Homework - 2

Sriram Karthik Badam

September 6, 2012

1. **Elimination of Projective Distortion from a camera image of a planar scene : Calculate Homography**

   The Homography (H) to transform a point on the world plane to the image plane is given by

$$x^{image} = H.x^{world}$$

$$\begin{bmatrix} {x_1}^i \\ {y_1}^i \\ {z_1}^i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} {x_1}^w \\ {y_1}^w \\ 1 \end{bmatrix}$$

   where $(\frac{{x_1}^i}{{z_1}^i}, \frac{{y_1}^i}{{z_1}^i})$ gives the actual coordinates of the point in 2D image plane.

   Given a point x in the world plane, we can easily identify its counterpart on the image plane (x^i). Further solving the above equation we end up with.,

$$\begin{aligned} {x_1}^i &= h_{11}{x_1}^w + h_{12}{y_1}^w + h_{13} \\ {y_1}^i &= h_{21}{x_1}^w + h_{22}{y_1}^w + h_{23} \\ {z_1}^i &= h_{31}{x_1}^w + h_{32}{y_1}^w + 1 \end{aligned}$$

   which implies that the corresponding 2D coordinates of the point on image plane are

$$({x_1}^{image}, {y_1}^{image}) = (\frac{h_{11}{x_1}^w + h_{12}{y_1}^w + h_{13}}{h_{31}{x_1}^w + h_{32}{y_1}^w + 1}, \frac{h_{21}{x_1}^w + h_{22}{y_1}^w + h_{23}}{h_{31}{x_1}^w + h_{32}{y_1}^w + 1})$$

   The above formula represents two equations which need to be solved to get the value of H. Since there are 8 unknowns in H, we need four such points to find H.

These set of equations can be written in form of Ax = b where A is a coefficient matrix (8x8), x consists of the variables and b(8x1) contains the constant terms.

$$\begin{bmatrix} {x_1}^w & {y_1}^w & 1 & 0 & 0 & 0 & -{x_1}^w{x_1}^i & -{y_1}^w{x_1}^i \\ 0 & 0 & 0 & {x_1}^w & {y_1}^w & 1 & -{x_1}^w{y_1}^i & -{y_1}^w{y_1}^i \\ {x_2}^w & {y_2}^w & 1 & 0 & 0 & 0 & -{x_2}^w{x_2}^i & -{y_2}^w{x_2}^i \\ 0 & 0 & 0 & {x_2}^w & {y_2}^w & 1 & -{x_2}^w{y_2}^i & -{y_2}^w{y_2}^i \\ {x_3}^w & {y_3}^w & 1 & 0 & 0 & 0 & -{x_3}^w{x_3}^i & -{y_3}^w{x_3}^i \\ 0 & 0 & 0 & {x_3}^w & {y_3}^w & 1 & -{x_3}^w{y_3}^i & -{y_3}^w{y_3}^i \\ {x_4}^w & {y_4}^w & 1 & 0 & 0 & 0 & -{x_4}^w{x_4}^i & -{y_4}^w{x_4}^i \\ 0 & 0 & 0 & {x_4}^w & {y_4}^w & 1 & -{x_4}^w{y_4}^i & -{y_4}^w{y_4}^i \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} {x_1}^i \\ {y_1}^i \\ {x_2}^i \\ {y_2}^i \\ {x_3}^i \\ {y_3}^i \\ {x_4}^i \\ {y_4}^i \end{bmatrix}$$

Solving the linear equations we get the matrix H.

2. **Use H to remove projective distortion**

   From H, we get the inverse of H (H^{-1}) which maps the image to world plane. Using H^{-1} we find the boundaries of the image in the world plane and set the width, height of the result.

