ECE 661: Homework 3

Monday, 17 September 2018

Dr. Avinash Kak

Naveen Madapana

1. Homography estimation

1.1 Definition

A homography is a linear transformation that maps physical points from domain plane to a range plane (image plane). A general homography (H), represented by $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is a 3×3 nonsingular matrix that transforms 2D physical points from one plane to the other. Also, note that homographies map straight lines to straight lines.
Let $x_1 = \begin{bmatrix} u_1 & v_1 & w_1 \end{bmatrix}^T$ and $x_2 = \begin{bmatrix} u_2 & v_2 & w_2 \end{bmatrix}^T$ be the homogeneous coordinates of two points p_1 and p_2. Suppose p_1 and p_2 represent a particular point in the real world captured by two different cameras. Then, we could estimate a homography H such that:

$$x_2 = Hx_1 \quad \longrightarrow \quad \begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix}$$

x_1 and x_2 are homogeneous coordinates. Hence, we can rewrite u_2 and v_2 as u_2/w_2 and v_2/w_2 respectively.

There are three ways in which we can estimate homography H in order to eliminate both the projective distortion and affine distortion. Note that projective distortion preservers straight lines and affine distortion keeps parallel lines parallel.

1. **Point to point correspondence**.
2. **Two steps**: Removing projective distortion (estimated via vanishing lines) followed by affine distortion
3. **One step**: Estimating the homography that removes both projective and affine distortion in one step

1.2 Point to point correspondence

In this approach, it is assumed that we have a one to one correspondence between pixels in the domain plane to the pixels in the range plane. Overall, we need atleast four correspondences (pairs of points) in order to compute the homography in this method.
Without the loss of generality, it can be assumed that $w_1 = 1$ as we are working with homogeneous coordinates. By simplifying the equation further and rearranging the equation in the form of $Ah = 0$, where h is a 9×1 vector consisting of elements in the homography H, we will get,

$$u_2(H_{31}u_1 + H_{32}v_1 + H_{33}) = H_{11}u_1 + H_{12}v_1 + H_{13}w_1$$
$$v_2(H_{31}u_1 + H_{32}v_1 + H_{33}) = H_{21}u_1 + H_{22}v_1 + H_{23}w_1$$

Hence, each pair of matching point will yield two equations (for x and y). where h is a 9×1 vector consisting of elements in the homography H and A is a $2m \times 9$ matrix (m being the number matching pairs of points selected to estimate h). Let a_u and a_v be the coefficients corresponding to x and y dimensions.

$$Ah = 0$$

Where:

$$\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1u_2 & v_1u_2 & u_2 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1v_2 & v_1v_2 & v_2 \end{bmatrix} h = 0$$

$$h = \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{21} & H_{22} & H_{23} & H_{31} & H_{32} & H_{33} \end{bmatrix}^T$$

Now if we choose m pairs of points we will get the system $Ah = 0$:

$$\begin{bmatrix} -u_{11} & -v_{11} & -1 & 0 & 0 & 0 & u_{11}u_{12} & v_{11}u_{12} & u_{12} \\ 0 & 0 & 0 & -u_{11} & -v_{11} & -1 & u_{11}v_{12} & v_{11}v_{12} & v_{12} \\ -u_{21} & -v_{21} & -1 & 0 & 0 & 0 & u_{21}u_{22} & v_{21}u_{22} & u_{22} \\ 0 & 0 & 0 & -u_{21} & -v_{21} & -1 & u_{21}v_{22} & v_{21}v_{22} & v_{22} \\ & & & & \vdots & & & & \\ -u_{n1} & -v_{n1} & -1 & 0 & 0 & 0 & u_{n1}u_{n2} & v_{n1}u_{n2} & u_{n2} \\ 0 & 0 & 0 & -u_{n1} & -v_{n1} & -1 & u_{n1}v_{n2} & v_{n1}v_{n2} & v_{n2} \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = 0$$

The vector h can be estimated using singular value decomposition. We want to estimate h such that:

$$\underset{h}{\text{argmin}} \ \frac{|Ah|^2}{|h|^2}$$

Let $Ah = \lambda h$, where λ is an eigenvalue and h is corresponding eigenvector.

$$\underset{h}{\text{argmin}} \ \frac{|\lambda h|^2}{|h|^2} \Rightarrow \underset{h}{\text{argmin}} \ \frac{\lambda^2|h|^2}{|h|^2} \Rightarrow \underset{h}{\text{argmin}} \ \lambda^2$$

Hence, h is the eigenvector of A with lowest eigenvalue. Therefore, we compute SVD of A i.e. $A = USV^T$ and choose the last column of V^T. Now, we reshape h in order to get a $H_{3\times3}$.

1.3 Two step method

This method is two fold. First, a homography to remove projective distortion was estimated using vanishing lines. Second, the homography for removing affine distortion was computed.

Pure projective distortion: Vanishing Lines

Assume that there are a set of parallel lines in the domain plane. These set of parallel lines intersect at infinity i.e. at ideal points ($[x, y, 0]^T$). Further, all of the ideal points pass through the line at infinity which is represented by $[0, 0, 1]^T$.
However, When the domain plane is subjected to pure projective distortion, the parallel lines **do not** remain parallel in the range plane. Hence the line at infinity in the domain plane is transformed to a physical line (which can be computed) in the range plane. This is illustrated in the figure 1.

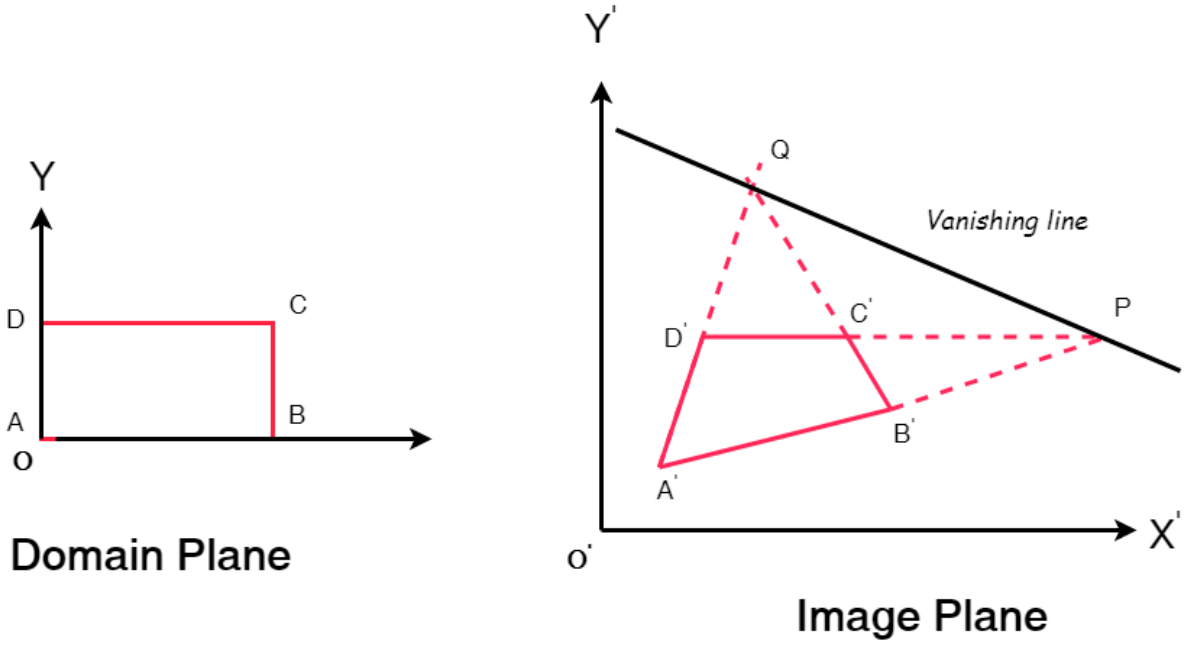


Figure 1: Concept of vanishing lines

Given the set of parallel lines ($A'B'$, $C'D'$) and ($A'D'$ and $B'C'$), we can estimate the vanishing line $l = [l_1, l_2, l_3]$. Then the homography that removes the pure projective distortion is given below. This homography H sends the vanishing line back to infinity thus keeping the parallel lines parallel.

$$H_{vl} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

Pure affine distortion

Affine distortion keeps the parallel lines parallel but however does not preserve the angle between the lines. For instance, perpendicular lines in the domain plane are not perpendicular in the range plane when the domain plane is subjected to pure affine distortion.

