3D Computer Vision (2023/24)

Exercise 1

Submitted by Group 25:

- Asmita Sengupta
- Juan Pablo Pino Bravo
- Mahmoud Abuhussien
- Vaishnavi Shirbhate

Upload: 08.11.2023 (11:30)

Deadline: 21.11.2023 (23:59)

Please hand in a single **.zip** file named according to the pattern "**groupXX_exerciseX**" (e.g. group00_exercise9). The contents of the .zip should be as follows:

- folder named according to the pattern groupXX_exerciseX
  - **.ipynb** file
  - **.html** export of .ipynb with all the outputs you got
  - **data** folder containing necessary files to run the code

I.e.

1. **unzip** the provided exerciseX.zip file
2. **rename** folder "exerciseX" according to the pattern "groupXX_exerciseX"
3. **solve** tasks inside .ipynb file
4. **export** notebook as .html (File > Download as > HTML)
5. **zip** folder groupXX_exerciseX
6. **submit** groupXX_exerciseX.zip

Theory

T1. Camera with lenses

(a)

Given a camera with a distance L between the film and the lens, derive the mathematical relationship (formula) between the height H_o of the object in front of the camera and the height H_i of its image. Additionally, explain the intuition behind the relationship. Assume a thin lens.

Solution

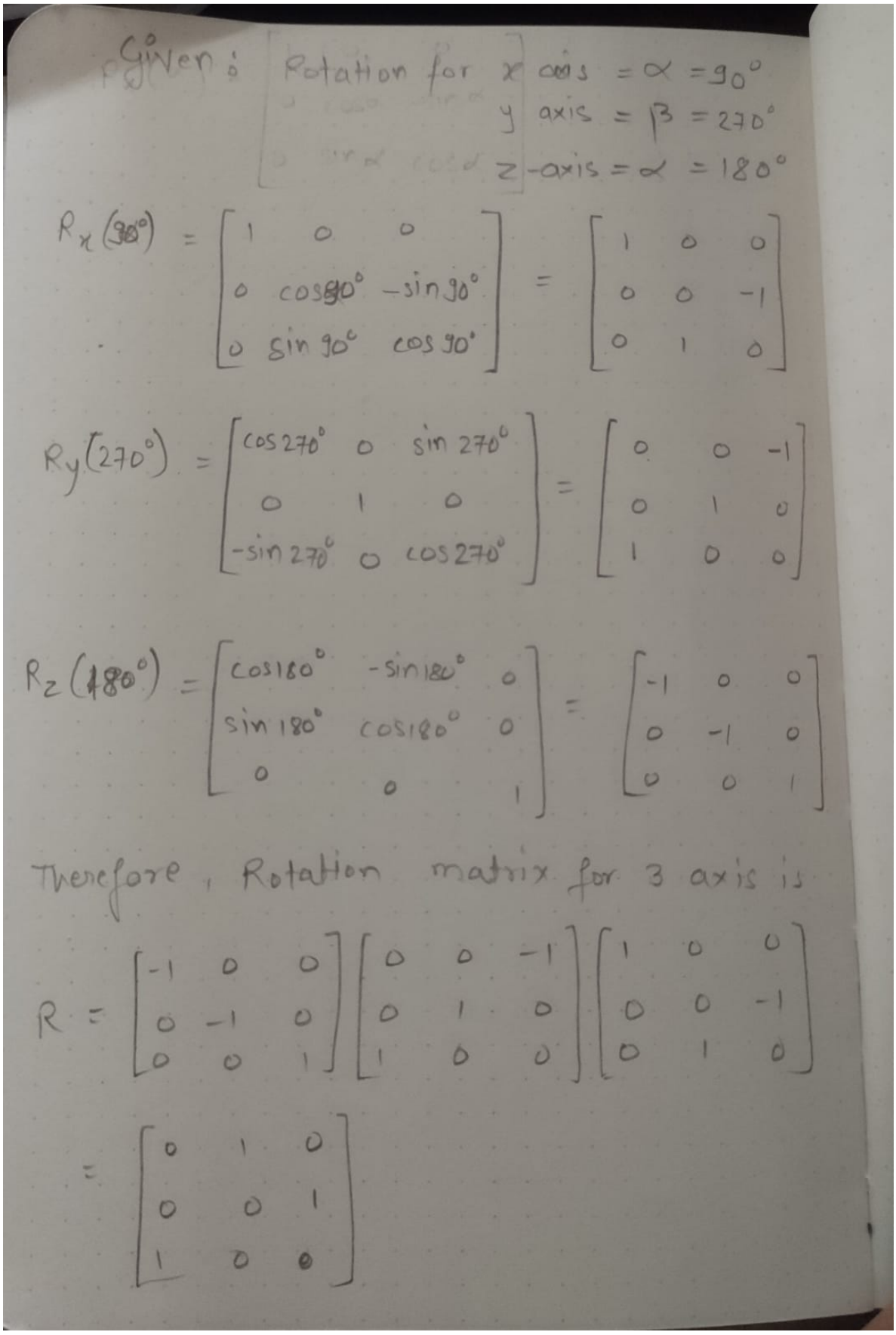
- Similar triangles: $\frac{y'}{D'} = \frac{y}{D} \rightarrow \frac{H_i}{L} = \frac{H_o}{D}$
- therefore $\frac{H_i}{H_o} = \frac{L}{D}$
- Interpretation:
  - For the same distance to the lense D, a bigger object (H_o) has a bigger image (H_i)
  - (OR) The further the object of height H_o is away from the lense (D), the smaller is its image H_i