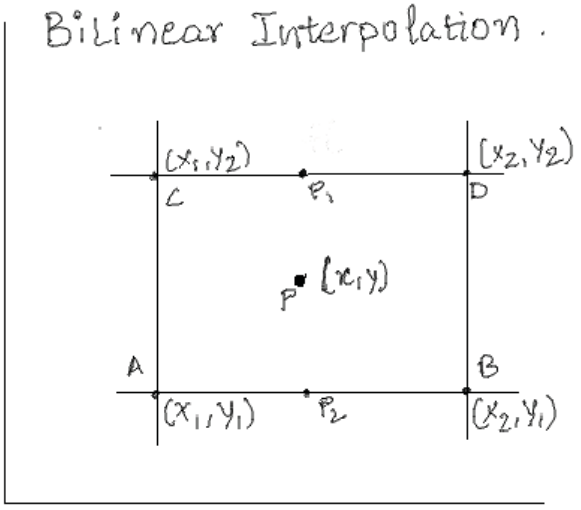

Figure 1: Interpolation

The coordinates on the world plane are Real but the pixel coordinates on the image plane are integers. So there can't be an one to one correspondence. To get around this predicament, we can use bilinear interpolation to estimate the pixel value on world plane. Bilinear Interpolation works as follows, Suppose a point P is surrounded by the points (x_1, y_1), (x_2, y_1), (x_1, y_2), (x_2, y_2) then the value of a function f at the point P(x, y) is approximated from the values at the surrounding points in 2D space.

$$f(x,y) = \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(x_1, y_1) + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(x_2, y_1)$$

$$+ \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(x_1, y_2) + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(x_2, y_2)$$

This formula is obtained by first interpolating in x-direction to get values at P_1, P_2 and then in y-direction to get the value at P.

The transformation occurs as follows

(a) The Boundaries of the distortion free image (the planar scene) are calculated using H^{-1} and the camera image boundaries.

(b) The new width, height are computed and using them a new image is created. Now pixel values are calculated for each point (of the grid) on world plane by finding the corresponding pixel coordinates in camera image. Since these pixel coordinates may not be integral, bilinear interpolation is used on the surrounding pixels as shown in the figure.

This process gives us a projective distortion free image.

3. **Similarity Transformation between two Projective Distortion free images.**

To calculate the Similarity Transform between two images we take four points on each image and use these points to calculate the Homography/Similarity transformation matrix using the method explained in Section 1 (in this case, the homography is equal to Similarity Tranformation(S) because the images are supposed to be similiar without any projective distortion).

Once we get the Similarity tranformation matrix (S) we can verify the results by transforming one image into the image plane of

the second image and overlapping it onto the latter image. This is done by applying the transform matrix S on each and every pixel in the first image to find the corresponding pixel in the second image. Then the pixels of the first image are aligned with the second. This gives an overlap effect with the second image on the back ground and the first image on foreground.(I overlapped them by calculating the mean of pixel values at each common pixel to retain the information of both images).

The Results of Homography and Similarity transformation are shown from the next page.

In each image, the points used to calculate the homography or Similarity transformation are marked with green arrows.

