Computer Vision SS22 Assignment 2: Camera Calibration

Felix Hamburger
Student ID: 35925
Computer Vision SS22
Computer Science Master
Ravensburg Weingarten University
Email: felix.hamburger@rwu.de

Mario Amann
Student ID: 35926
Computer Vision SS22
Computer Science Master
Ravensburg Weingarten University
Email: mario.amann@rwu.de

This paper shows in four steps how a pinhole camera model with barell distortion can be undistorted. It shows how the intrinsic and extrinsic properties of the camera can be used to undo the distortion for images taken with this camera.

1 INTRODUCTION

Some Pinhole cameras, especially cheap, uncalibrated ones introduce a distortions to an image as it can be seen in Figure 1.

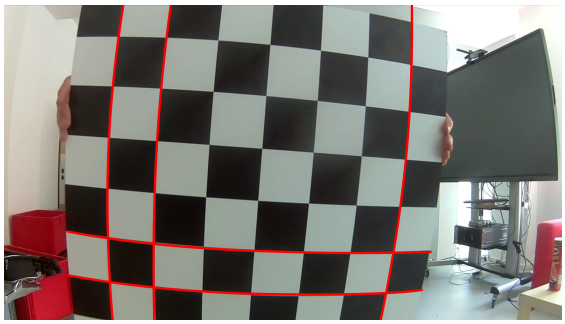

Fig. 1. Depiction of a image with a clear view of the distortion of a regular squared shaped chess board.

Thhis distortion can be be reversed by calculating and using the intrinsic and extrinsic properties of the camera. A step by step explanation will follow in the following sections.

2 PREPROCESSING

Before any preprocessing can be applied, the images have to be created from the given video. Therefore each frame of the video is saved separately. This is done with the imread function[1] of OpenCV which outputs the image as numpy array with the shape (Height, Width, Channel). By default the decoded image is stored in the BGR-format. Which means, the first channel contains the blue color channel, the second one contains the green color channel and the last the red one. In preparation for the detection of the checkerboard pattern, the images are converted to grayscale format. This convertion is made with the cvtColor function[2] of OpenCV. The convertion of colorspace is made with the following formula[3]:

$$Y = B * 0.114 + G * 0.587 + R * 0.299$$

3 CORNER DETECTION

After the preprocessing steps, the corners of the chessboard can be detected on the images. For this the OpenCV function findChessboardCorners[4] is used. As input parameters the function gets a grayscale image, the size of the pattern and already recognized corners. As size of the pattern the size 6 times 6 was chosen, because in about 500 pictures from 2000 the chessboard is recognized and the chessboard is recognized at completely different places. Since no corners are recognized before, nothing is passed at the already recognized corners. Furthermore, no flag is used and needed because the images were already converted to grayscale in preprocessing, incorrectly extracted squares were manually checked and did not occur, and there was no time pressure to speed up the detection of the corners. If corners of a chessboard are

found in the image, the corner points are added to a list of corner points from all images.

4 CAMERA CALIBRATION

With the list of corner points the intrinsic camera parameters can be estimated. The OpenCV function calibrateCamera[5] is used, which is based on the algorithm of the paper by Z.Zhang[6] from 2000. The first parameter passed is a matrix consisting of vectors of the calibration points in the coordinate space. In this case this matrix is a multi-dimensional meshgrid. Furthermore the list of corner points from the **??** section and the size of the image are passed to the function. The remaining passing parameters are passed as NoneType and no flags are set. The function returns the root mean squared reprojection error, the intrinsic camera matrix, the distortion coefficients and the rotation and translation vectors. The intrinsic camera matrix is described as follows:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The returned intrinsic camera matrix from the calibrateCamera function has the values: @@@@@@@@@@@@@@@@@@@@@

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The calibrateCamera function returns following distortion coefficients: @@@@@@@@@@@@@@@@@@@@@

$$\begin{aligned} k_1 &= s \\ k_2 &= s \\ p_1 &= s \\ p_2 &= s \\ k_1 &= s \\ k_1 &= s \end{aligned}$$

As next step the optimal camera intrinsic matrix has to be computed. For this purpose the getOptimalNewCameraMatrix function[7] is used. The function computes the optimal camera intrinsic matrix based on free scaling parameter alpha. The camera matrix, the distortion coefficients vector, the original and the new image size and the free scaling parameter alpha are passed as input. No further optional parameter are passed. The camera matrix and the distortion coefficients vector are the previously calculated values. As original and new image size the original size of the image is passed. The free scaling parameter alpha is set to one. Therefore all pixels are kept and no pixels are removed [8]. The optimal intrinsic camera matrix from getOptimalNewCameraMatrix has the following values: @@@@@@@@@@@@@@@@@@@@@

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

With the optimal intrinsic matrix the next step the undistortion can take place.

