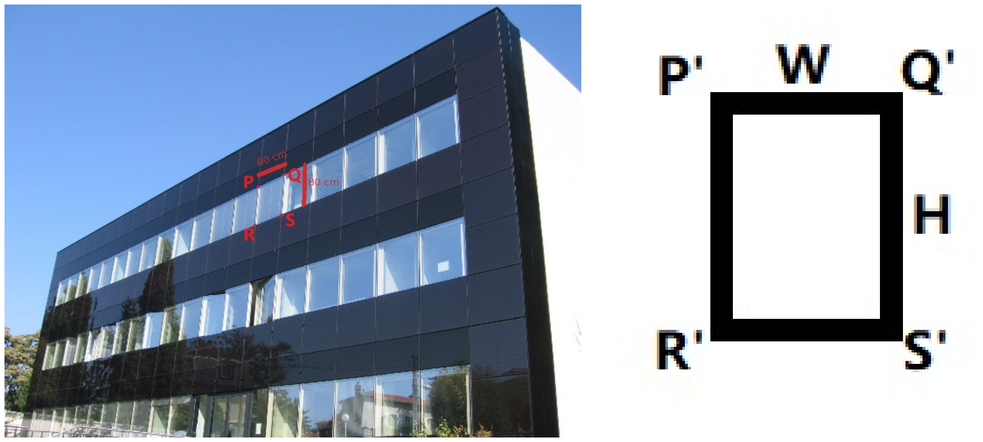
ECE 661 – Homework 3

Ran Xu

xu943@purdue.edu

09/13/2018

1. Logic – Point-to-Point Correspondences Method



(a) camera plane (b) world plane

Figure 1: Schematic diagram of point-to-point method, taking Building picture as an example

In the point-to-point method, we already know the real world measures of two perpendicular line segments. As shown in Fig. 1(a), the real measure of the window is given – the width PQ and height QS are W=60cm and H=80cm. Thus, we recover this rectangular in the world plane P'Q'S'R' with the width and height in homogeneous coordinates by $P' = (0,0,1)^T$, $Q' = (W,0,1)^T$, $R' = (0,H,1)^T$, $S' = (W,H,1)^T$

Denote the HC point coordinate $P(x_P, y_P, 1)$, $Q(x_Q, y_Q, 1)$, $R(x_R, y_R, 1)$, $S(x_S, y_S, 1)$, and the homography H from camera plane to the world plan as follows,

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

Then we can solve the required homography H by the following equation and apply it to the world plane image,

$$\begin{bmatrix} x_P & y_P & 1 & 0 & 0 & 0 & -x_P \cdot 0 & -y_P \cdot 0 \\ 0 & 0 & 0 & x_P & y_P & 1 & -x_P \cdot 0 & -y_P \cdot 0 \\ x_Q & y_Q & 1 & 0 & 0 & 0 & -x_Q W & -y_Q \mathrm{W} \\ 0 & 0 & 0 & x_Q & y_Q & 1 & -x_Q \cdot 0 & -y_Q \cdot 0 \\ x_R & y_R & 1 & 0 & 0 & 0 & -x_R \cdot 0 & -y_R \cdot 0 \\ 0 & 0 & 0 & x_R & y_R & 1 & -x_R \mathrm{H} & -y_R \mathrm{H} \\ x_S & y_S & 1 & 0 & 0 & 0 & -x_S \mathrm{W} & -y_S \mathrm{W} \\ 0 & 0 & 0 & x_S & y_S & 1 & -x_S \mathrm{H} & -y_S \mathrm{H} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathrm{W} \\ 0 \\ 0 \\ \mathrm{H} \\ \mathrm{W} \\ H \end{bmatrix}$$

2. Logic – 2-Step Method

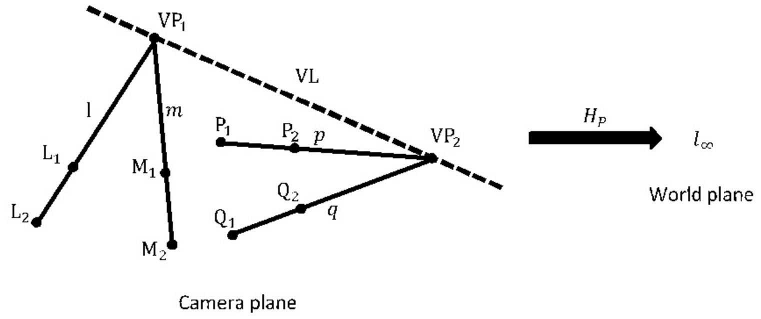


Figure 2: Schematic diagram of the first step -- removing projective transformation

Firstly, we remove the projective distortion as shown in Fig. 2. We particularly look at the line $l_\infty = (0,0,1)^{\mathrm{T}}$ in the world plane. This line maps to the vanishing line in the camera plane, denoted $VL = (VL_1, VL_2, VL_3)^{\mathrm{T}}$, with homography $\mathrm{H_P^{-1}}$. Thus, we can construct a qualified homograph H_P by

$$H_P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ VL_1 & VL_2 & VL_3 \end{bmatrix}$$

where the mapped vanishing line $\mathrm{H_P^{-T}} \cdot VL = (0,0,1)^{\mathrm{T}}$ is the line l_∞.

To find out the representation of the vanishing line in the camera plane. We can find out two sets of "parallel" line pairs (l, m) and (p, q) in the camera plane, which are supposed to be parallel in the world plane. Denote the two points on the four lines as shown in Fig. 2 as

$$\begin{cases} L_1 = \left(L_{1x}, L_{1y}, 1\right)^{\mathrm{T}} \\ L_2 = \left(L_{2x}, L_{2y}, 1\right)^{\mathrm{T}} \\ M_1 = \left(M_{1x}, M_{1y}, 1\right)^{\mathrm{T}} \\ M_2 = \left(M_{2x}, M_{2y}, 1\right)^{\mathrm{T}} \\ P_1 = \left(P_{1x}, P_{1y}, 1\right)^{\mathrm{T}} \\ P_2 = \left(P_{2x}, P_{2y}, 1\right)^{\mathrm{T}} \\ Q_1 = \left(Q_{1x}, Q_{1y}, 1\right)^{\mathrm{T}} \\ Q_2 = \left(Q_{2x}, Q_{2y}, 1\right)^{\mathrm{T}} \end{cases}$$

Then, the four lines can be computed as

$$\begin{cases} l = L_1 \times L_2 \\ m = M_1 \times M_2 \\ p = P_1 \times P_2 \\ q = Q_1 \times Q_2 \end{cases}$$

Then, the two vanishing points can be computed as

$$\begin{cases} VP_1 = l \times m \\ VP_2 = p \times q \end{cases}$$

Finally, the vanishing line can be computed as $VL = VP_1 \times VP_2$

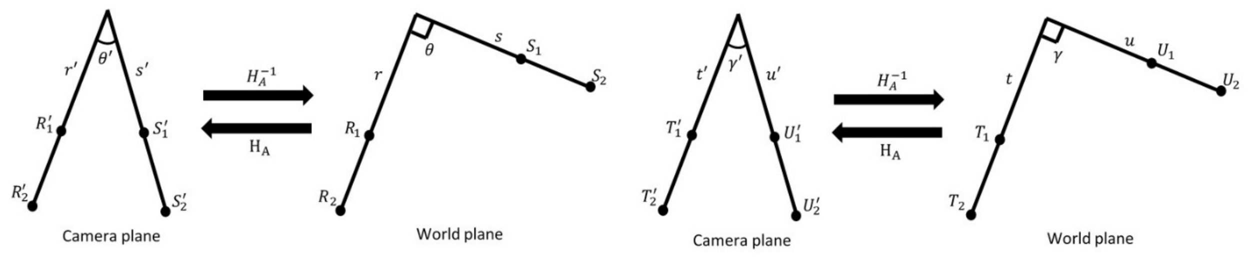


Figure 3 Schematic diagram of the second step -- removing affine transformation

Secondly, we remove the affine distortion after applying H_P to the camera image. As shown in the Fig. 3, a pair of perpendicular lines r and s in the world plane, with angle between $\theta = 90°$ satisfy the formula for $cos\theta$

$$cos\theta = \frac{r^T C_\infty^* s}{\sqrt{(r^T C_\infty^* r)(s^T C_\infty^* s)}} = 0$$

where the representation of r and s can be derived with the similar approach as discussed in the first method and C_∞^* is the dual degenerate conic at infinity, denoted

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

We denote an affine distortion from world plane to camera plane with homography

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

We can substitute r and s in the formula with the mapped lines $r' = (r_1', r_2', r_3')^T$ and $s' = (s_1', s_2', s_3')^T$ in the camera plane as

$$cos\theta = \frac{r'^T C_\infty^*{}' s'}{\sqrt{(r^T C_\infty^* r)(s^T C_\infty^* s)}} = 0$$

where $C_\infty^*{}'$ is the dual degenerate conic at infinity in the camera plane can be written as $C_\infty^*{}' = H_A C_\infty^* H_A^T$. Thus, we have

$$(r_1', r_2', r_3') \begin{bmatrix} A & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix} \begin{bmatrix} I & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix} \begin{bmatrix} A^T & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix} \begin{pmatrix} s_1' \\ s_2' \\ s_3' \end{pmatrix} = 0 \Rightarrow (r_1', r_2', r_3') \begin{bmatrix} AA^T & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix} \begin{pmatrix} s_1' \\ s_2' \\ s_3' \end{pmatrix} = 0 \xRightarrow{V=AA^T} (r_1', r_2') \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} \begin{pmatrix} s_1' \\ s_2' \end{pmatrix} = 0$$

Due to the symmetric property of V, we have

$$v_{11} r_1' s_1' + v_{12}(r_1' s_2' + r_2' s_1') + v_{22} r_2' s_2' = 0$$

To solve the matrix V, we can get another equation from another pair of "perpendicular" lines t' and u' in the camera plane.

$$v_{11} t_1' u_1' + v_{12}(t_1' u_2' + t_2' u_1') + v_{22} t_2' u_2' = 0$$

Due to the homogenous property of matrix V, we can solve V by the following equation and assume $v_{22} = 1$.

$$\begin{bmatrix} r_1' s_1' & r_1' s_2' + r_2' s_1' \\ t_1' u_1' & t_1' u_2' + t_2' u_1' \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} -r_2' s_2' \\ -t_2' u_2' \end{bmatrix}$$

To solve the matrix A in the homography H_A, we can use the eigendecomposition of V, denoted by $V = U\Sigma U^T$, where U is the eigenvector of V and Σ is a diagonal matrix with eigenvalues. The matrix A in the homography H_A can be calculated as $A = U\sqrt{\Sigma}$.