4. **Results: Projective Distortion Elimination**

Adams1.jpg

Figure 2: input: adams1.jpg

Figure 3: output

Adams2.jpg

Figure 4: input: adams2.jpg

Figure 5: output

board1.jpg

Figure 6: input: board1.jpg

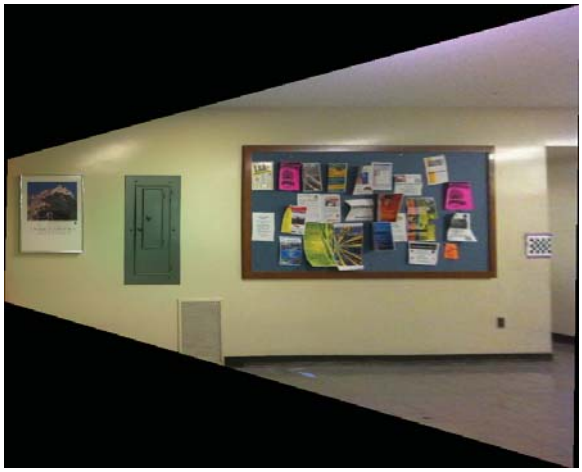

Figure 7: output

board2.jpg

Figure 8: input: board2.jpg

Figure 9: output

door1.jpg

Figure 10: input: door1.jpg

Figure 11: output

door2.jpg

Figure 12: input: door2.jpg

Figure 13: output

tree1.jpg

Figure 14: input: tree1.jpg

Figure 15: output

tree2.jpg

Figure 16: input: tree2.jpg

Figure 17: output

My pictures : Apartment door

Figure 18: input: apartment door.jpg

Figure 19: output

My pictures : Box

Figure 20: input: box1.jpg

Figure 21: output

My pictures : Box2

Figure 22: input: box2.jpg

Figure 23: output

5. **Results of similarity tranformation**

Figure 24: input: Corrected adams2

Figure 25: input: Corrected adams1

Figure 26: Overlapped Adams

Overlapped boards

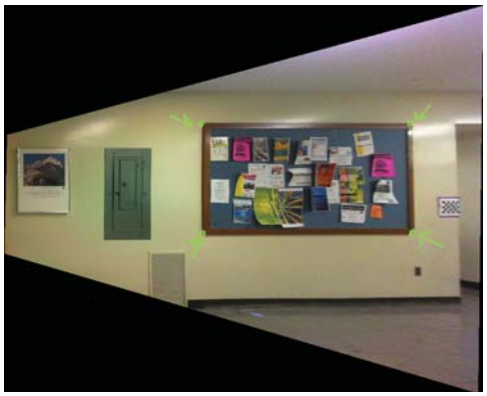

Figure 27: input: Corrected board1

Figure 28: input: Corrected board2

Figure 29: Overlapped boards

Overlapped Doors

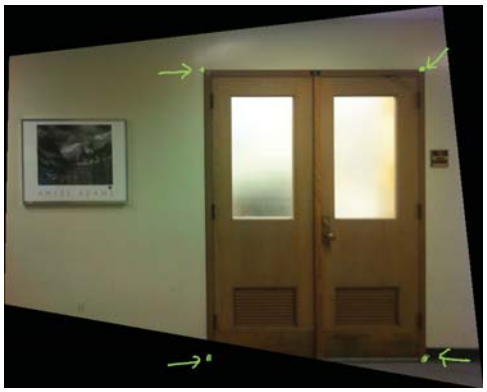

Figure 30: input: Corrected door2

Figure 31: input: Corrected door1

Figure 32: Overlapped doors

Overlapped Trees

Figure 33: input: Corrected tree1

Figure 34: input: Corrected tree2

Figure 35: Overlapped trees

Overlapped Boxs

Figure 36: input: Corrected box1

Figure 37: input: Corrected box2

Result shown in next page

Figure 38: Overlapped boxs

6. **Source Code: Elimination of Projective Distortion (homography.py)**

```
#!/usr/bin/python
import sys, os
import cv
import numpy

def main():
  world_input = numpy.zeros(8, float)
  image_input = numpy.zeros(8, int)
  print "@Author: S.karthik badam"
  if len(sys.argv) > 1:
    filename = sys.argv[1]
    im = cv.LoadImageM(filename, cv.CV_LOAD_IMAGE_UNCHANGED
        )
    if im == 0:
      print "enter a valid imagefile path"
      exit()
  else:
    print "enter a valid imagefile path"

  #read input from user

  for i in range(4):
    print "Enter the World coordinate x,y ",i
    world_input[2*i],world_input[2*i+1] = raw_input().split
        (",")
    print "The world coordinates of point ",i," are ",
        world_input[2*i],",",world_input[2*i+1]

    print "Enter the corresponding Image coordinate"
    image_input[2*i],image_input[2*i+1] = raw_input().split