5 UNDISTORTION

The used pinhole camera distorts the image with a barell distortion as seen in figure 1.
To calculate the undistortion openCV provides the function:

```
cv.undistort(src, cameraMatrix, distCoeffs
[, dst[, newCameraMatrix]]) -> dst
```

The function takes as input values the image to be rectified, the intrinsic matrix already explained in the Section 4, the distortion coefficients and the intrinsic camerea matrix based on the scaling parameters.

The image is then cropped to the appropriate size using the region of interest roi which was also calculated in Section 4. The result of the undistortian can be viewed in Figure. 4 The image will then be set to the appropriate output dimension using the roi also obtained in chapter 2.

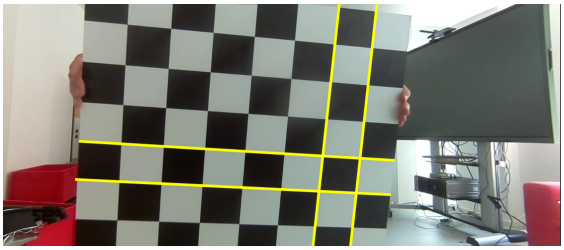

Fig. 2. Depiction of the undistorted image with lines indicating the correct dimensions of the chess board.

6 SUMMARY

With the four steps from Section 2, 3,4 and 5 the individual images of the video could be undistorted sufficiently. Finally, all the important values that were decisive for the calculation of the undistortion are given here once again. On the one hand the intrinsic camera matrix as well as the scaled and shifted camera matrix:

$$CameraMatrix = \begin{bmatrix} 1.23 & 0 & 8.96 \\ 0 & 1.23 & 5.17 \\ 0 & 0 & 1 \end{bmatrix}$$

$$ScaledCameraMatrix = \begin{bmatrix} 720.50 & 0 & 733.48 \\ 0 & 678.07 & 464.96 \\ 0 & 0 & 1 \end{bmatrix}$$

The corresponding distortion coefficients are as follows:

$$\begin{aligned} k_1 &= -0.332 \\ k_2 &= 0.142 \\ p_1 &= -0.001 \\ p_2 &= -0.000 \\ k_3 &= -0.035 \end{aligned}$$

An overall error of 0.27 over all images in the video could be achieved which leads to an output where no distortion can be seen with the human eye. The following illustrations show the change from a distorted image to an undistorted image.

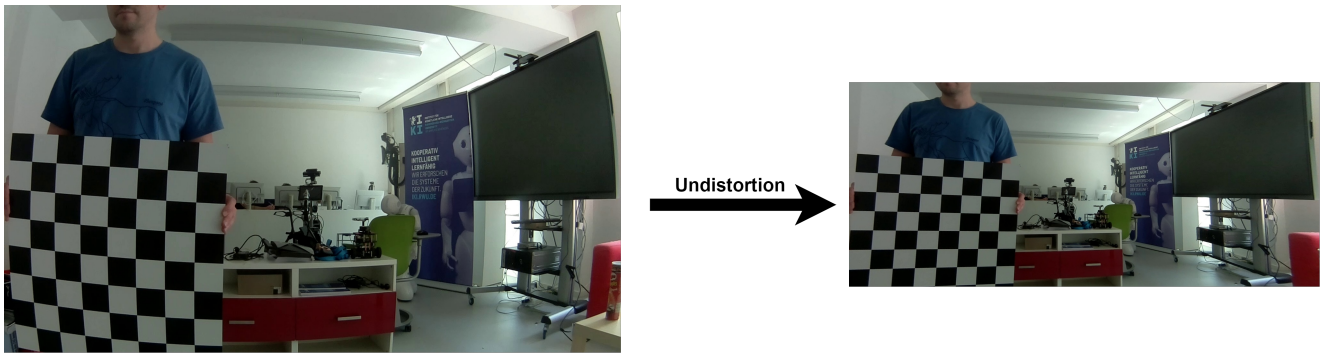

Fig. 3. Depiction of the final undistortion with the chess board located in the bottom left corner.

