RWU Hochschule Ravensburg-Weingarten University of Applied Sciences


Faculty for Electrical Engineering and Computer Science

# PROJECT REPORT: TASK - 2

## CAMERA CALIBRATION AND DEPTH ESTIMATION

*Guided by:*

**Dr. Prof. Stefan Elser**

*S*ubmitted by:

**Akshay Chobe-36316**

Master-Mechatronics

**Prasad Sonani-36202**

Master-Electrical Engineering and Embedded Systems

September 11, 2023

# Contents

# Chapter 1

# Introduction

The concepts of camera calibration and depth estimation will be investigated in this project, with a specific focus on the pinhole camera model. The goal is to accurately project a 3D environment onto a 2D image and estimate the object distance from the camera.

In the initial phase, the video file "videoWW1_calibration.avi" along with the distortion coefficients and three images (distorted, undistorted, and undistorted plus cropped versions) will be utilized. The estimation of the intrinsic matrix will be performed using recordings of a checkerboard pattern. An accurate estimate will be obtained by analyzing various positions of the checkerboard. The problem of lens distortion, where image objects may not appear as anticipated due to geometric distortions, will be addressed using checkerboard recordings and the undistortion technique.

In the second section, the focus will be on depth estimation. All intrinsic camera parameters will be calculated using camera calibration process on only one frame of the "videoHD1.avi" video file. Also, we got accurate intrinsic camera matrix(K) from it.Which is use to find depth of Object images. In the next part, the input for depth estimation will rely on undistorted images. Distances will be estimated using the geometry of similar triangles, utilizing the known dimensions of the checkerboard pattern. Finally, the verification of depth estimation is conducted.

# Chapter 2

# Undistortion

## 2.1 Distortion Coefficient

The Distortion Coefficient is found by performing Camera calibration using Opencv. For this, a few frames of the video "videoWW1 calibration.avi" are taken, specifically selecting frames where the checkerboard appears in different positions in each image.

```
checkerboard_size = (7, 7)

objp = np.zeros((np.prod(checkerboard_size), 3), np.float32)
objp[:,:2] = np.mgrid[0:checkerboard_size[0], 0:checkerboard_size[1]].T.reshape(-1, 2)

objpoints = []
imgpoints = []

for i in range(12):
    img = cv2.imread(f'C:/Users//Goal_1_Dataset/frame_{i}.jpg')
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)

    if ret == True:
        objpoints.append(objp)
        imgpoints.append(corners)
        cv2.drawChessboardCorners(img, checkerboard_size, corners, ret)

ret, k,Distrotion_Coefficient, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
```

Figure 2.1: Code Snippet for Estimating the Distortion Coefficients

The aim is to have camera calibration performed using a checkerboard pattern with "checkerboard_size", which is considered here as "7 X 7". A loop is set up to iterate through a series of images (in this case, 12 images) containing the checkerboard pattern. The corners of the checkerboard in the grayscale image, which is converted using "cv2.cvtColor", are detected using the "cv2.findChessboardCorners" function. Then, the detected corners are rendered on

the image using "cv2.drawChessboardCorners". After processing all the images, the camera calibration is performed using the function "cv2.calibrateCamera". The required distortion coefficients, along with the camera matrix (K), are returned by the function.

| Parameters | Value |
|---|---|
| k1 | -1.2290072759022985 |
| k2 | -18.622917428554544 |
| s1 | 0.00794332465289252 |
| s2 | -0.019843624835973368 |
| k3 | 136.1991269948859 |

Table 2.1: Distortion Coefficient

The objective of the code snippet below is to undistort an image using previously determined camera calibration settings where "Img.shape[1::-1]", returns the width and height of the image in reverse order, is then used to extract the shape of the image.

```
image=cv2.imread(f'C:/Users/VISHAL.000/Downloads/Camera Calibration and Depth Estimation-20230506/Dataset/frame_12.jpg')
image_shape = image.shape[1::-1]
newcameramtx, _ = cv2.getOptimalNewCameraMatrix(mtx, dist,image_shape, 1)

# Undistort image
undistorted_img = cv2.undistort(img, mtx, dist, None, newcameramtx)

x1 = 100
y1 = 40
x2 = 1150
y2 = 900

# Crop the image
cropped_image = undistorted_img[y1:y2,x1:x2]
```

