Homework - 3

Sriram Karthik Badam

September 18, 2012

1. **Elimination of Projective and Affine Distortion - Two Step Method**

   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.

   The first method to remove both Projective and Affine distortion, discussed in this document, is a two step method that achieves the goal by first getting rid of the Projective Distortion by mapping the vanishing line of the image to l_∞ and then removing the Affine distortion by using dual degenerate conic.

   (a) **Step - 1: Removing Projective Distortion**

   To eliminate the projective distortion we take coordinates of 4 points as input which form a rectangular segment in the world plane. The intersections of the opposite sides of the distorted rectangle are calculated by simple matrix operations. These intersection points are called Vanishing Points (P, Q) and the line joining them is the vanishing line. See the Figure 1 of image plane for further clarity.

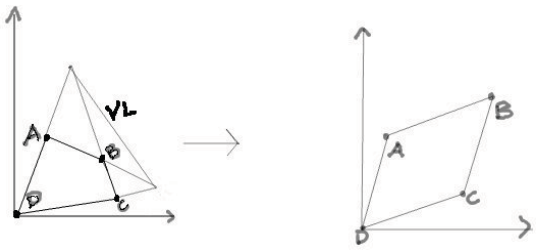

Figure 1: step one

   The Vanishing line has to be mapped to l_∞ to make the opposite sides of rectangular segment parallel. Let $L(l_1, l_2, l_3)$ be the Vanishing line, then the homography that maps it to l_∞ is

$$l_\infty = \begin{bmatrix} 1 & 0 & -l_1/l_3 \\ 0 & 1 & -l_2/l_3 \\ 0 & 0 & 1/l_3 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

   By the properties of Duality between lines and points, the Homography(H_{proj}) that tranforms a point in the world plane to one in the image plane (during which, it creates the projective distortion) is given by

$$\begin{bmatrix} {x_1}^i \\ {y_1}^i \\ {z_1}^i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} \begin{bmatrix} {x_1}^w \\ {y_1}^w \\ 1 \end{bmatrix}$$

Thus the coordinates of the Vanishing line give the Homography that maps the world plane to image plane with projective distortion. The inverse of this Homography removes the projective distortion because now the lines corresponding to the opposite sides of the rectangle meet on l_∞ which means they are parallel.
The properties of Homogeneous coordinates system used for the above procedure are:

i. The cross product of two points in Homogeneous coordinates system gives the line joining the two points.
ii. The intersection of two lines is computed by finding the cross product of the lines.

(b) **Step: 2 Removing Affine Distortion**

Now that we removed the projective Distortion, the next step would be to find a way to get rid of the Affine Distortion. The cosine of the angle between two orthogonal lines is 0. We use this property to find a Homography to transform the Distorted Rectangular feature (looks like a parallelogram) obtained from above procedure into a Rectangle. If l (l_1, l_2), m(m_1, m_2) are two orthogonal lines in real world then (assuming they are orthonormal) the cosine of angle between them is cos(Θ)

$$cos(\Theta) = \vec{l}.\vec{m}$$

$\vec{l}$, $\vec{m}$ are two lines in the world plane and their corresponding counterparts in the image plane (with Affine distortion) are $\vec{l'}$,$\vec{m'}$
The mapping between $\vec{l}$ and $\vec{l'}$ is given by

$$\vec{l} = H_{affine}^{-T}.\vec{l'}$$

Substituting this to the equation defining the Cosine of angle between them, we get

$$\vec{l'^T} H_{affine} C_\infty^* H_{affine}^T \vec{m'} = 0$$

where

$$C_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \; and \; H_{affine} = \begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix}$$

By further simplication we have

$$\begin{bmatrix} l_1' \\ l_2' \end{bmatrix} \begin{bmatrix} AA^T & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \end{bmatrix} = 0$$

Let S = AA^T and since S has two degrees of freedom two orthogonal line pairs would be enough to solve it. I used one pair from the rectangle and the other was given as an input to the program. Once we get the elements of S by finding the nullspace of the matrix having $\begin{bmatrix} l_1' * m_1' & l_1' * m_2' + l_2' * m_1' & l_2' * m_2' \end{bmatrix}$ as rows (Simplify above equation for justification), we can find A by Singular Value Decomposition of S. Thus we get the Homography(H_{affine}) whose inverse could eliminate the affine distortion from the image.

$$H = H_{proj} * H_{affine}$$

gives the overall mapping.

2. **One Step Method -**

An Alternate method to get rid of both the Projective and Affine Distortions in a simple step, would use the dual degenerate conic (C_∞^*)) directly without removing Projective Distortion.

The Homography(H) that maps the world plane to the image plane in this case is given by

$$\begin{bmatrix} {x_1}^i \\ {y_1}^i \\ {z_1}^i \end{bmatrix} = H \begin{bmatrix} {x_1}^w \\ {y_1}^w \\ 1 \end{bmatrix}$$

where

$$H = \begin{bmatrix} K & 0 \\ \vec{v} & 1 \end{bmatrix}$$

Applying the same cosine formula here we have

$$\vec{l'^T} C'^*_\infty \vec{m'} = 0$$

where

$$C'^*_\infty = \begin{bmatrix} KK^T & K\vec{v} \\ \vec{v}^T K^T & vv^T \end{bmatrix}$$

(a) To compute C'^*_∞, we can write it as follows

$$C'^*_\infty == \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

since its symmetric and homogeneous we have five variables to solve. So we need five pairs of orthogonal lines $(\vec{l'}, \vec{m'})$. Simplifying futher.,

$$\begin{bmatrix} l'_1 & l'_2 & l'_3 \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \end{bmatrix} = 0$$

$$\begin{bmatrix} l'_1 * m'_1 & \frac{l'_2 * m'_1 + l'_1 * m'_2}{2} & l'_2 * m'_2 & \frac{l'_1 * m'_3 + m'_1 * l'_3}{2} & \frac{l'_2 * m'_3 + m'_2 * l'_3}{2} & l'_2 * m'_2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = 0$$

The values can be obtained from the co-ordinates of the five orthogonal lines.

