Prateek Gupta
gupta198@purdue.edu
ECE 661 Homework 3

Overview

In this report, we will remove projective and affine distortion from images in three different ways: the point to point method, two step method, and one step method. The point to point method entails constructing a homography using a set of four points on the distorted image and the corresponding set of points in the corrected image. In the two-step method we will first remove only the projective distortion, then remove the affine distortion. The one step method allows us to construct a homography using the dual degenerate conic. Each of these will be discussed in more detail and the results for each will be presented.

Each of these methods will employ the same basic strategy of first calculating the homography, then applying the homography. We will first discuss how to apply the homography since that is common to all methods. Then there will be a discussion on how the homography for each method is calculated, and results of its application will be shown on the three images provided and on two additional images found on the internet.

Applying the Homography:

Once the homography is calculate (which will be discussed as we go through each method), it can be applied to the original image for the distortion to be removed. We essentially need to move each pixel from its original position to a new position. For each pixel, we will take the coordinates of the original and augment 1. This will give us the coordinates of the pixel *x* in the representational space, allowing us to apply the 3x3 homography **H** to get *x'* in the following manner:

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

Here *x'* is the result of applying the homography and gives the coordinates in the representational space. In order to get the coordinates in the physical space, we will divide x_1 and x_2 by x_3. We will then replace that pixel on the destination image with pixel we started with.

The Distorted Images

We will use the following five images and remove the distortion on each feature discussed using the each of strategies discussed above.

Image 1: The windows are distorted. We will attempt to straighten out one of them

Image 2: The lower window on the house is distorted. We attempt to straighten this out

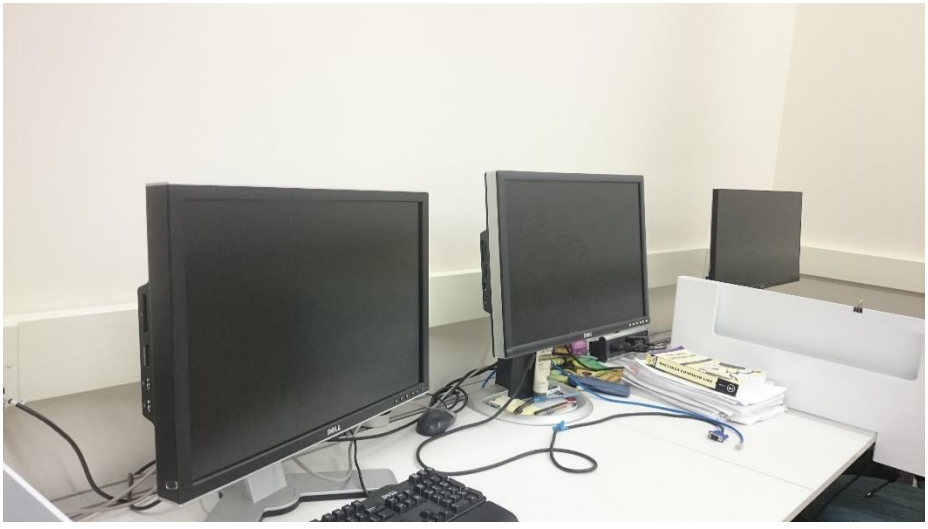


Image 3: The computer screen in the middle is not straight. The goal will be to straighten it back out

Image 4: This is a picture of a poster someone on campus created and is selling. It appears he believes that no one can reproduce it themselves since the picture is taken at an angle. Obviously, he does not know about this class. We will attempt to straighten it.

Image 5: Here we have an advertisement for a television

We will discuss the three ways to straighten the images in detail and show the results of each method.

Point to Point Method

This strategy is very similar to what was done in the previous report, so much of this section will be adapted from there.

The homography in the point to point method is calculated by selecting at least four mappings of points. So, we pick the points of the corners of a feature on the distorted image and determine corresponding points by looking at the measurements of the feature. This will give us a mapping of points we currently have and where we want those points to end up.

$(x_1, y_1) \rightarrow (x_1', y_1')$
$(x_2, y_2) \rightarrow (x_2', y_2')$
$(x_3, y_3) \rightarrow (x_3', y_3')$
$(x_4, y_4) \rightarrow (x_4', y_4')$

Using these points, we will calculate the homography H

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

The homography H can be multiplied to all points in the 2-dimensional representational space to get the distorted image. The vector *x* will represent the original point, and the vector *x'* will represent the distorted point. We get *x'* by multiplying *x* with the homography H.

$$x' = \boldsymbol{H}x$$
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad x' = \begin{bmatrix} x_1{}' \\ x_2{}' \\ x_3{}' \end{bmatrix}$$

Using the equation for finding the distorted points, we can construct a system of three equations:

$$x_1' = h_{11}x_1 + h_{12}x_2 + h_{13}x_3$$
$$x_2' = h_{21}x_1 + h_{22}x_2 + h_{23}x_3$$
$$x_3' = h_{31}x_1 + h_{32}x_2 + h_{33}x_3$$

Now, in order to find the coordinates of the distorted point, we are only interested in the ratios. So, the distorted x coordinate (x') can be found by computing the ratio between x_1 and x_3, and the distorted y coordinate (y') can be found by computing the ratio between x_2 and x_3.

$$x' = \frac{x_1'}{x_3'} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

$$y' = \frac{x_1'}{x_3'} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

Again, since we are only concerned with the ratios, we can set x_3 and h_{33} to 1. We will also change x_1 to x and x_2 to y, so that we can accommodate multiple sets of equations without getting too confused with the subscripts

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

And we can manipulate these algebraically to get

$$x' = h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx'$$
$$y' = h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy'$$

In order to solve for all values h, we need to utilize four pairs of points to construct 4 sets of equations, giving us a total of 8 equations with 8 unknowns. This can be represented by

$$\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}$$

We can now solve for all values h easily by plugging in the four pairs of points and using matrix algebra.