Figure 2.2: Code Snippet for undistortion

For given image, the optimal camera matrix is calculated using inbuilt function "cv2.getOptimalNewCameraMatrix". The new camera matrix (newcameramtx) and an optional mask that denotes valid pixels are returned by the function. The image is then subjected to the undistortion using "cv2.undistort". Next, specific coordinates are defined to crop the image. The variables x1, y1, x2, and y2 represent the top-left and bottom-right coordinates of the rectangular region of interest (ROI) to be cropped. Finally, the undistorted_img is cropped using the defined coordinates and stored in "cropped_image".

## 2.2 Undistortion Results

Figure 2.3: Image 1: Distorted, Undistorted, Undistorted+Cropped

Figure 2.4: Image 2: Distorted, Undistorted, Undistorted+Cropped

Figure 2.5: Image 3: Distorted, Undistorted, Undistorted+Cropped

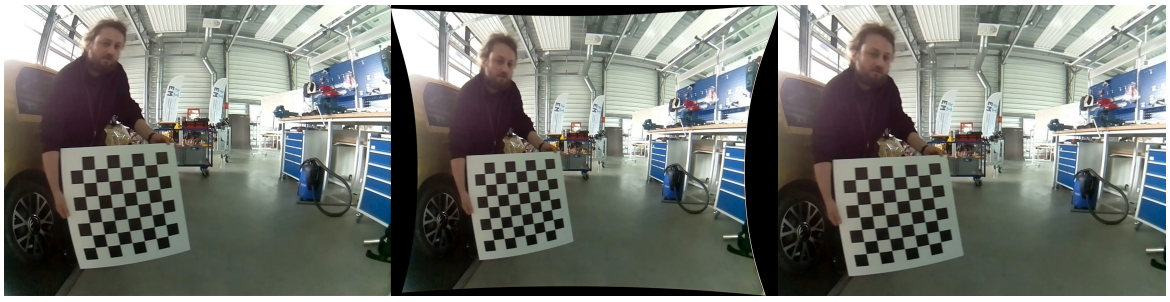

Figure 2.6: Image 4: Distorted, Undistorted, Undistorted+Cropped

# Chapter 3

# Depth Estimation

## 3.1 Intrinsic Camera Parameters

The intrinsic camera matrix, distortion coefficient, rotation vector, and translation vector were obtained after performing camera calibration with only one image "frame ID : 1334" from the video "videoHD1.avi". Additionally, the corners of the 7x7 checkerboard were captured and stored in the array "corners" using "cv2.findChessboardCorners".

Figure 3.1: Frame : 1334

After the camera calibration, the intrinsic matrix (K) is obtained. This matrix (K) is used in all frames to calculate the distance of object from the camera. Specifically, when converting the 2D coordinates into 3D coordinates of any point.

$$K = \begin{bmatrix} 1237.0027729241529 & 0 & 654.592485468364 \\ 0 & 1240.7005990933174 & 494.5558916358658 \\ 0 & 0 & 1 \end{bmatrix}$$

All intrinsic camera parameter values are mentioned in the table below. We can get the focal length of camera by $f = \frac{fx+fy}{2}$. The focal length obtained is 1238.85 pixel. Here, all values of parameters are in pixel.

| Parameter | Value |
|---|---|
| fx | 1237.0027729241529 |
| fy | 1240.7005990933174 |
| cx | 654.592485468364 |
| cy | 494.5558916358658 |

Table 3.1: Intrinsic Camera Parameters

## 3.2 Approach to find Depth

We have the physical dimension of checkerboard as 50.6 cm X 50.6 cm. and no. of cells are 8 X 8. So, every cell of the checkerboard is 6.325 cm long.