(b) To get the the matrix A we find the singular Value Decomposition of S which consists of the four corresponding elements in C'^*_∞.

$$S = UDV^T$$

then

$$K = UD^{0.5}U^T$$

(c) To get the value of $\vec{v}$ we solve $\mathrm{K}\vec{v} = \begin{bmatrix} d/2 \\ e/2 \end{bmatrix}$.

Once we get the Homography (H), its inverse can be used to eliminate distortions.

3. **Reconstructing the Distortion free image**

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.

To get the pixel values in the world frame we construct a grid in the world frame and for each point in the grid the corresponding image co-ordinate is found out. These co-ordinates maynot be integers and to get around that we use Bilinear interpolation on the 4 neighbouring pixels.

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

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

This formula is obtained by first interpolating in x-direction and then in y-direction to get the value at a pixel ($f(x, y)$).

4. **Comparision of the two methods**

   (a) Even though the one step method requires a lot of tweaking for the right set of lines, the results seem to be overall better than (or sometimes comparable to) the two step method.

   (b) The one step method fails when there are less orthogonal features in the picture or when the orthogonal features are parallel to each other in the image.

5. In my implementation input is taken from the user by catching the mouse click event when the user clicks on a pixel. The pixel and connecting lines are higlighted in the results.

   (a) For the twostep method the user has to first input the rectangular segment coordinates and then the orthogonal pair.

   (b) For onestep method, the user has to click on each point of the line irrespective of whether two lines share a point i.e., If AB and BC are two lines, one joining A,B points and the other B,C points, then user clicks on A once, B twice and C once.