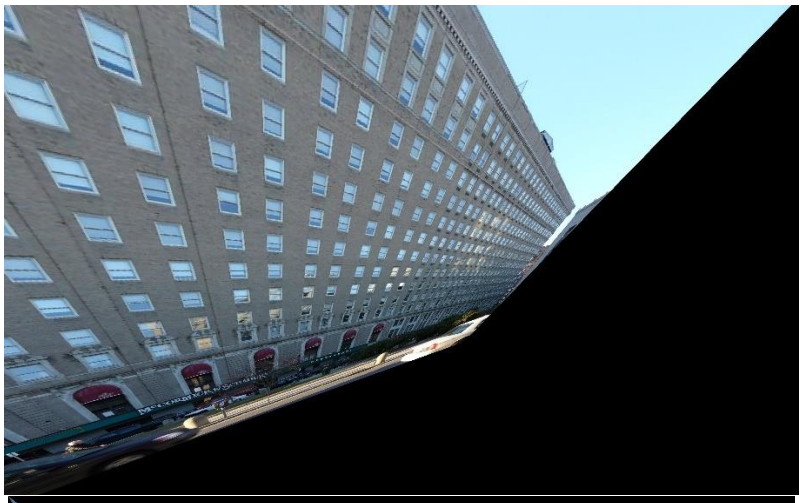
$$\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix}^{-1} \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}$$

This will give us the homography matrix

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

We can now apply this homography using the method discussed above. In order to do this, we will use the following point correspondences, assuming one centimeter is represented by one pixel:

Image 1:	*Image 2:*	*Image 3:*
(644, 500) → (0, 0)	(383, 577) → (0, 0)	(644, 500) → (0, 0)
(663, 503) → (75, 0)	(593, 552) → (84, 0)	(663, 503) → (75, 0)
(643, 501) → (0, 85)	(381, 834) → (0, 74)	(643, 531) → (0, 85)
(663, 535) → (75, 85)	(607, 922) → (84, 74)	(663, 535) → (75, 85)



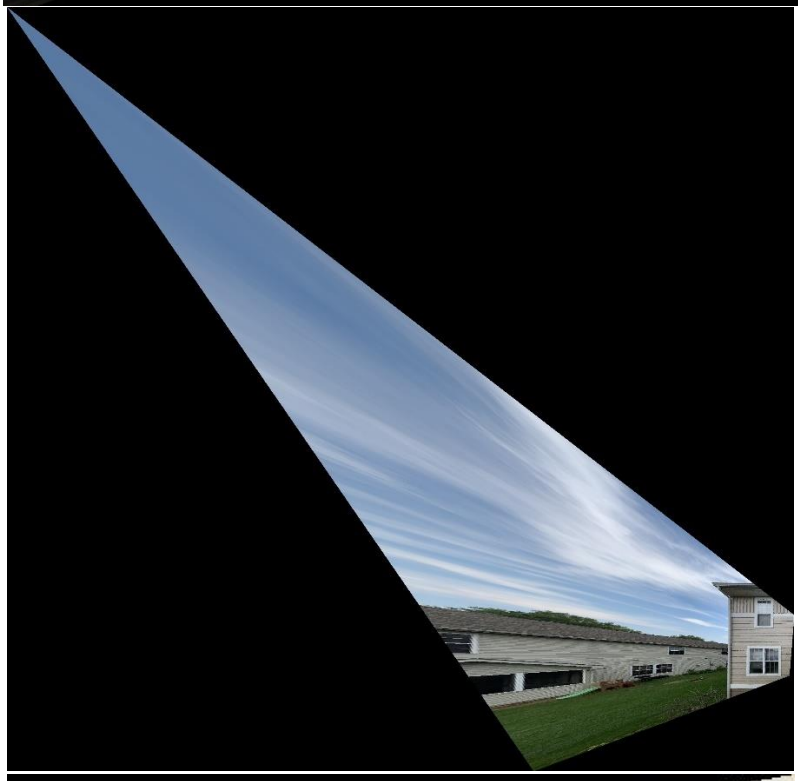


Image 4:
(234, 199) → (0, 0)
(626, 233) → (220, 0)
(256, 813) → (0, 280)
(647, 739) → (220, 280)

Image5:
(625, 521) → (0, 0)
(3851, 52) → (188, 0)
(625, 2418) → (0, 106)
(3852, 2869) → (188, 106)

Two Step Method

In this method, we will start by first removing the projective distortion using the vanishing lines technique. Once we do this, we will be left with only an affine distortion. We will then need to remove this.