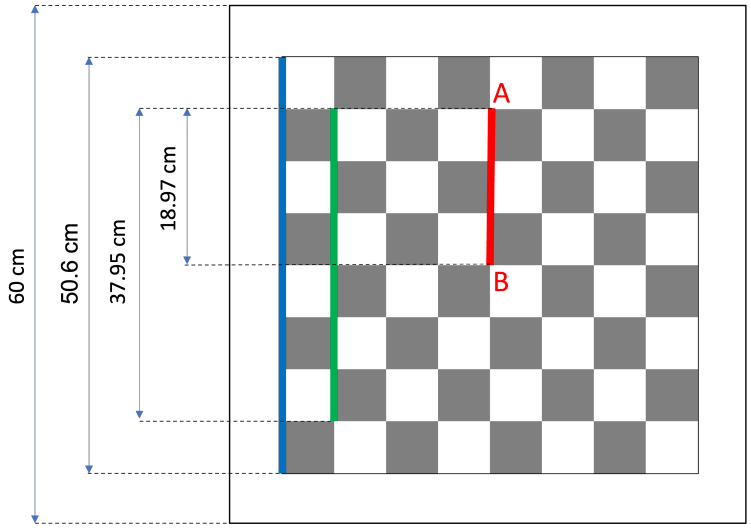


Figure 3.2: Checkerboard Size

Here, the 3D world coordinate and 3D camera coordinate are same. So we have to consider value of translation vector (t) = 0 and $R = E_3$. Another scenario we have considered that the checkerboard is perpendicular to camera. So, the image plane and object plane is parallel to each other as shown in Figure 3.3. As you can see in frame:1334 that the checkerboard is placed on the object. So, to find the distance of object we find the distance of checkerboard from its center. The Figure 3.3 is representation of camera projection in 3D world.

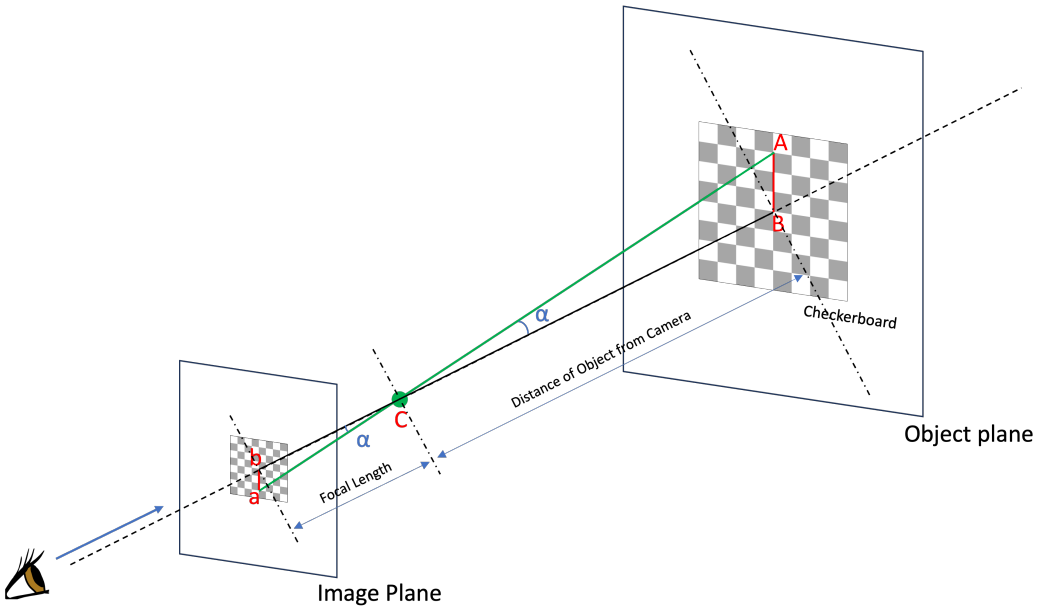


Figure 3.3: Ray Diagram

First image plane points 'a' and 'b' are converted into 3D with following equation;

$$L = \left\{ \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} \in \mathbb{R}^3; \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \mathbf{z} \cdot \mathbf{R}^t K^{-1} \begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} - \mathbf{R}^t \cdot \vec{\mathbf{t}} \text{ and } \mathbf{z} \in \mathbb{R} \right\}$$

here, R = $E_3$ and $\vec{\mathbf{t}} = \vec{\mathbf{0}}$.