T2. Rotation

An Object is roated around the x-axis by 90°, then around the y-axis by 270°, and finally around the z-axis by 180°.

(a)

Derive the 3D Roation Matrix that executes the same transformation.
Hint: the given values lead to 'nice' numbers.



(b)

Assume the object is a sphere with a radius of 3.5. Explain how the radius will change after the transformation.

No, the radius of the sphere does not change after transformation. It remains 3.5

(c)

What is the rotation matrix that transforms the object back to its original orientation?
Hint: this should be very short

It will be the transpose of the rotation matrix.

T3. Transformation Chain

(a)

Why are homogeneous coordinates used for transforming points between coordinate systems?

Homogeneous coordinates are used for transforming points between coordinate systems because they make all the transformations linear which can be represented by a single matrix.

(b)

Describe the transformation chain for mapping a point from the world coordinate system to the pixel coordinate system of an intrinsically and extrinsically calibrated camera. Use formulas and explain the intermediate steps in words.

We need to find a coordinate transformation from world to pixel coordinates.

- $M = (X_s, Y_s, Z_s)$: point in world coord.
- $M = (x_s, y_s, z_s)$: point in camera coordinates.
- $t = CO$: translation vector between origins.

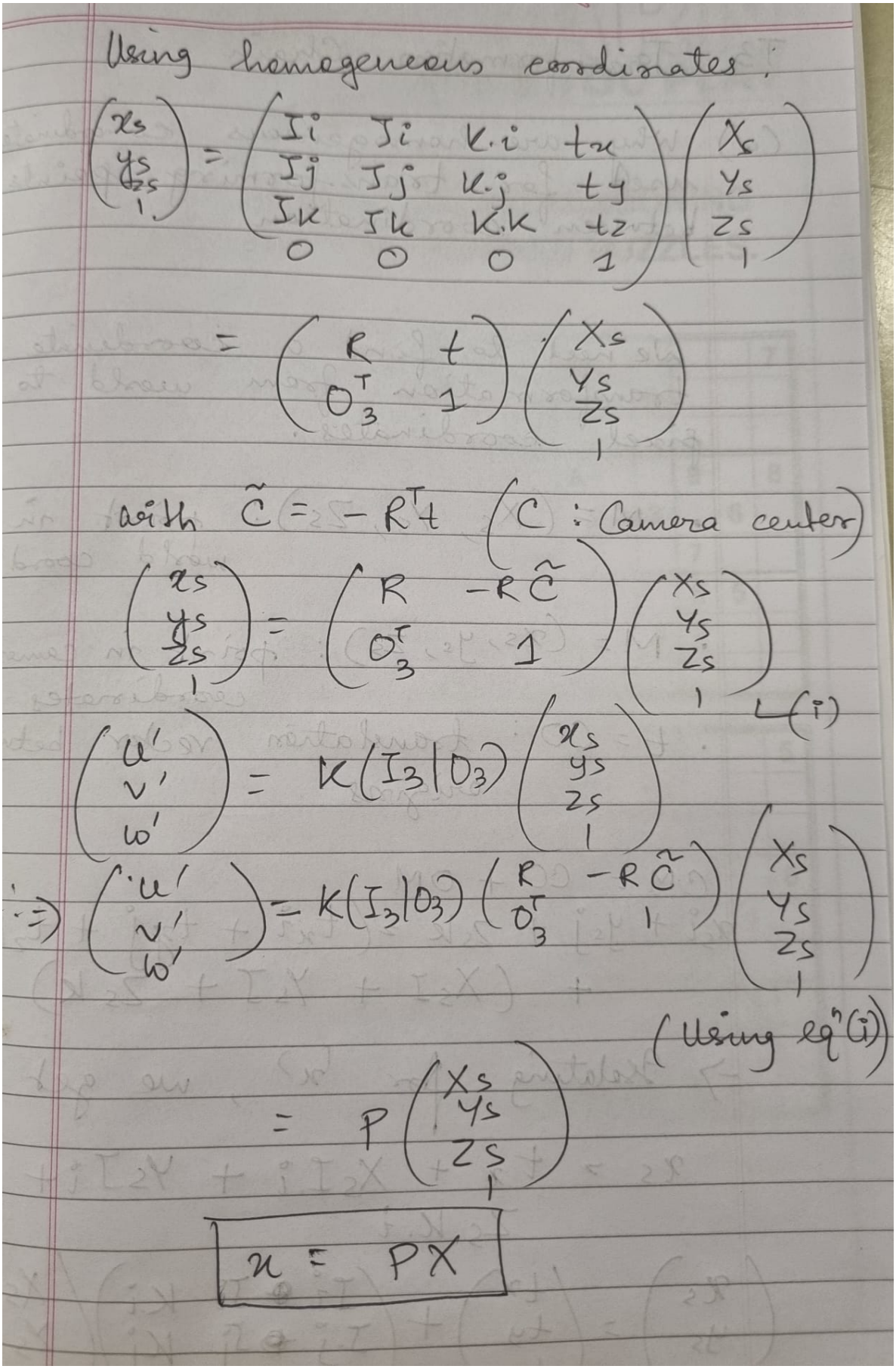
$$CM = CO + OM$$

$$x_s i + y_s j + z_s k = (t_x i + t_y j + t_z k) + (X_s I + Y_s J + Z_s K)$$

$\Rightarrow$ Isolating for x, we get:

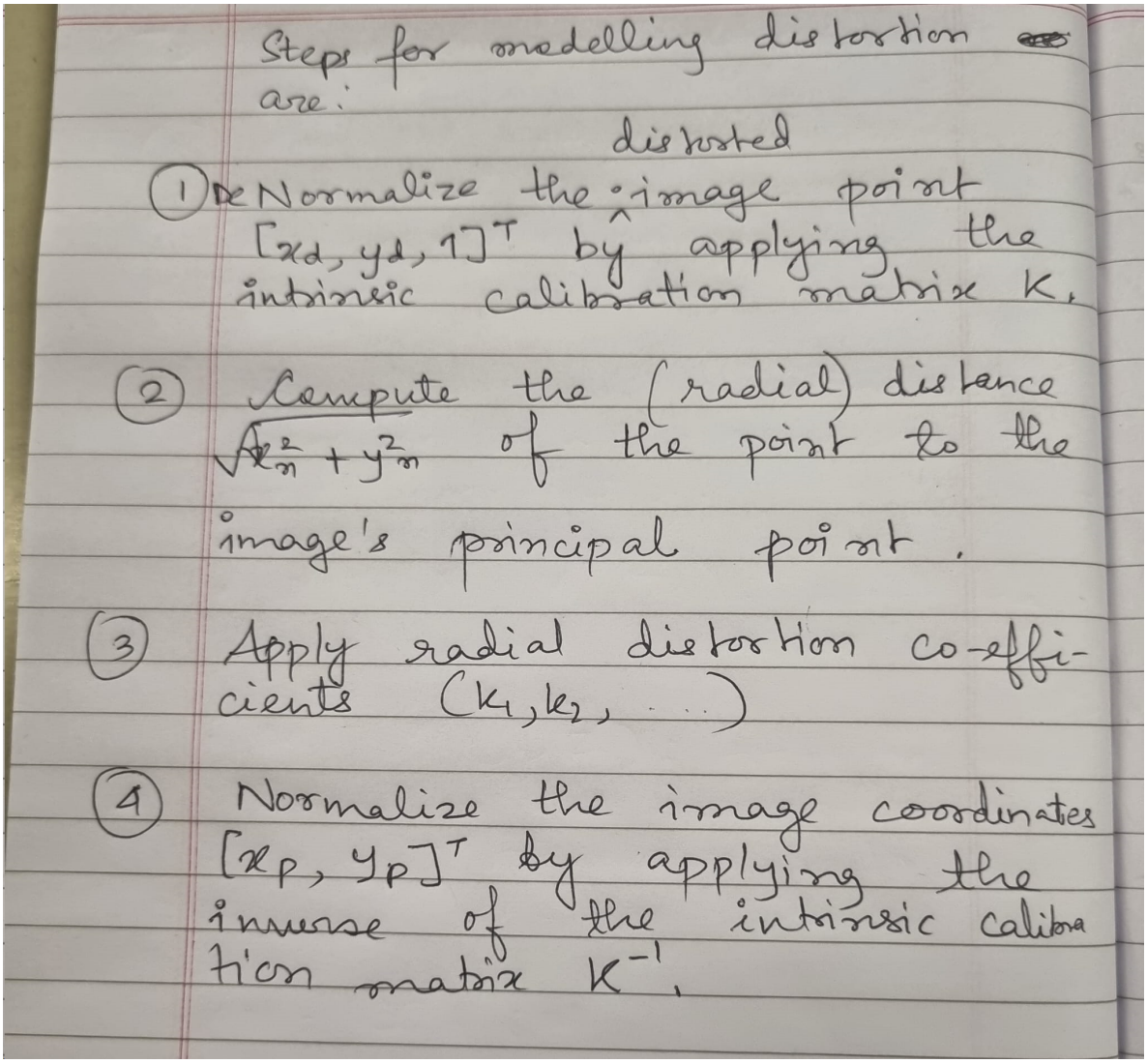
$$x_s = t_x + X_s I.i + Y_s J.i + Z_s K.i$$

$$\begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} + \begin{pmatrix} I.i & J.i & K.i \\ I.j & J.j & K.j \\ I.k & J.k & K.k \end{pmatrix} \begin{pmatrix} X_s \\ Y_s \\ Z_s \end{pmatrix}$$



(c)

Describe the steps for modelling distortion.



Implementation

I1. Distortion Modelling

The **./data/** directory contains images of a chessboard that were used for calibrating a camera with high radial distortion. The results of the calibration (intrinsics of the camera and extrinsics for each board) are stored in **./data/calib.mat**.