In order to remove the projective distortion, we must find the vanishing line. We will start by finding the equations of the two sets of parallel lines. The equation for a line is found by taking the cross product of the two coordinates in the representational space

$$l = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Here x and y represent the two points. Once we find the equations for each set of parallel lines, we can find their intersection by taking the cross product of the two lines

$$p = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} \times \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}$$

Here *l* and *m* represent the equations for the two lines. Once we have the coordinates for the intersection of both sets of parallel lines, we take their cross product again to find the equation of the vanishing line *v*. We then need to push this line to infinity. This can be done by applying the following homography:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ v_1 & v_2 & v_3 \end{bmatrix}$$

Where v_1, v_2, and v_3 are the parameters of the vanishing line v. This homography can be applied in the same method as discussed above. The following are the output.


RIP
NEON
CACTUS
1992-2020
THANKS
FOR THE
MEMORIES
WEST LAFAYETTE, IN
THE PIANO MAN
DANCING QUEEN
HAIL PURDUE
DUELING PIANOS
JELLO SHOTS
TAKE ME HOME, COUNTRY ROADS
DIXIELAND DELIGHT
MR. BRIGHTSIDE
SOCIAL!
BREAKFAST CLUB
PURDUE UNIVERSITY
QUARANTINE CACTUS
SWEET CAROLINE


SAMSUNG
QLED

These now have an affine distortion. In order to remove this, we know that the angles are not preserved. So, the simplest thing to do is to straighten out the 90-degree angles. We will find the equations for a set of orthogonal lines in the world plane by taking the cross product of two points on each line, similar to the method above. Suppose we have one line *L* and line *M.* The cosine of the angle between the is:

$$\cos\theta = \frac{L^T C_\infty^* M}{\sqrt{(L^T C_\infty^* L)(M^T C_\infty^* M)}}$$

$$C_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We are interested in finding the homography that results in the distorted image. We can obtain this homography by looking at the equations $L = H^T L'$ and $M = H^T M'$. We also know can set cosθ to 0 since we know that that θ is 90. This causes the denominator of the previous equation to drop out. Making the appropriate substitution for *L* and *M*:

$$L'^T H C_\infty^* H^T M' = 0$$

If we set $H = \begin{bmatrix} AA^T & 0 \\ 0 & 0 \end{bmatrix}$

$$[l_1' \quad l_2' \quad l_3'] \begin{bmatrix} AA^T & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix} = 0$$

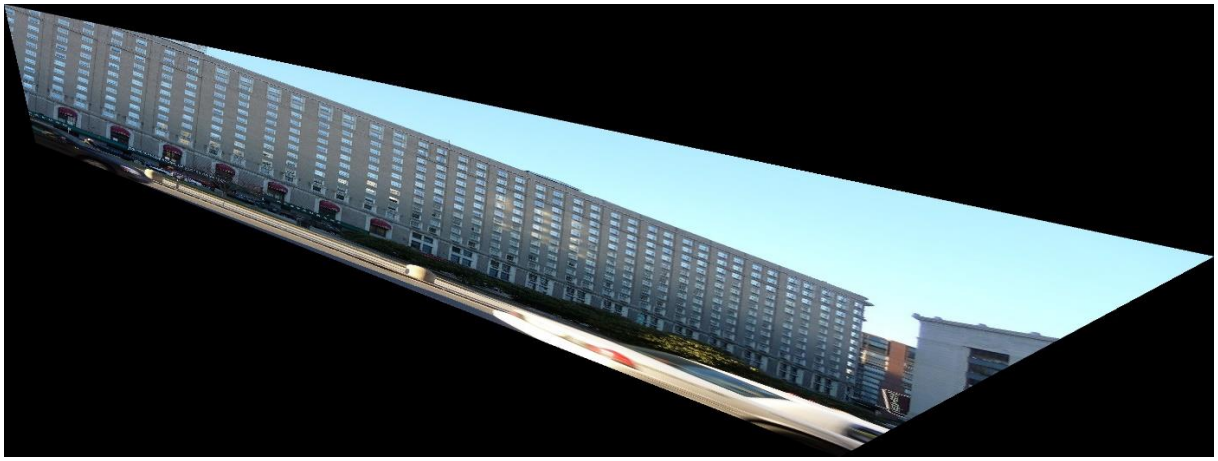
Now let $S = AA^T \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}$. This makes out equation

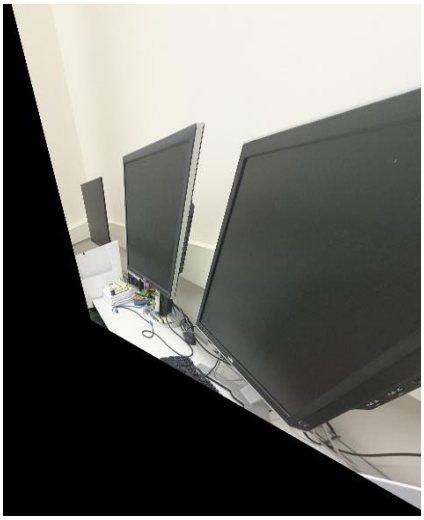
$$s_{11} m_1' l_1' + s_{12}(l_1' m_2' + l_2' m_1') + s_{22} l_2' m_2' = 0$$