Let $l = [l_1, l_2, l_3]$ and $m = [m_1, m_2, m_3]$ be the perpendicular lines in the domain plane. Additionally, let $l' = [l'_1, l'_2, l'_3]$ and $m' = [m'_1, m'_2, m'_3]$ be the corresponding lines in the range plane. Finally, let θ be the angle between l' and m'. Figure 2 shows this elements.
The angle θ can be described by the homogeneous coordinates of the lines l and m as follows.

$$\cos(\theta) = \frac{l_1 m_1 + l_2 m_2}{\sqrt{(l_1^2 + l_2^2)(m_1^2 + m_2^2)}}$$

If we write this equation in terms of the dual degenerate conic $C^*_\infty = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, we will get:

$$\cos(\theta) = \frac{C_N}{C_D} = \frac{l^T C^*_\infty m}{\sqrt{(l^T C^*_\infty l)(m^T C^*_\infty m)}}$$

Now lets represent $\cos(\theta)$ in terms of l', m' and $C^{*'}_\infty$. We use the formulae to map the conics and lines from domain to range plane. Let H be the homography that maps **domain plane to range plane**. Now, note

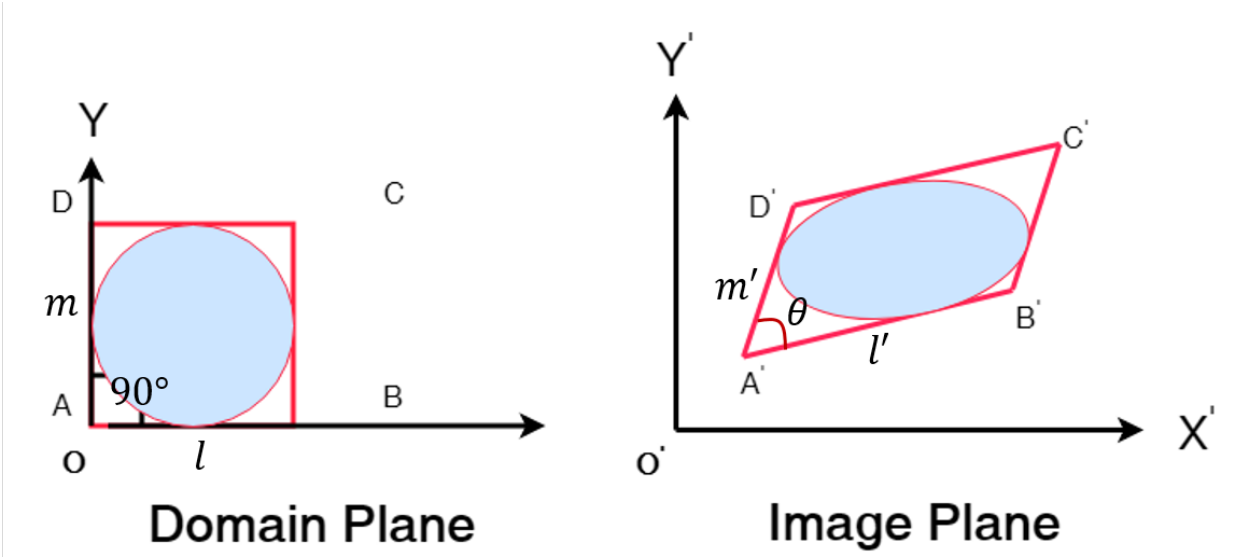


Figure 2: Illustrating affine distortion

that lines l in the domain plane correspond to $l' = H^T l$ in the range/image plane. Similarly, the conic C in domain plane corresponds to $C' = HCH^T$ in the range plane. By substituting, we will get,

$$C_N = (l'^T H)(H^{-1} C_\infty^{*'} H^{-T})(H^T m') = l' C_\infty^{*'} m'$$

Next, assuming that the angle in the original picture is 90 deg (As shown in Figure **??**), we will get the following. Also, note that we assume that H is purely affine without any projective distortion. Finally, we assume that the there is no translation ($t = [0, 0]^T$).

$$l' H C_\infty^* H^T m' = 0 \xrightarrow{H \text{ is affine}} \begin{bmatrix} l'_1 & l'_2 & l'_3 \end{bmatrix} \begin{bmatrix} A & \vec{t} \\ \vec{0}^T & 1 \end{bmatrix} \begin{bmatrix} I & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} \begin{bmatrix} A^T & \vec{0} \\ \vec{t}^T & 1 \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \end{bmatrix} = 0$$

$$\xrightarrow{\text{Multiply Matrices}} \begin{bmatrix} l'_1 & l'_2 & l'_3 \end{bmatrix} \begin{bmatrix} AA^T & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \end{bmatrix} = 0 \xrightarrow[m'_3 = l'_3 = 1]{S = AA^T} \begin{bmatrix} l'_1 & l'_2 \end{bmatrix} \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \end{bmatrix}$$

Given that S is symmetric we end up with an equation that has three unknowns S_{11}, S_{12}, S_{22}. Without loss of generality we can assume $S_{12} = 1$, since all that matters is the proportions and that results in two unknowns:

$$S_{11} l'_1 m'_1 + l'_1 m'_2 + l'_2 m'_1 + S_{22} l'_2 m'_2 = 0$$

We can estimate S using SVD. Now that we know S we need to find A. Both S and A are positive definite. Hence, we can express them in terms of their eigenvalues and eigenvectors. Let V be the matrix of eigenvectors of S and D be the diagonal matrix of eigenvalues of S. Given that $S = AA^T$ we obtain the following equation.

$$S = VDV^T \xrightarrow[\text{V is orthonormal}]{S = AA^T} A = V\sqrt{D}V^T$$

$$H_{af} = \begin{bmatrix} A & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix}$$

1.4 One step method

In this approach, the objective is to find a general homography that can remove both projective and affine distortion in one step. The structure of such general homography is given as follows. Note that v^T was set to zero, when we assumed that homography is purely affine in the previous section. Now, v^T is non zero.

$$H = \begin{bmatrix} A & \vec{0} \\ v^T & 1 \end{bmatrix}$$

Intuition: The dual degenerate conic at infinity in the domain plane transforms into a physical conic in the range plane when domain plane is subjected to a general homography. Now, we want to estimate the physical conic in the range plane ($C_\infty^{*'}$), so that we can compute the homography H.
Now, lets consider the general representation of the conic ($C_\infty^{*'}$):

$$C_\infty^{*'} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} = \begin{bmatrix} P & q \\ q^T & f \end{bmatrix}$$

As mentioned in the previous section, let us assume that $l \perp m$ in the domain plane, i.e. $\cos(\theta) = 0$. Therefore,

$$C_N = l' C_\infty^{*'} m' = 0$$

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

$$\begin{bmatrix} l_1 m_1 & l_2 m_1 + m_2 l_1 & l_2 m_2 & l_3 m_1 + m_3 l_1 & l_3 m_2 + m_3 l_2 & l_3 m_3 \end{bmatrix} \begin{bmatrix} a \\ b/2 \\ c \\ d/2 \\ e/2 \\ f \end{bmatrix} = 0$$

We have a total of six variables. Since we are working in homogeneous coordinates, we have only five variables. Hence we need at least five pairs of perpendicular lines in the range plane. We can use SVD to solve the homogeneous system $Ah = 0$ and compute the conic, $C_\infty^{*'}$. Normalize the conic so that $f = 1$.

Given $C_\infty^{*'}$, we need to estimate H.

$$C_\infty^{*'} = H C_\infty^{*} H^T$$

$$C_\infty^{*'} = \begin{bmatrix} A & \vec{0} \\ v^T & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} A & \vec{0} \\ v^T & 1 \end{bmatrix}^T = \begin{bmatrix} AA^T & Av \\ v^T A & v^T v \end{bmatrix}$$

Hence, $P = AA^T$ and $q^T = v^T A$. Note that $A = A^T$ (symmetric matrix).

Now, we need to find A and v in order to find H. Since we are working in homogeneous system, note that $f = 1$. lets perform SVD on P in order to get A.

$$P = VDV^T \Rightarrow A = V\sqrt{D}V^T$$

$$v^T = q^T \ A^{-1}$$

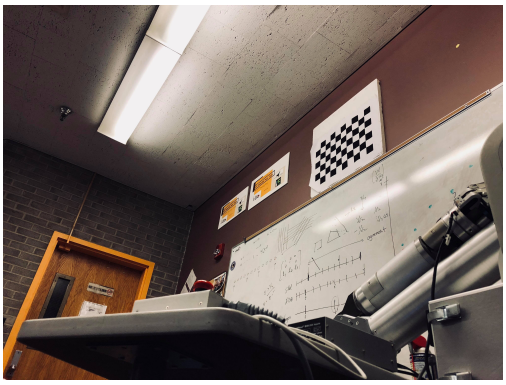
$$v^T = \frac{v^T}{v^T v}$$

2. Results

The given images are depicted in figure 3.