Note that we want to guarantee the positive definiteness of matrix A and thus the elements in A should satisfy the following equations,

$$\begin{cases} a_{11} > 0 \\ a_{11}a_{22} - a_{12}a_{21} > 0 \end{cases}$$

Finally, the homography to remove the affine distortion is the inverse of H_A and maps the camera plane back to the world plane. Thus the desired homography is

$$H_A^{-1} = \begin{bmatrix} A & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix}$$

3. Logic – 1-Step Method

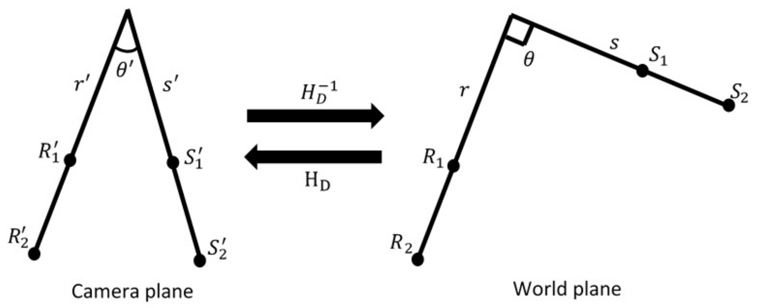


Figure 4: Schematic diagram of the 1-step method

As discussed in the two-step method, the angle θ of a pair of perpendicular lines r and s in the world plane satisfy

$$cos\theta = \frac{r^T C_\infty^* s}{\sqrt{(r^T C_\infty^* r)(s^T C_\infty^* s)}} = 0$$

where the representation of r and s can be derived with the similar approach as discussed in the first method and C_∞^* is the dual degenerate conic at infinity, denoted

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

We denote the projective and affine distortion from world plane to camera plane with homography H_D. We can substitute r and s in the formula with the mapped lines $r' = (r_1', r_2', r_3')^T$ and $s' = (s_1', s_2', s_3')^T$ in the camera plane as

$$cos\theta = \frac{r'^T C_\infty^*{}' s'}{\sqrt{(r^T C_\infty^* r)(s^T C_\infty^* s)}} = 0$$

where $C_\infty^*{}'$ is the dual degenerate conic at infinity in the camera plane, written as

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

With 5 independent pairs of perpendicular lines denoted $r^{[i]}$ and $s^{[i]}$ in the camera plane, where $i = 0,1,2,3,4$. We can solve $C_\infty^*{}'$ by the following equation, assuming $f = 1$ due to the homogenous property.

$$\begin{bmatrix} r_1^{[0]}s_1^{[0]} & r_1^{[0]}s_2^{[0]} + r_2^{[0]}s_1^{[0]} & r_2^{[0]}s_2^{[0]} & r_1^{[0]}s_3^{[0]} + r_3^{[0]}s_1^{[0]} & r_3^{[0]}s_2^{[0]} + r_2^{[0]}s_3^{[0]} \\ r_1^{[1]}s_1^{[1]} & r_1^{[1]}s_2^{[1]} + r_2^{[1]}s_1^{[1]} & r_2^{[1]}s_2^{[1]} & r_1^{[1]}s_3^{[1]} + r_3^{[1]}s_1^{[1]} & r_3^{[1]}s_2^{[1]} + r_2^{[1]}s_3^{[1]} \\ r_1^{[2]}s_1^{[2]} & r_1^{[2]}s_2^{[2]} + r_2^{[2]}s_1^{[2]} & r_2^{[2]}s_2^{[2]} & r_1^{[2]}s_3^{[2]} + r_3^{[2]}s_1^{[2]} & r_3^{[2]}s_2^{[2]} + r_2^{[2]}s_3^{[2]} \\ r_1^{[3]}s_1^{[3]} & r_1^{[3]}s_2^{[3]} + r_2^{[3]}s_1^{[3]} & r_2^{[3]}s_2^{[3]} & r_1^{[3]}s_3^{[3]} + r_3^{[3]}s_1^{[3]} & r_3^{[3]}s_2^{[3]} + r_2^{[3]}s_3^{[3]} \\ r_1^{[4]}s_1^{[4]} & r_1^{[4]}s_2^{[4]} + r_2^{[4]}s_1^{[4]} & r_2^{[4]}s_2^{[4]} & r_1^{[4]}s_3^{[4]} + r_3^{[4]}s_1^{[4]} & r_3^{[4]}s_2^{[4]} + r_2^{[4]}s_3^{[4]} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} -r_3^{[0]}s_3^{[0]} \\ -r_3^{[1]}s_3^{[1]} \\ -r_3^{[2]}s_3^{[2]} \\ -r_3^{[3]}s_3^{[3]} \\ -r_3^{[4]}s_3^{[4]} \end{bmatrix}$$

Finally, we perform eigendecomposition for $C_\infty^*{}'$, getting $C_\infty^*{}' = U\Sigma U^T$ and thus $H_D = U\sqrt{\Sigma}$. Note that the homography from image plane to world plane is the inversed form, i.e.

$$H_D{}^{-1} = \sqrt{\Sigma^{-1}}U^T$$

4.1 Results for the two given images

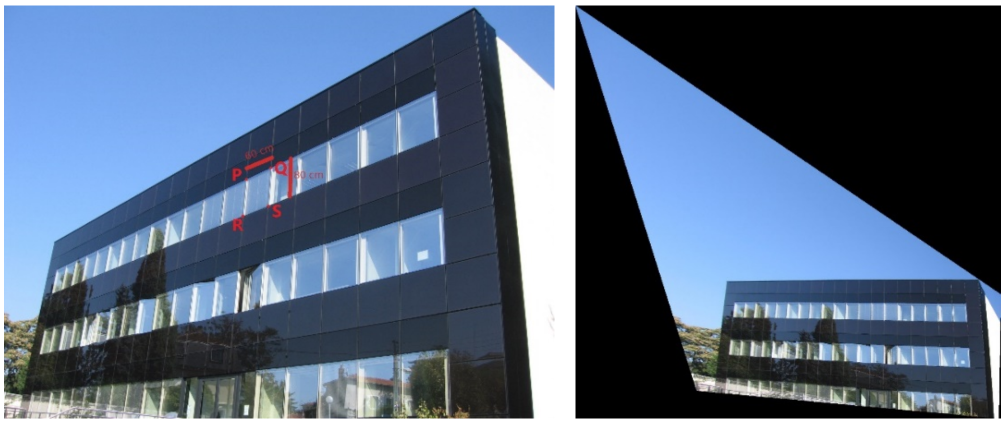
1. Using Point-to-Point Correspondences

The corresponding points in shown in Table 1 and the input/output images are shown in Fig. 5 and 6.

<u>Observation</u>: The distortions on the window and frame can be perfectly removed but those on the remaining part of the images are not guaranteed to be removed.

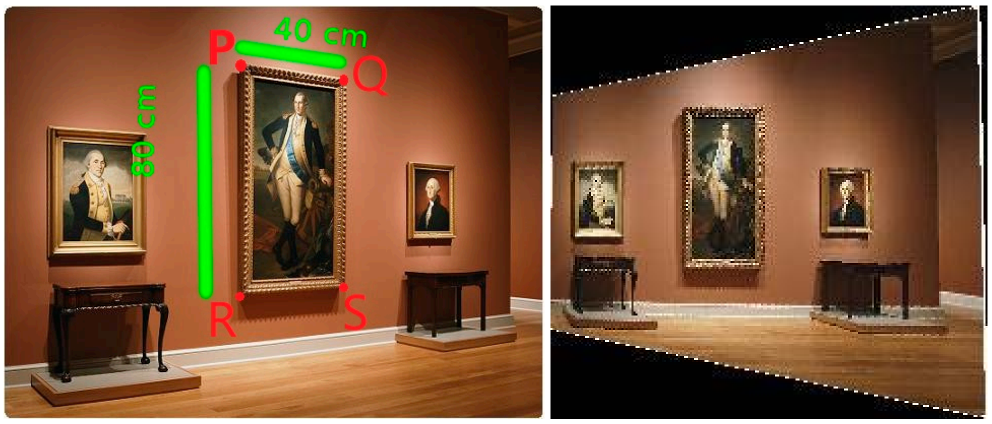
Table 1: corresponding points used in the first p2p method for the two given images

	$P\ (P')$	$Q\ (Q')$	$R\ (R')$	$S\ (S')$
Building image (image plane)	$(1141, 815, 1)^T$	$(1256, 761, 1)^T$	$(1126, 993, 1)^T$	$(1245, 945, 1)^T$
Original building (world plane)	$(0, 0, 1)^T$	$(60, 0, 1)^T$	$(0, 80, 1)^T$	$(60, 80, 1)^T$
Homography	$H = \begin{bmatrix} 1.37 & 0.115 & -1.65 \times 10^{3} \\ 0.543 & 1.18 & -1.58 \times 10^{3} \\ 1.01 \times 10^{-3} & 3.81 \times 10^{-4} & 1.00 \end{bmatrix}$			
Portraits image (image plane)	$(232, 57, 1)^T$	$(336, 73, 1)^T$	$(232, 286, 1)^T$	$(335, 278, 1)^T$
Original portraits (world plane)	$(0, 0, 1)^T$	$(40, 0, 1)^T$	$(0, 80, 1)^T$	$(40, 80, 1)^T$
Homography	$H = \begin{bmatrix} 0.278 & -3.42 \times 10^{-17} & -64.6 \\ -4.30 \times 10^{-2} & 0.278 & -5.96 \\ -8.12 \times 10^{-4} & -3.79 \times 10^{-5} & 1.00 \end{bmatrix}$			



