LAB 2: Correcting Perspective distortion

Oluwadara Adedeji

November 11th, 2022

Table of Contents

1. Introduction

Distortions are regarded as optical errors/aberration, which are usually caused by aberration in the optical lens. Specifically, perspective distortion arises when the lens is not perfectly aligned parallel to the imaging plane. This causes distortions in the image. This also causes the shapes not to be preserved. For example, circles become ellipses. In addition, the distance between two points is not preserved, hence making parallel lines to converge. Furthermore, angles are not preserved in the image. These effects are categorized as foreshortening, distance and pose respectively. Other forms of distortions are barrel and pincushion distortion.

As a result of perspective distortion, we get a false information of depth and pose of the image. This false information can mislead our model when we use these extracted features from a distorted image. It also distracts us from important information from the image. Hence, the need to correct this distortion. This is an important process in computer vision, photography and more.

1. Background

The H matrix transforms the distorted (D) coordinates to processed (P) coordinates. The relationship is given in equation 1. The relationship also makes use of the homogenous coordinates.

$$P = H * D - - - - eqn(1)$$

For the algorithm, the processed and distorted coordinates extracted are used to form a matrix X, and this matrix is inverted, and a dot product is done on P and X inverse, as shown in equation 2.

$$H = X^{-1}P - - - - eqn(2)$$

To expatiate on the concepts of matrix inverse, the inverse of a matrix is another matrix which when it is multiplied with it, it gives the multiplicative identity. In linear algebra, we use different methods to get the inverse of a matrix. One is the Cramer's rule method, and another is by Gaussian elimination. The inverse of a matrix is important in computation as it helps in solving linear equations, and this is important in many computations done by computers. For this exercise, the linear algebra function in numpy (np.linalg) is used to get the inverse. Unlike the Gaussian elimination which can find the inverse of a non-square matrix, the function in numpy only works on matrices which are square matrices. Further information on how linear algebra inverse works in numpy can be found in the documentation [1]. Hence, the need to ensure in this lab that every matrix for which the inverse was computed was a square matrix.

To get the correct perspective distortion, we use H^{-1} to map coordinates in the processed array to corresponding coordinates in the distortion array, then use the information in that distortion coordinates in our processed array. The overall operation is given in equation 3. Further explanation of this algorithm is given in section 3.3.

$$D = H^{-1}P ---- eqn(3)$$

3 Part 1:

Goal: Overall goal is to correct perspective distortion in an image. Subgoals are to extract the image coordinates for distorted and the processed images, generate H inverse and perform perspective distortion.

3.1 Extract distorted and processed coordinates

First, the coordinates are extracted for the distorted coordinates. The coordinates are extracted using Paint. This is done such that we pick 4 coordinates which fall within a trapezoid. The acute angles are assumed to be perpendicular (on the processed image). This means that for the processed image, as stated earlier, the acute angles are extended such that they are perpendicular to each other. It should be noted that the coordinates from Paint are in (col, row), so they are swapped to (row, col), which follows the image representation in the Numpy. Figure 1 shows how the coordinates were selected. The red line shows the distortion coordinates(c,d), and the green line shows the processed coordinates (a,b).

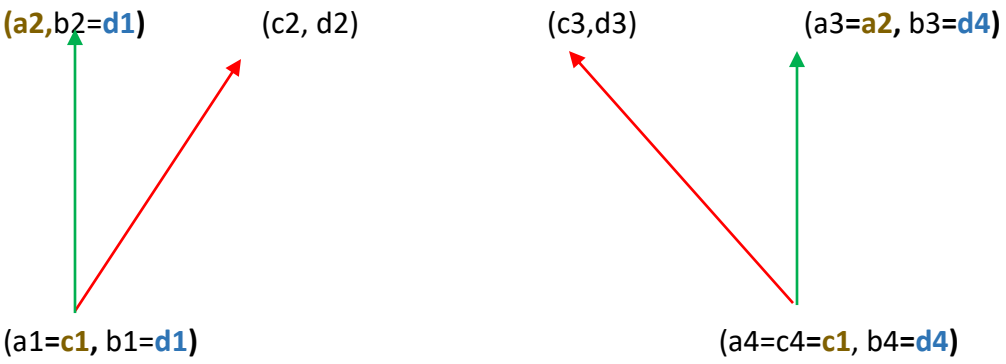


Figure 1: Illustration of the processed (a,b) and distorted (c,d) coordinates of the image