(a) Image 1

(b) Image 2



(c) Image 3

(d) Image 4

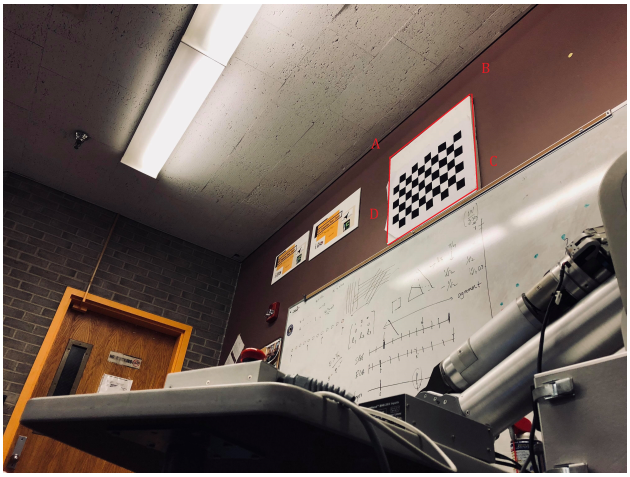
Figure 3: Original pictures

2.1 Point to point correspondence

Four pairs of points have been chosen to remove the projective distortion using pixel to pixel correspondences. The end points A, B, C and D show the four points chosen as depicted in figure 4. The output of removing the projective distortion is depicted in figure 5.

(a) Image 1

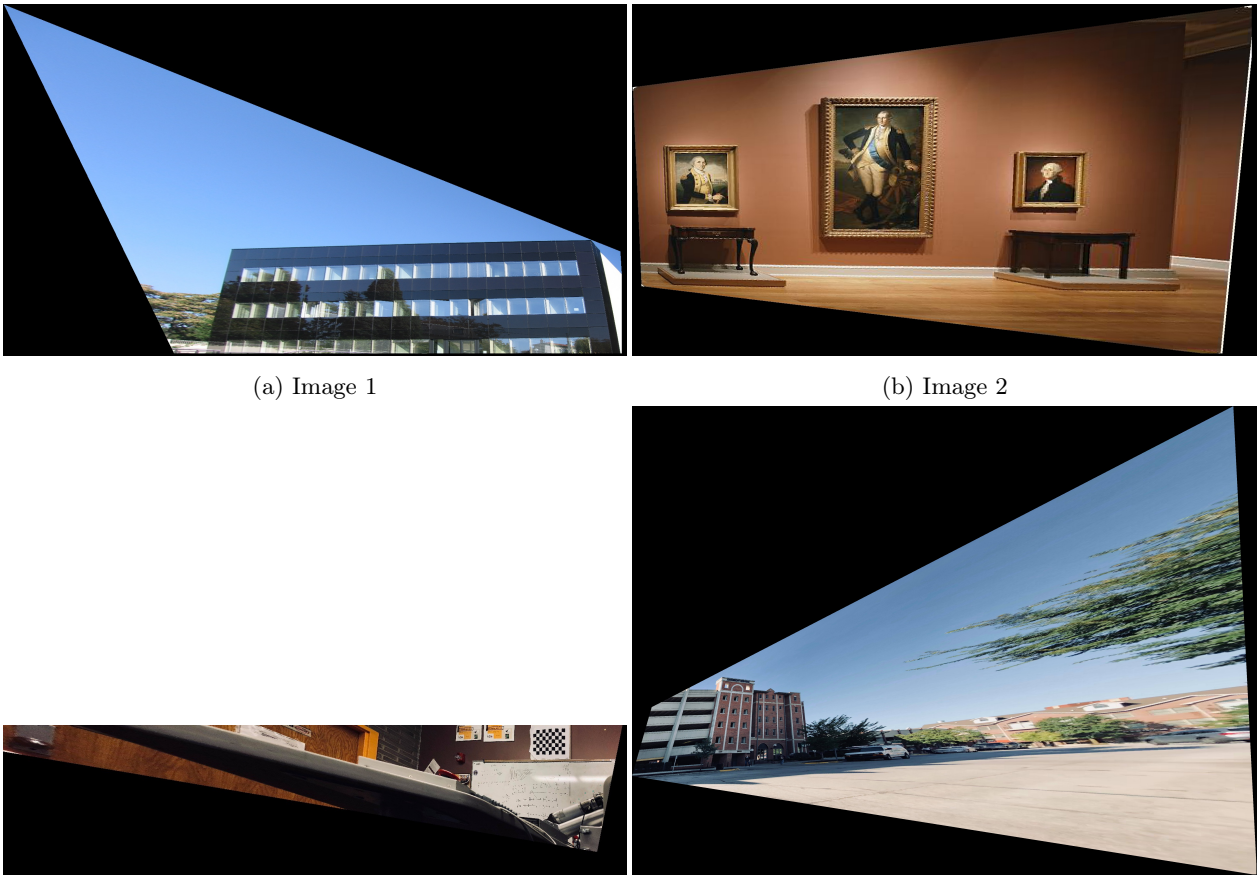
(b) Image 2



(c) Image 3

(d) Image 4

Figure 4: A, B, C and D are the point correspondences



(a) Image 1

(b) Image 2

(c) Image 3

(d) Image 4

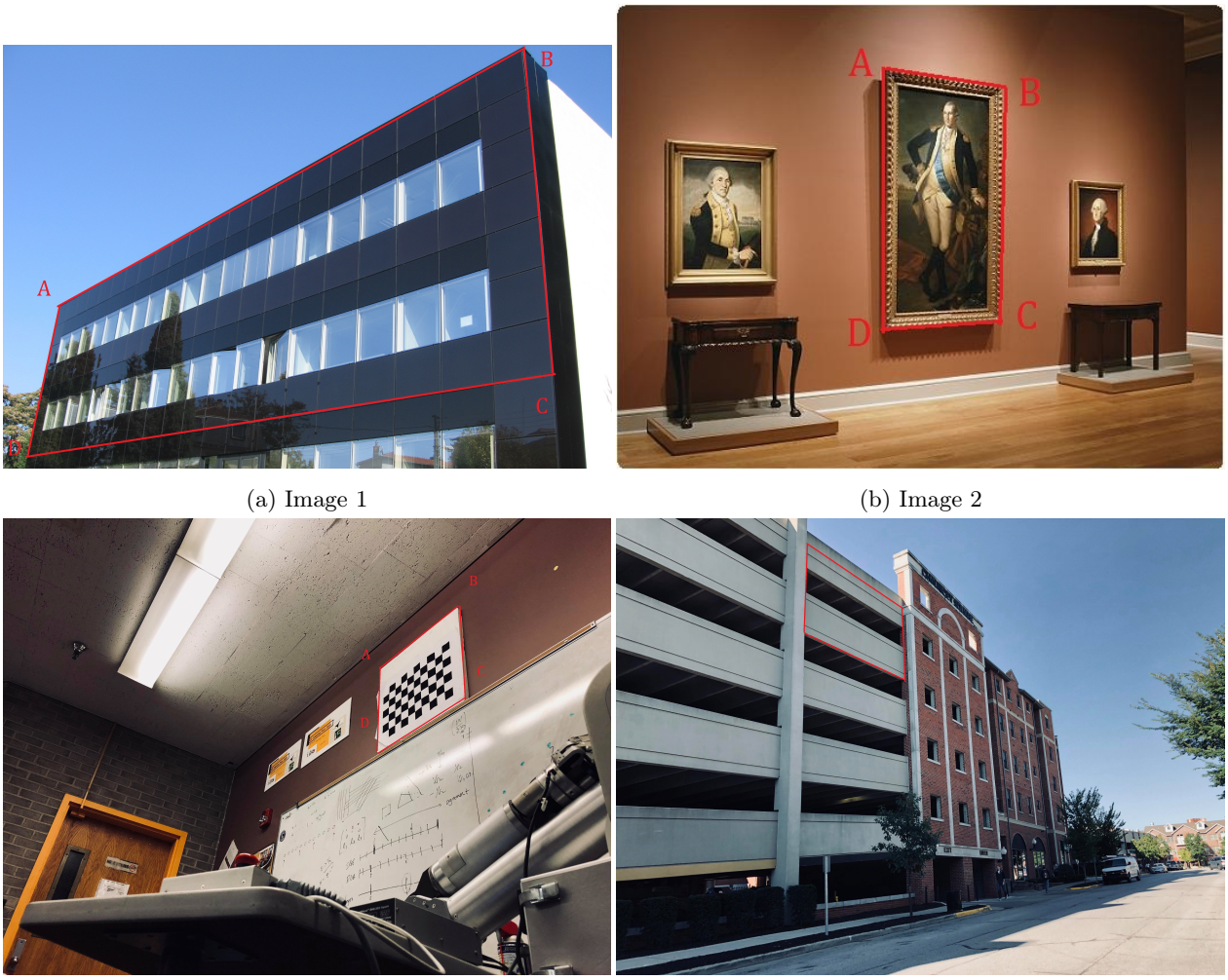
Figure 5: Output of removing projective distortion using point to point correspondences.