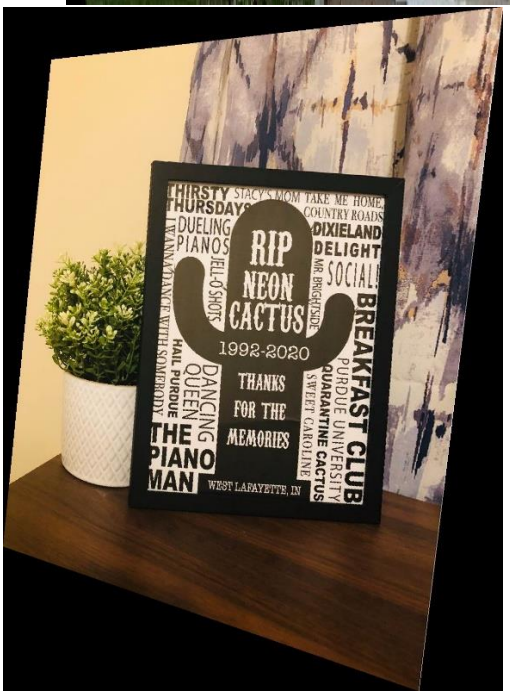
As always, we only care about the ratios, so we can set $s_{22} = 1$. We then use the single value decomposition to get A. We can use this value A to easily get the homography. It will be applied to the distorted images in the same way as above:

$$S = AA^T = VD^2V^T$$
$$A = VDV^T$$

The results of applying this homography are shown below:






RIP
NEON
CACTUS
1992-2020
THANKS
FOR THE
MEMORIES
BREAKFAST CLUB
DIXIELAND
DELIGHT
SOCIAL!
DUELING
PIANOS
JELL-O SHOT
DANCING
QUEEN
THE
PIANO
MAN
WEST LAFAYETTE, IN


SAMSUNG
QLED

One Step Method

In this process, we will remove the projective and affine distortions simultaneously. In order to do this, we will use the dual degenerate conic. Let us start with the general homography

$$H = \begin{bmatrix} \mathrm{A} & 0 \\ \mathrm{v^t} & 1 \end{bmatrix}$$

Using the result obtained in the precious part and the following

$$C_\infty^{*\prime} = HC_\infty^* H^T = \begin{bmatrix} AA^T & Av \\ v^T A^T & v^T v \end{bmatrix} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

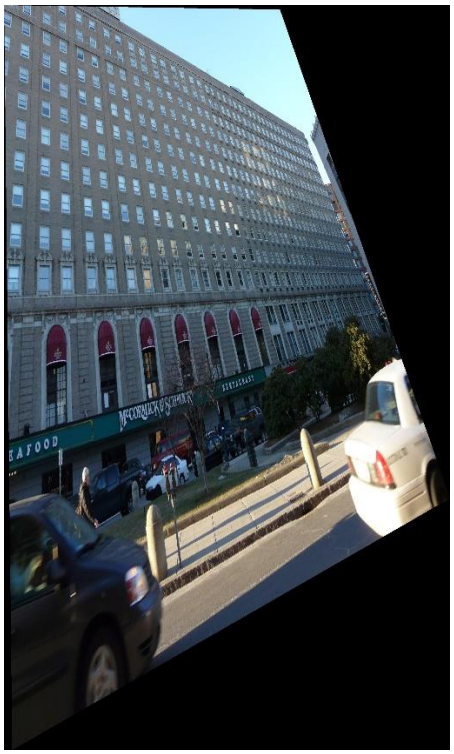
We obtain

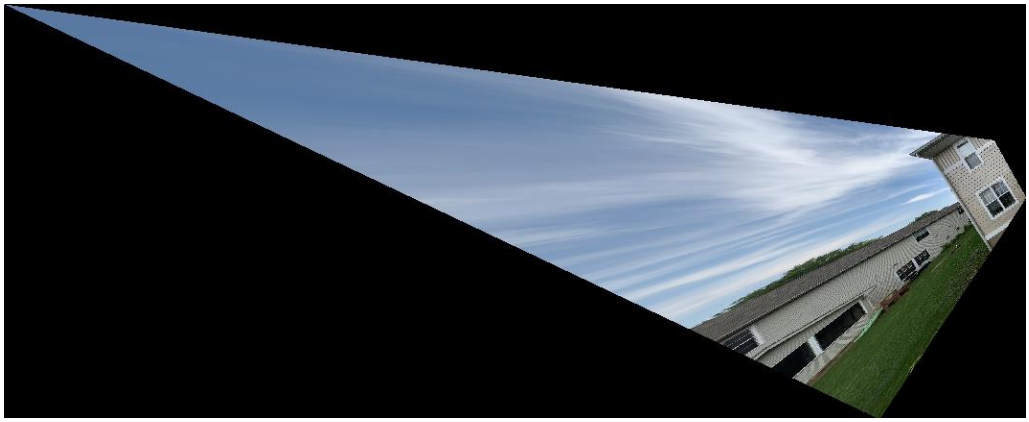
$$L^T C_\infty^{*\prime} M' = 0 = [l_1' \quad l_2' \quad l_3'] \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix}$$

