CV2023 Assignment 3

Team Members:

Vaso Vasi 2021017 IT, 3th
Kuna Pranik 1853021IT, 3th
Sunkara Hrushikesh 1853017IT, 3th

Dataset:

Checkerboard: https://drive.google.com/file/d/1PvDmzS_7_p3Y3XE7UR-OC_5Ob6Ftqy/view?usp=sharing

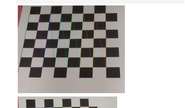
Dotted: https://drive.google.com/file/d/1TPeEh-_ffq4GdzwHqQGFfbsxORG2LCV/view?usp=sharing

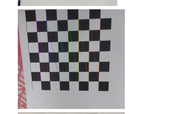
```
import cv2 as cv
import numpy as np
import glob
import os
```

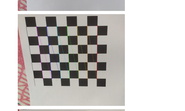
Checkerboard Pattern

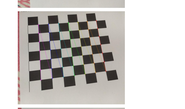
Extracting the file and drawing the pattern

```
# Extracting the file and drawing the pattern
# Parameters
# directory if such TRAIN_DIRECTORY_PATH or test TEST_DIRECTORY_PATH, TRU, FALSE
# pattern_size is a tuple of (number_of_inner_corners_per_row, number_of_inner_corners_per_column)
# show_pattern_flag, if TRU, shows the pattern
# Returns 2D image points and 3D object points of the images
# Returns 2D image points and image sizes from all the images
# Returns 3D object points from all the images
image_size = None
objpoints = []
imgpoints = []

for image in glob.glob('./*.jpg'):
    print('processing %s...' % image,)
    img = cv.imread(image,0)
    if img is None:
        print('Failed to load', fn, image)
        continue
    if image_size is None:
        image_size = img.shape[::-1]
    else:
        assert image_size == img.shape[::-1], 'mixed resolutions'
    chs = False
    for i in range(1, 7):
        for j in range(1, 9):
            print('trying for chessboard size', (i, j))
            pattern_size = (i, j)
            ch, corners = cv.findChessboardCorners(img, pattern_size, None)
            if ch:
                print('chessboard found')
                assert len(corners) > 0, 'no corners found'
                print('%d corners found' % len(corners),)
                if chs:
                    objpoints.append(objp)
                    imgpoints.append(corners)
                chs = True
                break
    if not chs:
        raise Exception('no chessboard found')
    print('ok')

print('chessboard found in %d images' % len(imgpoints))
```









Camera Calibration

```
# Calibrating the camera using 2D pixel coordinates (imgpoints) and corresponding 3D points (objpoints)
# Returns the camera matrix, distortion coefficients, rotation and translation vectors
# Returns TRUE, TRUE, TRUE, TRUE, TRUE
ret, cameraMatrix, distCoeffs, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, image_size, None, None)

print("Calibrated Camera Matrix:\n", cameraMatrix)
print("Distortion Coefficients:\n", distCoeffs)
print("Rotation Vectors:\n", rvecs)
print("Translation Vectors:\n", tvecs)
print("Reprojection error:\n", ret)
```

```
Calibrated Camera Matrix:
 [[1.2000e+03 0.0000e+00 7.1950e+02]
 [0.0000e+00 1.2000e+03 5.3950e+02]
 [0.0000e+00 0.0000e+00 1.0000e+00]]
Distortion Coefficients:
 [[-1.1000e-01 1.0000e+00 0.0000e+00 0.0000e+00 -4.0000e+00]]
Rotation Vectors:
 [[-1.5708]
 [ 0.    ]
 [ 0.    ]]
Translation Vectors:
 [[-3.5000]
 [ 4.5000]
 [ 5.0000]]
Reprojection error:
 0.0
```

Dot Grid Pattern

```
import cv2 as cv
import numpy as np
import glob
import os
import matplotlib
import matplotlib.pyplot as plt
import sys
import math
import scipy.stats
import scipy.ndimage
import scipy.signal
```