In [ ]:
```
'''You can add all your imports here'''
import os

import numpy as np
import cv2 as cv
import scipy.io as io
from PIL import Image
import matplotlib.pyplot as plt
```

(a)

Write a function that does the following:

- Draw any number of 2D pixel points onto an image

The inputs should be:

- A single image
- An Array of 3D world points
- Color of the points to be drawn

The output should be:

- An image with points drawn onto it

In [ ]:
```python
def draw_points(image, points, color):
    '''Draws points on an image'''
    for point in points:
        # print("Center:", point)
        # print("Center (tuple):", (point[0], point[1]))
        image = cv.circle(image, (int(point[0]), int(point[1])), 3, color
    return image
```

(b)

Write a function **project_points** that does the following:

- Convert 3D world points to 2D image points
- As an option: model radial distortions (using k1, k2, k5)

The inputs should be:

- Array of 3D world points
- Camera intrinsics
- Camera extrinsics
- Distortion coefficients (if needed)

The output should be:

- Array of 2D pixel coordinates

In [ ]:

```python
def project_points(points, K, R, t, dist_coef = None):
    '''Projects a 3D point into the image plane'''
    # Apply rotation
    X = R@points.T
    # print ("points after rotation", X.shape)

    # Apply extrinsics
    X = X + t
    # print("point after translation", X)

    # Apply instrinsics
    X = K@X

    # print("point after intrinsics", X)

    X /= X[2,:]
    X = X[:2, :].T

    if dist_coef is None:
        return X
    else:
        # Extract distortion coefficients
        k1 = dist_coef[0]
        k2 = dist_coef[1]
        k3 = dist_coef[2]
        k4 = dist_coef[3]
        k5 = dist_coef[4]

        fx = K[0,0]
        fy = K[1,1]


        # Calculate radial distance from center (480 / 2, 640 /2)
        x_c = 640/2
        y_c = 480/2
        x = (X[:,0] - x_c) / fx
        y = (X[:,1] - y_c) / fy
        r = np.sqrt(x**2 + y**2)

        # Calculate distorted point
        x_distort = x * (1 + k1*r**2 + k2*r**4 + k5*r**6)
        y_distort = y * (1 + k1*r**2 + k2*r**4 + k5*r**6)

        X[:,0] = x_distort * fx + x_c
        X[:,1] = y_distort * fy + y_c

        return X
```

(c)

Write a function **project_and_draw** that does the following:

- Execute **project_points**
- Execute **draw_points**
- Save the result as an image file

The inputs should be:

- The data that is necessary to run your functions
- Needs to run on **all images** with a single call

In [ ]:
```
def project_and_draw(imgs, points, color, K, R, t, dist_coef = None):
  for i in range(len(imgs)):
    projected_points = project_points(points[i], K, R[i], t[i], dist_coef
    image = imgs[i].copy()
    image = draw_points(image, projected_points, color)
    if dist_coef is None:
      cv.imwrite("results/no_distortion/{0}.jpg".format(i), image)
    else:
      cv.imwrite("results/distortion/{0}.jpg".format(i), image)
```