so finally we get the following equation:

$$L = \left\{ \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} \in \mathbb{R}^3; \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \mathbf{z}\mathbf{K}^{-1} \begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} \text{ and } \mathbf{z} \in \mathbb{R} \right\}$$

we make function 'Convert_to_3d' of this equation, which gave the 3d coordinates of the point. Here, we update '1' in pixel coordinate and changed its shape from 2x1 to 3x1. It is called as homogeneous pixel coordinate. After that it was multiplied with inverse of camera Intrinsic matrix(k) and scalar(z).

```
def Convert_to_3D(pixel_coordinate,K, scalar):
    K_inverse = np.linalg.inv(K)
    pixel_homogeneous = np.append(pixel_coordinate, 1).reshape(3, 1)
    ray_direction = np.dot(K_inverse, pixel_homogeneous)
    Point__3D = scalar*ray_direction.flatten()
    return Point__3D
```

Figure 3.4: function 'Convert_to_3d'

The scalar value defines the new position of that point in 3D projection of camera. Here, we change 2d points 'a' and 'b' into 3d points 'P' and 'Q'. The points 'P' and 'Q' are scalar(z) distance away from camera(C).The value of 'z' should be positive or negative. In below image we consider the positive value for best projection from side view and easy understanding.

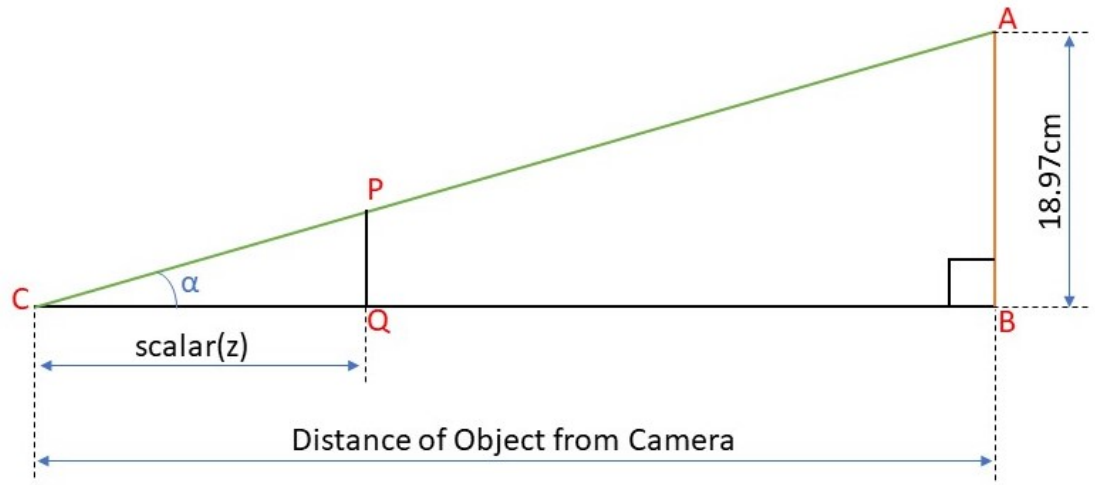


Figure 3.5: Right Side view of Figure 3.3

The △ABC and △PQC are symmetric triangles.

$$\triangle ABC \sim \triangle PQC$$

So,

$$\frac{PQ}{AB} = \frac{QC}{BC} = \frac{CP}{CA}$$

BC = Object Distance,

here considering,

$$\frac{PQ}{AB} = \frac{QC}{BC}$$
$$\therefore \quad BC = QC \cdot \frac{AB}{PQ}$$

```python
def calculate_midpoint(point1, point2):
    point1 = np.array(point1)
    point2 = np.array(point2)
    midpoint = (point1 + point2) / 2
    return midpoint
```

Figure 3.6: Midpoint calculation function

```python
C=np.array([0,0,0])
corner_points = np.array([corners[0],      # Corners of outer Edge from 7x7 chekkerboard.
                          corners[6],
                          corners[42],
                          corners[48]])

b = np.mean(corner_points, axis=0)         # Finding the center(b) of checkerbord in image plan.
Q = Convert_to_3D(b,camera_matrix,1)

a = calculate_midpoint(corners[0],corners[6])
P = Convert_to_3D(a,camera_matrix,1)

PQ=calculate_distance(Q,P)
QC=calculate_distance(Q,C)
AB= 0.18975                        # The value is in meter.
BC = QC*AB/PQ
print("Distance:",BC)
```

```
Distance: 4.718249889712598
```

Figure 3.7: Distance calculation

We got very close result of '4.718m' for the frame:1334 for which the actual answer given was '4.7m' and that's why we concluded that intrinsic matrix 'K' found for this frame is the most accurate one. We continued to calculate object distance for two other frames with the same method. Every time the values of array 'corners' are changed but the matrix 'K' is remained the same for all frames.