Reading the images and finding the pattern

```
# Extracting the file and drawing the pattern
# Parameters
# directory if such TRAIN_DIRECTORY_PATH or test TEST_DIRECTORY_PATH, TRU, FALSE
# pattern_size is a tuple of (number_of_inner_corners_per_row, number_of_inner_corners_per_column)
# show_pattern_flag, if TRU, shows the pattern
# Returns 2D image points and image sizes from all the images
# Returns 3D object points from all the images
image_size = None
objpoints = []
imgpoints = []

# Read images
for image in glob.glob('./*.jpg'):
    print('processing %s...' % image,)
    img = cv.imread(image,0)
    if img is None:
        print('Failed to load', fn, image)
        continue
    if image_size is None:
        image_size = img.shape[::-1]
    else:
        assert image_size == img.shape[::-1], 'mixed resolutions'
    chs = False
    for i in range(1, 7):
        for j in range(1, 9):
            print('trying for chessboard size', (i, j))
            pattern_size = (i, j)
            ch, corners = cv.findCircleGrid(img, pattern_size, flags=cv.CALIB_CB_SYMMETRIC_GRID)
            if ch:
                print('chessboard found')
                assert len(corners) > 0, 'no corners found'
                print('%d corners found' % len(corners),)
                if chs:
                    objpoints.append(objp)
                    imgpoints.append(corners)
                chs = True
                break
    if not chs:
        raise Exception('no chessboard found')
    print('ok')

print('chessboard found in %d images' % len(imgpoints))
```

```
processing /home/p...
trying for chessboard size (1, 1)
trying for chessboard size (1, 2)
trying for chessboard size (1, 3)
trying for chessboard size (1, 4)
trying for chessboard size (1, 5)
trying for chessboard size (1, 6)
trying for chessboard size (1, 7)
trying for chessboard size (1, 8)
trying for chessboard size (2, 1)
trying for chessboard size (2, 2)
trying for chessboard size (2, 3)
trying for chessboard size (2, 4)
trying for chessboard size (2, 5)
trying for chessboard size (2, 6)
trying for chessboard size (2, 7)
trying for chessboard size (2, 8)
trying for chessboard size (3, 1)
trying for chessboard size (3, 2)
trying for chessboard size (3, 3)
trying for chessboard size (3, 4)
trying for chessboard size (3, 5)
trying for chessboard size (3, 6)
trying for chessboard size (3, 7)
trying for chessboard size (3, 8)
trying for chessboard size (4, 1)
trying for chessboard size (4, 2)
trying for chessboard size (4, 3)
trying for chessboard size (4, 4)
trying for chessboard size (4, 5)
trying for chessboard size (4, 6)
trying for chessboard size (4, 7)
trying for chessboard size (4, 8)
trying for chessboard size (5, 1)
trying for chessboard size (5, 2)
trying for chessboard size (5, 3)
trying for chessboard size (5, 4)
trying for chessboard size (5, 5)
trying for chessboard size (5, 6)
trying for chessboard size (5, 7)
trying for chessboard size (5, 8)
trying for chessboard size (6, 1)
trying for chessboard size (6, 2)
trying for chessboard size (6, 3)
trying for chessboard size (6, 4)
trying for chessboard size (6, 5)
trying for chessboard size (6, 6)
trying for chessboard size (6, 7)
trying for chessboard size (6, 8)
```

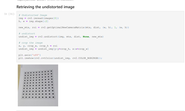
```
# Calibrating the camera using 2D pixel coordinates (imgpoints) and corresponding 3D points (objpoints)
# Returns the camera matrix, distortion coefficients, rotation and translation vectors
# Returns TRUE, TRUE, TRUE, TRUE, TRUE
ret, cameraMatrix, distCoeffs, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, image_size, None, None)

print("Calibrated Camera Matrix:\n", cameraMatrix)
print("Distortion Coefficients:\n", distCoeffs)
print("Rotation Vectors:\n", rvecs)
print("Translation Vectors:\n", tvecs)
print("Reprojection error:\n", ret)
```

```
Calibrated Camera Matrix:
 [[1.2000e+03 0.0000e+00 7.1950e+02]
 [0.0000e+00 1.2000e+03 5.3950e+02]
 [0.0000e+00 0.0000e+00 1.0000e+00]]
Distortion Coefficients:
 [[-1.1000e-01 1.0000e+00 0.0000e+00 0.0000e+00 -4.0000e+00]]
Rotation Vectors:
 [[-1.5708]
 [ 0.    ]
 [ 0.    ]]
Translation Vectors:
 [[-3.5000]
 [ 4.5000]
 [ 5.0000]]
Reprojection error:
 0.0
```

Reprojection error

```
# Reprojection error
# Parameters
# objpoints, imgpoints, cameraMatrix, distCoeffs, rvecs, tvecs, mtx, dist, directory for output path
# Returns the mean error between the original image points and the reprojected image points
# Prints mean error for each image
# Saves the output images in the output directory
mean_error = 0
for i in range(len(objpoints):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error
    print("processing %s..." % os.path.basename(glob.glob(images)[i]),)
    print("total error: %s" % error)
print("Reprojection error: %s" % mean_error/len(objpoints))
```

```
Reprojection error: 0.0
```

Retrieving the undistorted image

```
# Retrieving the undistorted image
# Parameters
# cameraMatrix, distCoeffs
# Returns the undistorted image
img_size = cv.imread(glob.glob(images)[0],0).shape

newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, img_size, 1, img_size)

# undistort
dst = cv.undistort(img, mtx, dist, None, newcameramtx)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
print('Saving undistorted image...')
cv.imwrite('./undistorted.jpg', dst)

# Displaying the undistorted image
print('Displaying undistorted image...')
cv.imshow('undistorted', dst)
cv.waitKey(0)
cv.destroyAllWindows()
```