Implementation details

1. A, B, C and D correspond to (0, 0), (59, 0), (59, 79) and (0, 79) respectively in the domain plane.
2. Note that, we need to estimate the homography H that maps range plane to the domain plane.
3. Normalize the homography such that $H(end, end) = 1$
4. First, map the corners of the image using H and identify the size of the transformed image (in domain plane).
5. Now, for each pixel in the transformed image, identify the corresponding pixel in the given image using H^{-1}. Make sure to convert the pixel coordinates into unsigned integer of eight bits.
6. Next, for each pixel in the transformed image, copy the intensities of the corresponding pixels in the given image.
7. For the pixels in the transformed image that fall outside the given image, set the intensity to zero. This would give black regions in the resulting images.
8. If we use only four points, the homography is very sensitive to the points chosen. Hence, choose the points carefully. It is very recommended to use more than four points.

2.2 Two Step Method

First, pure projective distortion is removed using the vanishing lines approach. In this method, the vanishing line in the range plane is estimated. Then, the homography that sends the physical vanishing line to line at infinity is computed.
Two sets of parallel lines are needed to estimate the vanishing line. Figure 6 shows the set of parallel lines chosen ($AB \parallel CD$ and $AD \parallel BC$).



(a) Image 1

(b) Image 2

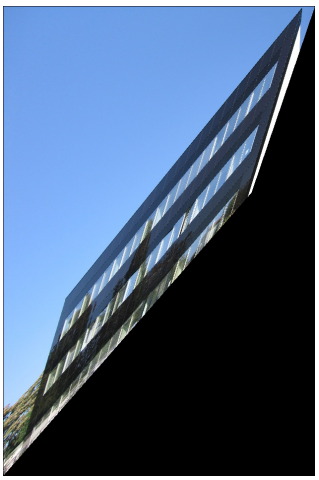
(c) Image 3

(d) Image 4

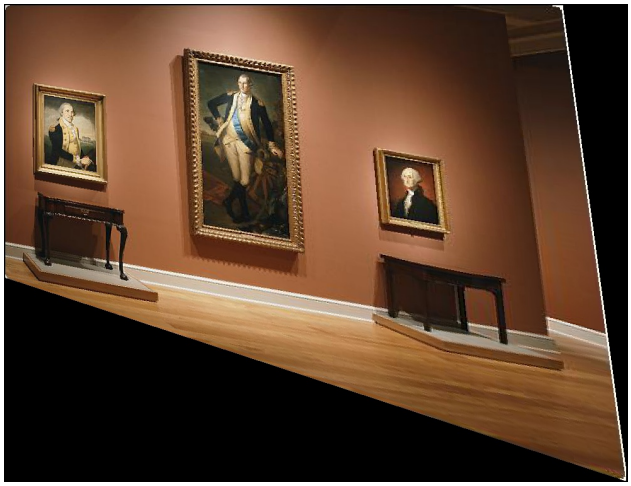
Figure 6: Pairs of parallel lines ($AB \parallel CD$ and $AD \parallel BC$) to estimate vanishing line.

The output of removing the pure projective distortion is depicted in figure 7. In these images, the parallel lines in the domain plane appear to be parallel.

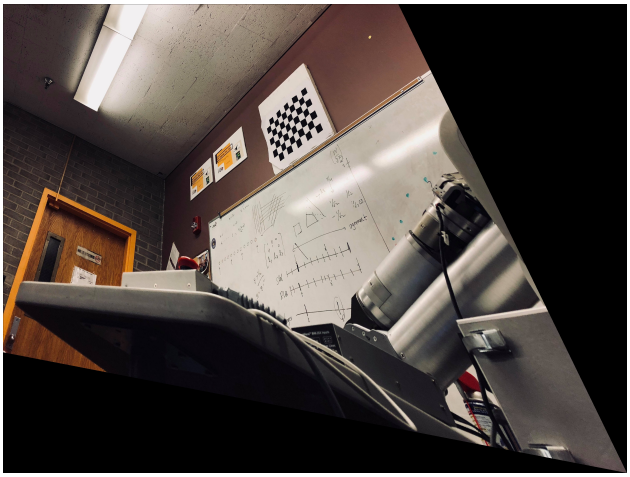
Now the next step is to to remove the affine distortion. We need two sets of perpendicular lines to remove affine distortion. The two sets of perpendicular lines chosen are depicted in figure 8 ($AB \perp BC, BC \perp CD, CD \perp DA, DA \perp AB, AC \perp BD$). The output of removing the affine distortion is depicted in figure 9.



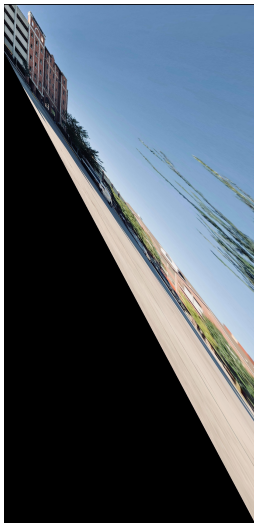
(a) Image 1



(b) Image 2

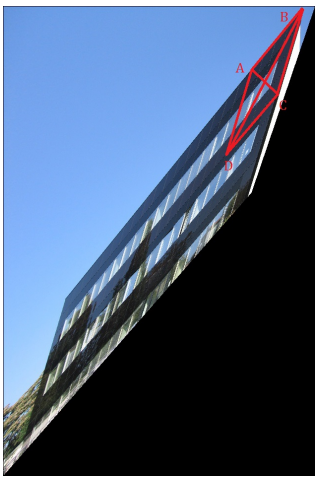


(c) Image 3

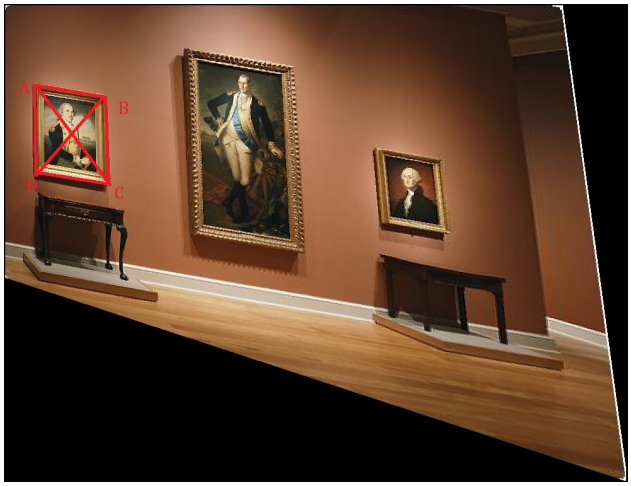


(d) Image 4

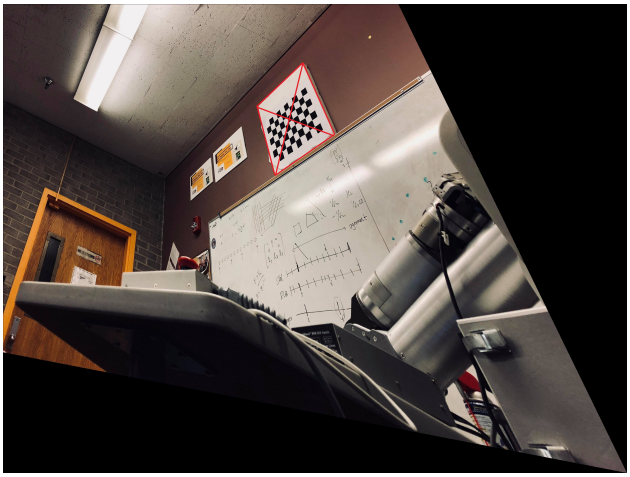
Figure 7: Output after estimating the vanishing lines and removing the projective distortion.