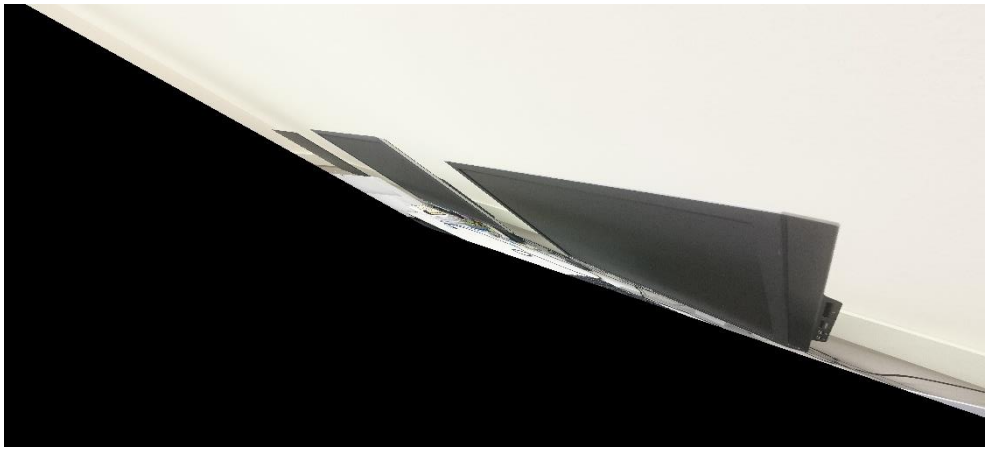
Similar to before, we will set *f* to 1 since we are only concerned with the ratios. This means we will have 5 unknowns, so we will need 5 equations. Multiplying out the above and rewriting in matrix form:

$$\begin{bmatrix} l_1'm_1' & \dfrac{l_1'm_2' + l_2'm_1'}{2} & l_2'm_1' & \dfrac{l_1'm_3' + l_3'm_1'}{2} & \dfrac{l_2'm_3' + l_3'm_2'}{2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = -l_3'm_3'$$

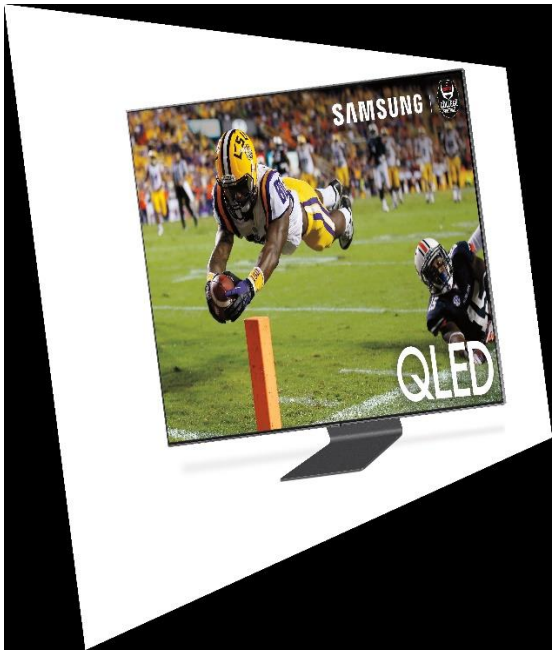
We can use matrix algebra to simply solve for all five unknowns. Once we have them, we can apply SVD to find the homography. It will then be applied the same as in the other two sections. Here are the results:








THIRSTY THURSDAYS
STACY'S MOM
TAKE ME HOME, COUNTRY ROADS
DUELING PIANOS
DIXIELAND DELIGHT
JELL-O SHOTS
MR. BRIGHTSIDE
SOCIAL!
RIP NEON CACTUS
1992-2020
THANKS FOR THE MEMORIES
BREAKFAST CLUB
I WANNA DANCE WITH SOMEBODY
HAIL PURDUE
DANCING QUEEN
PURDUE UNIVERSITY
QUARANTINE CACTUS
SWEET CAROLINE
THE PIANO MAN
WEST LAFAYETTE, IN