6. **Results**

adams

Figure 2: input: adams.jpg

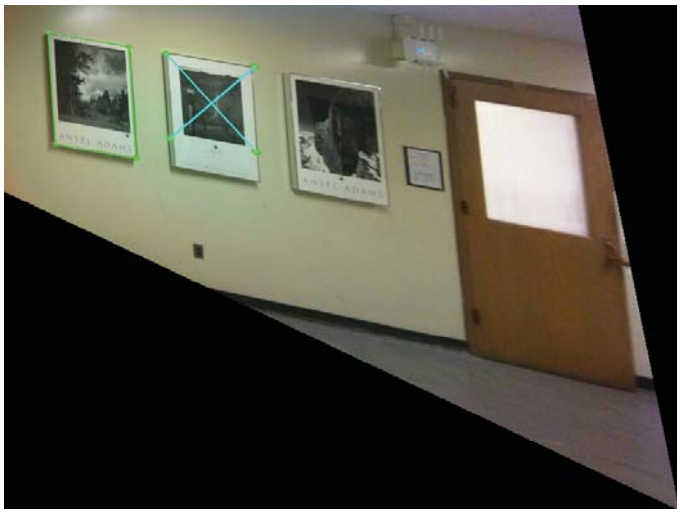

Figure 3: Step 1 result

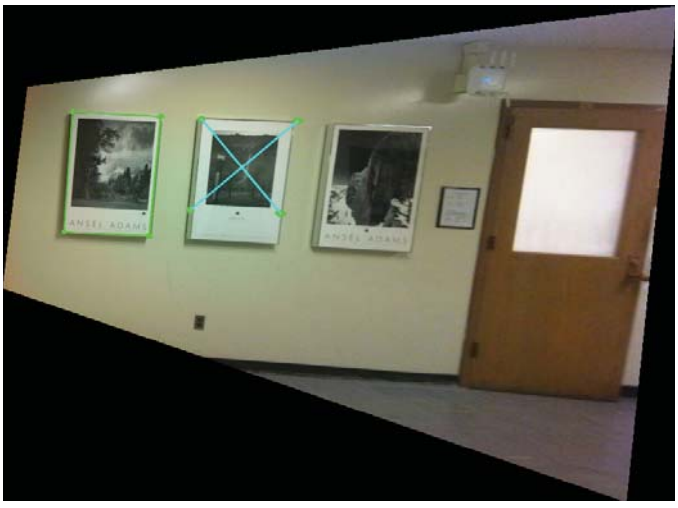

Figure 4: Step 2 Result

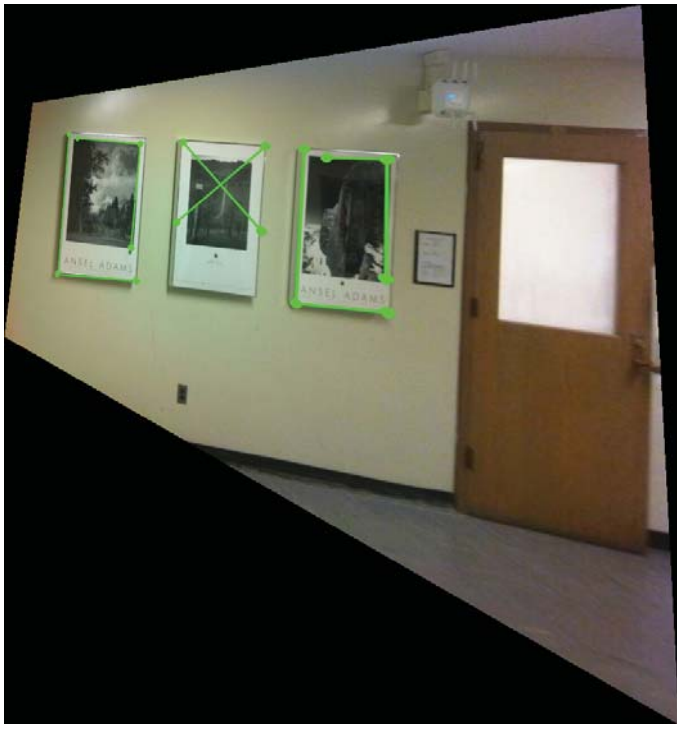

Figure 5: One Step Method result

adams2

Figure 6: input: adams.jpg

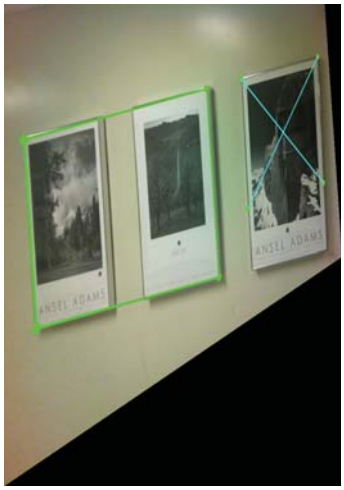

Figure 7: Step 1 result

Figure 8: Step 2 Result

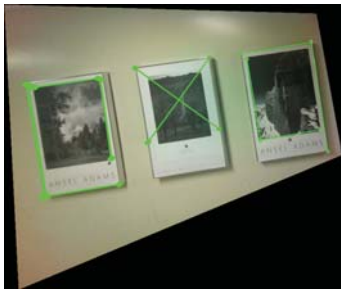

Figure 9: One Step Method result

Board

Figure 10: input: board.jpg

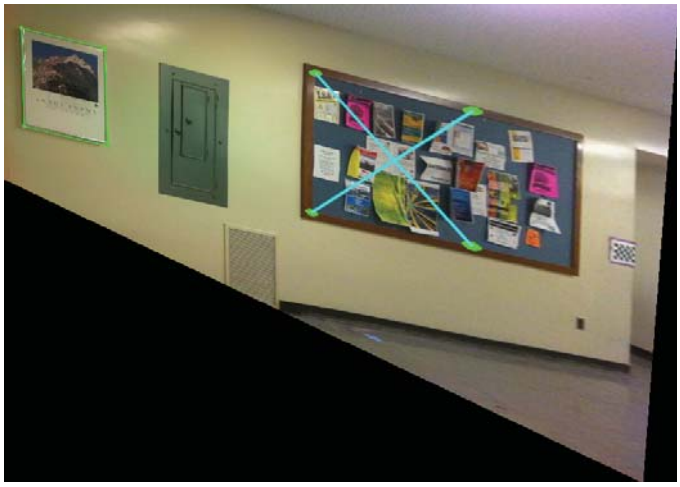

Figure 11: Step 1 result

Figure 12: Step 2 result

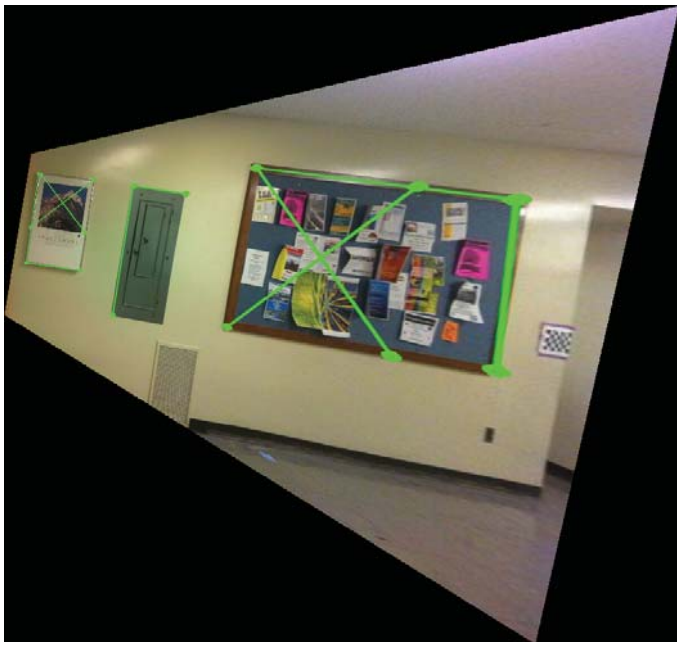

Figure 13: One Step Method result

building

Figure 14: input: building.jpg

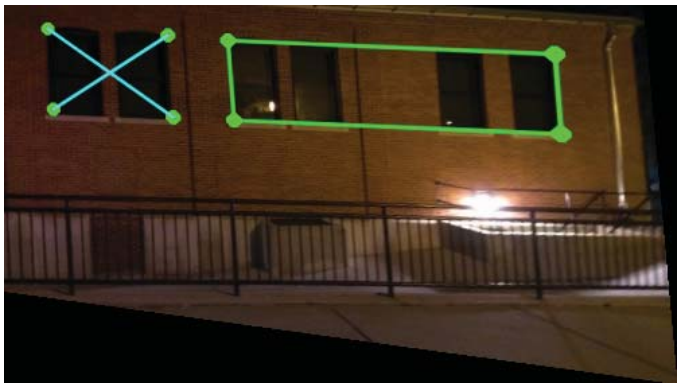

Figure 15: Step 1 result

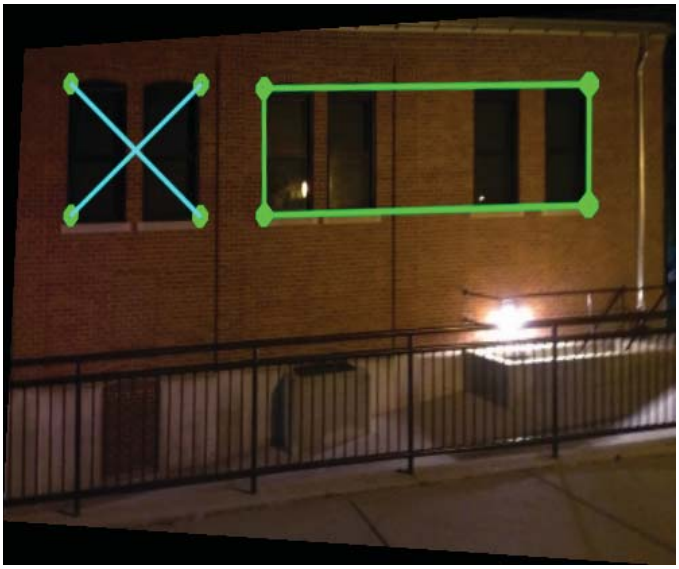

Figure 16: Step 2 result

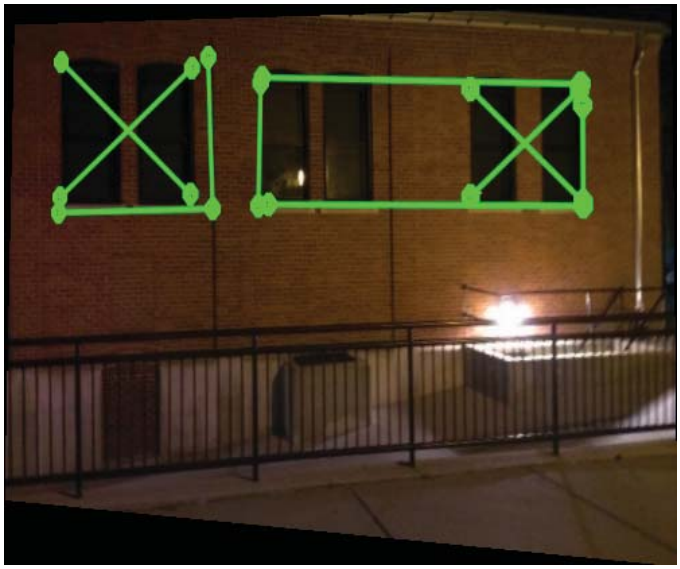

Figure 17: One Step Method Result

building2

