Lab 6

John Munyi

03/10/2018

Table of Contents

Introduction

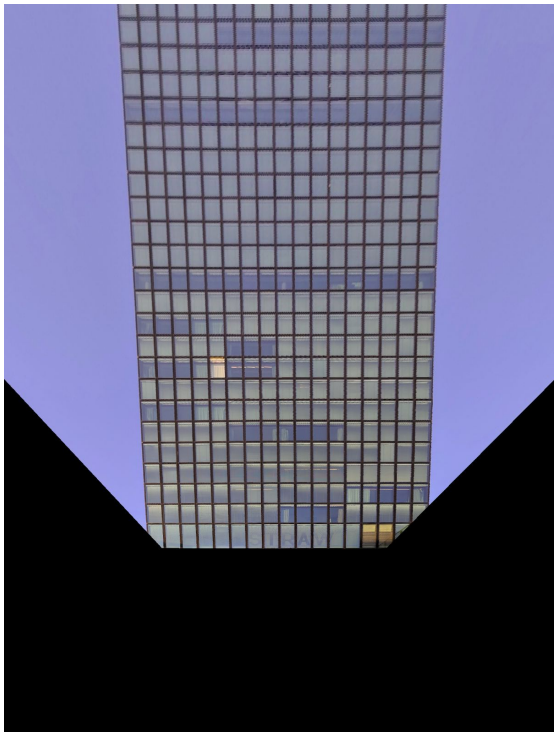
This assignment covers the concept of perspective distortion correction; this is where you have an image that appears to have 2 lines or points meet at infinity. A good example of this is how railway lines appear to meet at the horizon, but as you get closer to the horizon the meeting point keeps moving further. Distortion correction is an important aspect since we need to at times apply it so that the user can see images differently, this depends with circumstances.

1. Perspective distortion correction

The perspective phenomena is well explained when parallel edges appear to meet at infinity. Other notable examples are: circles become eclipse and where angles are involved the angles are not preserved.

The aim of this lab is to correct perspective distortion so that as the user observes the images from different camera angles the distortion is reduced or eliminated.

1.1 Results



The 2 pictures above show corrected perspectives for the images samples given. It's clearly notable that in first picture the perspective in completely reduced to parallel lines. In the second one due to the complexity of horizontal perspective I wasn't able to achieve 100% parallel look.

1.2 Discussion

In this section I will give a brief overview of how to perform distortion correction. At the heart of this process is a matrix H which has to be solved by picking arbitrary points in the distorted image and correct (perceived) image.

This is followed by stepping through a black image, which should be of equal or a little less in dimensions to the distorted image. We then do a bilinear interpolation of every point and update the pixels in the blank black image.

The corrected images shape may differ depending on where and how the points we pick are located in the distorted image.

2. Conclusion

To conclude, a few observations can be made.

- The direction of the perspective will dictate how you pick your points. Depending on if its horizontal or vertical perspective.
- The selection of the points on the distorted image should resemble a trapezium while the points on the corrected image should be square or rectangular.