SAMSUNG
QLED

Code for Point to Point Method:

```
import numpy as np
import cv2

def applyHomography(img, H):
    dIm = np.array([[0,0,img.shape[0],img.shape[0]],[0, img.shape[1], 0, img.shape[1]], [1, 1, 1, 1]])
    dImNew = np.matmul(H, dIm)
    dImNew[0] = dImNew[0]/dImNew[2]
    dImNew[1] = dImNew[1]/dImNew[2]

    minH = int(np.amin(dImNew[0]))
    maxH = int(np.amax(dImNew[0]))
    minW = int(np.amin(dImNew[1]))
    maxW = int(np.amax(dImNew[1]))
    imNew = np.zeros((maxH - minH, maxW - minW,3))
    invH = np.linalg.inv(H)

    for row in range(imNew.shape[0]):
        for col in range(imNew.shape[1]):
            coord = np.matmul(invH,[[row + minH], [col + minW], [1]])
            coord /= coord[2]
            if coord[0]<img.shape[0] and coord[0]>=0 and coord[1]<img.shape[1] and coord[1]>=0:
                imNew[row][col] = img[int(coord[0])][int(coord[1])]

    return imNew

def getHomography(im1Pts, im2Pts):
    px = im1Pts[0, 1]
    py = im1Pts[0, 0]
    qx = im1Pts[1, 1]
    qy = im1Pts[1, 0]
    rx = im1Pts[2, 1]
    ry = im1Pts[2, 0]
    sx = im1Pts[3, 1]
    sy = im1Pts[3, 0]

    pxp = im2Pts[0, 1]
    pyp = im2Pts[0, 0]
    qxp = im2Pts[1, 1]
    qyp = im2Pts[1, 0]
    rxp = im2Pts[2, 1]
    ryp = im2Pts[2, 0]
    sxp = im2Pts[3, 1]
    syp = im2Pts[3, 0]
```

```
    t = np.array([[pxp],[pyp],[qxp],[qyp],[rxp],[ryp],[sxp],[syp]])
    P = np.array([[px, py, 1, 0, 0, 0, -px*pxp, -py*pxp],
                  [0, 0, 0, px, py, 1, -px*pyp, -py*pyp],
                  [qx, qy, 1, 0, 0, 0, -qx*qxp, -qy*qxp],
                  [0, 0, 0, qx, qy, 1, -qx*qyp, -qy*qyp],
                  [rx, ry, 1, 0, 0, 0, -rx*rxp, -ry*rxp],
                  [0, 0, 0, rx, ry, 1, -rx*ryp, -ry*ryp],
                  [sx, sy, 1, 0, 0, 0, -sx*sxp, -sy*sxp],
                  [0, 0, 0, sx, sy, 1, -sx*syp, -sy*syp]])
    pInv = np.linalg.inv(P)
    h = pInv.dot(t)
    H = np.array([[h[0, 0], h[1, 0], h[2, 0]], [h[3, 0], h[4, 0], h[5, 0]], [h[6, 0], h[7, 0], 1]])
    return H

im1 = cv2.imread("hw3_Task1_Images/Images/Img1.JPG")
dest1Corners = np.array([[0,0], [75, 0], [0,85], [75,85]])
im1Corners = np.array([[644, 500], [663, 503], [643, 531], [663, 535]])
H1 = getHomography(im1Corners,dest1Corners)
cv2.imwrite('Results/Pt2Pt1.jpg', applyHomography(im1, H1))

im2 = cv2.imread("hw3_Task1_Images/Images/Img2.jpeg")
dest2Corners = np.array([[0,0], [84, 0], [0,74], [84,74]])
im2Corners = np.array([[383, 577], [593, 552], [381, 834], [607, 922]])
H2 = getHomography(im2Corners,dest2Corners)
cv2.imwrite('Results/Pt2Pt2.jpg', applyHomography(im2, H2))

im3 = cv2.imread("hw3_Task1_Images/Images/Img3.JPG")
dest3Corners = np.array([[0,0], [55, 0], [0,36], [55,36]])
im3Corners = np.array([[2063, 700], [2665, 718], [2097, 1479], [2695, 1334]])
H3 = getHomography(im3Corners,dest3Corners)
cv2.imwrite('Results/Pt2Pt3.jpg', applyHomography(im3, H3))

im1 = cv2.imread("hw3_Task1_Images/Images/Own1.jpg")
dest1Corners = np.array([[0, 0], [220, 0], [0,280], [220,280]])
im1Corners = np.array([[234, 199], [626, 233], [256, 813], [647, 739]])
H1 = getHomography(im1Corners,dest1Corners)
cv2.imwrite('Results/Pt2Pt4.jpg', applyHomography(im1, H1))

im2 = cv2.imread("hw3_Task1_Images/Images/Own2.jpg")
dest2Corners = np.array([[0,0], [188, 0], [0, 106], [188,106]])
im2Corners = np.array([[625, 521], [3851, 52], [625, 2418], [3852, 2869]])
H2 = getHomography(im2Corners,dest2Corners)
cv2.imwrite('Results/Pt2Pt5.jpg', applyHomography(im2, H2))
```

Code for Two Step Method

```
import numpy as np
import cv2

def applyHomography(img, H):
    dIm = np.array([[0,0,img.shape[0],img.shape[0]],[0, img.shape[1], 0, img.shape[1]], [1, 1, 1, 1]])
    dImNew = np.matmul(H, dIm)
    dImNew[0] = dImNew[0]/dImNew[2]
    dImNew[1] = dImNew[1]/dImNew[2]

    minH = int(np.amin(dImNew[0]))
    maxH = int(np.amax(dImNew[0]))
    minW = int(np.amin(dImNew[1]))
    maxW = int(np.amax(dImNew[1]))

    print([minH, maxH, minW, maxW])
    imNew = np.zeros((maxH - minH, maxW - minW,3))
    invH = np.linalg.inv(H)

    for row in range(imNew.shape[0]):
        for col in range(imNew.shape[1]):
            coord = np.matmul(invH,[[row + minH], [col + minW], [1]])
            coord /= coord[2]
            if coord[0]<img.shape[0] and coord[0]>=0 and coord[1]<img.shape[1] and coord[1]>=0:
                imNew[row][col] = img[int(coord[0])][int(coord[1])]

    return imNew

def getHomographyP(corners):
    lineLeft = np.cross(corners[0],corners[2])
    lineRight = np.cross(corners[1],corners[3])
    pt1 = np.cross(lineLeft,lineRight)

    lineTop = np.cross(corners[0],corners[1])
    lineBottom = np.cross(corners[2],corners[3])
    pt2 = np.cross(lineTop,lineBottom)

    lineVanishing = np.cross(pt1,pt2)
    lineVanishing = lineVanishing / lineVanishing[2]
    H = np.array([[1,0,0],[0,1,0],lineVanishing])
    return H
```

```
def getHomographyA(pts):
   lineLeft = np.cross(pts[0],pts[2])
   #lineLeft = lineLeft / lineLeft[2]
   lineRight = np.cross(pts[1],pts[3])
   #lineRight = lineRight / lineRight[2]
   lineTop = np.cross(pts[0],pts[1])
   #lineTop = lineTop / lineTop[2]
   lineBottom = np.cross(pts[2],pts[3])
   #lineBottom = lineBottom/ lineBottom[2]

   A = np.array([[lineLeft[0]*lineBottom[0], lineLeft[0]*lineBottom[1]+lineLeft[1]*lineBottom[0], 0],
            [lineRight[0]*lineTop[0], lineRight[0]*lineTop[1]+lineRight[1]*lineTop[0], 0],
            [lineLeft[0]*lineTop[0], lineLeft[0]*lineTop[1]+lineLeft[1]*lineTop[0], 0],
            [lineRight[0]*lineBottom[0], lineRight[0]*lineBottom[1]+lineRight[1]*lineBottom[0], 0]])

   B = np.array([[-lineLeft[1]*lineBottom[1]],[-lineRight[1]*lineTop[1]],[-lineLeft[1]*lineTop[1]],[-
lineRight[1]*lineBottom[1]]],dtype = 'f')
   x = np.matmul(np.linalg.pinv(A),B)

   S = np.array([[x[0],x[1]],[x[1],1]],dtype = 'f')
   svd = np.linalg.svd(S)
   Aad = np.matmul(np.matmul(svd[2].conj().T,np.diag(np.sqrt(svd[1]))),svd[2])

   H = np.append(Aad,[[0,0]],0)
   H = np.append(H,[[0],[0],[1]],1)
   return H

img = cv2.imread("hw3_Task1_Images/Images/Img1.JPG")
imgCorners = np.array([[111,296,1], [197, 564,1 ], [683,242,1], [714, 548, 1]],dtype = 'f')
HProj = getHomographyP(imgCorners)
cv2.imwrite('Results/TwoStepProj1.jpg', applyHomography(img, HProj))
HAff = getHomographyA(np.matmul(HProj,imgCorners.T).T)
H = np.matmul(np.linalg.inv(HAff),HProj)
cv2.imwrite('Results/TwoStepAff1.jpg',applyHomography(img, H))

img = cv2.imread("hw3_Task1_Images/Images/Img2.jpeg")
imgCorners = np.array([[57700,38300,1],[55200,59300,1],[83400,38100,1],[92200,60700,1]],dtype = 'f')
HProj = getHomographyP(imgCorners)
cv2.imwrite('Results/TwoStepProj2.jpg', applyHomography(img, HProj))
HAff = getHomographyA(np.matmul(HProj,imgCorners.T).T)
H = np.matmul(np.linalg.inv(HAff),HProj)
cv2.imwrite('Results/TwoStepAff2.jpg',applyHomography(img, H))

img = cv2.imread("hw3_Task1_Images/Images/Img3.JPG")
```

```
imgCorners = np.array([[500*3,644*3,1],[503*3,663*3,1],[531*3,643*3,1],[535*3,663*3,1]],dtype = 'f')
HProj = getHomographyP(imgCorners)
cv2.imwrite('Results/TwoStepProj3.jpg', applyHomography(img, HProj))
HAff = getHomographyA(np.matmul(HProj,imgCorners.T).T)
H = np.matmul(np.linalg.inv(HAff),HProj)
cv2.imwrite('Results/TwoStepAff3.jpg',applyHomography(img, H))

img = cv2.imread("hw3_Task1_Images/Images/Own1.jpg")
imgCorners = np.array([[199,234,1],[233,626,1],[813,256,1],[739,647,1]],dtype = 'f')
HProj = getHomographyP(imgCorners)
cv2.imwrite('Results/TwoStepProj4.jpg', applyHomography(img, HProj))
HAff = getHomographyA(np.matmul(HProj,imgCorners.T).T)
H = np.matmul(np.linalg.inv(HAff),HProj)
cv2.imwrite('Results/TwoStepAff4.jpg',applyHomography(img, H))

img = cv2.imread("hw3_Task1_Images/Images/Own2.jpg")
imgCorners = np.array([[521,625,1],[52,3851,1],[2418,625,1],[2869,3852,1]],dtype = 'f')
HProj = getHomographyP(imgCorners)
cv2.imwrite('Results/TwoStepProj5.jpg', applyHomography(img, HProj))
HAff = getHomographyA(np.matmul(HProj,imgCorners.T).T)
H = np.matmul(np.linalg.inv(HAff),HProj)
cv2.imwrite('Results/TwoStepAff5.jpg',applyHomography(img, H))
```

