")
H = cv2.getPerspectiveTransform(objPts, imgPts)  # transform matrix, from
Z = H[2, 2]  # Z value, view height
# Z = 2.0  # Z value, view height
S = 10.0  # Scale value
C = True
quit = False  # should we quit?
shape = img.shape[:2][::-1]  # img shape
```

- construct the scale image for image:

scale matrix, scale down the matrix to make the result scale up：

```
scale = np.diag([1/S, 1/S, 1])
```

- **Procedure:**

This is actually the basic procedure of what warpPerspective except that it maps from dstination coordinates to src image coordinates and get the color values

#Get the inverse of the mapping matrix：

```
invM = np.linalg.inv(M)  # this one maps coordinates in src image to coordinates in dst image
ones = np.ones((1, board_n))  # to make stuff homographic
log.info(f"Getting corners.T of shape {corners.T.shape} and ones of shape {ones.shape}")
expC = np.concatenate([corners.T, ones], 0)  # construct the homographic original corner points
log.info(f"Getting expanded corners of shape {expC.shape}")
invC = np.matmul(invM, expC).T  # get the mapped coordinates of the corners
invC[:, 0] /= invC[:, 2]  # apply Z value to corners
invC[:, 1] /= invC[:, 2]  # apply Z value to corners
invC = invC[:, :2]  # get only the first two columns
invC = invC.astype("float32")  # mark: strange error if you don't convert stuff to float32
```

Update significant corner points base on the user's specification of Z and S value to draw circles on later:

```
imgPts[0] = invC[0]
imgPts[1] = invC[board_w - 1]
imgPts[2] = invC[(board_h - 1) * board_w]
imgPts[3] = invC[(board_h - 1) * board_w + board_w - 1]
```

when the flag WARP_INVERSE_MAP is set. Otherwise, the transformation is first inverted with invert and then put in the formula above instead of M. The function cannot operate in-place.dst(x, y) are defined by src(homo(matmul(H, (x, y, 1)))) so the transformation actually acts on the coordinates of the dstination image thus OpenCV would firstly apply a inversion by default:

```
birdseye = cv2.warpPerspective(
    img, M, shape,
    flags=cv2.INTER_LINEAR | cv2.WARP_INVERSE_MAP,  # don't do matrix inverse before transform
    borderMode=cv2.BORDER_CONSTANT  # use constant value to fill the border
)
```

whether user want to display the checker board points or not:

```
if C:  # whether user want to display the checker board points or not
    cv2.drawChessboardCorners(birdseye, board_sz, invC, True)
    # mark the image points to be used to generate birdseye view
    cv2.circle(birdseye, tuple(imgPts[0].astype(int).tolist()), 9, (255, 0, 0), 3)
    cv2.circle(birdseye, tuple(imgPts[1].astype(int).tolist()), 9, (0, 255, 0), 3)
    cv2.circle(birdseye, tuple(imgPts[2].astype(int).tolist()), 9, (0, 0, 255), 3)
    cv2.circle(birdseye, tuple(imgPts[3].astype(int).tolist()), 9, (0, 255, 255), 3)
```

- **Update:**

update view height:

```
if k == ord('u'):
    Z += 0.1
if k == ord('d'):
    Z -= 0.1
```

update scale factor:

```
if k == ord('i'):
    S += 0.5
if k == ord('o'):
    S -= 0.5
```

toggle checker board:

```
if k == ord('t'):
    C = not C
```

Reset:

```
if k == ord('r'):
    Z = H[2, 2]
    S = 10.0
```

Chapter 4: Experimental environment and operation method

Development environment

- macOS 10.14.6

- python 3.9

- opencv-python 4.5.1.48

Operation mode

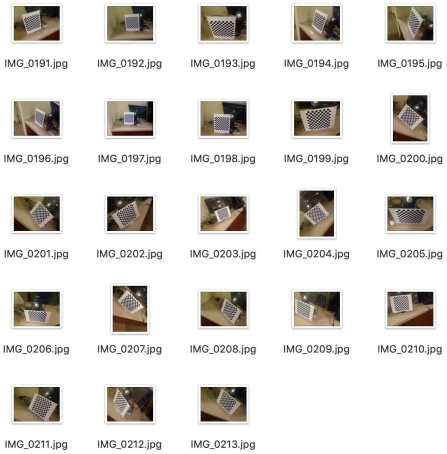
-python calibrate.py -c 12 -r 12 (column and row)

-python birdseye.py -c 12 -r 12 (column and row)

Chapter 5: Experimental results

Calibration

INPUT:

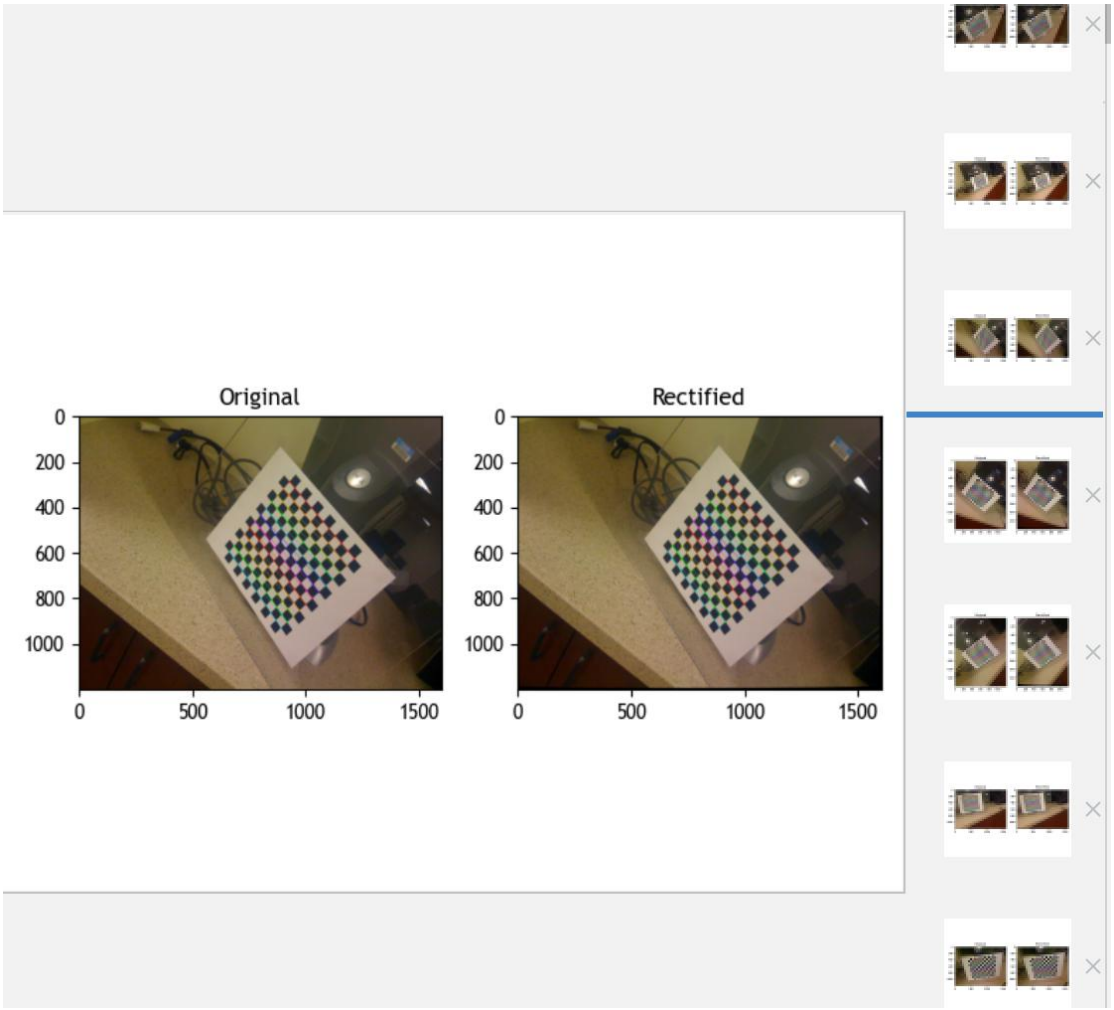


Run:

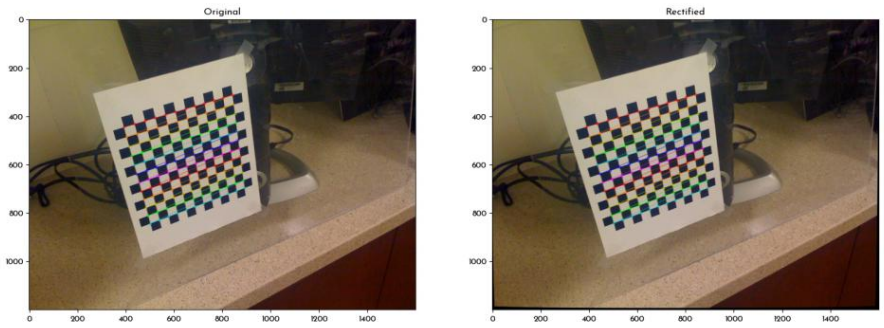
```
Python Console
>>> runfile('/Users/gakiara/Desktop/PycharmProjects/ComputerVision/calibrate.py', wdir='/Users/gakiara/Desktop/
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0202.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0203.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0201.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0200.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0204.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0210.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0199.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0198.jpg
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:50 gakiaradeMacBook-Pro.local __main__[9348] INFO Begin processing ./calibration/IMG_0211.jpg
```