## 3.3 Depth Estimation Results

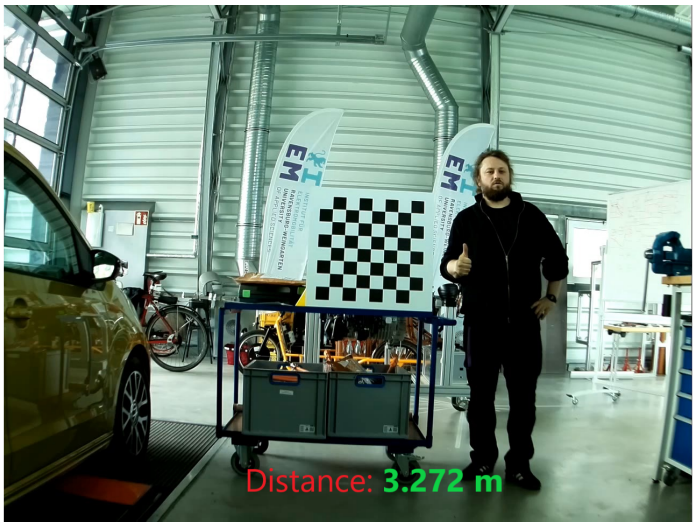


Figure 3.8: Frame ID: 900


Figure 3.9: Frame ID: 1334


Figure 3.10: Frame ID: 1900

## 3.4 Verification

This code segment is used to verify distance estimation accuracy. The code segment computes the 2D coordinates of two points in the picture and compares it with the distance of ground truth value for the same points.

```
def calculate_distance(point1, point2):
    point1 = np.array(point1)
    point2 = np.array(point2)
    distance = np.linalg.norm(point2 - point1)
    return distance
```

Figure 3.11: Distance calculation Function

```
Corner_0=(Convert_to_3D(corners[0],camera_matrix,Distance))*100
Corner_6=(Convert_to_3D(corners[6],camera_matrix,Distance))*100
Real_width =37.95
calculated_width=calculate_distance(Corner_0,Corner_6)
Error=calculated_width-Real_width
print("Real Width of 7X7 checkerbord(cm):",Real_width)
print("Calculated width of 7X7 checkerbord(cm):",calculated_width)
print("Error in cemtimeter:",Error)
```

```
Real Width of 7X7 checkerbord(cm): 37.95
Calculated width of 7X7 checkerbord(cm): 38.67455001995976
Error in cemtimeter: 0.7245500199597572
```

Figure 3.12: Code for Verification

The width of 7 X 7 checkerboard is 37.95 cm as shown in Figure 3.2. The corner points of the top edge were chosen and the distance between them was found with our algorithm, and those two points were converted into 3d. The found distance between those Two 3D points is further compared with the original distance 37.95 cm. Here we found that for frame:1334 we got an error of almost 2%. As shown in Figure 3.12 . The reason behind choosing frame:1334 is that we got best camera matrix from it.

$$\text{Algorithm Error} = \frac{0.72455}{37.95} \times 100 = 1.9\%$$

# Chapter 4

# Conclusion

Analysing the results we can say that symmetric triangle method is very accurate to find approximate distance of object in picture with checkerboard. However after verification we encountered an error of 1.9% in the result. Few parameters are responsible for affecting these results such as slight lens distortion in the image we used for finding matrix 'K'. Also, we assumed that the checkerboard is perpendicular to the camera plane, but due to some human errors it is not always possible. To overcome this issue we have to consider rotation of checkerboard along X-axis and Y-axis, while calculating the corners of checkerboard in the image plane.