(a) Input image with real world measures (b) Resulting image

Figure 5: Removing distortions of building image with p2p method



(a) Input image with real world measures (b) Resulting image

Figure 6: Removing distortions of portrait image with p2p method

2. Using 2-step method

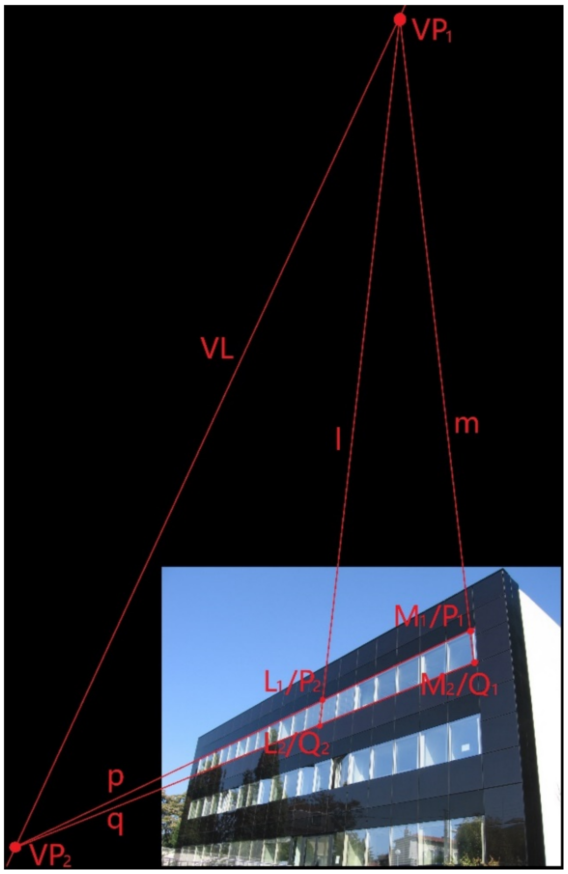
(a1) Using 2-step method – removing projective distortion from the building image

In this step, I got 2 pairs of “parallel” lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 7(a) and the HC of the 8 points in summarized in Table 2. The resulting image in shown in Fig. 7(b).

<u>Observation</u>: the annotation of the 8 points must be very accurate to calculate the vanishing line. Otherwise it may drift dramatically and thus H_p may not perfectly remove the projective distortion.

Table 2: Point, line and homography representations in this step

	L_1	L_2	M_1	M_2	P_1	P_2	Q_1	Q_2
HC	$(1044,869,1)^T$	$(1025,1030,1)^T$	$(2024,420,1)^T$	$(2038,619,1)^T$	$(2024,420,1)^T$	$(1044,869,1)^T$	$(2038,619,1)^T$	$(1025,1030,1)^T$
	VP_1				VP_2			
HC	$(1678,-4501,1)^T$				$(-1879,2208,1)^T$			
	VL							
HC	$(1.41\times10^{-3},7.48\times10^{-4},1)$							
	H_p (mapping from camera plane to world plane)							
HC	$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 1.41\times10^{-3} & 7.48\times10^{-4} & 1\end{bmatrix}$							



(a) Input image with selected “parallel” lines

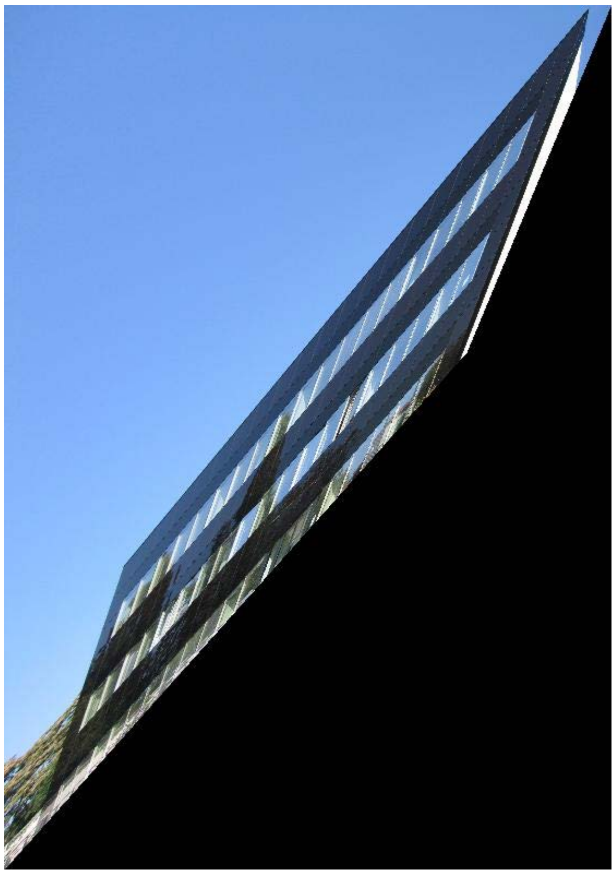

(b) Resulting image

Figure 7: Removing the projective distortion of building image with 2-step method

(a2) Using 2-step method – removing affine distortion from the building image

In this step, I first apply the homography H_p in the first step to the input image and regard the output image from the first step as the input image in this step.

I find 2 pairs of "perpendicular" lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 8(a) and the HC of the 8 points in summarized in Table 3. The resulting image in shown in Fig. 8(b).

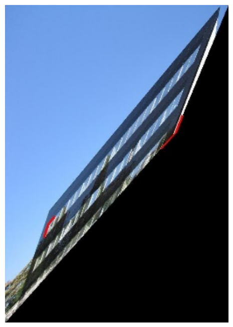
<u>Observation</u>:

(1) As shown in Fig. 8(b), the "perpendicular" lines in the image plane are truly recovered to be perpendicular. However, there is still similarity distortion and thus the build is not positioned in the direction as in the real world.

(2) According to the equations in the first section, the matrix to solve V may be highly ill-conditioned and thus induce large error. The reason is that only the first two coordinates in the four lines, a.k.a the slope, are used and in practice there are only 1 obvious pair of perpendicular lines (horizontal lines and vertical lines) available in the image, resulting in redundant second pair of perpendicular lines.

Table 3: Point, line and homography representations in this step

	R_1	R_2	S_1	S_2	T_1	T_2	U_1	U_2
HC	$(108,546,1)^T$	$(97,579,1)^T$	$(108,546,1)^T$	$(124,527,1)^T$	$(425,318,1)^T$	$(389,358,1)^T$	$(325,318,1)^T$	$(442,275,1)^T$
	r		s		t		u	
HC	$(-33,-11,9570)^T$		$(19,16,-10788)^T$		$(-40,-36,28448)^T$		$(43,17,-23681)^T$	
	H_a^{-1} (mapping from camera plane to world plane)							
HC	$\begin{bmatrix} 3.95 & 2.59 & 0 \\ -0.46 & 0.70 & 0 \\ 0 & 0 & 1 \end{bmatrix}$							


(a) Output image from the first step

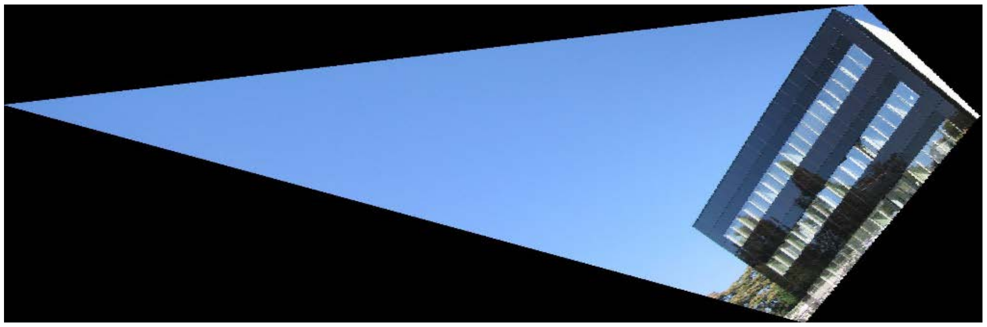

(b) Resulting image

Figure 8: Removing the affine distortion of building image with 2-step method

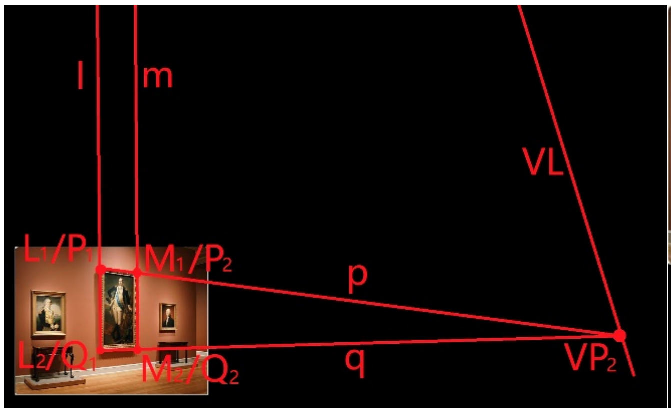
(b1) Using 2-step method– removing projective distortion from the portrait image

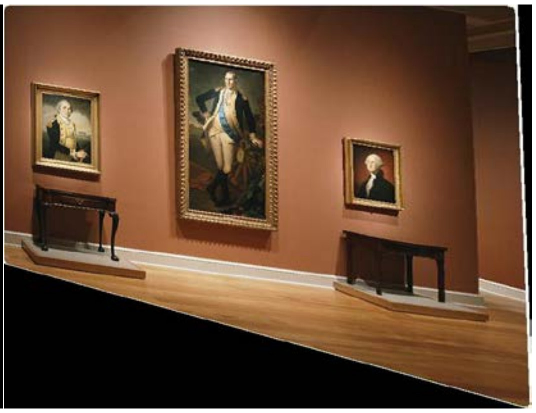
In this step, I find 2 pairs of "parallel" lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 9(a) and the HC of the 8 points in summarized in Table 4. The resulting image in shown in Fig. 9(b).

<u>Observation</u>: the annotation of the 8 points must be very accurate to calculate the vanishing line. Otherwise it may drift dramatically and thus H_p may not perfectly remove the projective distortion.

Table 4: Point, line and homography representations in this step

	L_1	L_2	M_1	M_2	P_1	P_2	Q_1	Q_2
HC	$(232,57,1)^T$	$(231,281,1)^T$	$(338,73,1)^T$	$(336,278,1)^T$	$(232,57,1)^T$	$(338,73,1)^T$	$(231,281,1)^T$	$(336,278,1)^T$
	VP_1				VP_2			
HC	$(142,20117,1)^T$				$(1480, 245,1)$			
	VL							
HC	$(-6.68 \times 10^{-4}, -4.50 \times 10^{-5}, 1)$							
	H_p (mapping from camera plane to world plane)							
HC	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -6.68 \times 10^{-4} & -4.50 \times 10^{-5} & 1 \end{bmatrix}$							





(a) Input image with selected "parallel" lines (b) Resulting image

Figure 9: Removing the projective distortion of portrait image with 2-step method

(b2) Using 2-step method – removing affine distortion from the portrait image

In this step, I first apply the homography H_p in the first step to the input image and regard the output image from the first step as the input image in this step.

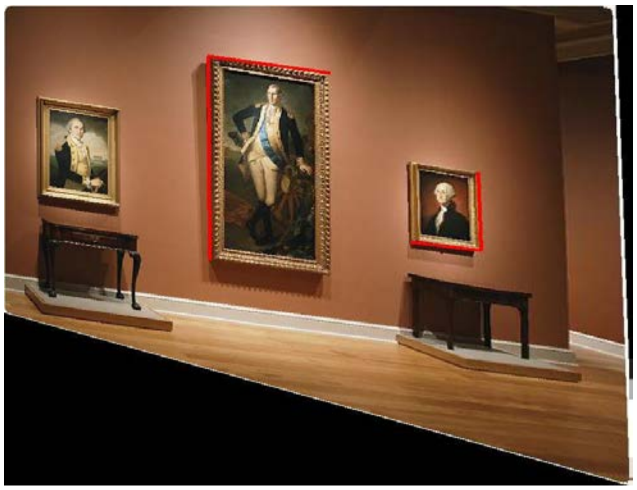
I find 2 pairs of "perpendicular" lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 10(a) and the HC of the 8 points in summarized in Table 5. The resulting image in shown in Fig. 10(b).

<u>Observation</u>:

As shown in Fig. 10(b), the "perpendicular" lines in the image plane are not perfectly recovered to be perpendicular. The reason is that the two pairs of lines should lie on the same world plane, while they do not actually. The different distortions on each part of the image make it impossible to remove the affine distortion of the whole image.

Table 5: Point, line and homography representations in this step

	R_1	R_2	S_1	S_2	T_1	T_2	U_1	U_2
HC	$(275,68,1)^T$	$(277,343,1)^T$	$(275,68,1)^T$	$(439,91,1)^T$	$(643,330,1)^T$	$(548,319,1)^T$	$(643,330,1)^T$	$(639,225,1)^T$
	r		s		t		u	
HC	$(-275,2,75489)^T$		$(-23,164,-4827)^T$		$(11,-95,24277)^T$		$(105,-4,-66195)^T$	
	H_a^{-1} (mapping from camera plane to world plane)							
HC	$\begin{bmatrix} 0.762 & 0.412 & 0 \\ -0.501 & 0.926 & 0 \\ 0 & 0 & 1 \end{bmatrix}$							


(a) Output image from the first step

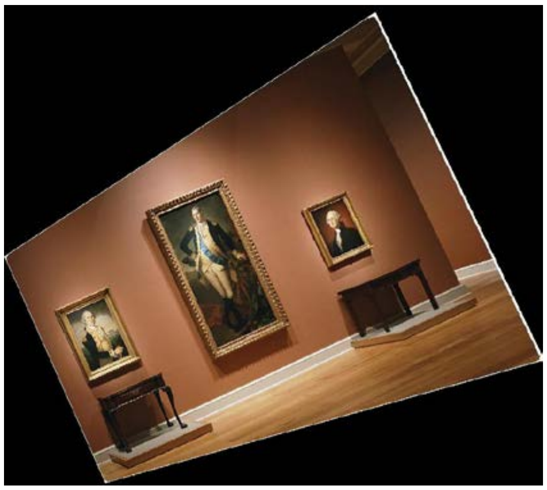

(b) Resulting image

Figure 10: Removing the affine distortion of portrait image with 2-step method

3a. Using 1-step method from the building image

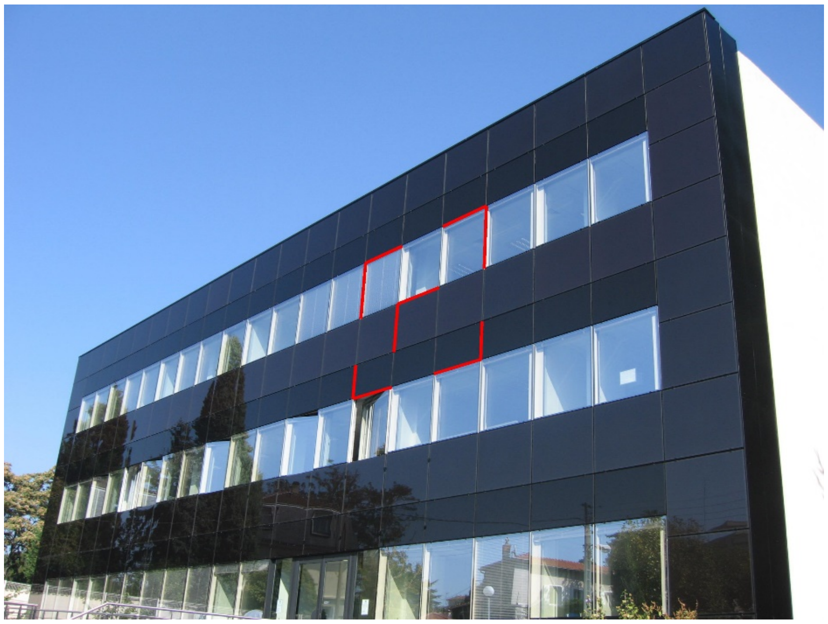
In this step, I find 5 pairs of "perpendicular" lines in the camera plane, with a total of 20 points to derive the 10 lines. The selection of the 20 points in shown in Fig. 11(a) and the HC of the 20 points in summarized in Table 6. The resulting image in shown in Fig. 11(b).

<u>**Observation**</u>**:**

As shown in Fig. 11(b), the image is nearly perfectly recovered although similarity distortion still exists in the resulting image because by removing the projective and affine distortion, we can only recover "parallel" and "perpendicular" lines in the image plane to be parallel and perpendicular in the world plane.

Table 6: Point, conic and homography representations in this step

Perpendicular lines	R_1	R_2	S_1	S_2
1st pair	$(1142,817,1)^T$	$(1127,993,1)^T$	$(1142,817,1)^T$	$(1258,763,1)^T$
2nd pair	$(1524,637,1)^T$	$(1387,702,1)^T$	$(1524,637,1)^T$	$(1517,835,1)^T$
3rd pair	$(1507,1122,1)^T$	$(1511,1001,1)^T$	$(1507,1122,1)^T$	$(1359,1167,1)^T$
4th pair	$(1103,1245,1)^T$	$(1112,1140,1)^T$	$(1103,1245,1)^T$	$(1225,1209,1)^T$
5th pair	$(1244,946,1)^T$	$(1375,894,1)^T$	$(1244,946,1)^T$	$(1232,1098,1)^T$
Conic and homogrphy	$C'_\infty = \begin{bmatrix} 1.69 \times 10^6 & -2.49 \times 10^6 & -1.14 \times 10^3 \\ -2.49 \times 10^6 & 5.08 \times 10^6 & 1.85 \times 10^3 \\ -1.14 \times 10^3 & 1.85 \times 10^3 & 1 \end{bmatrix}$		$H_D^{-1} = \begin{bmatrix} 0.185 & -0.349 & -1.34 \times 10^{-4} \\ 1.45 & 0.764 & -6.40 \times 10^{-4} \\ 5.04 \times 10^{-4} & -1.16 \times 10^{-4} & 1 \end{bmatrix}$	


(a) Input image

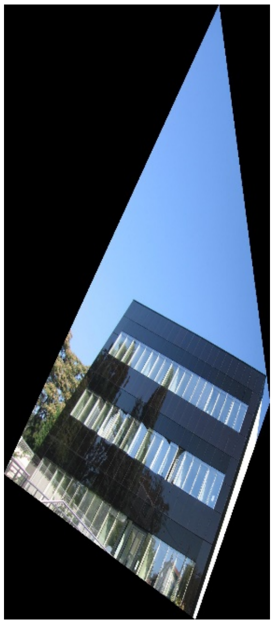

(b) Resulting image

Figure 11: Removing the projective and affine distortions of the building image with 1-step method

3b. Using 1-step method from the portrait image

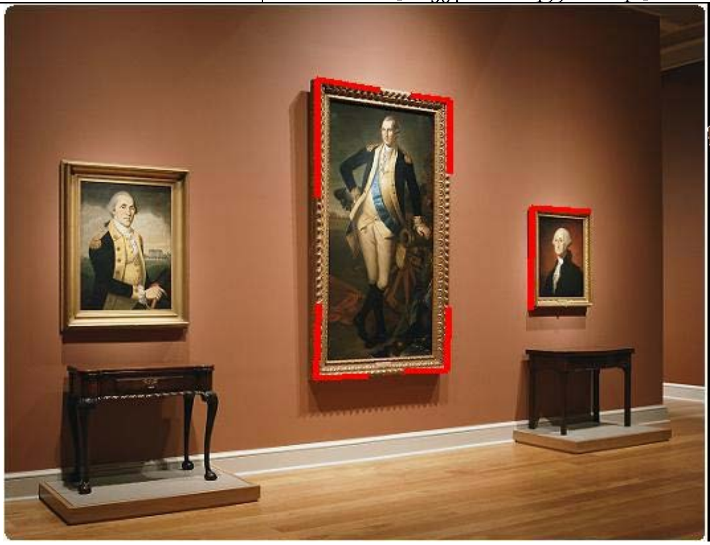
In this step, I find 5 pairs of "perpendicular" lines in the camera plane, with a total of 20 points to derive the 10 lines. The selection of the 20 points in shown in Fig. 12(a) and the HC of the 20 points in summarized in Table 7. The resulting image in shown in Fig. 12(b).

<u>Observation</u>:

As shown in Fig. 12(b), the image is also nearly perfectly recovered although similarity distortion still exists in the resulting image because by removing the projective and affine distortion, we can only recover "parallel" and "perpendicular" lines in the image plane to be parallel and perpendicular in the world plane.