```
2021-12-24 22:18:51 gakiaradeMacBook-Pro.local __main__[9348] INFO Found 144 checkerboard corners
2021-12-24 22:18:51 gakiaradeMacBook-Pro.local __main__[9348] INFO Refined 144 checkerboard corners
2021-12-24 22:18:51 gakiaradeMacBook-Pro.local __main__[9348] INFO Beginning calibration using images in: ./calibration, files: ['./calibration/IMG_0202.jpg',
2021-12-24 22:18:52 gakiaradeMacBook-Pro.local __main__[9348] INFO Got camera intrinsics:
[[1.76650014e+03 0.00000000e+00 8.17488855e+02]
 [0.00000000e+00 1.76253327e+03 6.71145499e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]] and distortion coefficients:
[[ 0.03552487 -0.27852743  0.0114769   0.00258806  0.88162491]]
2021-12-24 22:18:52 gakiaradeMacBook-Pro.local __main__[9348] INFO Opening intrinsics.xml for output
2021-12-24 22:18:52 gakiaradeMacBook-Pro.local __main__[9348] INFO camera_matrix and distortion_coefficients stored to intrinsics.xml
```

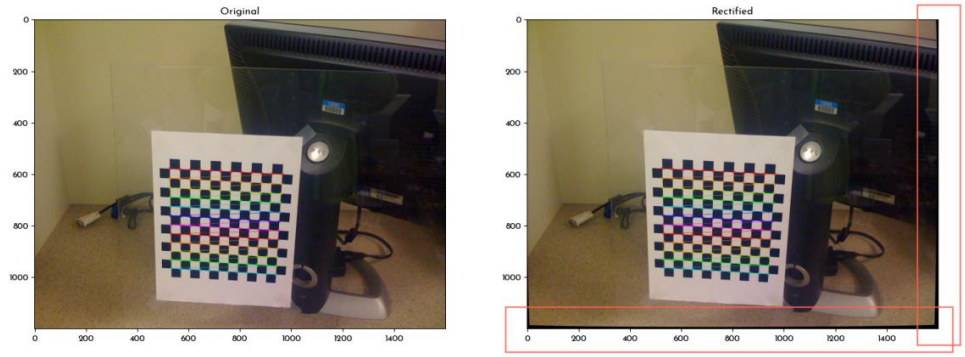
OUTPUT:



This one was pretty fine before calibration:



This one's pin cushion distortion is visually corrected:



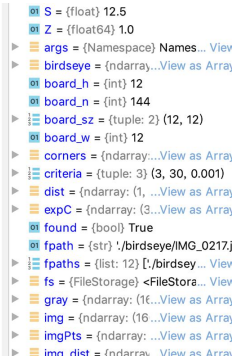
Birdseye

Note that we saved the camera intrinsic parameters and distortion coefficients to intrinsics.xml in our first implementation:

```
  <data>
    1.7665001354193548e+03 0. 8.1748885450142586e+02 0.
    1.7625332695956029e+03 6.7114549896100618e+02 0. 0. 1.</data></camera_matrix>
<distortion_coefficients type_id="opencv-matrix">
  <rows>1</rows>
  <cols>5</cols>
  <dt>d</dt>
  <data>
    3.5524872710714153e-02 -2.7852742568991956e-01
    1.1476904001174463e-02 2.5880585944984386e-03 8.8162491028320267e-01</data></distortion_coefficients>
```

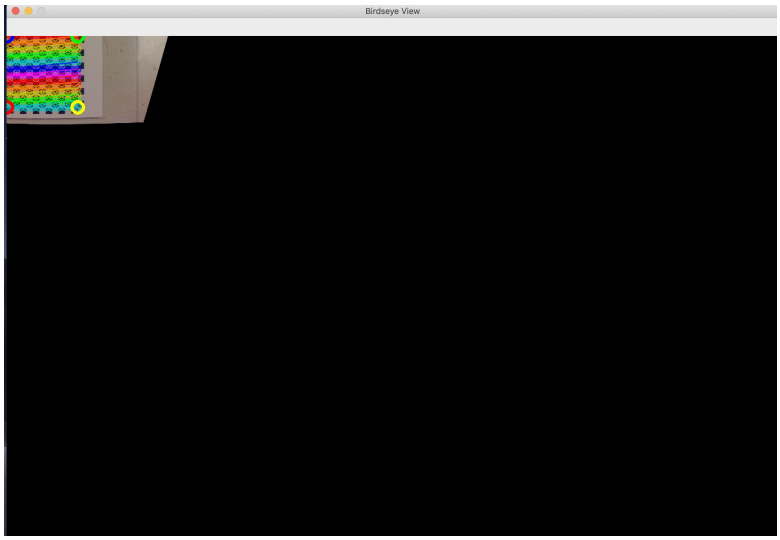
Run:

```
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting corners.T of shape (2, 144) and ones of shape (1, 144)
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting expanded corners of shape (3, 144)
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting inverted corners of shape (144, 2)
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting key: i or order 105
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting H of [[ 2.64129520e+00 -7.76851126e-01  3.53435059e+02]
 [-1.85435543e-01  1.65648614e-01  1.21219727e+03]
 [-1.36704478e-04 -1.29846150e-03  1.00000000e+00]], Z of 1.0
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting corners.T of shape (2, 144) and ones of shape (1, 144)
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting expanded corners of shape (3, 144)
2021-12-24 22:32:10 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting inverted corners of shape (144, 2)
2021-12-24 22:32:11 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting key: i or order 105
2021-12-24 22:32:11 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting H of [[ 2.53564339e+00 -7.45777081e-01  3.53435059e+02]
 [-1.78018121e-01  1.59022670e-01  1.21219727e+03]
 [-1.31236298e-04 -1.24652304e-03  1.00000000e+00]], Z of 1.0
2021-12-24 22:32:11 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting corners.T of shape (2, 144) and ones of shape (1, 144)
2021-12-24 22:32:11 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting expanded corners of shape (3, 144)
2021-12-24 22:32:11 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting inverted corners of shape (144, 2)
2021-12-24 22:32:27 gakiaradeMacBook-Pro.local __main__[11226] INFO Getting key:   or order 27
```



And it opens up the first image with a birds eye transform to:

```
objPts = np.zeros((4, 2), "float32")  # object points to be used to generate birdseye view
# currently using the borad's default coordinates
objPts[0][0] = 0
objPts[0][1] = 0
objPts[1][0] = board_w - 1
objPts[1][1] = 0
objPts[2][0] = 0
objPts[2][1] = board_h - 1
objPts[3][0] = board_w - 1
objPts[3][1] = board_h - 1
```

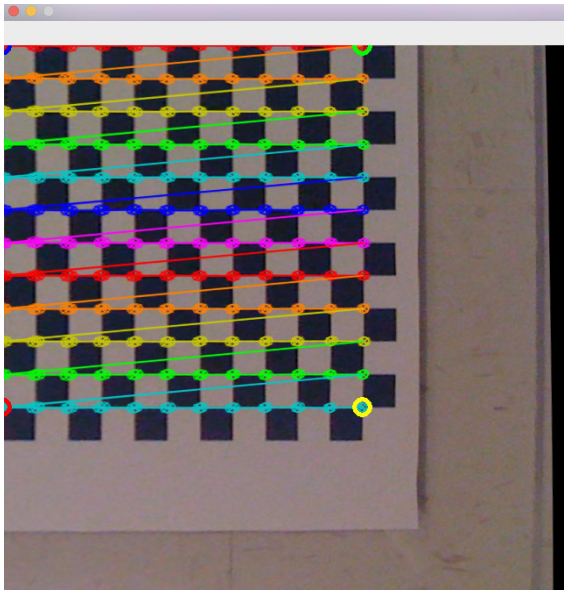


We used OpenCV's HighGUI for simple user interaction:
i (scale up: zoom in) or o (scale down: zoom out)
u (for up) and d (for down)
t to toggle for the checkerboard indicator
r to reset stuff
n for next image

Rectified Original Image:

- **i (scale up: zoom in) or o (scale down: zoom out) ：**

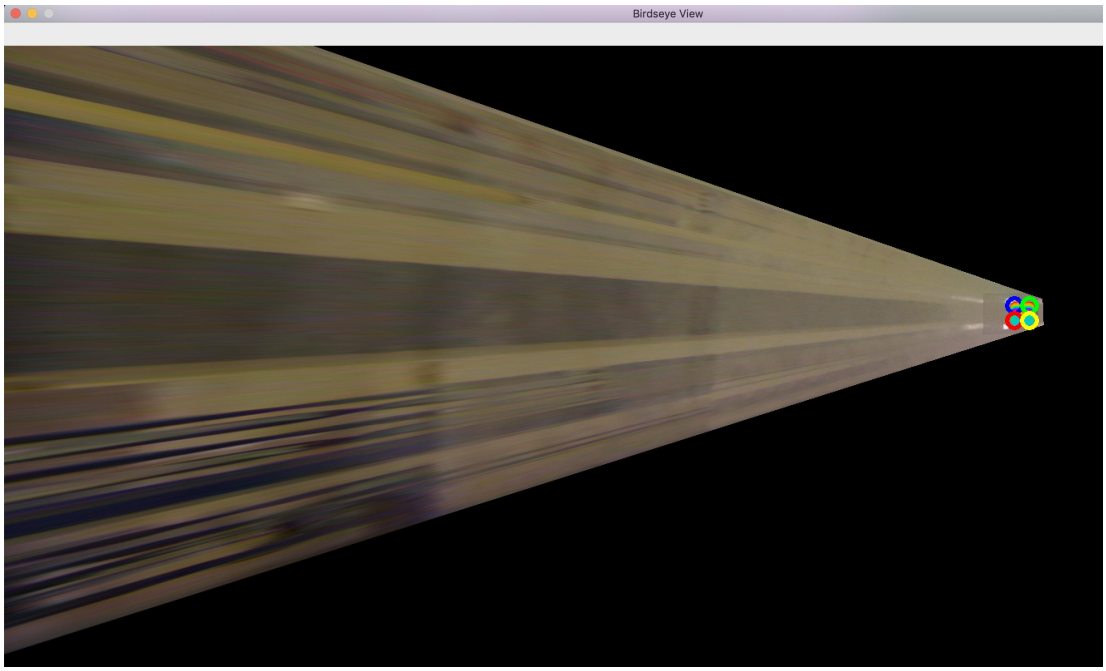
zoom in：

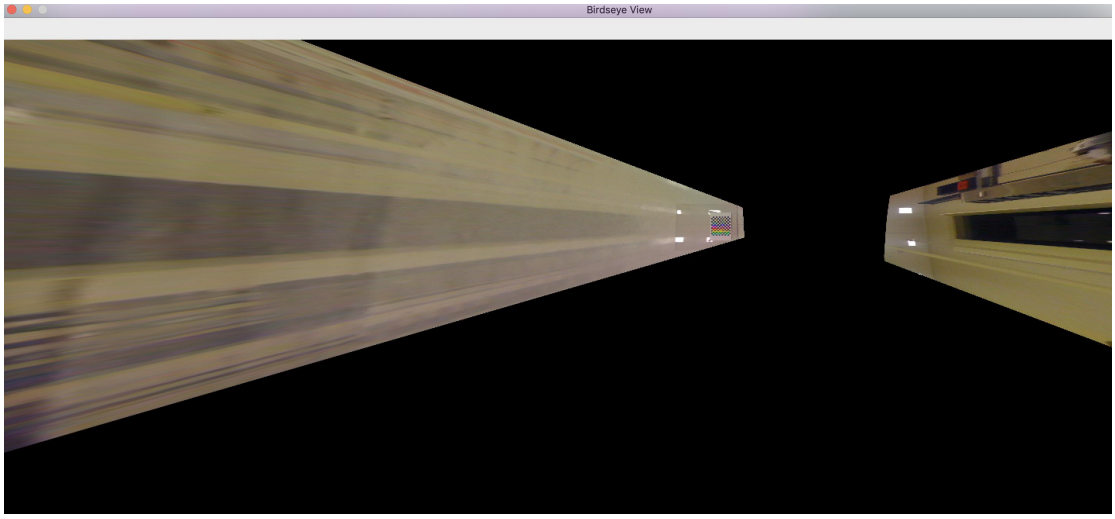


zoom out：

- **u (for up) and d (for down) ：**

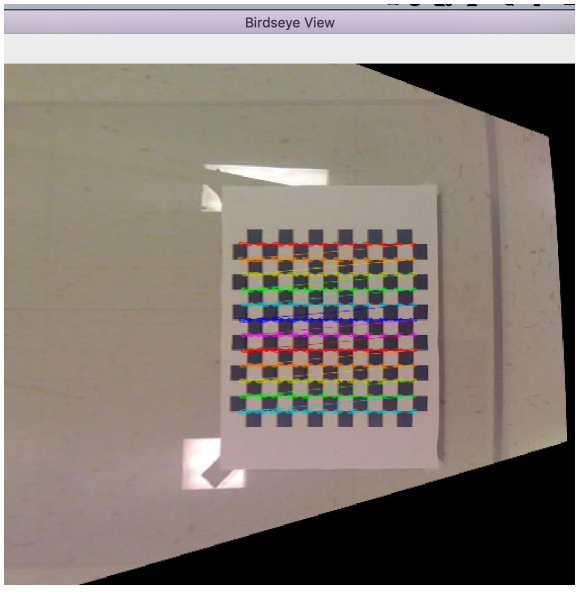
Up：

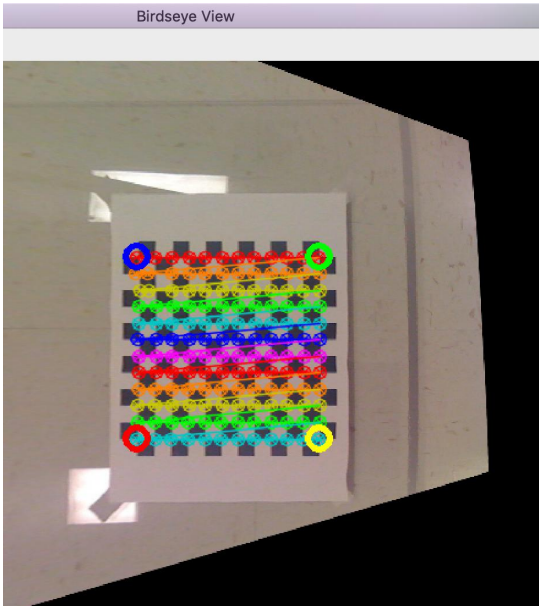


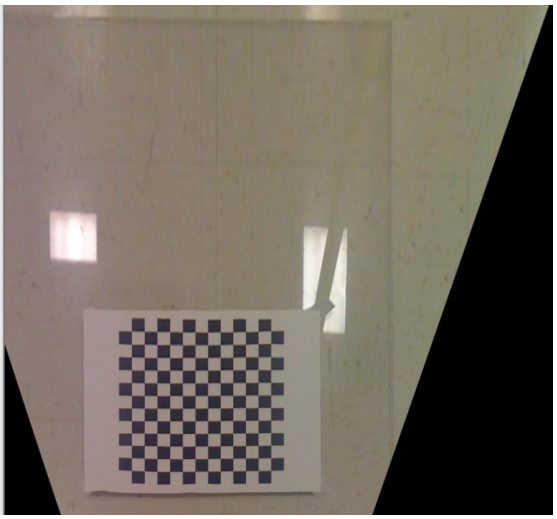


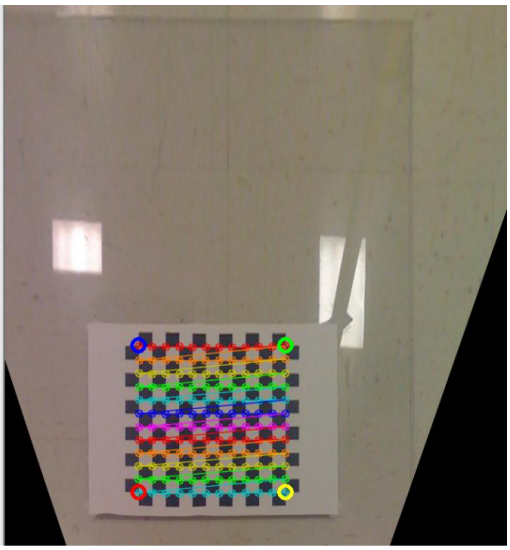
Down:

- **t for Toggle Checkerboard Marker:**









Chapter 6: Thoughts

OpenCV in python and C++