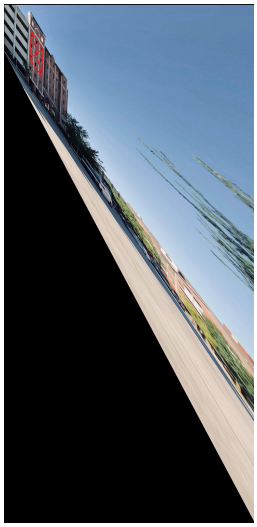
(a) Image 1



(b) Image 2

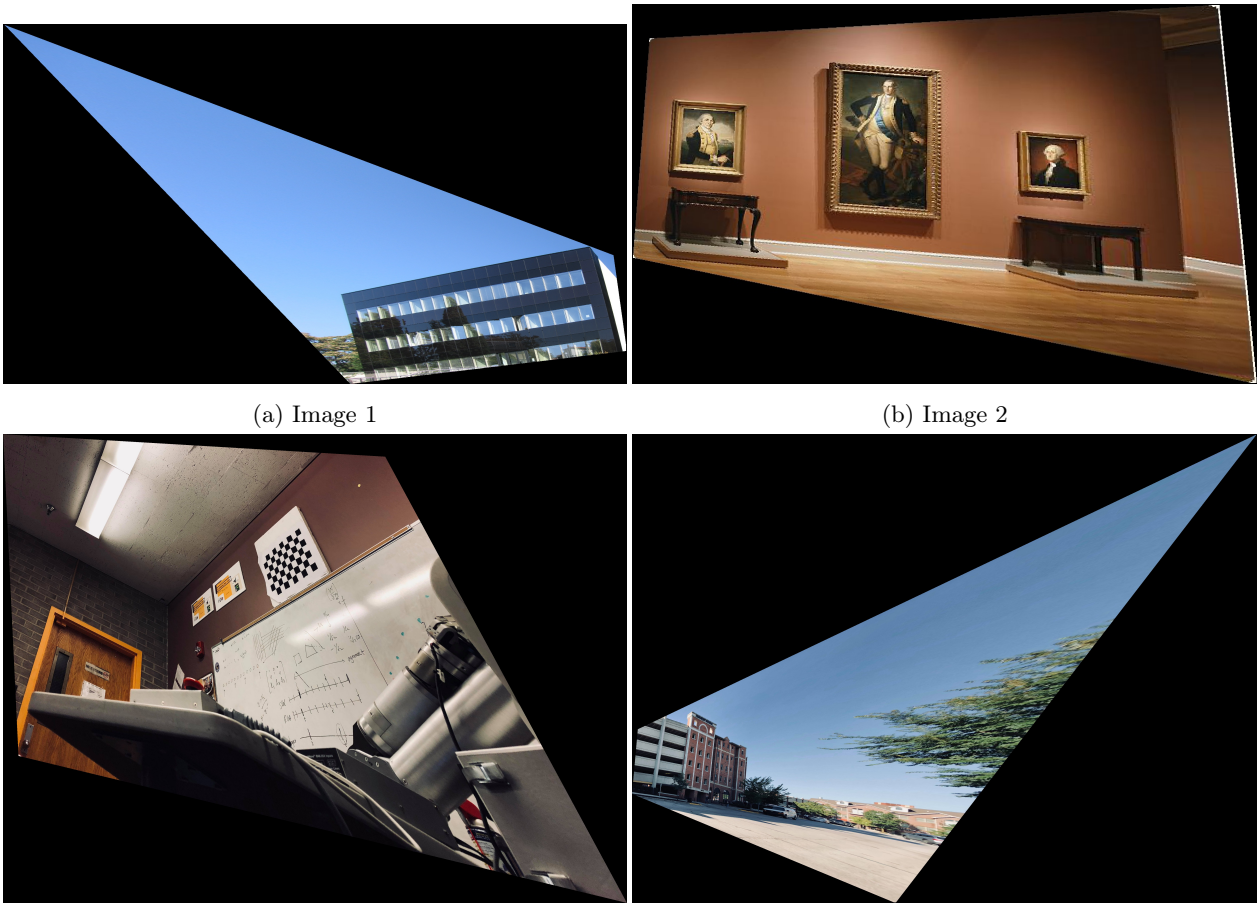


(c) Image 3



(d) Image 4

Figure 8: Perpendicular lines selected in the images where the projective distortion is removed. ($AB \perp BC, BC \perp CD, CD \perp DA, DA \perp AB, AC \perp BD$)



(a) Image 1 (b) Image 2

(c) Image 3 (d) Image 4

Figure 9: Output after removing the affine distortion from the images that have no projective distortion.

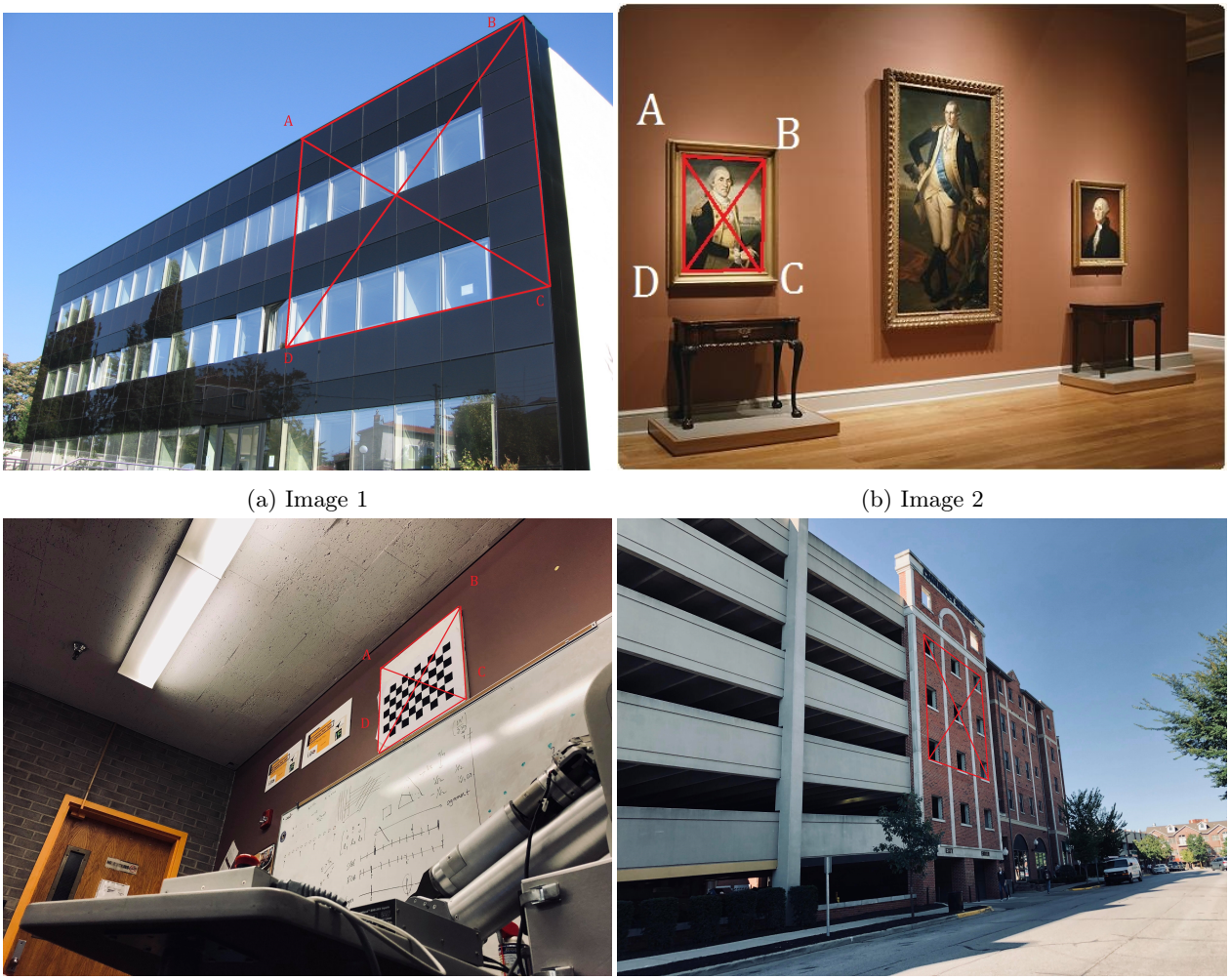
Implementation details

1. Normalize the vanishing line before forming the homography. Note that this homography H_{vl} transforms the given image into domain plane where the selected lines are parallel. Hence this homography removes the pure projective distortion.
2. **NOTE that, pairs of perpendicular lines should be chosen on the image obtained after removing the pure projective distortion**
3. The homography computed to remove affine distortion H_{af} transforms domain plane to image plane. Hence the final homography is $H_{af}^{-1}\ H_{vl}$. Use this homography to remove both projective and affine distortion in the image.
4. Make sure to normalize both the H_{af} and H_{vl} before forming the final homography.