        (",")
    print "The image coordinates of point are ",image_input
        [2*i],",",image_input[2*i+1]
    print "-------------------------------------------------"

  #create the matrices A and B to solve Ax = B

  A_matrix_as_vector = numpy.zeros(64, float)
  B_vector = numpy.zeros(8, float)

  for i in range(4):
    #even rows in terms of i
    A_matrix_as_vector[16*i+0] = world_input[2*i]
    A_matrix_as_vector[16*i+1] = world_input[2*i+1]
    A_matrix_as_vector[16*i+2] = 1
    A_matrix_as_vector[16*i+3] = 0
```

```
    A_matrix_as_vector[16*i+4] = 0
    A_matrix_as_vector[16*i+5] = 0
    A_matrix_as_vector[16*i+6] = -image_input[2*i]*
        world_input[2*i]
    A_matrix_as_vector[16*i+7] = -image_input[2*i]*
        world_input[2*i+1]

    #odd rows
    A_matrix_as_vector[8*(2*i+1)+0] = 0
    A_matrix_as_vector[8*(2*i+1)+1] = 0
    A_matrix_as_vector[8*(2*i+1)+2] = 0
    A_matrix_as_vector[8*(2*i+1)+3] = world_input[2*i]
    A_matrix_as_vector[8*(2*i+1)+4] = world_input[2*i+1]
    A_matrix_as_vector[8*(2*i+1)+5] = 1
    A_matrix_as_vector[8*(2*i+1)+6] = -image_input[2*i+1]*
        world_input[2*i]
    A_matrix_as_vector[8*(2*i+1)+7] = -image_input[2*i+1]*
        world_input[2*i+1]


    #fill vector B
    B_vector[2*i] = image_input[2*i]
    B_vector[2*i+1] = image_input[2*i+1]


#Solve Input to get Homography H

A = cv.CreateMatHeader(8, 8, cv.CV_64FC1)
cv.SetData(A, A_matrix_as_vector, cv.CV_AUTOSTEP)
B = cv.CreateMatHeader(8, 1, cv.CV_64FC1)
cv.SetData(B, B_vector, cv.CV_AUTOSTEP)
x = cv.CreateMat(8,1, cv.CV_64FC1)

cv.Solve(A, B, x, cv.CV_LU)

H = cv.CreateMat(3, 3, cv.CV_64FC1)
for i in range(3):
  for j in range(3):
    if (i*3+j)!=8:
      cv.mSet(H, i , j, cv.mGet(x, i*3 + j, 0))
    else:
      cv.mSet(H, 2, 2, 1)

H_inverse = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.Invert(H, H_inverse, cv.CV_LU)

#printing the H Matrix
temp_array = numpy.asarray(H[:,:])
print "Homography H is :"
```

```
print temp_array

#Transform the image to world plane

boundaries_world = cv.CreateMat(3, 4, cv.CV_64FC1)
boundaries_image = cv.CreateMat(3, 4, cv.CV_64FC1)
cv.mSet(boundaries_image, 0, 0, 0)
cv.mSet(boundaries_image, 0, 1, im.width-1)
cv.mSet(boundaries_image, 0, 2, 0)
cv.mSet(boundaries_image, 0, 3, im.width-1)
cv.mSet(boundaries_image, 1, 0, 0)
cv.mSet(boundaries_image, 1, 1, 0)
cv.mSet(boundaries_image, 1, 2, im.height-1)