3.2 Generate H inverse

For the algorithm, first an array X is formed from the distorted and processed coordinates. This array is then inverted using np.linalg, and it is confirmed that the array is indeed converted when it is multiplied with itself to give an 8 x 8 identity matrix. The H matrix is then generated by multiplying this inverse with the 8 x 1 processed coordinates (as in equation 2). This is then converted to a list and 1 is appended to the list of H arrays to be able to reshape to 3 x 3 matrix

(i.e 9 values required). The reshape is done and that gives the H matrix. Again, we use np.linalg to get the inverse. This inverted H is then returned.

3.3 Correct Perspective distortion

The general overview of the algorithm is to correct perspective distortion by using the H inverse. First, the processed array is set to all zeros and to the same size as the image array. Then, I looped through each pixel in the processed array. Let's say the row is *i* and the column is *j.* For every *i , j* in the processed array, the H inverse was used to get *c* and *d* in the distorted array. Then the RGB values at *c, d* (of the distorted image) are placed into the *i , j* in the processed image array. However, not all pixel values of *c, d* are within the image range. Hence, a conditional was used to only consider *c, d* values greater than or equal to 0 and less than the row and column sizes respectively.

The algorithm was tested with two images and the following results were generated. The zoom in and out is also tried with test 2 image.

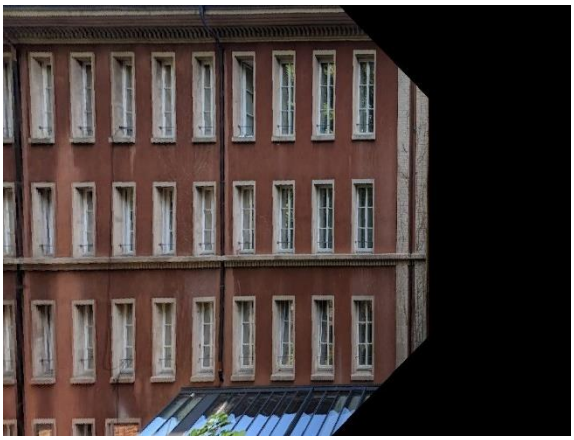
Test 1:

Distorted image

Processed image

Test 2

Distorted image



Processed image (zoomed out)

Processed image (zoomed in, although image rotated, showing that the ordering of coordinates also matters)

Observation

It is observed that some pixels are lost, and these are black in the processed image. Also, the quality of the processed coordinates determines how good the processed image will be. In the case where the processed coordinates used to generate the H matrix do not form a perfect rectangle or square, the correction result won't be as good. It was also observed that to achieve a Zoom in effect, the coordinates picked need to be close. As shown in the zoomed in image, the coordinate of reference also matters, and determines the order of the coordinates. Starting with a coordinate far away from the right (in the second image) or bottom(in the first image) can cause the processed image to rotate.

4.Conclusion

In this lab, I have been able to perform perspective distortion correction. I realized how important it is to generate quality processed coordinates (and distortion coordinates), which will then determine how accurate the H matrix will be in generating corresponding distortion coordinates for a particular processed coordinate and vice versa. For the processed coordinates to be of high quality, they have to form a perfect rectangle or square. I was further exposed to array manipulation using Numpy.

5. References

[1] "Numpy linear algebra inverse documentation." https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html (accessed Nov. 11, 2022).

6. Appendix

```
# -*- coding: utf-8 -*-
#!/Users/Hp/AppData/Local/Programs/Python/Python311

"""
Created on Fri Nov 11 21:03:20 2022

@author: oadedeji
"""

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from numpy.linalg import inv
```

```
#To run, use python filename.py

def generate_H_inverse(processed_cord, distorted_cord):
    """



    Parameters
    ----------
    processed_cord : (row,col)
    distorted_cord : (row,col)

    Returns
    -------
    h_inv : 3 by 3 matrix

    """
    # Extract the X matrix
    X = np.array([[distorted_cord[0],distorted_cord[1],1,0,0,0, -
processed_cord[0]*distorted_cord[0], -
processed_cord[0]*distorted_cord[1]],[0,0,0,distorted_cord[0],distorted_cord[1],1
,-processed_cord[1]*distorted_cord[0],-
processed_cord[1]*distorted_cord[1]],[distorted_cord[2],distorted_cord[3],1,0,0,0
, -processed_cord[2]*distorted_cord[2], -
processed_cord[2]*distorted_cord[3]],[0,0,0,distorted_cord[2],distorted_cord[3],1
,-processed_cord[3]*distorted_cord[2],-
processed_cord[3]*distorted_cord[3]],[distorted_cord[4],distorted_cord[5],1,0,0,0
, -processed_cord[4]*distorted_cord[4], -
processed_cord[4]*distorted_cord[5]],[0,0,0,distorted_cord[4],distorted_cord[5],1
,-processed_cord[5]*distorted_cord[4],-
processed_cord[5]*distorted_cord[5]],[distorted_cord[6],distorted_cord[7],1,0,0,0
, -processed_cord[6]*distorted_cord[6], -
processed_cord[6]*distorted_cord[7]],[0,0,0,distorted_cord[6],distorted_cord[7],1
,-processed_cord[7]*distorted_cord[6],-processed_cord[7]*distorted_cord[7]]])

    #Get the inverse of X and comfirm that the product of inverse and X gives
identity matrix
    distorted_inv = inv(X)
    np.allclose(np.dot(X, distorted_inv), np.eye(8))

    #Get the H matrix
    H = np.dot(distorted_inv, processed_cord)

    #Convert the array to a list
    h_reshaped = list(H)
```

```
    #Append 1 to the list so as to have 9 items(to be able to reshape to 3 by 3)
    h_reshaped.append(1)

    #Convert back to an array and reshape to 3 by 3
    h_reshaped = np.array(h_reshaped)
    h_reshaped = h_reshaped.reshape(3,3)

    # Get the inverse of the resultant H matrix and return it
    h_inv = inv(h_reshaped)

    return h_inv




def perspective_distortion(image_array, processed_array,h_inv):
    """


    Parameters
    ----------
    image_array : Image array
    processed_array : Processed array
    h_inv : H inverse

    Returns
    -------
    processed_array : resultant processed array

    """

    # Loop through all the pixels in the processed array
    #For every i and j in the processed array, get the corresponding c and d in
the distortion array
    #Place the RGB value at that c,d pixel value in the i,j position
    #Return the processed array

    for i in range(processed_array.shape[0]):
        for j in range(processed_array.shape[1]):
            a_b = np.array([i,j,1])
            c_d = np.dot(h_inv,a_b)
            c = int(c_d[0]/c_d[2])
            d = int(c_d[1]/c_d[2])
```

```
            if (c >= 0 and c < image_array.shape[0]) and (d >= 0 and d <
image_array.shape[1]):
                processed_array[i,j] = image_array[c,d]

    return processed_array

#Load the image, extract the array, and set the processed array as zero
my_image1 = Image.open("PC_test_1.jpg")
image_array_1 = np.asarray(my_image1, np.float64)
processed_array_1 = np.where(image_array_1 <= 256, 0,0)

my_image2 = Image.open("PC_test_2.jpg")
image_array_2 = np.asarray(my_image2, np.float64)
processed_array_2 = np.where(image_array_2 <= 256, 0,0)

#Get the coordinates distorted = [c1,d1,c2,d2,c3,d3....], likewise for processed
(a,b)
processed_cord_1 = [433,137,345,137,345,477,433,477]
distorted_cord_1 = [433,137,345,179,345,425,431,477]

processed_cord_2 = [540,473,324,473,324,621,540,621]
distorted_cord_2 = [500,469,343,473,324,621,540,617]

#Get the H inverse
h_inv_1 = generate_H_inverse(processed_cord_1,distorted_cord_1)
h_inv_2 = generate_H_inverse(processed_cord_2,distorted_cord_2)

#Perform perspective distortion correction
processed_array_1 = perspective_distortion(image_array_1,
processed_array_1,h_inv_1)
processed_array_2 = perspective_distortion(image_array_2,
processed_array_2,h_inv_2)

#Save the processed image in that directory
Image.fromarray(processed_array_1.astype(np.uint8)).save("processed_test_1.jpg")
Image.fromarray(processed_array_2.astype(np.uint8)).save("processed_test_2.jpg")

#Show that operation has been completed
print("Done")
```