2.3 One Step Method

In this method, both the projective and affine distortion are removed in one step. This method requires five sets of perpendicular lines. The pairs of orthogonal lines chosen are depicted in figure 10 ($AB \perp BC, BC \perp CD, CD \perp DA, DA \perp AB, AC \perp BD$).

The output of this step is depicted in figure 12.



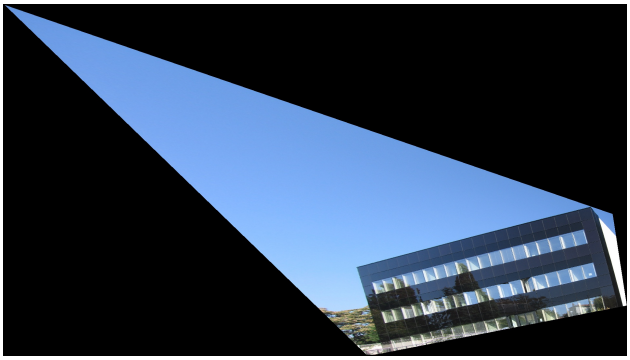
(a) Image 1

(b) Image 2

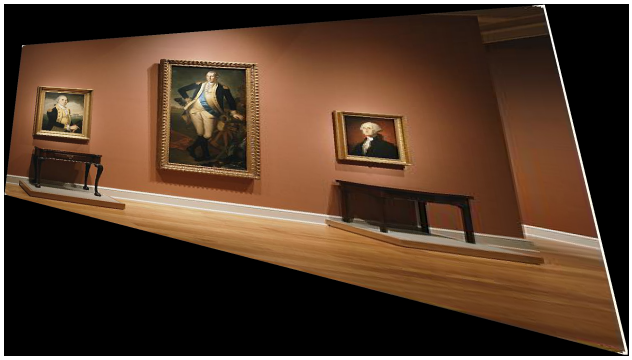
(c) Image 3

(d) Image 4

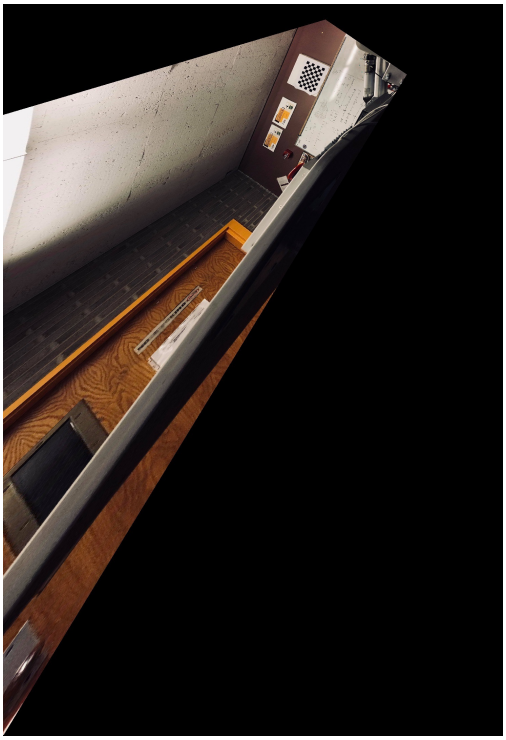
Figure 10: Five pairs of perpendicular lines in order to remove both projective and affine distortion in one step. ($AB \perp BC, BC \perp CD, CD \perp DA, DA \perp AB, AC \perp BD$)



(a) Image 1



(b) Image 2



(c) Image 3

(d) Image 4

Figure 11: Output of removing both projective and affine distortions in one step.

Implementation details

1. This method is very sensitives to the points/lines chosen. Further, it requires some additional hacks for it to work.

2. The homography estimated using this method (H_{gh}) maps domain plane to image plane. Since we want to transform the image plane back to domain plane, make sure to use H_{gh}^{-1} to transform the image to domain plane.

3. If you use SVD to solve the homogeneous system, $Ah = 0$, make sure to normalize h, i.e. $h(end) = 1$

4. **HACK:** In this method, our objective is to make sure that all of the pairs of lines selected are perpendicular in the transformed image. However, we are not concerned with the scale of the transformed

image. It is possible that all of the pixels in the original image might fall in the range $[0, 1]$. Hence we need modify the obtained homography to scale up the transformed image.

Let us say the bottom right corner $(N-1, M-1)$ was mapped to (α_x, α_y) when we use the homography H_{gh}^{-1}. Let us say $\alpha_x, \alpha_y \in [0, 1]$. Now, say we want the transformed image to be of size (W, H) where W and H is the width and height respectively. For instance, $(W, H) = (1920, 1080)$.

Now, estimate $\lambda_x = \dfrac{W}{\alpha_x}$ and $\lambda_y = \dfrac{H}{\alpha_y}$. Form the matrix K:

$$K = \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The modified homography $\tilde{H_{gh}} = KH_{gh}^{-1}$. Use the $\tilde{H_{gh}}$ to transform the image to the domain plane.

3. Method comparison

The **one step method** is great in the sense that we are eliminating both affine and projective distortions in one step. However, it is very hard to find a set of five perpendicular lines such that each set is not parallel to the other set of perpendicular lines. Also, the one step approach requires additional hacks to get good results. Overall the one step approach is very elegant but we need to be careful when we use this method.

On the other hand, **two step approach** is straight forward. Further, it is easy to choose two pairs of parallel lines and two pairs of perpendicular lines. Also, two step approach shows better performance and takes minimum number of points/lines required to estimate the homography. The two step method required reasonable number of trials to obtain good lines.

4. Extra credit: Different vanishing lines

In this section, an image with multiple sets of parallel has been chosen to verify the uniqueness of the vanishing line. The image is shown in figure 12

Figure 12: Image 4

The different sets of vanishing lines are shown in the figure 13. The vanishing lines for:

Set 1 $[-4.91e-04 \quad -4.14e-04 \quad 1.0]$

Set 2 $[5.35e-04 \quad 1.05e-03 \quad 1.0]$

Set 3 $[-7.94e-04 \quad -1.10e-03 \quad 1.0]$

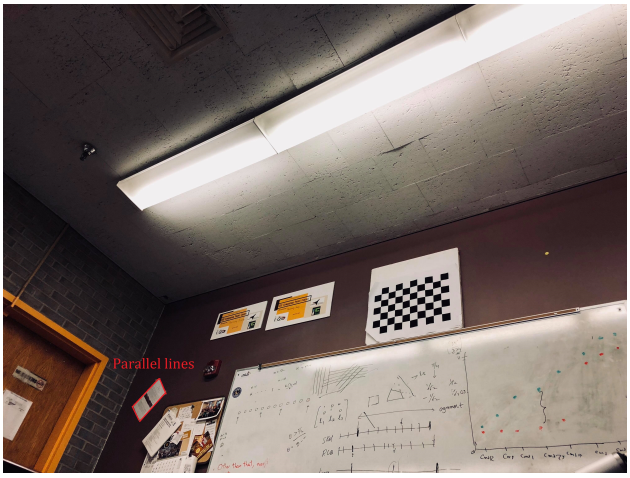
Set 4 $[-3.69e-04 \quad -5.00e-04 \quad 1.0]$

The results obtained using different sets of parallel lines is depicted in figure 14.

Conclusion

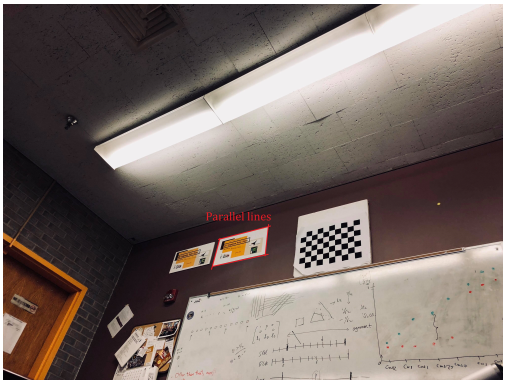
By looking at the equations of the vanishing lines and the resulting images shown in figure 14, it can be concluded that the vanishing lines change depending on the set of parallel lines chosen. The slopes of the vanishing lines is also different for all of the four sets considered.