Figure 18: input: building2.jpg

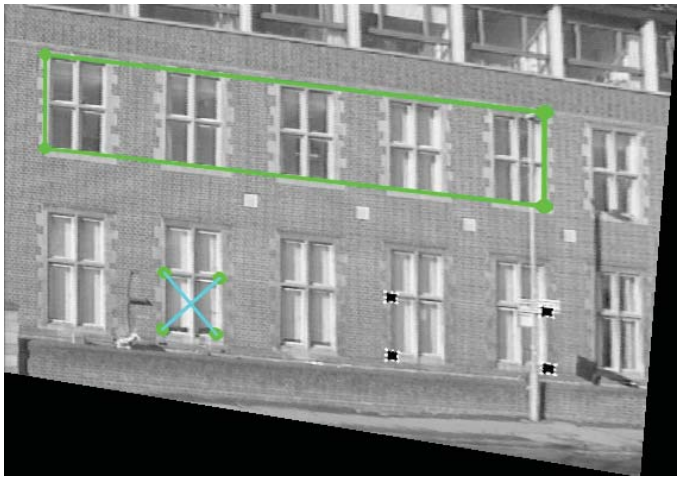

Figure 19: Step 1 result

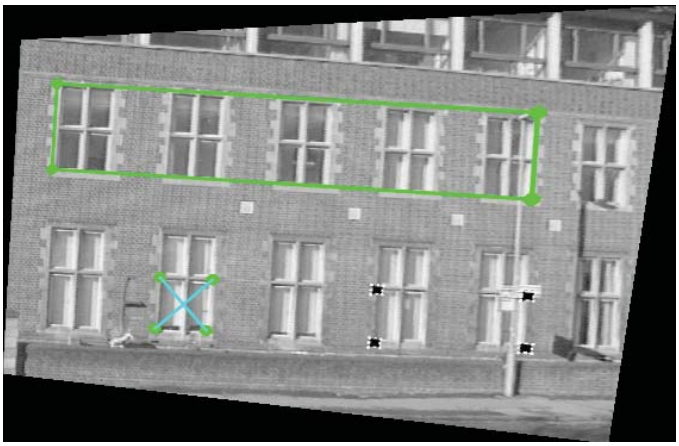

Figure 20: Step 2 Result

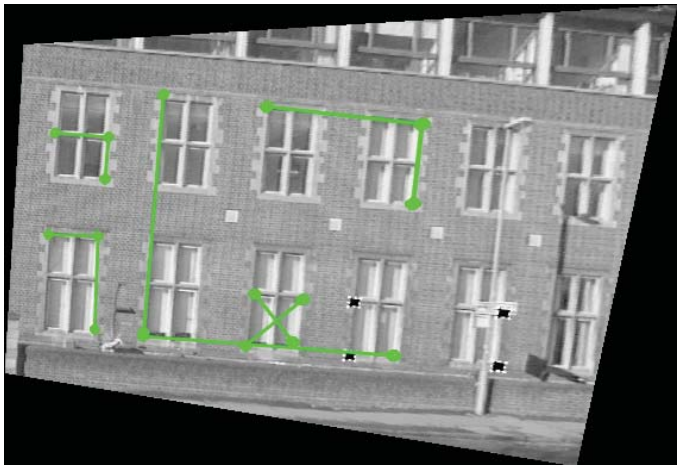

Figure 21: One Step Method Result

door

Figure 22: input: door.jpg

Figure 23: Step 1 result

Figure 24: Step 2 Result

Figure 25: One Step Method Result

door1

Figure 26: input: door1.jpg

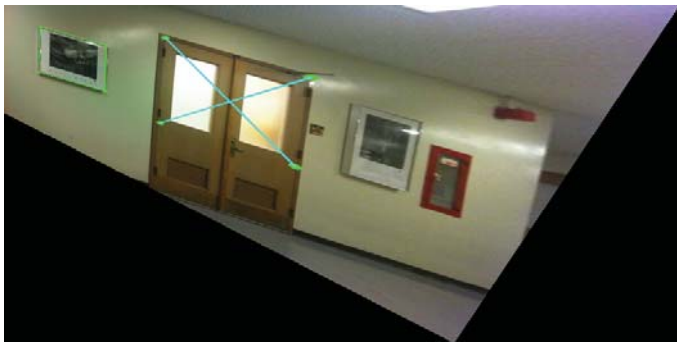

Figure 27: Step 1 result

Figure 28: Step 2 Result

Figure 29: One Step Method Result

notice

Figure 30: input: notice.jpg

Figure 31: Step 1 result

Figure 32: Step 2 result

Figure 33: One Step Method Result

tree

Figure 34: input: tree1.jpg

Figure 35: Step 1 result

Figure 36: Step 2 result

Figure 37: One Step Method Result

Observe that one step method performs worse than the other, because there are very less orthogonal lines in the image (except the picture in the middle).