cv.mSet(boundaries_image, 1, 3, im.height-1)
cv.mSet(boundaries_image, 2, 0, 1)
cv.mSet(boundaries_image, 2, 1, 1)
cv.mSet(boundaries_image, 2, 2, 1)
cv.mSet(boundaries_image, 2, 3, 1)

cv.MatMul(H_inverse, boundaries_image, boundaries_world)

#check boundaries in world coordinates to get the min and
     max values

x_min = 1e10
y_min = 1e10
x_max = 0
y_max = 0

for i in range(4):
  x = cv.mGet(boundaries_world, 0, i)/cv.mGet(
      boundaries_world, 2,i)
  y = cv.mGet(boundaries_world, 1, i)/cv.mGet(
      boundaries_world, 2,i)
  if x < x_min:
    x_min = x
  if x > x_max:
    x_max = x
  if y < y_min:
    y_min = y
  if y > y_max:
    y_max = y

#scale
scale = float(im.width/(x_max-x_min))
final_height = int((y_max-y_min)*scale)

new_image = cv.CreateImage((im.width, final_height), cv.
    IPL_DEPTH_64F,3)
```

```
  #fill the image using intepolation
  world_coordinates = cv.CreateMat(3, 1, cv.CV_64FC1)
  image_coordinates = cv.CreateMat(3, 1, cv.CV_64FC1)
  cv.mSet(world_coordinates, 2, 0, 1)

  for i in range(new_image.width):
    cv.mSet(world_coordinates, 0, 0, float(x_min+i/scale))
    for j in range(new_image.height):
      cv.mSet(world_coordinates, 1, 0, float(y_min+j/scale)
          )
      cv.MatMul(H, world_coordinates, image_coordinates)

      x = cv.mGet(image_coordinates, 0, 0)/cv.mGet(
          image_coordinates, 2, 0)
      y = cv.mGet(image_coordinates, 1, 0)/cv.mGet(
          image_coordinates, 2, 0)

      if x>=0 and y>=0 and x<im.width-1 and y<im.height-1:

        temp = [0,0,0]
        for k in range(3):
          temp[k] = 0
          temp[k] = temp[k]+(x-int(x))*(y-int(y))*(im[int(y
              +1),int(x+1)][k])
          temp[k] = temp[k]+(1.0-(x-int(x)))*(y-int(y))*(im
              [int(y+1),int(x)][k])
          temp[k] = temp[k]+(x-int(x))*(1.0-(y-int(y)))*(im
              [int(y),int(x+1)][k])
          temp[k] = temp[k]+(1.0-(x-int(x)))*(1.0-(y-int(y)
              ))*(im[int(y),int(x)][k])

        new_image[j,i] =  temp

  #write image to file
  cv.SaveImage("final_image.png", new_image)

  #show Original and final images
  cv.NamedWindow("Original", cv.CV_WINDOW_AUTOSIZE)
  cv.ShowImage("Original", im)
  cv.NamedWindow("Result", cv.CV_WINDOW_AUTOSIZE)
  im = cv.LoadImageM("final_image.png", cv.
      CV_LOAD_IMAGE_UNCHANGED)
  cv.ShowImage("Result", im)
  cv.WaitKey()

if __name__=="__main__":
  main()
```