Code For One Step Method

```
import numpy as np
import cv2

def applyHomography(img, H):
   dIm = np.array([[0,0,img.shape[0],img.shape[0]],[0, img.shape[1], 0, img.shape[1]], [1, 1, 1, 1]])
   dImNew = np.matmul(H, dIm)
   dImNew[0] = dImNew[0]/dImNew[2]
   dImNew[1] = dImNew[1]/dImNew[2]

   minH = int(np.amin(dImNew[0]))
   maxH = int(np.amax(dImNew[0]))
   minW = int(np.amin(dImNew[1]))
   maxW = int(np.amax(dImNew[1]))

   imNew = np.zeros((maxH - minH, maxW - minW,3))
   invH = np.linalg.inv(H)

   for row in range(imNew.shape[0]):
      for col in range(imNew.shape[1]):
         coord = np.matmul(invH,[[row + minH], [col + minW], [1]])
```

```
        coord /= coord[2]
        if coord[0]<img.shape[0] and coord[0]>=0 and coord[1]<img.shape[1] and coord[1]>=0:
            imNew[row][col] = img[int(coord[0])][int(coord[1])]

    return imNew


# Function to find homography for distortion by Dual Conic
def getHomography(pts):
    l1 = np.cross(pts[2],pts[3])
    l1 = l1 / l1[2]
    m1 = np.cross(pts[0],pts[2])
    m1 = m1 / m1[2]

    l2 = np.cross(pts[0],pts[2])
    l2 = l2 / l2[2]
    m2 = np.cross(pts[0],pts[1])
    m2 = m2 / m2[2]

    l3 = np.cross(pts[1],pts[3])
    l3 = l3 / l3[2]
    m3 = np.cross(pts[0],pts[1])
    m3 = m3 / m3[2]

    l4 = np.cross(pts[1],pts[3])
    l4 = l4 / l4[2]
    m4 = np.cross(pts[2],pts[3])
    m4 = m4 / m4[2]

    l5 = np.cross(pts[0],pts[3])
    l5 = l5 / l5[2]
    m5 = np.cross(pts[1],pts[2])
    m5 = m5 / m5[2]


    # LSE Ax = b
    A = np.array([[l1[0]*m1[0], 0.5*(l1[1]*m1[0]+l1[0]*m1[1]), l1[1]*m1[1],
0.5*(l1[0]*m1[2]+l1[2]*m1[0]), 0.5*(l1[2]*m1[1]+l1[1]*m1[2])],
                  [l2[0]*m2[0], 0.5*(l2[1]*m2[0]+l2[0]*m2[1]), l2[1]*m2[1], 0.5*(l2[0]*m2[2]+l2[2]*m2[0]),
0.5*(l2[2]*m2[1]+l2[1]*m2[2])],
                  [l3[0]*m3[0], 0.5*(l3[1]*m3[0]+l3[0]*m3[1]), l3[1]*m3[1], 0.5*(l3[0]*m3[2]+l3[2]*m3[0]),
0.5*(l3[2]*m3[1]+l3[1]*m3[2])],
                  [l4[0]*m4[0], 0.5*(l4[1]*m4[0]+l4[0]*m4[1]), l4[1]*m4[1], 0.5*(l4[0]*m4[2]+l4[2]*m4[0]),
0.5*(l4[2]*m4[1]+l4[1]*m4[2])],
```

```
            [l5[0]*m5[0], 0.5*(l5[1]*m5[0]+l5[0]*m5[1]), l5[1]*m5[1], 0.5*(l5[0]*m5[2]+l5[2]*m5[0]),
0.5*(l5[2]*m5[1]+l5[1]*m5[2])]])

   B = np.array([[-l1[2]*m1[2]],[-l2[2]*m2[2]],[-l3[2]*m3[2]],[-l4[2]*m4[2]],[-l5[2]*m5[2]]])

   C = np.matmul(np.linalg.pinv(A), B)
   C = C/np.max(C)

   S = np.zeros((2,2))
   S[0,0] = C[0]
   S[1,0] = 0.5 * C[1]
   S[0,1] = 0.5 * C[1]
   S[1,1] = C[2]

   svd = np.linalg.svd(S)
   D = np.diag(np.sqrt(svd[1]))
   K = np.dot(svd[2], np.dot(D, svd[2]))
   V = np.array([0.5*C[3], 0.5*C[4]])
   v = np.dot(np.linalg.inv(K), V)
   H = np.array([[K[0, 0], K[0, 1], 0],
            [K[1, 0], K[1, 1], 0],
            [v[0, 0], v[1, 0], 1]])
   return H

img = cv2.imread("hw3_Task1_Images/Images/Img1.JPG")
imgCorners = np.array([[296,111,1], [564, 197, 1], [242,683,1], [548, 714, 1]],dtype = 'f')
H = getHomography(imgCorners)
cv2.imwrite('Results/OneStep1.jpg',applyHomography(img,np.linalg.inv(H)))

img = cv2.imread("hw3_Task1_Images/Images/Img2.jpeg")
imgCorners = np.array([[57700,38300,1],[55200,59300,1],[83400,38100,1],[92200,60700,1]],dtype = 'f')
H = getHomography(imgCorners)
cv2.imwrite('Results/OneStep2.jpg',applyHomography(img,np.linalg.inv(H)))

img = cv2.imread("hw3_Task1_Images/Images/Img3.JPG")
imgCorners = np.array([[500*3,644*3,1],[503*3,663*3,1],[531*3,643*3,1],[535*3,663*3,1]],dtype = 'f')
H = getHomography(imgCorners)
cv2.imwrite('Results/OneStep3.jpg',applyHomography(img,np.linalg.inv(H)))

img = cv2.imread("hw3_Task1_Images/Images/Own1.jpg")
imgCorners = np.array([[199,234,1],[233,626,1],[813,256,1],[739,647,1]],dtype = 'f')
H = getHomography(imgCorners)
cv2.imwrite('Results/OneStep4.jpg',applyHomography(img,np.linalg.inv(H)))
```

```
img = cv2.imread("hw3_Task1_Images/Images/Own2.jpg")
imgCorners = np.array([[521,625,1],[52,3851,1],[2418,625,1],[2869,3852,1]],dtype = 'f')
H = getHomography(imgCorners)
cv2.imwrite('Results/OneStep5.jpg',applyHomography(img,np.linalg.inv(H)))
```