tree2

Figure 38: input: tree2.jpg

Figure 39: Step 1 result

Figure 40: Step 2 result

Figure 41: One Step Method Result

7. Source Code: twostepmethod.py

```
#!/usr/bin/python
#
# Author: Sriram Karthik Badam
# Date: Sep 12, 2012
#

import sys, os
import cv
import numpy

#global variables
pointCount = 0
image = 0

"""
This function collects the input coordinates when user clicks on them
The coordinates are stores in "param" varaible which is a cvMat in this case.
"""
def on_mouse (event, x, y, flags, param):
  global pointCount
  global image

  #got a new point
  if event == cv.CV_EVENT_LBUTTONDOWN:
    point = (int(x),int(y))
    cv.Circle(image, point, 4, (0, 255, 0), 3, 8, 0)
    cv.mSet(param, 0, pointCount, x)
    cv.mSet(param, 1, pointCount, y)
    #point belong to rectangle
    if pointCount > 0 and pointCount < 4:
      x_ini = cv.mGet(param, 0, pointCount - 1 )
      y_ini = cv.mGet(param, 1, pointCount - 1 )
      point_ini = (int(x_ini),int(y_ini))
      cv.Line(image, point, point_ini, (0,255,0), 2, cv.CV_AA, 0)
    #for the fourth point an extra line has to be drawn with the first point
    if pointCount == 3:
      x_ini = cv.mGet(param, 0, 0)
      y_ini = cv.mGet(param, 1, 0)
      point_ini = (int(x_ini),int(y_ini))
      cv.Line(image, point, point_ini, (0, 255, 0), 2, cv.CV_AA, 0)
    #point belongs to the orthogonal pair
    if pointCount > 4 and pointCount < 8 and pointCount != 6:
      x_ini = cv.mGet(param, 0, pointCount-1)
      y_ini = cv.mGet(param, 1, pointCount-1)
      point_ini = (int(x_ini), int(y_ini))
      cv.Line(image, point, point_ini, (255, 255, 0), 2, cv.CV_AA, 0)

    cv.ShowImage("Input Image", image)
    pointCount = pointCount + 1

###########
"""
prints the input cvMat
"""
def printMatrix(mat):
  temp_array = numpy.asarray(mat[:,:])
  print temp_array


###########
"""
Computes the boundaries in the world plane for the image using Homography H.
```

```
"""
def findBoundaries (H, width, height):
  #computes boundaries in the projective distortion free space
  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, width-1)
  cv.mSet(boundaries_image, 0, 2, 0)
  cv.mSet(boundaries_image, 0, 3, width-1)
  cv.mSet(boundaries_image, 1, 0, 0)
  cv.mSet(boundaries_image, 1, 1, 0)
  cv.mSet(boundaries_image, 1, 2, height-1)
  cv.mSet(boundaries_image, 1, 3, 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, boundaries_image, boundaries_world)
  return boundaries_world

###########
"""
Creates the distortion free image from the input image using the computed homography
Does Bilinear Interpolation
"""
def createNewImage(image, scale, new_height, H, x_min, y_min):
  im_result =  cv.CreateImage((image.width, new_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(im_result.width):
    cv.mSet(world_coordinates, 0, 0, float(x_min+i/scale))
    for j in range(im_result.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<image.width-1 and y<image.height-1:
        #interpolation
  temp = [0,0,0]
  for k in range(3):
    temp[k] = 0
    temp[k] = temp[k]+(x-int(x))*(y-int(y))*(image[int(y+1),int(x+1)][k])
    temp[k] = temp[k]+(1.0-(x-int(x)))*(y-int(y))*(image[int(y+1),int(x)][k])
    temp[k] = temp[k]+(x-int(x))*(1.0-(y-int(y)))*(image[int(y),int(x+1)][k])
    temp[k] = temp[k]+(1.0-(x-int(x)))*(1.0-(y-int(y)))*(image[int(y),int(x)][k])

  im_result[j,i] =  temp
  return im_result

"""
main function
"""
def main():
  print "@Author: S.Karthik Badam"
  global image

  #loads image
  if len(sys.argv) > 1:
```

```
  filename = sys.argv[1]
  image = cv.LoadImage(filename, cv.CV_LOAD_IMAGE_UNCHANGED)
  if image == 0:
    print "enter a valid Image Path"
else:
  print "Enter image file path as argument"

#initializes input variables and takes input from user
point_image = cv.CreateMat(2, 8, cv.CV_64FC1)
cv.NamedWindow('Input Image', cv.CV_WINDOW_AUTOSIZE)
cv.SetMouseCallback('Input Image', on_mouse, point_image)
cv.ShowImage('Input Image', image)
cv.WaitKey()
cv.DestroyWindow('Input Image')

#print input
print "input is :"
printMatrix(point_image)

#Step 1: Eliminate Projective Distortion
#finds the lines joining the vertices of the Rectanglar Element
lines = cv.CreateMat(3, 6, cv.CV_64FC1)
point1 = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable
point2 = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable
temp_line1 = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable
temp_line2 = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable
vanishing_point1 = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable
vanishing_point2 = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable

for i in range(4):
  index1 = 2*(i%2) + int(i/2)
  index2 = (index1+1) % 4
  cv.mSet(point1, 0, 0, cv.mGet(point_image, 0, index1))
  cv.mSet(point1, 1, 0, cv.mGet(point_image, 1, index1))