3. Appendix

distortion_correction.py

```
import numpy as np
import bmp_io_c
import math
from numpy.linalg import inv

def readImage(image):
  rows, cols, distorted_pixels = bmp_io_c.input_bmp_c(image)
  return rows, cols, distorted_pixels

# rowsimage1, colsimage1, distorted_pixels1 = readImage("image_pd_01.bmp")
# print(rowsimage1, colsimage1)

rowsimage1, colsimage1, distorted_pixels1 = readImage("image_pd_02.bmp")
print(rowsimage1, colsimage1)

# # select four point in the fixed image
# c1, d1 = 381, 1057
# c2, d2 = 1193, 1051
# c3, d3 = 319, 1173
# c4, d4 = 1271, 1171

# # select four point in the fixed image
# a1, b1 = 381, 1057
```

```
# a2, b2 = 1193, 1051
# a3, b3 = 381, 1173
# a4, b4 = 1193, 1173

# select four point in the fixed image
c1, d1 = 390, 519
c2, d2 = 1017, 132
c3, d3 = 408, 932
c4, d4 = 1005, 1248

# select four point in the fixed image
a1, b1 = 390, 519
a2, b2 = 1023, 558
a3, b3 = 408, 932
a4, b4 = 1005, 1248



# solving for H
points = [[c1, d1, 1, 0, 0, 0, -c1*d1, -a1*d1],
        [0, 0, 0, c1, d1, 1, -b1*c1, -b1*d1],
        [c2, d2, 1, 0, 0, 0, -c2*d2, -a2*d2],
        [0, 0, 0, c2, d2, 1, -b2*c2, -b2*d2],
        [c3, d3, 1, 0, 0, 0, -c3*d3, -a3*d3],
        [0, 0, 0, c3, d3, 1, -b3*c3, -b3*d3],
        [c4, d4, 1, 0, 0, 0, -c4*d4, -a4*d4],
        [0, 0, 0, c4, d4, 1, -b4*c4, -b4*d4],]

Inv_points = np.linalg.inv(points)
ab = [a1, b1, a2, b2, a3, b3, a4, b4]
H = np.matmul(Inv_points, ab)

# add a one at the end of h
h = H.tolist()
h.append(1)
H = np.array([h]).reshape(3,3)
```

```
H_inverse = np.linalg.inv(H)


# black image
blank_image = np.zeros([3, rowsimage1, colsimage1])
bmp_io_c.output_bmp_c("blank_image.bmp", blank_image)

#Bi-linear inrerpolation
def lininterp (distorted_pixels, x, y):

  if x < 0 or x > (colsimage1 - 1):
     return 0, 0, 0

  if y < 0 or y > (rowsimage1 - 1):
     return 0, 0, 0

  x1 = math.floor(x)
  x2 = math.ceil(x)
  y1 = math.floor(y)
  y2 = math.ceil(y)


  if x1 == x2 or y1 == y2:
     return distorted_pixels[0, y1, x1], distorted_pixels[1, y1, x1],
distorted_pixels[2, y1, x1]

  mat = np.asarray([[1, x1, y1, x1*y1], [1, x1, y2, x1*y2], [1, x2, y1, x2*y1], [1,
x2, y2, x2*y2]])

  mat = np.transpose(np.linalg.inv(mat))

  coordinates = np.matmul(mat, np.asarray([[1], [x], [y], [x * y]]))

  R = (float(coordinates[0]) * distorted_pixels[0, y1, x1]) + (float(coordinates[1]) *
distorted_pixels[0, y1, x2]) \
  + (float(coordinates[2]) * distorted_pixels[0, y2, x1]) + (float(coordinates[3]) *
```

```
distorted_pixels[0,y2, x2])

  G = (float(coordinates[0]) * distorted_pixels[1, y1, x1]) + (float(coordinates[1]) *
distorted_pixels[1, y1, x2]) \
  + (float(coordinates[2]) * distorted_pixels[1, y2, x1]) + (float(coordinates[3]) *
distorted_pixels[1, y2, x2])

  B = (float(coordinates[0]) * distorted_pixels[2, y1, x1]) + (float(coordinates[1]) *
distorted_pixels[2, y1, x2]) \
  + (float(coordinates[2]) * distorted_pixels[2, y2, x1]) + (float(coordinates[3]) *
distorted_pixels[2, y2, x2])

  return R, G, B

for y in range(rowsimage1):
  for x in range(colsimage1):
      vector = np.matmul(H_inverse, np.array([x,y,1]))
      a = vector[0]/vector[2]
      b = vector[1]/vector[2]

      r,g,b = lininterp(distorted_pixels1, a, b)
      blank_image[0,y,x] = r
      blank_image[1,y,x] = g
      blank_image[2,y,x] = b

bmp_io_c.output_bmp_c("blank_image2.bmp", blank_image)
```