7. Source Code: Similarity transformation (similarity.py)

```
#!/usr/bin/python
import sys, os
import cv
import numpy


def main():
  image1_input = numpy.zeros(8, int)
  image2_input = numpy.zeros(8, int)
  print "@Author: S.karthik badam"
  if len(sys.argv) > 1:
    filename1 = sys.argv[1]
    im1 = cv.LoadImageM(filename1, cv.
        CV_LOAD_IMAGE_UNCHANGED)
    if im1 == 0:
      print "enter a valid imagefile path"
      exit()
    filename2 = sys.argv[2]
    im2 = cv.LoadImageM(filename2, cv.
        CV_LOAD_IMAGE_UNCHANGED)
    if im2 == 0:
      print "enter a valid imagefile path"
      exit()

  else:
    print "enter a valid imagefile path"
  #read input from user

  print "Enter the cordinates of matching points in image 1
      and image 2"
  for i in range(4):
    print "Enter the image 1 coordinate x,y ",i
    image1_input[2*i],image1_input[2*i+1] = raw_input().
        split(",")
    print "The image1 coordinates of point ",i," are ",
        image1_input[2*i],",",image1_input[2*i+1]

    print "Enter the corresponding Image 2 coordinate"
    image2_input[2*i],image2_input[2*i+1] = raw_input().
        split(",")
    print "The image coordinates of point are ",
        image2_input[2*i],",",image2_input[2*i+1]
    print "-----------------------------------------------"

  #create the matrices A and B to solve Ax = B
```

```
A_matrix_as_vector = numpy.zeros(64, float)
B_vector = numpy.zeros(8, float)

for i in range(4):
  #even rows in terms of i
   A_matrix_as_vector[16*i+0] = image1_input[2*i]
   A_matrix_as_vector[16*i+1] = image1_input[2*i+1]
   A_matrix_as_vector[16*i+2] = 1
   A_matrix_as_vector[16*i+3] = 0
   A_matrix_as_vector[16*i+4] = 0
   A_matrix_as_vector[16*i+5] = 0
   A_matrix_as_vector[16*i+6] = -image2_input[2*i]*
       image1_input[2*i]
   A_matrix_as_vector[16*i+7] = -image2_input[2*i]*
       image1_input[2*i+1]

  #odd rows

   A_matrix_as_vector[8*(2*i+1)+0] = 0
   A_matrix_as_vector[8*(2*i+1)+1] = 0
   A_matrix_as_vector[8*(2*i+1)+2] = 0
   A_matrix_as_vector[8*(2*i+1)+3] = image1_input[2*i]
   A_matrix_as_vector[8*(2*i+1)+4] = image1_input[2*i+1]
   A_matrix_as_vector[8*(2*i+1)+5] = 1
   A_matrix_as_vector[8*(2*i+1)+6] = -image2_input[2*i+1]*
       image1_input[2*i]
   A_matrix_as_vector[8*(2*i+1)+7] = -image2_input[2*i+1]*
       image1_input[2*i+1]


  #fill vector B
   B_vector[2*i] = image2_input[2*i]
   B_vector[2*i+1] = image2_input[2*i+1]


#Solve Input to get Similarity transform S

A = cv.CreateMatHeader(8, 8, cv.CV_64FC1)
cv.SetData(A, A_matrix_as_vector, cv.CV_AUTOSTEP)
B = cv.CreateMatHeader(8, 1, cv.CV_64FC1)
cv.SetData(B, B_vector, cv.CV_AUTOSTEP)
x = cv.CreateMat(8,1, cv.CV_64FC1)

cv.Solve(A, B, x, cv.CV_LU)

S = cv.CreateMat(3, 3, cv.CV_64FC1)

for i in range(3):
  for j in range(3):
```

```
    if (i*3+j)!=8:
      cv.mSet(S, i , j, cv.mGet(x, i*3 + j, 0))
    else:
      cv.mSet(S, 2, 2, 1)


#printing the Similarity Matrix
temp_array = numpy.asarray(S[:,:])
print "Similarity transform is :"
print temp_array

S_inverse = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.Invert(S, S_inverse, cv.CV_LU)

#overlap both images
image1_coordinates = cv.CreateMat(3, 1, cv.CV_64FC1)
image2_coordinates = cv.CreateMat(3, 1, cv.CV_64FC1)
cv.mSet(image1_coordinates, 2, 0, 1)

for i in range(im1.width):
  cv.mSet(image1_coordinates, 0, 0, i)
  for j in range(im1.height):
    cv.mSet(image1_coordinates, 1, 0, j)
    cv.MatMul(S, image1_coordinates, image2_coordinates)

    x = cv.mGet(image2_coordinates, 0, 0)/cv.mGet(
        image2_coordinates, 2, 0)
    y = cv.mGet(image2_coordinates, 1, 0)/cv.mGet(
        image2_coordinates, 2, 0)

    if x>=0 and y>=0 and x<im2.width-1 and y<im2.height
        -1:

      temp = [0,0,0]
      for k in range(3):
        if im2[int(y),int(x)][k] != 0 and im1[j,i][k] !=
            0:
          temp[k] = (im1[j,i][k] + im2[int(y),int(x)][k])
              /2
        else:
          temp[k] = im1[j,i][k]
      if temp!=[0,0,0]:
        im2[int(y),int(x)] =  temp

#write image to file
cv.SaveImage("final_image.png", im2)

cv.NamedWindow("Result", cv.CV_WINDOW_AUTOSIZE)
im = cv.LoadImageM("final_image.png", cv.
```

```
        CV_LOAD_IMAGE_UNCHANGED)
    cv.ShowImage("Result", im)
    cv.WaitKey()

if __name__=="__main__":
    main()
```