  cv.mSet(point1, 2, 0, 1)

  cv.mSet(point2, 0, 0, cv.mGet(point_image, 0, index2))
  cv.mSet(point2, 1, 0, cv.mGet(point_image, 1, index2))
  cv.mSet(point2, 2, 0, 1)

  #finds cross product
  if i%2==0:
    cv.CrossProduct(point1, point2, temp_line1)
    for k in range(3):
      cv.mSet(lines, k, i, cv.mGet(temp_line1, k, 0))

  elif i%2==1:
    cv.CrossProduct(point1, point2, temp_line2)
    for k in range(3):
      cv.mSet(lines, k, i, cv.mGet(temp_line2, k, 0))

  #finds the vanishing points
  if int(i/2)==0:
    cv.CrossProduct(temp_line1, temp_line2, vanishing_point1)
  elif int(i/2)==1:
    cv.CrossProduct(temp_line1, temp_line2, vanishing_point2)

#calculates the final two orthogonal lines
for i in range(2):
  index1 = 2*i + 4
  index2 = index1 + 1
  cv.mSet(point1, 0, 0, cv.mGet(point_image, 0, index1))
  cv.mSet(point1, 1, 0, cv.mGet(point_image, 1, index1))
  cv.mSet(point1, 2, 0, 1)
```

```
    cv.mSet(point2, 0, 0, cv.mGet(point_image, 0, index2))
    cv.mSet(point2, 1, 0, cv.mGet(point_image, 1, index2))
    cv.mSet(point2, 2, 0, 1)

    cv.CrossProduct(point1, point2, temp_line1)
    for k in range(3):
      cv.mSet(lines, k, i+4, cv.mGet(temp_line1, k, 0))


#computes the vanishing line from vanishing points
vanishing_line = cv.CreateMat(3, 1, cv.CV_64FC1) #temporary variable
cv.CrossProduct(vanishing_point1, vanishing_point2, vanishing_line)
for i in range(3):
  cv.mSet(vanishing_line, i, 0, cv.mGet(vanishing_line, i, 0)/cv.mGet(vanishing_line
      , 2, 0))


#Determines H by mapping vanishing line to l_inifinity
H_proj_inverse = cv.CreateMat(3, 3, cv.CV_64FC1)
H_proj = cv.CreateMat(3, 3, cv.CV_64FC1)

cv.Zero(H_proj_inverse)
cv.mSet(H_proj_inverse, 0, 0, 1)
cv.mSet(H_proj_inverse, 1, 1, 1)
cv.mSet(H_proj_inverse, 2, 0, cv.mGet(vanishing_line, 0, 0))
cv.mSet(H_proj_inverse, 2, 1, cv.mGet(vanishing_line, 1, 0))
cv.mSet(H_proj_inverse, 2, 2, cv.mGet(vanishing_line, 2, 0))

cv.Zero(H_proj)
cv.mSet(H_proj, 0, 0, 1)
cv.mSet(H_proj, 1, 1, 1)
cv.mSet(H_proj, 2, 0, -cv.mGet(vanishing_line, 0, 0)/cv.mGet(vanishing_line, 2, 0))
cv.mSet(H_proj, 2, 1, -cv.mGet(vanishing_line, 1, 0)/cv.mGet(vanishing_line, 2, 0))
cv.mSet(H_proj, 2, 2, 1 / cv.mGet(vanishing_line, 2, 0))

print "H projective is"
printMatrix(H_proj_inverse)

H_transpose = cv.CreateMat(3, 3, cv.CV_64F)
cv.Transpose(H_proj, H_transpose)

#Removes projective distortion from the lines
cv.MatMul(H_transpose, lines, lines)



#Step 2: Affine distortion
#Mx = b is solved to x which gives the null vector
M = cv.CreateMat(2, 2, cv.CV_64FC1)
b = cv.CreateMat(2, 1, cv.CV_64FC1)

l1 = cv.mGet(lines, 0, 1) / cv.mGet(lines, 2, 1)
l2 = cv.mGet(lines, 1, 1) / cv.mGet(lines, 2, 1)
m1 = cv.mGet(lines, 0, 3) / cv.mGet(lines, 2, 3)
m2 = cv.mGet(lines, 1, 3) / cv.mGet(lines, 2, 3)

cv.mSet(M, 0, 0, l1*m1)
cv.mSet(M, 0, 1, l1*m2+l2*m1)
cv.mSet(b, 0, 0, -l2*m2)

l1 = cv.mGet(lines, 0, 4) / cv.mGet(lines, 2, 4)
l2 = cv.mGet(lines, 1, 4) / cv.mGet(lines, 2, 4)
m1 = cv.mGet(lines, 0, 5) / cv.mGet(lines, 2, 5)
```

```
m2 = cv.mGet(lines, 1, 5) / cv.mGet(lines, 2, 5)

cv.mSet(M, 1, 0, l1*m1)
cv.mSet(M, 1, 1, l1*m2+l2*m1)
cv.mSet(b, 1, 0, -l2*m2)

#finding S
x = cv.CreateMat(2, 1, cv.CV_64FC1)
cv.Solve(M, b, x, cv.CV_LU)
S = cv.CreateMat(2, 2, cv.CV_64FC1)
for i in range(2):
  for j in range(2):
    e1 = (i==1) and (j==0)
    e2 = (i==0) and (j==1)
    if e1 or e2:
cv.mSet(S, i, j, cv.mGet(x, 1, 0))
    elif i == 0 and j == 0:
cv.mSet(S, i, j, cv.mGet(x, 0, 0))
    else:
cv.mSet(S, i, j, 1)

#finding A
#S = UDVT  where VT-> U transpose
#A = U D^0.5 UT
D_1 = cv.CreateMat(2, 2, cv.CV_64FC1)
D_pow_half = cv.CreateMat(2, 2, cv.CV_64FC1)
U_1 = cv.CreateMat(2, 2, cv.CV_64FC1)
U_D = cv.CreateMat(2, 2, cv.CV_64FC1)
U_Transpose = cv.CreateMat(2, 2, cv.CV_64FC1) #temp variable
A = cv.CreateMat(2, 2, cv.CV_64FC1)

print "S is"
printMatrix(S)

cv.SVD(S, D_1, U_1, None, 0)
cv.Pow(D_1, D_pow_half, 0.5)
cv.MatMul(U_1, D_pow_half, U_D)
cv.Transpose(U_1, U_Transpose)
cv.MatMul(U_D, U_Transpose, A)

#check if AA^T is S sometimes it maynot if the input is not good enough, for example
#line in orthogonal pair is parallel to one of the sides of the rectangle.
temp = cv.CreateMat(2, 2, cv.CV_64FC1)
cv.GEMM(A, A, 1.0, None, 0, temp, cv.CV_GEMM_B_T)
print "AA^T is "
printMatrix(temp)


#Computes homography that removes affine distortion
H_affine_inverse = cv.CreateMat(3, 3, cv.CV_64FC1)
H_affine = cv.CreateMat(3, 3, cv.CV_64FC1)

cv.Zero(H_affine)
for i in range(2):
  for j in range(2):
    cv.mSet(H_affine, i, j, cv.mGet(A, i, j))
cv.mSet(H_affine, 2, 2, 1)

cv.Invert(H_affine, H_affine_inverse, cv.CV_LU)
print "H affine is"
printMatrix(H_affine_inverse)

H_inverse = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.MatMul(H_proj_inverse, H_affine_inverse, H_inverse)#removes both distortions!
```