Table 7: Point, conic and homography representations in this step

Perpendicular lines	R_1	R_2	S_1	S_2
1st pair	$(237,57,1)^T$	$(284,65,1)^T$	$(237,57,1)^T$	$(237,146,1)^T$
2nd pair	$(236,283,1)^T$	$(238,227,1)^T$	$(236,283,1)^T$	$(275,281,1)^T$
3rd pair	$(335,277,1)^T$	$(303,278,1)^T$	$(335,277,1)^T$	$(336,230,1)^T$
4th pair	$(337,73,1)^T$	$(308,69,1)^T$	$(337,73,1)^T$	$(337,128,1)^T$
5th pair	$(399,154,1)^T$	$(398,231,1)^T$	$(399,154,1)^T$	$(443,157,1)^T$
Conic and homogrphy	$C'_\infty = \begin{bmatrix} 5.28\times10^5 & 1.20\times10^5 & 564 \\ 1.20\times10^5 & 1.42\times10^5 & 199 \\ 564 & 199 & 1 \end{bmatrix}$		$H_D^{-1} = \begin{bmatrix} 1.28 & 0.366 & 1.42\times10^{-3} \\ -0.835 & 2.93 & 1.04\times10^{-3} \\ -9.28\times10^{-4} & -6.20\times10^{-4} & 1 \end{bmatrix}$	


(a) Input image

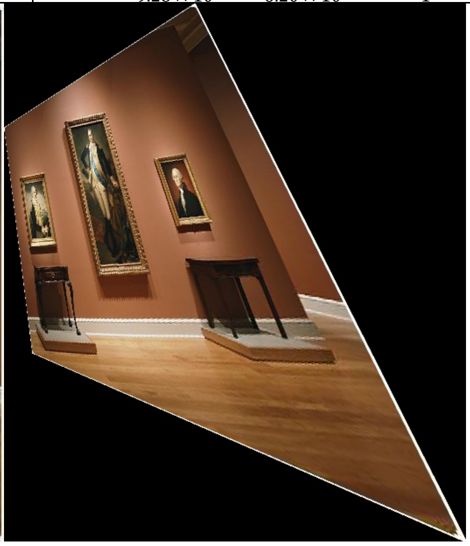

(b) Resulting image

Figure 12: Removing the projective and affine distortions of the portrait image with 1-step method

<u>Observation on the relative performance of the two methods:</u>

The two-step method is more stable than the one-step method because it is relatively easier to find out two pairs of parallel lines and two pairs of perpendicular lines in the image. However, finding 5 pairs of perpendicular lines as in the one-step method may cause ill-conditioned matrix because there are no 5 irrelevant pairs of perpendicular lines in most of the real world images. Thus the sigular value may be negative sometimes abd the solution is very vulnerable to the selection of line pairs. So, I believe the two-step method is better than the one-step ones.

4.2 Results for the two own images

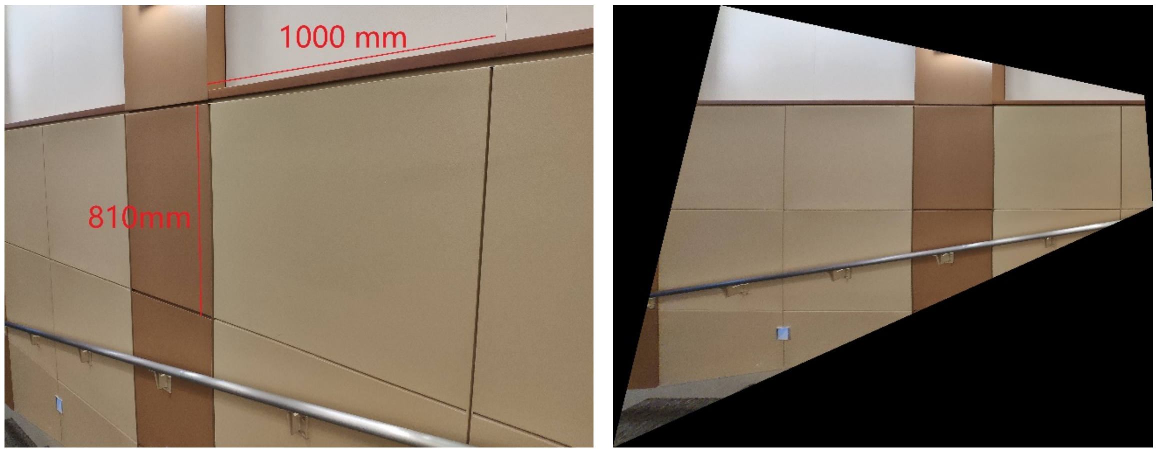
1. Using Point-to-Point Correspondences

The corresponding points in shown in Table 8 and the input/output images are shown in Fig. 13 and 14.

<u>Observation</u>: The distortions on the both images are perfectly removed.

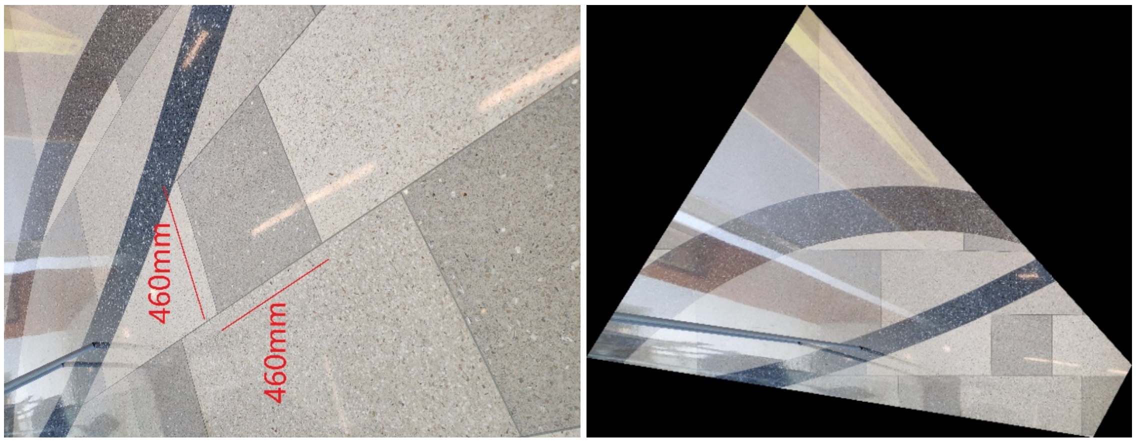
Table 8: corresponding points used in the first p2p method for the two given images

	$P\ (P')$	$Q\ (Q')$	$R\ (R')$	$S\ (S')$
Wall image (image plane)	$(1407,675,1)^T$	$(3333,420,1)^T$	$(1434,2145,1)^T$	$(3189,2790,1)^T$
Original wall (world plane)	$(0,0,1)^T$	$(1000,0,1)^T$	$(0,810,1)^T$	$(1000,810,1)^T$
Homography	$H = \begin{bmatrix} 1.54 & -2.83\times10^{-2} & -2.15\times10^{3} \\ 0.123 & 0.925 & -797 \\ 6.02\times10^{-4} & -8.41\times10^{-5} & 1.00 \end{bmatrix}$			
Floor image (image plane)	$(1216,1224,1)^T$	$(1774,558,1)^T$	$(1498,2158,1)^T$	$(2226,1666,1)^T$
Original floor (world plane)	$(0,0,1)^T$	$(460,0,1)^T$	$(0,460,1)^T$	$(460,460,1)^T$
Homography	$H = \begin{bmatrix} 1.65 & -0.497 & -1.39\times10^{3} \\ 1.05 & 0.877 & -2.35\times10^{3} \\ 9.75\times10^{-4} & -1.78\times10^{-5} & 1.00 \end{bmatrix}$			



(a) Input image with real world measures (b) Resulting image

Figure 13: Removing distortions of the wall image with p2p method



(a) Input image with real world measures (b) Resulting image

Figure 14: Removing distortions of the floor image with p2p method

2. Using 2-step method

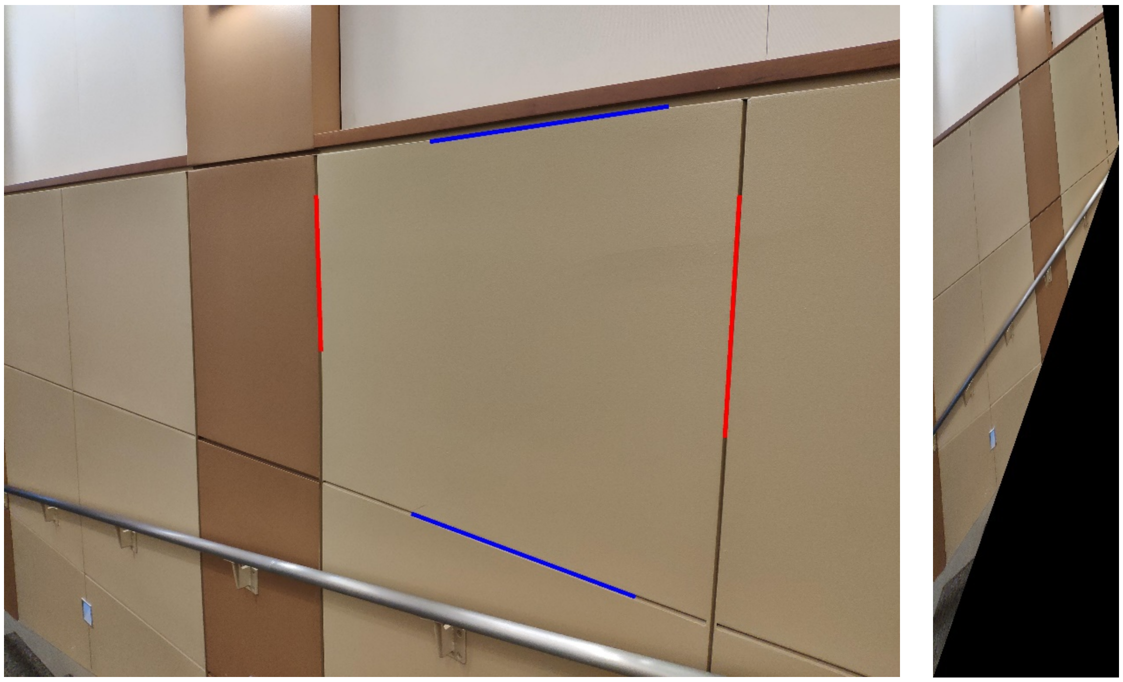
(a1) Using 2-step method – removing projective distortion from the wall image

In this step, I got 2 pairs of "parallel" lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 15(a) and the HC of the 8 points in summarized in Table 9. The resulting image in shown in Fig. 15(b).

Observation: the annotation of the 8 points must be very accurate to calculate the vanishing line. Otherwise it may drift dramatically and thus H_p may not perfectly remove the projective distortion.

Table 9: Point, line and homography representations in this step

	L_1	L_2	M_1	M_2	P_1	P_2	Q_1	Q_2
HC	$(1404,855,1)^T$	$(1425,1557,1)^T$	$(3309,855,1)^T$	$(3243,1947,1)^T$	$(2991,456,1)^T$	$(1917,615,1)^T$	$(2841,2664,1)^T$	$(1833,2289,1)^T$
	VP_1				VP_2			
HC	$(2035,21938,1)^T$				$(-1361,1100,1)^T$			
	VL							
HC	$(6.49\times10^{-4},-1.06\times10^{-4},1)^T$							
	H_p (mapping from camera plane to world plane)							
HC	$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 6.49\times10^{-4} & -1.06\times10^{-4} & 1\end{bmatrix}$							



(a) Input image with selected "parallel" lines (b) Resulting image

Figure 15: Removing the projective distortion of the wall image with 2-step method

(a2) Using 2-step method – removing affine distortion from the wall image

In this step, I first apply the homography H_p in the first step to the input image and regard the output image from the first step as the input image in this step.

I find 2 pairs of "perpendicular" lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 16(a) and the HC of the 8 points in summarized in Table 10. The resulting image in shown in Fig. 16(b).

<u>**Observation**</u>**:**

As shown in Fig. 16(b), the "perpendicular" lines in the image plane are truly recovered to be perpendicular. However, there is still similarity distortion and thus the build is not positioned in the direction as in the real world.

Table 10: Point, line and homography representations in this step

	R_1	R_2	S_1	S_2	T_1	T_2	U_1	U_2
HC	$(567,516,1)^T$	$(339,684,1)^T$	$(567,516,1)^T$	$(600,968,1)^T$	$(305,1701,1)^T$	$(284,1425,1)^T$	$(305,1701,1)^T$	$(464,1575,1)^T$
	r		s		t		u	
HC	$(-168,-228,212904)^T$		$(-452,33,239256)^T$		$(276,-21,-48459)^T$		$(126,159,-308889)^T$	
	H_a^{-1} (mapping from camera plane to world plane)							
HC	$\begin{bmatrix} 4.15 & -0.14 & 0 \\ 3.3\times10^{-2} & 1.00 & 0 \\ 0 & 0 & 1 \end{bmatrix}$							

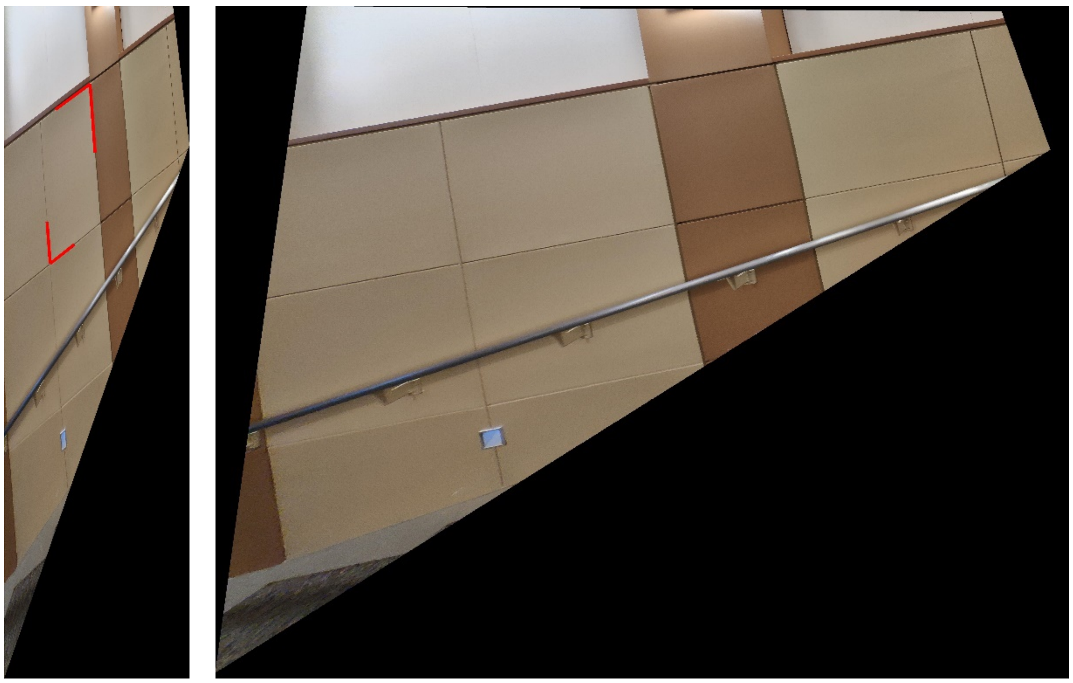

(a) Output image from the first step (b) Resulting image

Figure 16: Removing the affine distortion of building image with 2-step method

(b1) Using 2-step method– removing projective distortion from the floor image

In this step, I find 2 pairs of “parallel” lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 17(a) and the HC of the 8 points in summarized in Table 9. The resulting image in shown in Fig. 17(b).

<u>Observation</u>: the annotation of the 8 points must be very accurate to calculate the vanishing line. Otherwise it may drift dramatically and thus H_p may not perfectly remove the projective distortion.

Table 11: Point, line and homography representations in this step

	L_1	L_2	M_1	M_2	P_1	P_2	Q_1	Q_2
HC	$(1254,1353,1)^T$	$(1428,1929,1)^T$	$(1884,825,1)^T$	$(2133,1428,1)^T$	$(1671,681,1)^T$	$(1341,1071,1)^T$	$(2112,1749,1)^T$	$(1653,2058,1)^T$
	VP_1				VP_2			
HC	$(-1057,-6297,1)^T$				$(-1013,3852,1)$			
	VL							
HC	$(9.71\times10^{-4},-4.25\times10^{-6},1)$							
	H_p (mapping from camera plane to world plane)							
HC	$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\9.71\times10^{-4} & -4.25\times10^{-6} & 1\end{bmatrix}$							

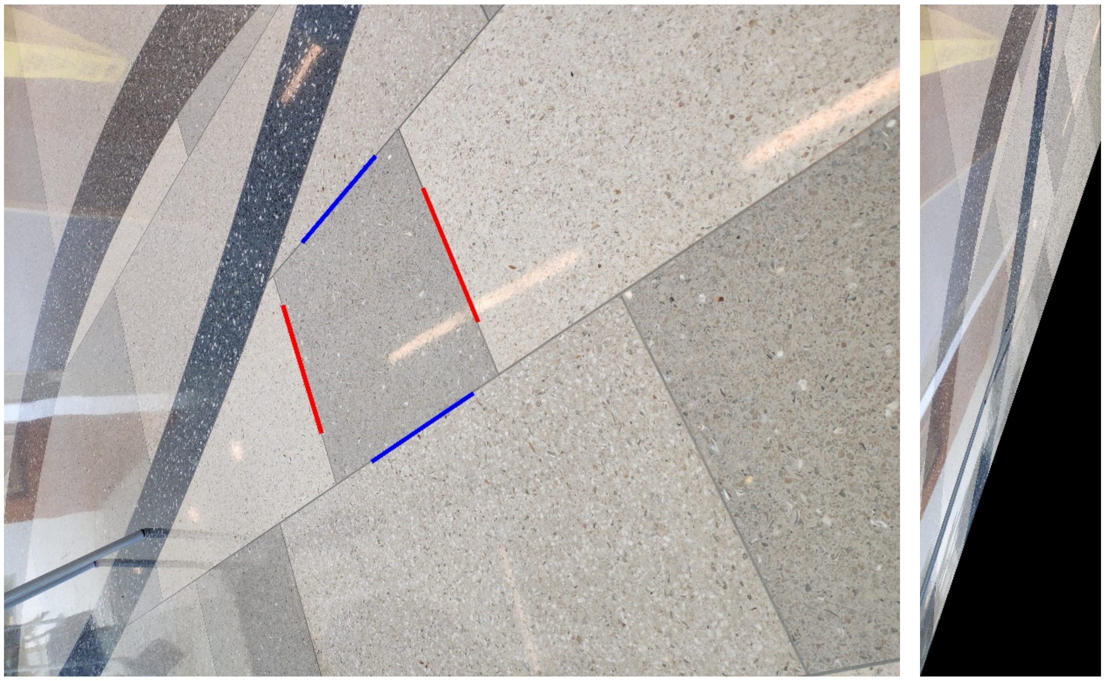

(a) Input image with selected “parallel” lines (b) Resulting image

Figure 17: Removing the projective distortion of portrait image with 2-step method

(b2) Using 2-step method – removing affine distortion from the floor image

In this step, I first apply the homography H_a in the first step to the input image and regard the output image from the first step as the input image in this step.

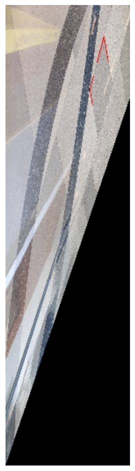
I find 2 pairs of "perpendicular" lines in the camera plane, with a total of 8 points to derive the 4 lines. The selection of the 8 points in shown in Fig. 18(a) and the HC of the 8 points in summarized in Table 12. The resulting image in shown in Fig. 18(b).

<u>**Observation**</u>**:**

As shown in Fig. 18(b), the "perpendicular" lines in the image plane is perfectly recovered to be perpendicular in the world plane.