However, it is very hard to decide if the vanishing lines are changing due to the inaccurate selection of points and floating point precisions. Though the vanishing lines obtained by Set 3 and Set 4 are different, the resulting images look very similar, implying that the vanishing lines are different due to the slight inaccuracy in point selection.



(a) Set 1

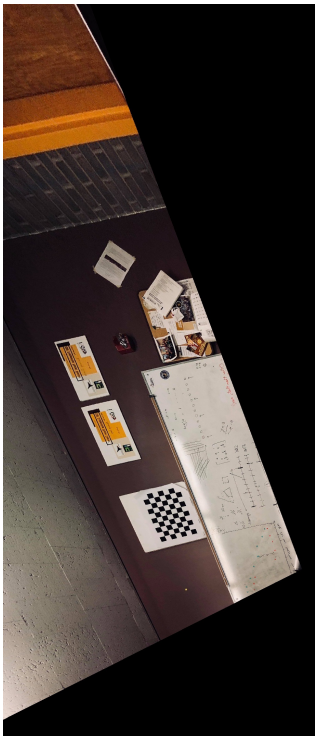
(b) Set 2



(c) Set 3

(d) Set 4

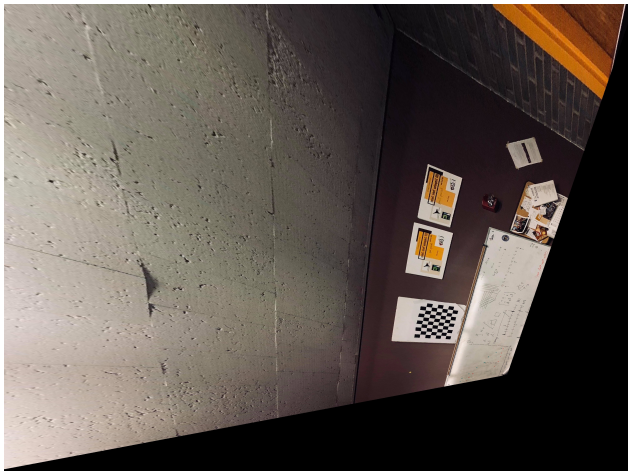
Figure 13: Different sets of parallel lines



(a) Set 1

(b) Set 2

(c) Set 3



(d) Set 4

Figure 14: Results obtained by different sets of vanishing lines followed by affine removal. The square pattern used previously was used to remove the affine distortion.