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

  boundaries = findBoundaries(H_proj_inverse, image.width, image.height)
  #checks boundaries in world coordinates to get the min and max values
  x_min = 1e50
  y_min = 1e50
  x_max = 0
  y_max = 0
  for i in range(4):
    x = cv.mGet(boundaries, 0, i)/cv.mGet(boundaries, 2, i)
    y = cv.mGet(boundaries, 1, i)/cv.mGet(boundaries, 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

  #finds the scale and in turn the width and height of final image
  scale = float(image.width/(x_max-x_min))
  new_height = int((y_max-y_min)*scale)

  #creates a projective distortion free image
  Result_1 = createNewImage(image, scale, new_height, H_proj, x_min, y_min)
  cv.SaveImage("projection_free_image.png", Result_1)

  #applys H_inverse to get boundaries in world plane
  boundaries_world = findBoundaries(H_inverse, image.width, image.height)

  #checks boundaries in world coordinates to get the min and max values
  x_min = 1e100
  y_min = 1e100
  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

  #finds scale and in turn the new width, height
  scale = float(image.width/(x_max-x_min))
  new_height = int((y_max-y_min)*scale)

  #creates the final distortion free image
  Result_final = createNewImage(image, scale, new_height, H, x_min,y_min)
  cv.SaveImage("Final_image.png", Result_final)

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

8. **Source Code: onestepmethod.py**

```
#!/usr/bin/python
#
# Author: Sriram Karthik Badam
# Date: Sep 15, 2012
#

import sys, os
import cv
import numpy

#global variables
pointCount = 0
image = 0

"""
This function collects the coordinates of the points selected by the user
and stores into variable "param" which is a cvMat in this case.
"""
def on_mouse (event, x, y, flags, param):
  global pointCount
  global image

  #got a new point
  if event == cv.CV_EVENT_LBUTTONDOWN:
    point = (int(x),int(y))
    cv.Circle(image, point, 4, (0, 255, 0), 3, 8, 0)
    cv.mSet(param, 0, pointCount, int(x))
    cv.mSet(param, 1, pointCount, int(y))
    cv.mSet(param, 2, pointCount, 1)
    if pointCount % 2 != 0 and pointCount < 20:
      x_ini = cv.mGet(param, 0, pointCount - 1 )
      y_ini = cv.mGet(param, 1, pointCount - 1 )
      point_ini = (int(x_ini),int(y_ini))
      cv.Line(image, point, point_ini, (0,255,0), 2, cv.CV_AA, 0)

    cv.ShowImage("Input Image", image)
    pointCount = pointCount + 1

###########
"""
prints a matrix
"""
def printMatrix(mat):
  temp_array = numpy.asarray(mat[:,:])
  print temp_array


############
"""
finds the world co-ordinates of the boundaries of the image
"""
def findBoundaries (H_inv, width, height):
  #computes boundaries in the distortion free space
  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, width-1)
  cv.mSet(boundaries_image, 0, 2, 0)
  cv.mSet(boundaries_image, 0, 3, width-1)
  cv.mSet(boundaries_image, 1, 0, 0)
  cv.mSet(boundaries_image, 1, 1, 0)
  cv.mSet(boundaries_image, 1, 2, height-1)
```

```
  cv.mSet(boundaries_image, 1, 3, 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_inv, boundaries_image, boundaries_world)
  return boundaries_world

###########
"""
performs the transform to obtain the distortion free image
using bilinear interpolation
"""

def createNewImage(image, scale, new_height, H, x_min, y_min):
  im_result =  cv.CreateImage((image.width, new_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(im_result.width):
    cv.mSet(world_coordinates, 0, 0, float(x_min+i/scale))
    for j in range(im_result.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<image.width-1 and y<image.height-1:
        temp = [0,0,0]
  for k in range(3):
    #interpolate
    temp[k] = 0
    temp[k] = temp[k]+(x-int(x))*(y-int(y))*(image[int(y+1),int(x+1)][k])
    temp[k] = temp[k]+(1.0-(x-int(x)))*(y-int(y))*(image[int(y+1),int(x)][k])
    temp[k] = temp[k]+(x-int(x))*(1.0-(y-int(y)))*(image[int(y),int(x+1)][k])
    temp[k] = temp[k]+(1.0-(x-int(x)))*(1.0-(y-int(y)))*(image[int(y),int(x)][k])

  im_result[j,i] =  temp
  return im_result

"""
main function
"""
def main():
  print "@Author: S.Karthik Badam"
  global image

  #loads image
  if len(sys.argv) > 1:
    filename = sys.argv[1]
    image = cv.LoadImage(filename, cv.CV_LOAD_IMAGE_UNCHANGED)
    if image == 0:
      print "enter a valid Image Path"
  else:
    print "Enter image file path as argument"

  #initializes input variables and takes input from user
  point_image = cv.CreateMat(3, 20, cv.CV_64FC1)
  cv.NamedWindow('Input Image', cv.CV_WINDOW_AUTOSIZE)
  cv.SetMouseCallback('Input Image', on_mouse, point_image)
```

```
cv.ShowImage('Input Image', image)
cv.WaitKey()
cv.DestroyWindow('Input Image')

#print input
print "input is :"
printMatrix(point_image)

point1 = cv.CreateMat(3, 1, cv.CV_64FC1)
point2 = cv.CreateMat(3, 1, cv.CV_64FC1)
point3 = cv.CreateMat(3, 1, cv.CV_64FC1)
point4 = cv.CreateMat(3, 1, cv.CV_64FC1)

l = cv.CreateMat(3, 1, cv.CV_64FC1)
m = cv.CreateMat(3, 1, cv.CV_64FC1)
A = cv.CreateMat(5, 5, cv.CV_64FC1)
x = cv.CreateMat(5, 1, cv.CV_64FC1)
b = cv.CreateMat(5, 1, cv.CV_64FC1)

#constructs the matrix A from the given 5 pair of
#orthogonal lines
#Ax=b is solved to get the null vector

for i in range(5):
  for j in range(3):
    cv.mSet(point1, j, 0, cv.mGet(point_image, j, 4*i))
    cv.mSet(point2, j, 0, cv.mGet(point_image, j, 4*i+1))
    cv.mSet(point3, j, 0, cv.mGet(point_image, j, 4*i+2))
    cv.mSet(point4, j, 0, cv.mGet(point_image, j, 4*i+3))

  cv.CrossProduct(point1, point2, l)
  cv.CrossProduct(point3, point4, m)
  #fill a row in A
  cv.mSet(A, i, 0, cv.mGet(l, 0, 0)*cv.mGet(m, 0, 0))
  cv.mSet(A, i, 1, (cv.mGet(l, 0, 0)*cv.mGet(m, 1, 0) + cv.mGet(l, 1, 0)*cv.mGet(m,
      0, 0))/2)
  cv.mSet(A, i, 2, cv.mGet(l, 1, 0)*cv.mGet(m, 1, 0))
  cv.mSet(A, i, 3, (cv.mGet(l, 0, 0)*cv.mGet(m, 2, 0) + cv.mGet(l, 2, 0)*cv.mGet(m,
      0, 0))/2)
  cv.mSet(A, i, 4, (cv.mGet(l, 1, 0)*cv.mGet(m, 2, 0) + cv.mGet(l, 2, 0)*cv.mGet(m,
      1, 0))/2)
  cv.mSet(b, i, 0, -cv.mGet(l, 2, 0)*cv.mGet(m, 2, 0))

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

#constructs the dual degenerate conic

C = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.mSet(C, 0, 0, cv.mGet(x, 0, 0))
cv.mSet(C, 0, 1, cv.mGet(x, 1, 0)/2)
cv.mSet(C, 0, 2, cv.mGet(x, 3, 0)/2)
cv.mSet(C, 1, 0, cv.mGet(x, 1, 0)/2)
cv.mSet(C, 1, 1, cv.mGet(x, 2, 0))
cv.mSet(C, 1, 2, cv.mGet(x, 4, 0)/2)
cv.mSet(C, 2, 0, cv.mGet(x, 3, 0)/2)
cv.mSet(C, 2, 1, cv.mGet(x, 4, 0)/2)
cv.mSet(C, 2, 2, 1)

#obtains H from C
print "C is "
printMatrix(C)
#finds K where KK^T = S
#S = UDVT (VT means V Transpose)
#K = U(D^0.5)UT
```