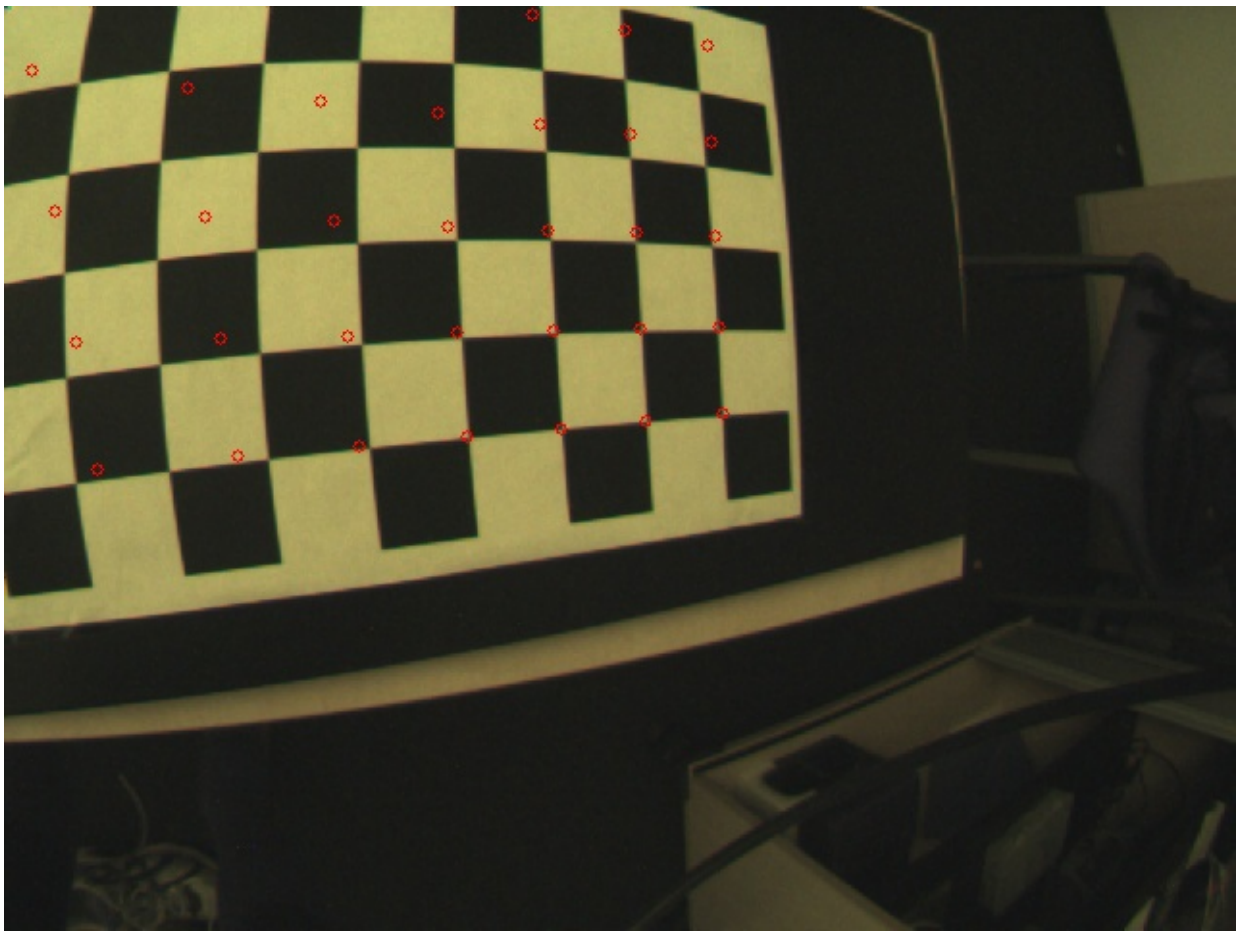
(d)