Source Code

```
import cv2
import numpy as np
import os, time, sys

##########################
#   HELPER FUNCTIONS     #
##########################
```

```
def get_pts(shap, corners = False, H=None, targ_shap = None):
    #####
    #
    # Description:
    #
    # Input:
    #   shap: tuple (num_rows, num_cols, optional) - for given image
    #   corners: if True, only corner points, if False, all points
    #   H: 3 x 3 ndarray - given image to target image
    #   targ_shap: tuple (num_rows, num_cols, optional) - for target image
    #
    #   if H is not None, apply the homography
    #   if targ_shap is not None, clip the transformed pts accordingly.
    #
    # Return:
    #   trasformed points. In (x, y) format. It depends on if H, targ_shap are None
    #
    #####

    M, N = shap[0], shap[1]
    if(corners):
        pts = np.array([[0, 0],[N-1, 0],[N-1, M-1], [0, M-1]])
    else:
        xv, yv = np.meshgrid(range(N), range(M))
        pts = np.array([xv.flatten(), yv.flatten()]).T
    if H is None: return pts, None

    # else
    t_pts = np.dot(H, real_to_homo(pts).T)
    t_pts = homo_to_real(t_pts.T).astype(int)
    if(targ_shap is None): return pts, t_pts

    #else
    t_pts[:,0] = np.clip(t_pts[:,0], 0, targ_shap[1]-1)
    t_pts[:,1] = np.clip(t_pts[:,1], 0, targ_shap[0]-1)
    return pts, t_pts

def find_homography_2d(pts1, pts2):
     # Estimate homography using the point to point correspondences
    # There should be a one to one correspondence between the pts1 and pts2.

    # Assertion
    assert pts1.shape[1] == 2, 'pts1 should have two columns'
    assert pts2.shape[1] == 2, 'pts2 should have two columns'
    assert pts1.shape[0] == pts2.shape[0], 'pts1 and pts2 should have same number of rows'

    # Forming the matrix A (8 x 9)
    A = []
    for (x1, y1), (x2, y2) in zip(pts1, pts2):
        A.append([x2, y2, 1, 0, 0, 0, -1*x1*x2, -1*x1*y2, -1*x1])
        A.append([0, 0, 0, x2, y2, 1, -1*y1*x2, -1*y1*y2, -1*y1])
    A = np.array(A)
```

```
    [U, S, V] = np.linalg.svd(A, full_matrices = True)
    h = V.T[:,-1]
    h = h / h[-1]

    # # Finding the homography. H[3,3] is assumed 1.
    # h = np.dot(np.linalg.pinv(A[:,:-1]), -1*A[:,-1])
    # h = np.append(h, 1)

    H = np.reshape(h, (3, 3))
    return H, hinv(H)

def find_homography_vl(pts):
    ## Find homography resulting from vanishing line approach.
    # pts should contain points either in clockwise or anti clockwise order
    # Assertion
    assert isinstance(pts, np.ndarray), 'pts should be a numpy array'
    assert pts.shape[1] == 2, 'pts should have two columns'
    assert pts.shape[0] == 4, 'pts should have four rows'

    pts = real_to_homo(pts)
    line1 = np.cross(pts[0,:], pts[1,:])
    line2 = np.cross(pts[2,:], pts[3,:])
    point1 = np.cross(line1, line2)

    line3 = np.cross(pts[0,:], pts[3,:])
    line4 = np.cross(pts[1,:], pts[2,:])
    point2 = np.cross(line3, line4)

    van_line = np.cross(point1, point2)
    van_line = van_line / van_line[-1]
    print 'van_line: ', van_line

    H = np.eye(3)
    H[2, 0] = van_line[0]
    H[2, 1] = van_line[1]

    return H

def find_homography_af(pts):
    ## Remove affine distortion using two pairs of perpendicular lines.
    # pts should contain points either in clockwise or anti clockwise order
    # Assertion
    assert isinstance(pts, np.ndarray), 'pts should be a numpy array'
    assert pts.shape[1] == 2, 'pts should have two columns'
    assert pts.shape[0] == 4, 'pts should have four rows'

    pts = real_to_homo(pts)
    ln_pairs = []
    ln_pairs.append((np.cross(pts[0,:], pts[1,:]), np.cross(pts[1,:], pts[2,:])))
    ln_pairs.append((np.cross(pts[1,:], pts[2,:]), np.cross(pts[2,:], pts[3,:])))
    ln_pairs.append((np.cross(pts[2,:], pts[3,:]), np.cross(pts[3,:], pts[0,:])))
    ln_pairs.append((np.cross(pts[0,:], pts[2,:]), np.cross(pts[1,:], pts[3,:])))
```

```
    A = []
    for line1, line2 in ln_pairs:
        r1 = line1[0]*line2[0]
        r2 = line1[0]*line2[1] + line1[1]*line2[0]
        r3 = line1[1]*line2[1]
        A.append([r1, r2, r3])
    print A
    [_, _, V] = np.linalg.svd(A, full_matrices = True)
    h = V.T[:,-1]
    h = h / h[-1]
    S = np.array([[h[0], h[1]],[h[1], h[2]]])
    print S
    [_, D2, V] = np.linalg.svd(S, full_matrices = True)
    H = np.eye(3)
    H[0:-1,0:-1] = np.dot(np.dot(V, np.diag(np.sqrt(D2))), V.T)
    return H

def find_homography_gh(pts_list):
    ## Find general homography that removes both projective and affine distortion.
    ## Remove both projective and affine distortion using five pairs of perpendicular lines.
    # pts should contain points either in clockwise or anti clockwise order
    # Assertion
    # assert isinstance(pts, np.ndarray), 'pts should be a numpy array'
    # assert pts.shape[1] == 2, 'pts should have two columns'
    # assert pts.shape[0] == 4, 'pts should have four rows'

    ln_pairs = []
    for pts in pts_list:
        pts = real_to_homo(pts)
        ln_pairs.append((nmlz(np.cross(pts[0,:], pts[1,:])), nmlz(np.cross(pts[1,:], pts[2,:]))))
        ln_pairs.append((nmlz(np.cross(pts[1,:], pts[2,:])), nmlz(np.cross(pts[2,:], pts[3,:]))))
        ln_pairs.append((nmlz(np.cross(pts[2,:], pts[3,:])), nmlz(np.cross(pts[3,:], pts[0,:]))))
        ln_pairs.append((nmlz(np.cross(pts[3,:], pts[0,:])), nmlz(np.cross(pts[0,:], pts[1,:]))))
        ln_pairs.append((nmlz(np.cross(pts[0,:], pts[2,:])), nmlz(np.cross(pts[1,:], pts[3,:]))))
    Y = []
    for line1, line2 in ln_pairs:
        r1 = line1[0]*line2[0]
        r2 = line1[0]*line2[1] + line1[1]*line2[0]
        r3 = line1[1]*line2[1]
        r4 = line1[0]*line2[2] + line1[2]*line2[0]
        r5 = line1[1]*line2[2] + line1[2]*line2[1]
        r6 = line1[2]*line2[2]
        Y.append([r1, r2, r3, r4, r5, r6])
    # print Y
    [_, _, V] = np.linalg.svd(Y, full_matrices = True)
    h = V.T[:,-1]
    h = h / h[-1]
    S = np.array([[h[0], h[1], h[3]],[h[1], h[2], h[4]],[h[3], h[4], h[5]]])
    # print S

    # Find 2 x 2
    [U, D2, V] = np.linalg.svd(S[:-1, :-1], full_matrices = True)
```

```
    H = np.eye(3)
    H[:-1,:-1] = np.dot(np.dot(V, np.diag(np.sqrt(D2))), V.T)
    vv = np.dot(np.linalg.inv(H[:-1,:-1]), np.array([h[3], h[4]])).T
    vv = vv / np.linalg.norm(vv)
    H[2,:-1] = vv
    return H

def apply_homography(img_path, H, num_partitions = 1, suff = ''):
    img = cv2.imread(img_path)
    img[0,:], img[:,0], img[-1,:], img[:,-1] = 0, 0, 0, 0
    xv, yv = np.meshgrid(range(0, img.shape[1], img.shape[1]-1), range(0, img.shape[0], img.shape[0]-
    img_pts = np.array([xv.flatten(), yv.flatten()]).T
    trans_img_pts = np.dot(H, real_to_homo(img_pts).T)
    trans_img_pts = homo_to_real(trans_img_pts.T).astype(int)

    xmin, ymin = np.min(trans_img_pts[:,0]), np.min(trans_img_pts[:,1])
    xmax, ymax = np.max(trans_img_pts[:,0]), np.max(trans_img_pts[:,1])
    W_new = xmax - xmin
    H_new = ymax - ymin

    img_new = np.zeros((H_new+1, W_new+1, 3), dtype = np.uint8)

    x_batch_sz = int(W_new/float(num_partitions))
    y_batch_sz = int(H_new/float(num_partitions))
    for x_part_idx in range(num_partitions):
        for y_part_idx in range(num_partitions):
            x_start, x_end = x_part_idx*x_batch_sz, (x_part_idx+1)*x_batch_sz
            y_start, y_end = y_part_idx*y_batch_sz, (y_part_idx+1)*y_batch_sz
            xv, yv = np.meshgrid(range(x_start, x_end), range(y_start, y_end))
            xv, yv = xv + xmin, yv + ymin
            img_new_pts = np.array([xv.flatten(), yv.flatten()]).T
            trans_img_new_pts = np.dot(hinv(H), real_to_homo(img_new_pts).T)
            trans_img_new_pts = homo_to_real(trans_img_new_pts.T).astype(int)
            trans_img_new_pts[:,0] = np.clip(trans_img_new_pts[:,0], 0, img.shape[1]-1)
            trans_img_new_pts[:,1] = np.clip(trans_img_new_pts[:,1], 0, img.shape[0]-1)
            img_new_pts = img_new_pts - [xmin, ymin]
            # This is the bottle nect step. It takes the most time.
            img_new[img_new_pts[:,1].tolist(), img_new_pts[:,0].tolist(), :] = img[trans_img_new_pts

    fname, ext = tuple(os.path.basename(img_path).split('.'))
    write_filepath = os.path.join(os.path.dirname(img_path), fname+suff+'.'+ext)
    print write_filepath
    cv2.imwrite(write_filepath, img_new)

def real_to_homo(pts):
    # pts is a 2D numpy array of size _ x 2/3
    # This function converts it into _ x 3/4 by appending 1
    if(pts.ndim == 1):
        return np.append(pts, 1)
    else:
        return np.concatenate((pts, np.ones((pts.shape[0], 1))), axis = 1)

def homo_to_real(pts):
```

```
    # pts is a 2D numpy array of size _ x 3/4
    # This function converts it into _ x 2/3 by removing last column
    if(pts.ndim == 1):
        pts = pts / pts[-1]
        return pts[:-1]
    else:
        pts = pts.T
        pts = pts / pts[-1,:]
        return pts[:-1,:].T

def save_mps(event, x, y, flags, param):
    fac, mps = param
    if(event == cv2.EVENT_LBUTTONUP):
        mps.append([int(fac*x), int(fac*y)])
        print(int(fac*x), int(fac*y))

def create_matching_points(img_path):
    npz_path = img_path[:-4] + '.npz'
    flag = os.path.isfile(npz_path)
    if(not flag):
        img = cv2.imread(img_path)
        fac = max(float(int(img.shape[1]/960)), float(int(img.shape[0]/540)))
        if(fac < 1.0): fac = 1.0
        resz_img = cv2.resize(img, None, fx=1.0/fac, fy=1.0/fac, interpolation = cv2.INTER_CUBIC)
        cv2.namedWindow('TempImage')
        mps = []
        cv2.setMouseCallback('TempImage', save_mps, param=(fac, mps))
        cv2.imshow('TempImage', resz_img)
        cv2.waitKey(0)
        np.savez(npz_path, mps = np.array(mps))
    return np.load(npz_path)

def nmlz(x):
    assert isinstance(x, np.ndarray), 'x should be a numpy array'
    assert x.ndim > 0 and x.ndim < 3, 'dim of x >0 and <3'
    if(x.ndim == 1 and x[-1]!=0): return x/float(x[-1])
    if(x.ndim == 2 and x[-1,-1]!=0): return x/float(x[-1,-1])
    return x

#########################
#          MAIN         #
#########################

import sys, os
import numpy as np
from glob import glob

sys.path.insert(0, os.path.join('..', 'utils'))
from helpers import *

import time

## Initialization
```

```
base_img_path = os.path.join('.', 'images')
all_img_paths = [os.path.join(base_img_path,'1.jpg'), os.path.join(base_img_path,'2.jpg'), os.path.jo
275
ref_img_size = (40, 40, 3)
clear_npz = True
vp_suff = '_vp'
gh_suff = '_gh'
280
if clear_npz:
    for _file in glob(os.path.join(base_img_path, '*.npz')):
        os.remove(_file)

285 M, N, _ = ref_img_size
cv2.imwrite(all_img_paths[-1], 255*np.ones(ref_img_size, dtype=np.uint8))

img_path = all_img_paths[3]

290 img_path_ref = all_img_paths[2]ag
pts = create_matching_points(img_path, suff = '_p2p')['mps']
pts_ref = np.array([[0,0],[N-1,0],[N-1,M-1],[0,M-1]])
## Point to point correspondence
H, Hinv = find_homography_2d(pts, pts_ref)
295 apply_homography2(img_path, Hinv, num_partitions = 4, suff = '_p2p')

# Vanishing line approach - Homework 3
# Two step approach
pts = create_matching_points(img_path, suff = vp_suff)['mps']
300 H_vl = find_homography_vl(pts)
print H_vl

apply_homography2(img_path, H_vl, num_partitions = 4, suff = vp_suff)
vp_img_path = os.path.splitext(img_path)[0] + vp_suff + '.jpg'
305 pts_vl = create_matching_points(vp_img_path, suff = '_')['mps']
H_af = find_homography_af(pts_vl)
apply_homography2(img_path, np.dot(hinv(H_af), H_vl), num_partitions=4, suff = '_out')



310 # ## General homography estimation (One step) - Homework 3
pts1 = create_matching_points(img_path, suff = gh_suff+'1')['mps']
# pts2 = create_matching_points(img_path, suff = gh_suff+'2')['mps']
# pts3 = create_matching_points(img_path, suff = gh_suff+'3')['mps']
H = find_homography_gh([pts1])
315 print H
print hinv(H)
apply_homography2(img_path, hinv(H), num_partitions = 4, suff = gh_suff)
```