Table 12: Point, line and homography representations in this step

	R_1	R_2	S_1	S_2	T_1	T_2	U_1	U_2
HC	$(651,207,1)^T$	$(604,384,1)^T$	$(651,207,1)^T$	$(680,379,1)^T$	$(558,563,1)^T$	$(575,663,1)^T$	$(558,563,1)^T$	$(582,470,1)^T$
	r		s		t		u	
HC	$(-177,-47,124956)^T$		$(-172,29,105969)^T$		$(-100,17,46229)^T$		$(93,24,-65406)^T$	
	H_a^{-1} (mapping from camera plane to world plane)							
HC	$\begin{bmatrix} 6.37 & -0.662 & 0 \\ 0.103 & 0.989 & 0 \\ 0 & 0 & 1 \end{bmatrix}$							


(a) Output image from the first step

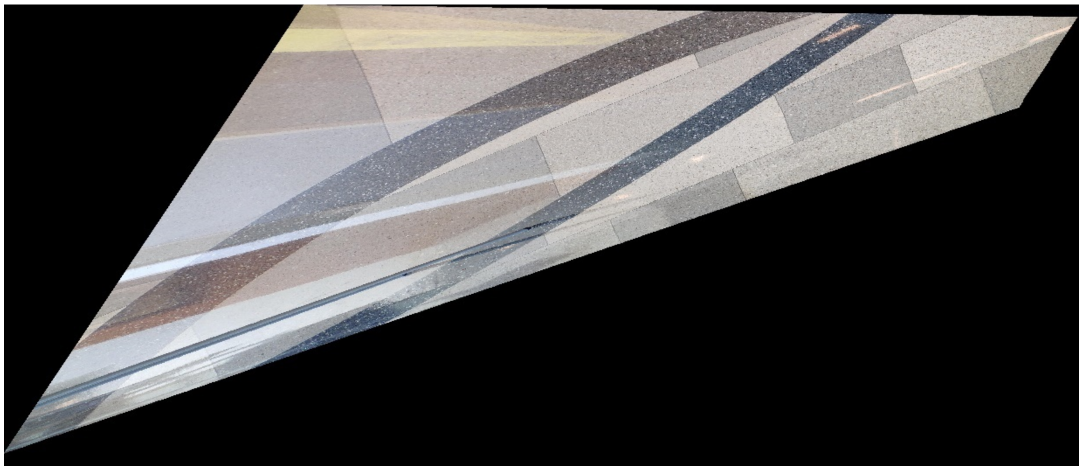
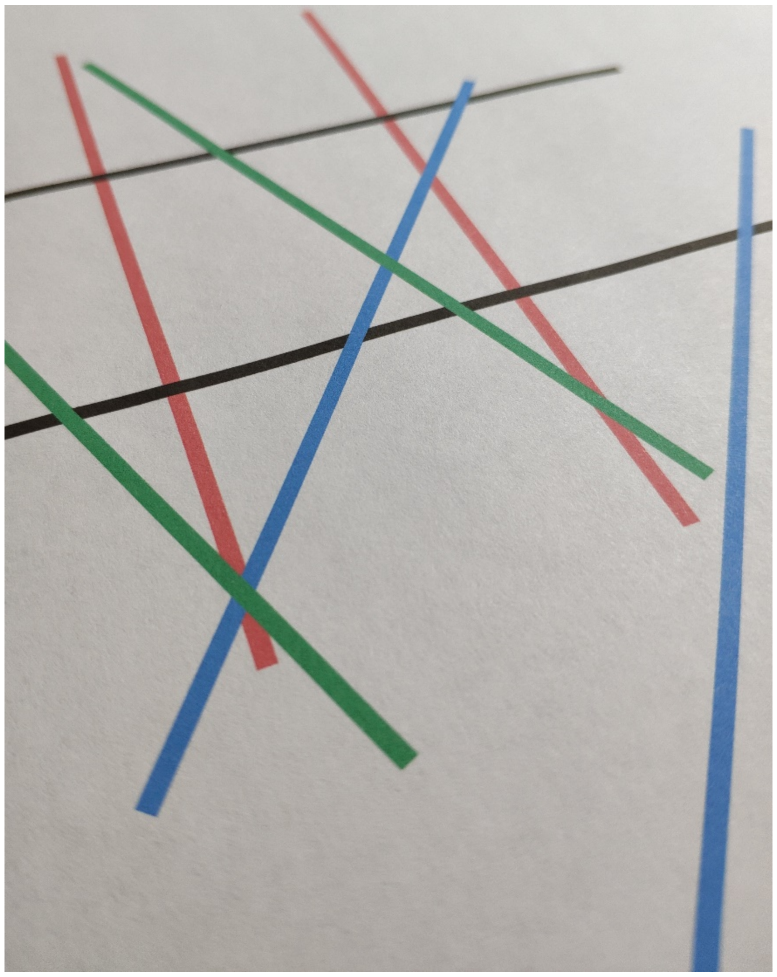

(b) Resulting image

Figure 18: Removing the affine distortion of the floor image with 2-step method

Bonus section: will multiple sets of parallel lines give the same vanishing line?

The answer is no. Typically, vanishing line is very far away and sensitive to the human's annotation of the points on the lines. As shown in the following figure. The vanishing line computed based on black and blue pairs of lines is [-4.97e-06, 4.24e-04, 1] with slope of 0.0117, while the vanishing line computed based on red and green pairs of lines is [-2.69e-06, 4.245e-04, 1] with slope of 0.00632.



3a. Using 1-step method from the wall image

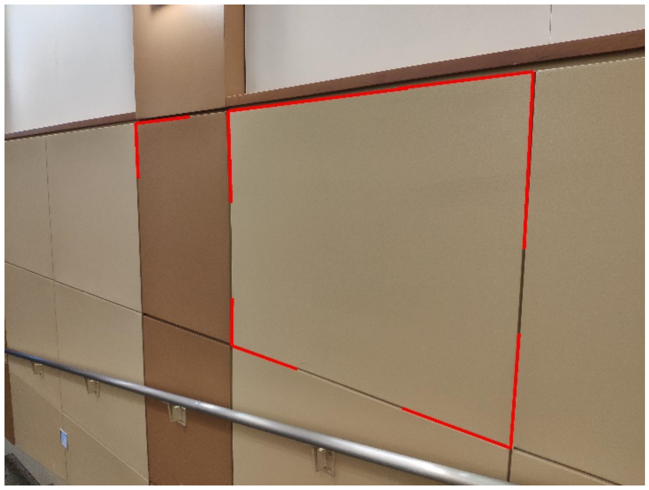
In this step, I find 5 pairs of "perpendicular" lines in the camera plane, with a total of 20 points to derive the 10 lines. The selection of the 20 points in shown in Fig. 19(a) and the HC of the 20 points in summarized in Table 13. The resulting image in shown in Fig. 19(b).

<u>**Observation**</u>**:**

As shown in Fig. 19(b), the image is nearly perfectly recovered although similarity distortion still exists in the resulting image because by removing the projective and affine distortion, we can only recover "parallel" and "perpendicular" lines in the image plane to be parallel and perpendicular in the world plane.

Table 13: Point, line and homography representations in this step

Perpendicular lines	R_1	R_2	S_1	S_2
1st pair	$(1407,669,1)^T$	$(1425,1245,1)^T$	$(1407,669,1)^T$	$(2541,525,1)^T$
2nd pair	$(3327,423,1)^T$	$(2478,528,1)^T$	$(3327,423,1)^T$	$(3270,1539,1)^T$
3rd pair	$(3186,2790,1)^T$	$(3240,2067,1)^T$	$(3186,2790,1)^T$	$(2502,2538,1)^T$
4th pair	$(1431,2145,1)^T$	$(1842,2292,1)^T$	$(1431,2145,1)^T$	$(1434,1845,1)^T$
5th pair	$(825,747,1)^T$	$(840,1092,1)^T$	$(825,747,1)^T$	$(1164,705,1)^T$
Conic and homogrphy	$C'_\infty = \begin{bmatrix} 1.69\times10^6 & -2.49\times10^6 & -1.14\times10^3 \\ -2.49\times10^6 & 5.08\times10^6 & 1.85\times10^3 \\ -1.14\times10^3 & 1.85\times10^3 & 1 \end{bmatrix}$		$H_D^{-1} = \begin{bmatrix} 0.185 & -0.349 & -1.34\times10^{-4} \\ 1.45 & 0.764 & -6.40\times10^{-4} \\ 5.04\times10^{-4} & -1.16\times10^{-4} & 1 \end{bmatrix}$	


(a) Input image

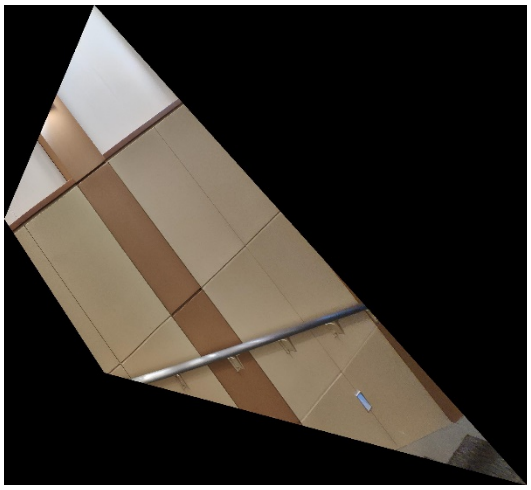

(b) Resulting image

Figure 19: Removing the projective and affine distortions of the wall image with 1-step method

3b. Using 1-step method from the floor image

In this step, I find 5 pairs of “perpendicular” lines in the camera plane, with a total of 20 points to derive the 10 lines. The selection of the 20 points in shown in Fig. 20(a) and the HC of the 20 points in summarized in Table 14. The resulting image in shown in Fig. 20(b).

Observation:

As shown in Fig.20(b), the image is nearly perfectly recovered although similarity distortion still exists in the resulting image because by removing the projective and affine distortion, we can only recover “parallel” and “perpendicular” lines in the image plane to be parallel and perpendicular in the world plane.