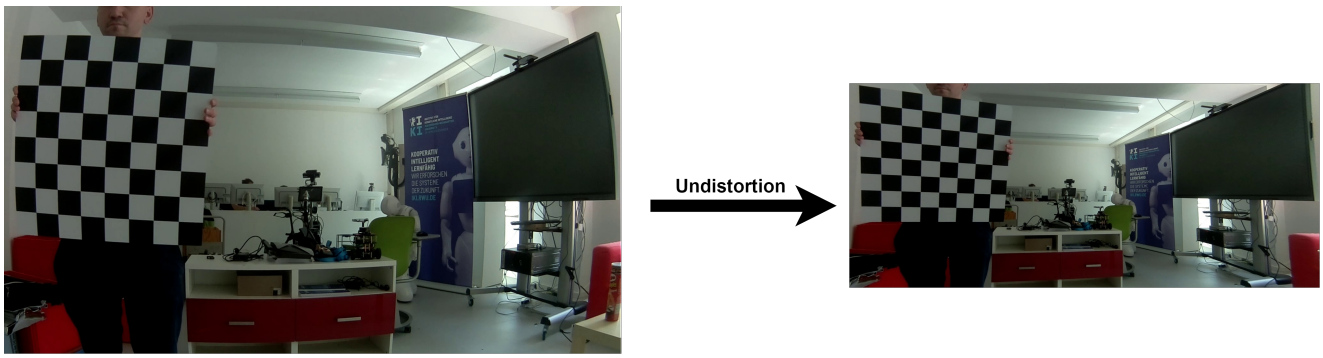

Fig. 4. Depiction of the final undistortion with the chess board located in the upper left corner.

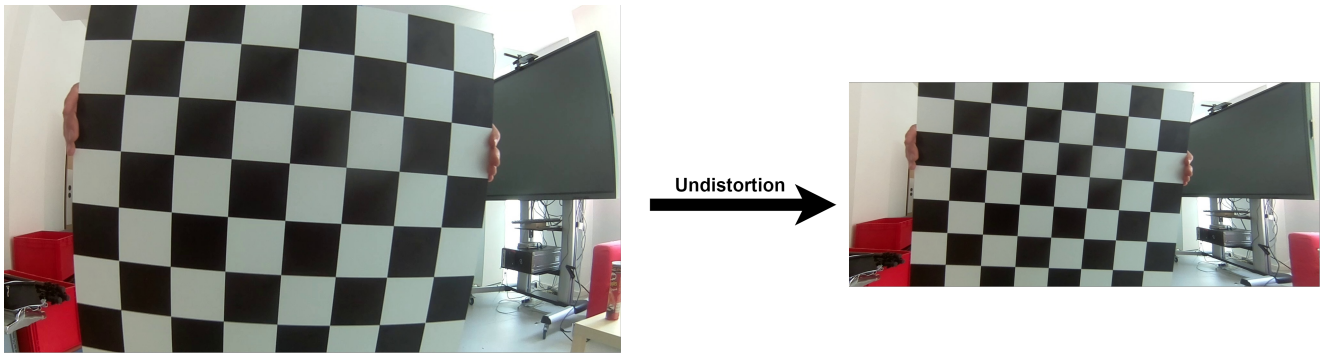

Fig. 5. Depiction of the final undistortion with the chess board located in the center.