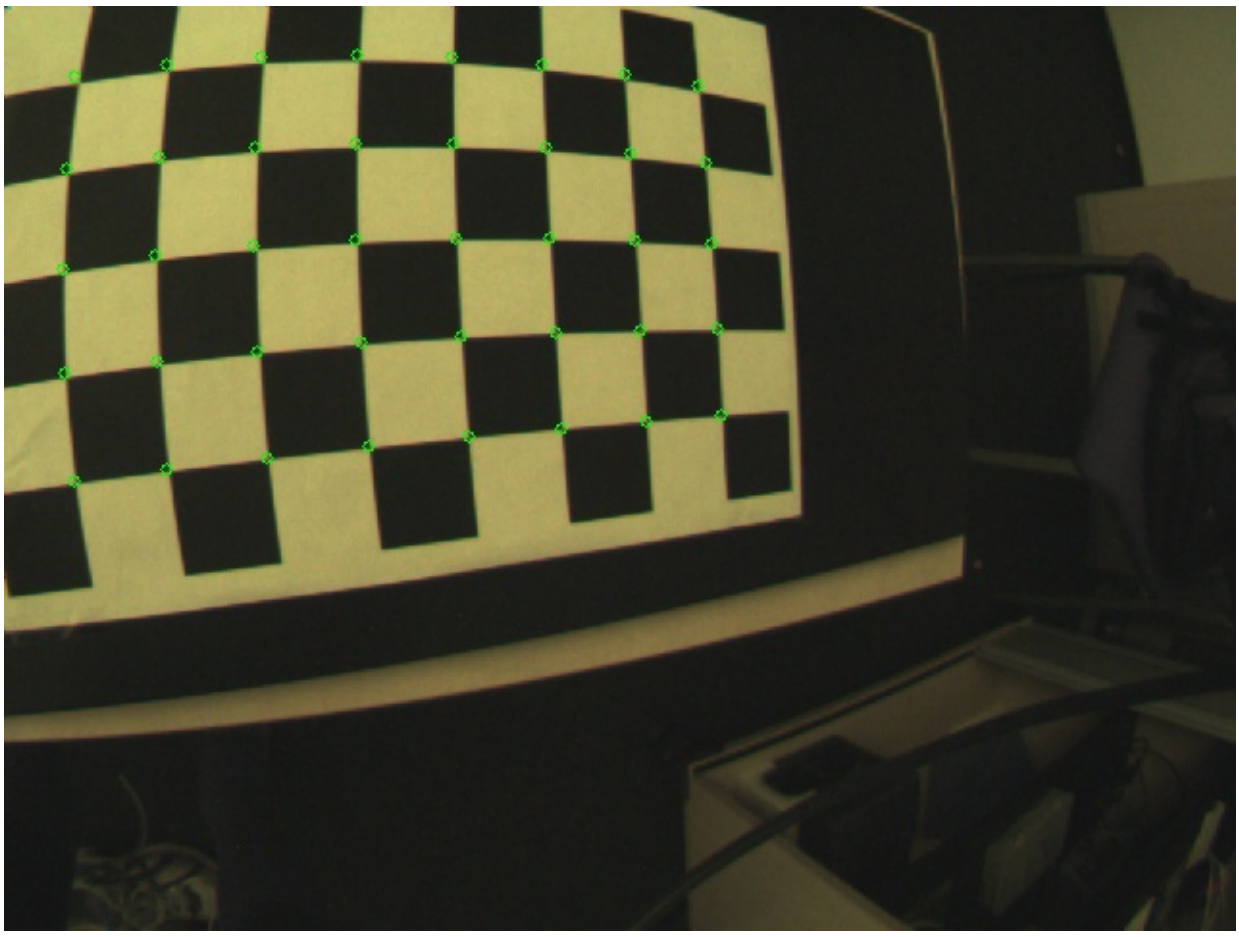
Run your **project_and_draw** function once without and once with distortion modelling. Then display the following:

- Your output for 00000.jpg **without any distortion modelling** in **red**
- Your output for 00000.jpg **with added radial distortion** (k1, k2, and k5) in **green**

In [ ]:

```
os.makedirs(name=f'results/no_distortion', exist_ok=True)
os.makedirs(name=f'results/distortion', exist_ok=True)

base_folder = './data/'

# Load the data
# There are 25 views/or images/ and 40 3D points per view
data = io.loadmat('./data/calib.mat')

# 3D points in the world coordinate system
x_3d_w = data['x_3d_w'] # shape=[25, 40, 3]

# Translation vector: as the world origin is seen from the camera coordin
t_vecs = data['translation_vecs'] # shape=[25, 3, 1]

# Rotation matrices: converts coordinates from world to camera
rot_mats = data['rot_mats'] # shape=[25, 3, 3]

# Five distortion coefficients
dist_coef = data['distortion_params'] # shape=[5, 1]

# K matrix of the cameras
k_matrix = data['k_mat'] # shape=[3, 3]

# Images corresponding to the 3D points
imgs_list = [cv.imread(base_folder+str(i).zfill(5)+'.jpg') for i in range
imgs = np.asarray(imgs_list) # shape=[25, 480, 640, 3]

#Call project_and_draw twice: once without and once with distortion model
project_and_draw(imgs, x_3d_w, (0, 0, 255), k_matrix, rot_mats, t_vecs)
project_and_draw(imgs, x_3d_w, (0, 255, 0), k_matrix, rot_mats, t_vecs, d

without_distortion = Image.open('results/no_distortion/0.jpg')
display(without_distortion)

with_distortion = Image.open('results/distortion/0.jpg')
display(with_distortion)
```





In [ ]: