ECE 661 Homework 2

Michael Goldberg

September 3 2024

1 Task 1

1.1 Undistorting Using Point to Point Comparisons

The goal of this task is to take two images and undistort them to remove projective and affine distortion. One strategy taken is to collect points of interest that we know to be straight and orthogonal to one another in World3D and map those points in the image to a rectangle. Below these images can be seen, with their corresponding faces to be undistorted.

(a) Image1

(b) Image2

(c) Image1 with Dimensions

(d) Image2 with Dimensions

It can be seen that the desired image output will be proportional to the real-life dimensions as depicted in the images, which are the corners used for mapping the homography. Therefore, to find the homography, I found the pixel coordinates of the corners of the object of interest and mapped them to:

$$[0,0],[width,0],[width,height],[0,height] \tag{1}$$

where $width$ and $height$ are those widths and heights specified in the dimensioned images.

This mapping can be done for each corner in the following manner:

Since $x' = Hx$, with

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \tag{2}$$

then

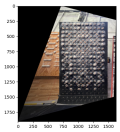
$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{3}$$

For clarity, I will refer to $\frac{u}{w}$ as x and $\frac{v}{w}$ as y.

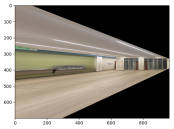
Expanding out, rearranging, and dividing by w (as we only care about the ratios in homogeneous coordinates (HC), we are left with:

$$\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{pmatrix} = \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{4}$$

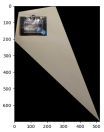
It can be clearly seen that there are only 2 equations and 8 unknowns for each point, therefore a minimum of 4 points must be collected to solve for the homography matrix coefficients. However, once these are solved, the homography can be applied to the entire image and the resulting images can be seen below. Note that the output raster size was adaptively created by recognizing the width and height of the four **image** corners (not the corners of the ROI).



(a) P2P Board



(b) P2P Corridor



(c) P2P Test Image1

(d) P2P Test Image2

1.2 2 Step Homography

This task is to first remove projective distortion by finding the image's vanishing line and mapping it to the line at infinity, and then the second step is to remove affine distortion by mapping the dual degenerate conic to the dual degenerate conic at infinity.

Step 1: Removing projective distortion

The homography that maps the vanishing line to the line at infinity is:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{pmatrix} \tag{5}$$

Therefore, all we need to do is find the vanishing line. This can be easily done by taking the four corners of the ROI we found and finding the cross products to get the edge lines. Parallel edge lines can then be crossed together

to produce the vanishing points, and these 2 vanishing points can be crossed to form the vanishing line:

$$l1 = \begin{pmatrix} u1 \\ v1 \\ 1 \end{pmatrix} \times \begin{pmatrix} u2 \\ v2 \\ 1 \end{pmatrix} \; l2 = \begin{pmatrix} u2 \\ v2 \\ 1 \end{pmatrix} \times \begin{pmatrix} u3 \\ v3 \\ 1 \end{pmatrix} \; l3 = \begin{pmatrix} u3 \\ v3 \\ 1 \end{pmatrix} \times \begin{pmatrix} uv \\ v4 \\ 1 \end{pmatrix} \; l4 = \begin{pmatrix} u4 \\ v4 \\ 1 \end{pmatrix} \times \begin{pmatrix} u1 \\ v1 \\ 1 \end{pmatrix} \tag{6}$$

Then,

$$vp1 = l1 \times l3 \tag{7}$$

and

$$vp2 = l2 \times l4 \tag{8}$$

Finally,

$$line_{vanishing} = vp1 \times vp2 = \begin{pmatrix} l_1 \\ l_2 \\ l3 \end{pmatrix} \tag{9}$$

Applying the above homography

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{pmatrix} \tag{10}$$

will then produce an image with no projective distortion. Now, the task is to transform this image into one without affine distortion as well. This can be done through the dual degenerate conic. The formula for $\cos\theta$ can be written as:

$$\cos\theta = \frac{l^T C_\infty^* m}{\sqrt{(l^T C_\infty^* l)(m^T C_\infty^* m)}} \tag{11}$$