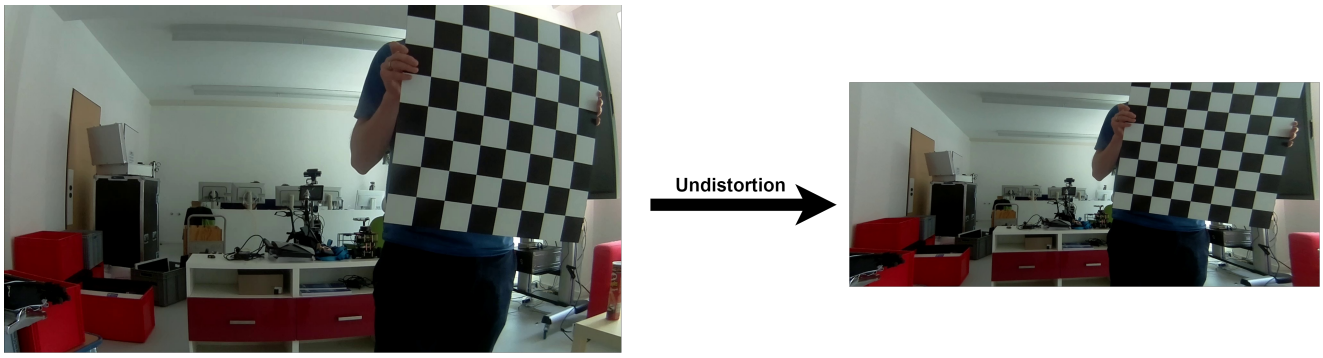

Fig. 6. Depiction of the final undistortion with the chess board located in the upper right corner.

7 CODE

```

import numpy as np
import cv2 as cv
import os
import pickle

path_used_image = "used_images/"
path_result = "result/"
path_vid_images = "vid_images/"
path_summary_results = "summary_results/"

def calibration():
    objp = np.zeros((6*6,3), np.float32)
    objp[:,:2] = np.mgrid[0:6,0:6].T.reshape(-1,2)
    objpoints = []
    imgpoints = []


    images = os.listdir(path_vid_images)
    images = [path_vid_images+i for i in images]

    count = 0
    max_it = len(images)
    for i, fname in enumerate(images):
        print("iteration1: ", i, " of ", max_it)
        img = cv.imread(fname)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        ret, corners = cv.findChessboardCorners(gray, (6,6), None)
        if ret == True:
            objpoints.append(objp)
            imgpoints.append(corners)
            cv.imwrite(path_used_image + "frame%d.jpg" % count, img)
            count +=1

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

    img = cv.imread(path_used_image + "frame0.jpg")
    h,  w = img.shape[:2]
    newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
    dst = cv.undistort(img, mtx, dist, None, newcameramtx)
    x, y, w, h = roi
    dst = dst[y:y+h, x:x+w]
    cv.imwrite(path_result + "result.jpg", dst)


    mean_error = 0
    max_it = len(objpoints)
    for i in range(len(objpoints)):
        print("iteration2: ", i, " of ", max_it)
        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

    t_error = mean_error/len(objpoints)

    with open(path_summary_results + 'objs.pkl', 'wb') as f:
        pickle.dump([t_error, ret, mtx, roi, newcameramtx, rvecs, tvecs, dist], f)

    print(t_error)
```

```
60
61 def test_calibration():
62
63     with open(path_summary_results + 'objs.pkl', 'rb') as f:
64         t_error, ret, mtx, roi, newcameramtx, rvecs, tvecs, dist = pickle.load(f)
65         print(f"{t_error}\n\n{ret}\n\n{mtx}\n\n{newcameramtx}\n\n{dist}")
66
67         images = os.listdir(path_vid_images)
68         images = [path_used_image+i for i in images]
69
70         max_it = len(images)
71         for i, image in enumerate(images):
72             print(i, " of ", max_it)
73             img = cv.imread(image)
74             h,  w = img.shape[:2]
75             dst = cv.undistort(img, mtx, dist, None, newcameramtx)
76             x, y, w, h = roi
77             dst = dst[y:y+h, x:x+w]
78             if "/" in image:
79                 image = image.split("/")[-1]
80                 image = image.split(".")[1]
81             else:
82                 image = image.split("\\")[-1]
83                 image = image.split(".")[0]
84
85
86             cv.imwrite(path_result + image + "_result.jpg", dst)
87
88 if __name__ == '__main__':
89     calibration()
90     test_calibration()
```

REFERENCES

[1] Documentary, O. imread https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56 Accessed: 2022-05-09.

[2] Documentary, O. cvtcolor https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab Accessed: 2022-05-09.

[3] Documentary, O. Color convertion rgb to gray https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html#color_convert_rgb_gray Accessed: 2022-05-09.

[4] Documentary, O. findchessboardcorners https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a Accessed: 2022-06-14.

[5] Documentary, O. calibratecamera https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d Accessed: 2022-06-15.

[6] Zhang, Z., 2000, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence,* **22**(11), pp. 1330–1334 Accessed: 2022-06-15.

[7] Documentary, O. getoptimalnewcameramatrix https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga7a6c4e032c97f03ba747966e6ad862b1 Accessed: 2022-06-15.

[8] Documentary, O. Camera calibration https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html Accessed: 2022-06-15.