I use mac-os to finish my work, and find the visual studio in mac is hard to use in C++, so I try to use python instead. And surprisingly, python is much easier than C++ that not only in its readable, but also it did not require you to learn about a new class Mat to be able to operate on images. In a nutshell, for man using a mac os, python would be a better choice, and the deployment of environment is easier.

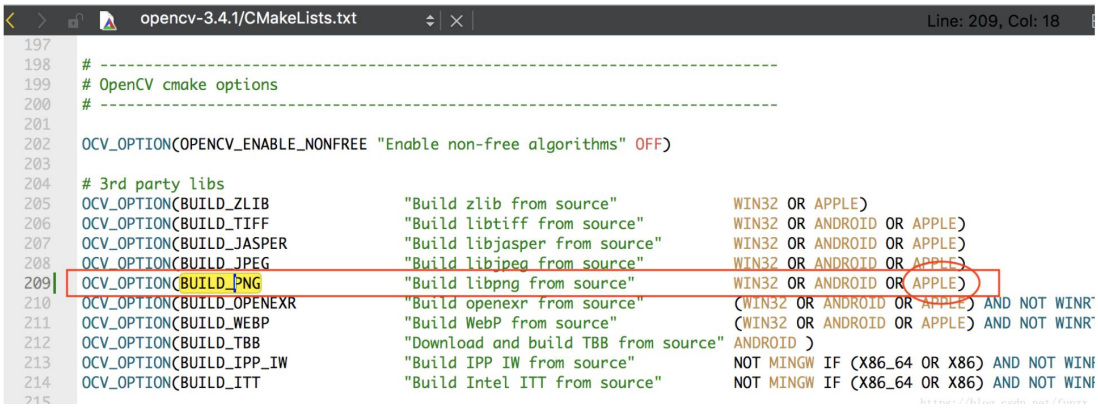
A little bug

"libpng warning: Application built with libpng-1.4.12 but running with 1.6.37"

I have checked the system's version: 1.6.37 and downloaded a matched version.

But at first, I have not found the way to link the old version in mac, and I don't know why the pycharm would link to the libpng in the mac-os.

Finally I found the answer to the questions: why does OpenCV's own libpng built-in when compiling on mac, but it depends on the system's libpng library on centOS? ? It seems that the answer to all this is to be found in the cmake configuration file. Open the OpenCV/CMakeLists.txt file and immediately found the clue (as shown below):



Did you see it? It is necessary to "build png" under Apple system! ! But "build png" is not used under centOS, it is naturally linked to the library of the system!

Now that we know that BUILD_PNG is used to control "compile built-in libpng" or "depend on system libpng", then we only need to set BUILD_PNG to ON when compiling OpenCV, then we can compile the libpng that comes with OpenCV forcibly, without being affected by libpng in the system. Impact. In this way, there will be no version mismatch problem, so the solution is:

Solution 1:
When compiling OpenCV, add parameters when executing cmake command: -D BUILD_PNG=ON
which is: cmake -D BUILD_PNG=ON -D CMAKE_BUILD_TYPE=DEBUG -D CMAKE_INSTALL_PREFIX=/usr/local/opencv-3.4.1…

Solution 2:
Replace the existing libpng library in the system with the libpng library of the same version as the libpng that comes with OpenCV. However, this method is not very elegant, and it is troublesome to operate. You need to download or compile a new version of the libpng library yourself, so it is not recommended.