Note - The author has solved for the 6 variables in the dual degenrate conic using 5 equations, by fixing one of the six variables namely 'f', to be 1 and thus converting it to a determined system of equations.

This would work only if 'f' is guaranteed to be non-zero as would be the case if you had significant projective distortion.However, if the distortion in the image is purely affine, then 'f' would be zero and the above approach would not work.

The correct way to solve for six variables using 5 homogeneous equations would be to solve for the null space of the'A' matrix. Use SVD to find the eigenvectors and then choose the eigenvector corresponding to the smallest eigenvalue.

```
S = cv.CreateMat(2, 2, cv.CV_64FC1)
K = cv.CreateMat(2, 2, cv.CV_64FC1)
for i in range(2):
  for j in range(2):
    cv.mSet(S, i, j, cv.mGet(C, i, j))

U = cv.CreateMat(2, 2, cv.CV_64FC1)
D_2 = cv.CreateMat(2, 2, cv.CV_64FC1)
D_pow_half = cv.CreateMat(2, 2, cv.CV_64FC1)
U_D = cv.CreateMat(2, 2, cv.CV_64FC1)

cv.SVD(S, D_2, U, None, 0)
cv.Pow(D_2, D_pow_half, 0.5)
cv.MatMul(U, D_pow_half, U_D)
cv.GEMM(U_D, U, 1.0, None, 0, K, cv.CV_GEMM_B_T)

#solves Kv = B where B is the corresponding column values in C
v = cv.CreateMat(2, 1, cv.CV_64FC1)
B = cv.CreateMat(2, 1, cv.CV_64FC1)
cv.mSet(B, 0, 0, cv.mGet(C, 0, 2))
cv.mSet(B, 1, 0, cv.mGet(C, 1, 2))
print "K is"
printMatrix(K)

cv.Solve(K, B, v, cv.CV_LU)

H = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.Zero(H)
for i in range(2):
  for j in range(2):
    cv.mSet(H, i, j, cv.mGet(K, i, j))
cv.mSet(H, 2, 0, cv.mGet(v, 0, 0))
cv.mSet(H, 2, 1, cv.mGet(v, 1, 0))
cv.mSet(H, 2, 2, 1.0)

H_inverse = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.Invert(H, H_inverse, cv.CV_LU)
print " H is "
printMatrix(H)
print "H inverse is"
printMatrix(H_inverse)

C_star =  cv.CreateMat(3, 3, cv.CV_64FC1)
cv.Zero(C_star)
cv.mSet(C_star, 0, 0, 1)
cv.mSet(C_star, 1, 1, 1)
HC = cv.CreateMat(3, 3, cv.CV_64FC1)
HCHT = cv.CreateMat(3, 3, cv.CV_64FC1)
cv.MatMul(H, C_star, HC)
cv.GEMM(HC, H, 1.0, None, 0, HCHT, cv.CV_GEMM_B_T)

print "HCHT"
printMatrix(HCHT)
#check if this the same as C ., sometimes it maynot which means you didnt
#select a perfect input. Some lines can be parallel in which case you have
#supplied less input than required

#creates distortion free final image
boundaries_world = findBoundaries(H_inverse, image.width, image.height)
#checks boundaries in world coordinates to get the min and max values
x_min = 1e100
y_min = 1e100
x_max = -1e100
y_max = -1e100
```

```
  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

  #finds the scale and in turn the new width, height
  scale = float(image.width/(x_max-x_min))
  new_height = int((y_max-y_min)*scale)

  #creates the final distortion free image
  Result_final = createNewImage(image, scale, new_height, H, x_min,y_min)
  cv.SaveImage("program_2_Final_image.png", Result_final)

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