We want angles that are 90 degrees in World2D to be 90 degrees in our corrected images, so $\cos\theta = 0$. Therefore, we just want to look at the numerator and set it equal to zero:

$$\cos\theta_{numerator} = l'^T H H^{-1} C_\infty^{*'} H^{-T} H^T m' = 0 \tag{12}$$

Simplifying,

$$\cos\theta_{numerator} = l'^T H C_\infty^* H^T m' = 0 \tag{13}$$

If we then assume that:

$$H = \begin{pmatrix} A & t \\ 0^T & 1 \end{pmatrix} \tag{14}$$

we get the following relation:

$$\begin{pmatrix} l'_1 & l'_2 & l'_3 \end{pmatrix} \begin{pmatrix} AA^T & 0 \\ 0^T & 0 \end{pmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0 \tag{15}$$

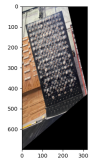
Setting $S = AA^T$, we get the following relation, which can be Single Value Decomposed to find s_11 and s_12:

$$\begin{pmatrix} l'_1 & l'_2 \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} \\ s_{12} & 1 \end{pmatrix} \begin{pmatrix} m'_1 \\ m'_2 \end{pmatrix} = 0 \tag{16}$$

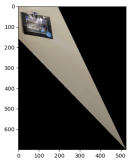
Therefore, we have 1 equation and 2 unknowns, which means we need 2 pairs of orthogonal lines to find the values of S and therefore create the homography of the form:

$$H = \begin{pmatrix} A & 0 \\ 0^T & 1 \end{pmatrix} \tag{17}$$

The results can be seen below. First, are the purely projective distortion corrections:



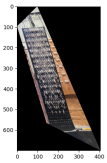
(a) 2Step Board



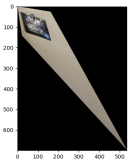
(b) 2Step Test Image1

(c) 2Step Test Image2

Next, the correction for affine as well:



(a) 2Step Board



(b) 2Step Test Image1

(c) 2Step Test Image2

1.3 1 Step Homography

The one-step approach allows both purely projective and affine to be corrected for in a single operation. This exploits the homography hierarchy, as all affine homographies are also projective homographies by nature.

Looking back at the dual degenerate conic at infinity formula:

$$\cos\theta_{numerator} = l'^T H C^*_\infty H^T m' = 0 \tag{18}$$

We can set up the following dual degenerate conic at infinity that eliminates the projective and affine distortions by observing a more general case:

$$\begin{pmatrix} l'_1 & l'_2 & l'_3 \end{pmatrix} \begin{pmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{pmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0 \tag{19}$$

We can then rewrite and rearrange to form the following relationship:

$$\frac{l'_1 m'_1 a}{-l'_3 m'_3} + \frac{l'_2 m'_1 + l'_1 m'_2}{-2l'_3 m'_3} b + \frac{l'_2 m'_2}{-l'_3 m'_3} c + \frac{l'_3 m'_1 + l'_1 m'_3}{-2l'_3 m'_3} d + \frac{l'_3 m'_2 + l'_2 m'_3}{-2l'_3 m'_3} e = 1 \quad (20)$$

It can be seen that there is 1 equation and 5 unknowns for each pair of orthogonal lines, so 5 pairs of orthogonal lines must be collected for the one-step method in order to find all coefficients for $C_\infty^{*'}$

Once this is found, we know that

$$C_\infty^{*'} = H C_\infty^* H^T \quad (21)$$

Let's assume:

$$H = \begin{pmatrix} A & 0 \\ v^T & 1 \end{pmatrix} \quad (22)$$

This results in:

$$C_\infty^{*'} = \begin{pmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{pmatrix} = \begin{pmatrix} A & 0 \\ v^T & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} A^T & v \\ 0^T & 1 \end{pmatrix} = \begin{pmatrix} AA^T & Av \\ v^T A^T & v^v \end{pmatrix} \quad (23)$$

It can be seen that clearly:

$$AA^T = \begin{pmatrix} a & \frac{b}{2} \\ \frac{b}{2} & c \end{pmatrix} \quad (24)$$

and

$$Av = \begin{pmatrix} \frac{d}{2} \\ \frac{e}{2} \end{pmatrix} \quad (25)$$

By performing SVD on AA^T, A can be determined, and as a result, $A^{-1}Av = v$

These can then be combined to generate the final one-step homography. The following result is shown below for the board image:

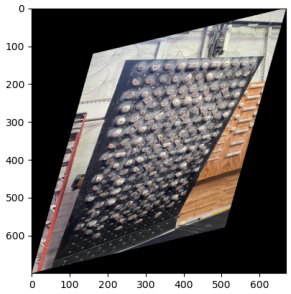


Figure 5: Board One Step

2 Discussion

My results were pretty bad because most of the time spent with this homework was trying to fix distortion errors from the corridor image. It was not made clear to the class that we were to ignore this image for submission, which caused me and some of my peers to sink more time into this than necessary. However, I think the P2P results that I did get working in time for this homework submission look pretty good

3 Code

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import time
import math


img1 = np.array(cv2.imread('HW3_images/board_1.jpeg'))
img2 = np.array(cv2.imread('HW3_images/corridor.jpeg'))

test_img1 = np.array(cv2.imread('HW3_images/test_img1.jpg'))
test_img2 = np.array(cv2.imread('HW3_images/test_img2.jpg'))

## Uncomment for pixel location visualization
# plt.imshow(img1)
# plt.show()

# plt.imshow(test_img2)
# plt.show()


width1 = 800
height1 = 1200
width2 = 300
height2 = 600


img1_points = np.array([[71, 421], [1220, 140], [1354, 1950], [421, 1789]])
test_img1_points = np.array([[923, 1300], [2330, 1165], [2241, 2134], [1006, 2668]])
test_img2_points = np.array([[730, 1014], [2266, 1116], [2037, 2882], [821, 2638]])


img1_2step_points = np.array([[71, 421], [1220, 140], [1354, 1950], [71, 421], [421, 1789],
test_img1_2step_points = np.array([[923, 1300], [2330, 1165], [2241, 2134], [923, 1300], [10
```

```
test_img2_2step_points = np.array([[730, 1014], [2266, 1116], [2037, 2882], [730, 1014], [82


img1_1step_points = np.array([[71, 421], [1220, 140], [1354, 1950], [71, 421], [421, 1789],
img2_1step_points = np.array([[1083, 528], [1306, 487], [1296, 1340], [916, 1127], [811, 106



transformed_img1_points = np.array([[0, 0], [width1, 0], [width1, height1], [0, height1]])
transformed_img2_points = np.array([[0, 0], [width2, 0], [width2, height2], [0, height2]])

transformed_test_img1_points = np.array([[0, 0], [height1, 0], [height1, width1], [0, width1
transformed_test_img2_points = np.array([[0, 0], [width1, 0], [width1, height1], [0, height1


def HC(point):
 return [point[0], point[1], 1.001]

def find_homgoraphy(domain_set, range_set):
 num_points = domain_set.shape[0]

 A = []
 p = []

 for i in range(num_points):
  # Convert to HC
  u, v, w = [domain_set[i][0], domain_set[i][1], 1]
  up, vp, wp = [range_set[i][0], range_set[i][1], 1]

  # Set up A and p matrices
  A.append([u, v, w, 0, 0, 0, -up * u, -up * v])
  A.append([0, 0, 0, u, v, w, -vp * u, -vp * v])
  p.append(up)
  p.append(vp)

 A = np.array(A)
 p = np.array(p)

 H = np.dot(np.linalg.inv(A), p.T)
 H = np.append(H, 1)
 H = np.reshape(H, (3, 3))

 return(H)


def apply_homography_with_inverse(src_img, H):
```

```
 dest_img, H = calculate_raster_and_resize(src_img, H)


 domain_height, domain_width, _ = src_img.shape
 range_height, range_width, _ = dest_img.shape


 H_inv = np.linalg.inv(H)

 H_inv /= H_inv[2, 2]

 for v in range(dest_img.shape[0]):
  for u in range(dest_img.shape[1]):
   range_point = np.array([u, v, 1])
   domain_point = H_inv @ range_point

   domain_point /= domain_point[2]

   domain_u, domain_v = int(domain_point[0]), int(domain_point[1])

   if(0 <= domain_u < domain_width and 0 <= domain_v < domain_height):
    dest_img[v, u] = src_img[domain_v, domain_u]


 return dest_img

def calculate_raster(img, H):
 img_height, img_width, _ = img.shape

 corners = np.array([[0, 0, 1], [img_width, 0, 1], [img_width, img_height, 1], [0, img_heigh
 corners_hc = corners.T

 corners_transformed_hc = np.matmul(H, corners_hc)
 corners_transformed_hc = corners_transformed_hc / (corners_transformed_hc[-1, :] + 1e-6)


 min_x = int(min(corners_transformed_hc[0,:]))
 max_x = int(max(corners_transformed_hc[0,:]))
 min_y = int(min(corners_transformed_hc[1,:]))
 max_y = int(max(corners_transformed_hc[1,:]))

 # print("min_x: ", min_x)
 # print("max_x: ", max_x)

 # print("min_y: ", min_y)
```

```
 # print("max_y: ", max_y)

 w_out = max_x - min_x
 h_out = max_y - min_y

 H_trans = np.array([[1, 0, -min_x], [0, 1, -min_y], [0, 0, 1]])

 return w_out, h_out, H_trans, min_x, max_x

def calculate_raster_and_resize(img, H):

 # print(H)

 w_out, h_out, _, _, _ = calculate_raster(img, H)

 #resizing
 aspect_ratio = w_out / (h_out + 1e-6)
 h_out_final = 700
 w_out_final = aspect_ratio * h_out_final
 H_resize = np.array([[w_out_final / (w_out + 1e-6), 0, 0], [0, h_out_final / (h_out + 1e-6)

 H = np.matmul(H_resize, H)

 w_out, h_out, H_trans, min_x, max_x = calculate_raster(img, H)

 # print("w_out, h_out: ", w_out, h_out)

 H = np.matmul(H_trans, H)

 new_img = np.zeros((h_out, w_out, 3), dtype=np.uint8)

 return new_img, H


def two_step(img_points, img, img_affine_points):

 img_height, img_width, _ = img.shape

 straight_line_horizontal1 = np.cross(HC(img_points[0]), HC(img_points[1]))
 straight_line_vertical1 = np.cross(HC(img_points[1]), HC(img_points[2]))

 straight_line_horizontal2 = np.cross(HC(img_points[3]), HC(img_points[2]))
 straight_line_vertical2 = np.cross(HC(img_points[0]), HC(img_points[3]))

 vanishing_point1 = np.cross(straight_line_horizontal1, straight_line_horizontal2)
 vanishing_point2 = np.cross(straight_line_vertical1, straight_line_vertical2)
```

```
vanishing_line = np.cross(vanishing_point1, vanishing_point2)
vanishing_line = vanishing_line / (vanishing_line[2])


Hp = np.array([[1, 0, 0], [0, 1, 0], [vanishing_line[0], vanishing_line[1], vanishing_line


new_img1 = apply_homography_with_inverse(img, Hp)


straight_line_horizontal1 = np.transpose(np.linalg.inv(Hp)) @ straight_line_horizontal1
straight_line_horizontal1 = straight_line_horizontal1 / straight_line_horizontal1[2]

straight_line_horizontal2 = np.transpose(np.linalg.inv(Hp)) @ straight_line_horizontal2
straight_line_horizontal2 = straight_line_horizontal2 / straight_line_horizontal2[2]

straight_line_vertical1 = np.transpose(np.linalg.inv(Hp)) @ straight_line_vertical1
straight_line_vertical1 = straight_line_vertical1 / straight_line_vertical1[2]

straight_line_vertical2 = np.transpose(np.linalg.inv(Hp)) @ straight_line_vertical2
straight_line_vertical2 = straight_line_vertical2 / straight_line_vertical2[2]


L = np.array([[straight_line_horizontal1[0]*straight_line_vertical1[0], straight_line_horiz
R = np.array([[-straight_line_horizontal1[1]*straight_line_vertical1[1]], [-straight_line_h


L_inv = np.linalg.inv(L)

result = np.squeeze(L_inv @ R)

S = np.array([[result[0], result[1]], [result[1], 1]])

U, D, Vt = np.linalg.svd(S)

D_sqrt = np.diag(np.sqrt(D))

A = U @ D_sqrt @ Vt


Ha = np.array([[A[0, 0], A[0, 1], 0], [A[1, 0], A[1, 1], 0], [0, 0, 1]])

H = Ha @ Hp

new_img2 = apply_homography_with_inverse(img, H)
```

```
 new_img1 = cv2.cvtColor(new_img1, cv2.COLOR_BGR2RGB)
 new_img2 = cv2.cvtColor(new_img2, cv2.COLOR_BGR2RGB)

 plt.imshow(new_img1)
 plt.show()

 plt.imshow(new_img2)
 plt.show()


def one_step(img_1step_points, img):
 line11 = np.cross(HC(img_1step_points[0]), HC(img_1step_points[1]))
 line12 = np.cross(HC(img_1step_points[1]), HC(img_1step_points[2]))

 line21 = np.cross(HC(img_1step_points[3]), HC(img_1step_points[4]))
 line22 = np.cross(HC(img_1step_points[4]), HC(img_1step_points[5]))

 line31 = np.cross(HC(img_1step_points[6]), HC(img_1step_points[7]))
 line32 = np.cross(HC(img_1step_points[7]), HC(img_1step_points[8]))

 line41 = np.cross(HC(img_1step_points[9]), HC(img_1step_points[10]))
 line42 = np.cross(HC(img_1step_points[10]), HC(img_1step_points[11]))

 line51 = np.cross(HC(img_1step_points[12]), HC(img_1step_points[13]))
 line52 = np.cross(HC(img_1step_points[13]), HC(img_1step_points[14]))


 line_pairs = [[line11, line12], [line21, line22], [line31, line32], [line41, line42], [line


 L = []
 R = []

 for line_pair in line_pairs:
  l1 = line_pair[0][0]
  l2 = line_pair[0][1]
  l3 = line_pair[0][2]

  m1 = line_pair[1][0]
  m2 = line_pair[1][1]
  m3 = line_pair[1][2]

  L.append([l1 * m1, (l2*m1 + l1*m2) / 2, l2 * m2, (l3*m1 + l1*m3) / 2, (l3*m2 + l2*m3) / 2]
```

```
 R.append([-l3*m3])

L = np.array(L)
R = np.array(R)

L_inv = np.linalg.inv(L)

result = np.squeeze(L_inv @ R)

a = result[0]
b = result[1]
c = result[2]
d = result[3]
e = result[4]

AAt = np.array([[a, b/2], [b/2, c]])
Av = np.array([[d/2], [e/2]])

U, D, Vt = np.linalg.svd(AAt)

A = U @ np.diag(np.sqrt(D)) @ Vt.T

A_inv = np.linalg.inv(A)

v = np.squeeze(A_inv @ Av)

H = np.array([[A[0, 0], A[0, 1], 0], [A[1, 0], A[1, 1], 0], [v[0], v[1], 1]])

H = H.T

new_img = apply_homography_with_inverse(img, H)

new_img = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)

plt.imshow(new_img)
plt.show()



state = '1step'

if(state == 'p2p'):
 # ## Finding Homographies
 H1 = find_homgoraphy(img1_points, transformed_img1_points)
 H1_test = find_homgoraphy(test_img1_points, transformed_test_img1_points)
```

```
 H2_test = find_homgoraphy(test_img2_points, transformed_test_img2_points)

 transformed_img1 = apply_homography_with_inverse(img1, H1)
 transformed_test_img1 = apply_homography_with_inverse(test_img1, H1_test)
 transformed_test_img2 = apply_homography_with_inverse(test_img2, H2_test)

 transformed_img1 = cv2.cvtColor(transformed_img1, cv2.COLOR_BGR2RGB)
 transformed_test_img1 = cv2.cvtColor(transformed_test_img1, cv2.COLOR_BGR2RGB)
 transformed_test_img2 = cv2.cvtColor(transformed_test_img2, cv2.COLOR_BGR2RGB)

 plt.imshow(transformed_img1)
 plt.show()

 plt.imshow(transformed_test_img1)
 plt.show()

 plt.imshow(transformed_test_img2)
 plt.show()

elif(state == '2step'):
 two_step(img1_points, img1, img1_2step_points)
 two_step(test_img1_points, test_img1, test_img1_2step_points)
 two_step(test_img2_points, test_img2, test_img2_2step_points)

elif(state == "1step"):
 one_step(img1_1step_points, img1)
```