Table 14: Point, line and homography representations in this step

Perpendicular lines	R_1	R_2	S_1	S_2
1st pair	$(1215,1227,1)^T$	$(1305,1518,1)^T$	$(1215,1227,1)^T$	$(1341,1071,1)^T$
2nd pair	$(1782,567,1)^T$	$(1602,765,1)^T$	$(1782,567,1)^T$	$(1914,909,1)^T$
3rd pair	$(2229,1665,1)^T$	$(2115,1386,1)^T$	$(2229,1665,1)^T$	$(1965,1842,1)^T$
4th pair	$(1500,2166,1)^T$	$(1713,2019,1)^T$	$(1500,2166,1)^T$	$(1410,1875,1)^T$
5th pair	$(495,1269,1)^T$	$(576,1638,1)^T$	$(495,1269,1)^T$	$(624,1047,1)^T$
Conic and homogrphy	$C'_\infty = \begin{bmatrix} 5.28\times10^5 & 1.20\times10^5 & 564 \\ 1.20\times10^5 & 1.42\times10^5 & 199 \\ 564 & 199 & 1 \end{bmatrix}$		$H_D^{-1} = \begin{bmatrix} 1.28 & 0.366 & 1.42\times10^{-3} \\ -0.835 & 2.93 & 1.04\times10^{-3} \\ -9.28\times10^{-4} & -6.20\times10^{-4} & 1 \end{bmatrix}$	

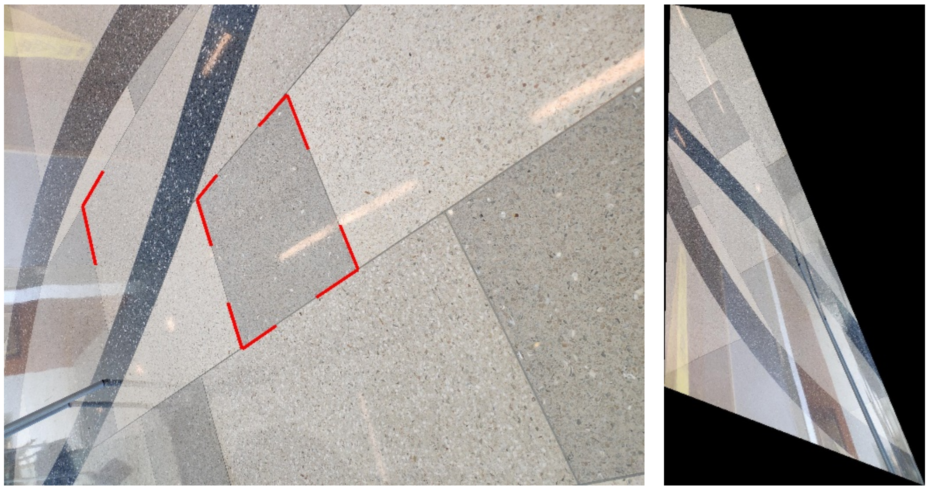

(a) Input image (b) Resulting image

Figure 20: Removing the projective and affine distortions of the floor image with 1-step method

5.1 Helper function to calculate homograph and image mapping

```
# This is the utility functions for ECE 661 hw3
import numpy as np
import copy
def Homography(A,B):
    # Find the homography from A to B, for example
    #A = np.transpose(np.array([[1141,817,1], [1257,761,1], [1126,994,1],
[1243,944,1]]))
    #B = np.transpose(np.array([[0,0,1], [60,0,1], [0,80,1], [60,80,1]]))
    Projection = np.zeros((8,1))   # 8 x 1
    Coefficient = np.zeros((8, 8))  # 8 x 8
    for index in range(4):
        Projection[index*2, 0] = B[0,index]    # xB
        Projection[index*2+1, 0] = B[1,index]  # yB
        Coefficient[index*2, 0] = A[0, index]  # xA
        Coefficient[index*2, 1] = A[1, index]  # yA
        Coefficient[index*2, 2] = 1
        Coefficient[index*2, 6] = - A[0, index] * B[0, index]  # -xA*xB
        Coefficient[index*2, 7] = - A[1, index] * B[0, index]  # -yA*xB
        Coefficient[index*2+1, 3] = A[0, index]  # xA
        Coefficient[index*2+1, 4] = A[1, index]  # yA
        Coefficient[index*2+1, 5] = 1
        Coefficient[index*2+1, 6] = - A[0, index] * B[1, index]  # -xA*yB
        Coefficient[index*2+1, 7] = - A[1, index] * B[1, index]  # -yA*yB
    h = np.matmul(np.linalg.inv(Coefficient), Projection)
    H_AB = 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_AB

def Mapped(imgA, H):
    # Boundary mapping
    HC_Boundary_A = np.array([[0,imgA.shape[1]-1,0,imgA.shape[1]-1],
                              [0,0,imgA.shape[0]-1,imgA.shape[0]-1],
                              [1,1,1,1]]).astype(int)
    HC_Boundary_MappedA = np.matmul(H, HC_Boundary_A)
    HC_Boundary_MappedA = (HC_Boundary_MappedA/HC_Boundary_MappedA[2,:]).a
stype(int)

    x_min = np.min(HC_Boundary_MappedA[0,:])
    x_max = np.max(HC_Boundary_MappedA[0,:])
    y_min = np.min(HC_Boundary_MappedA[1,:])
    y_max = np.max(HC_Boundary_MappedA[1,:])
    x_lim = x_max-x_min+1
    y_lim = y_max-y_min+1

    H_inverse = np.linalg.inv(H)
    # Construct a 3-D coordinate matrix "HC_mappedA" of mapped A (real HC)
    # [ 0, 0, 0, ........., 1, 1, 1, ........., (x_lim-1) ..] + x_min
    # [ 0, 1, 2, ..., (y_lim-1), 0, 1, 2, ..., (y_lim-1), ... ] + y_min
    # [ 1, 1, 1, 1, 1, 1, 1, ...................,1  ]
    # Meaning
    # [ X coordinate = Width direction, e.g. x_lim]
    # [ Y coordinate = Height direction, e.g. y_lim]
    # [ Ones ]
    HC_mappedA = np.ones((3, x_lim*y_lim)).astype(int)
    for indexH in range(x_lim):
        HC_mappedA[0, indexH*y_lim:indexH*y_lim+y_lim] = \
            np.repeat([indexH], y_lim) + x_min
        HC_mappedA[1, indexH*y_lim:indexH*y_lim+y_lim] = \
            np.arange(y_lim) + y_min
    HC_A = np.matmul(H_inverse, HC_mappedA)
    HC_A = (np.round(HC_A/HC_A[2,:])).astype(int)
    #print(np.max(HC_A[1,:]))

    A = np.array([[0,0], [imgA.shape[0],0],
                  [0,imgA.shape[1]], [imgA.shape[0],imgA.shape[1]]])
    # Check what mapped cooredinates is inside A
    EditVector = np.logical_and(HC_A[0,:]>=HC_Boundary_A[0,0], HC_A[0,:]<=
HC_Boundary_A[0,1])
    EditVector = np.logical_and(HC_A[1,:]>=HC_Boundary_A[1,0], EditVector)
    EditVector = np.logical_and(HC_A[1,:]<=HC_Boundary_A[1,2], EditVector)

    # Refill the image
    # Map all pixels inversely from mappedA plane to imgA plane, replace t
he pixels that mapped to A
    mappedA = np.zeros((y_lim,x_lim,3)).astype(int)
    for index in np.arange(x_lim*y_lim)[EditVector]:
        mappedA[HC_mappedA[1,index]-y_min, HC_mappedA[0,index]-x_min, :] =
 \
            imgA[HC_A[1,index], HC_A[0,index], :]
    return (x_min,y_min,mappedA)
```

5.2 P2P Correspondences Method

```
# This is the main python script for ECE 661 hw3
# To run: python3 main.py
import numpy as np
from PIL import Image
from homography_util import *

#Method 1
A_camera = np.transpose(np.array([[1141,815,1], [1258,761,1], [1126,993,
1], [1245,945,1]]))
A_world = np.transpose(np.array([[0,0,1], [60,0,1], [0,80,1], [60,80,1]]))
B_camera = np.transpose(np.array([[232,57,1], [336,73,1], [232,286,1], [33
5,278,1]]))
B_world = np.transpose(np.array([[0,0,1], [40,0,1], [0,80,1], [40,80,1]]))
C_camera = np.transpose(np.array([[1407,675,1], [3330,420,1], [1434,2145,
1], [3189,2790,1]]))
C_world = np.transpose(np.array([[0,0,1], [1000,0,1], [0,810,1], [1000,81
0,1]]))
D_camera = np.transpose(np.array([[1216,1224,1], [1774,558,1], [1498,2158,
1], [2226,1666,1]]))
D_world = np.transpose(np.array([[0,0,1], [460,0,1], [0,460,1], [460,460,
1]]))
```

```
img1 = np.array(Image.open("1.jpg"))
img2 = np.array(Image.open("2.jpg"))
img3 = np.array(Image.open("3.jpg"))
img4 = np.array(Image.open("4.jpg"))
```

```
# Homography from camera to world
H1_p2p = Homography(A_camera, A_world)
H2_p2p = Homography(B_camera, B_world)
H3_p2p = Homography(C_camera, C_world)
H4_p2p = Homography(D_camera, D_world)
print("(H1_p2p, H2_p2p) = ", H1_p2p, H2_p2p)
print("(H3_p2p, H4_p2p) = ", H3_p2p, H4_p2p)

(x_min,y_min,mapped) = Mapped(img1, H1_p2p)
mapped = Image.fromarray(np.uint8(mapped), 'RGB')
mapped.save("img1-m1.jpg")
print("(x_min,y_min) = (%d,%d)" %(x_min,y_min))

(x_min,y_min,mapped) = Mapped(img2, H2_p2p)
mapped = Image.fromarray(np.uint8(mapped), 'RGB')
mapped.save("img2-m1.jpg")
print("(x_min,y_min) = (%d,%d)" %(x_min,y_min))

(x_min,y_min,mapped) = Mapped(img3, H3_p2p)
mapped = Image.fromarray(np.uint8(mapped), 'RGB')
mapped.save("img3-m1.jpg")
print("(x_min,y_min) = (%d,%d)" %(x_min,y_min))

(x_min,y_min,mapped) = Mapped(img4, H4_p2p)
mapped = Image.fromarray(np.uint8(mapped), 'RGB')
mapped.save("img4-m1.jpg")
print("(x_min,y_min) = (%d,%d)" %(x_min,y_min))
```

5.3 2-step method

```
import numpy as np
from PIL import Image, ImageDraw
from homography_util import *

# Method 2 - projective distortion
img1 = np.array(Image.open("1.jpg"))
L1 = np.array([1044,869,1])
L2 = np.array([1025,1030,1])
M1 = np.array([2024,420,1])
M2 = np.array([2038,619,1])
P1 = np.array([2024,420,1])
P2 = np.array([1044,869,1])
Q1 = np.array([2038,619,1])
Q2 = np.array([1025,1030,1])
l = np.cross(L1, L2)
m = np.cross(M1, M2)
p = np.cross(P1, P2)
```

```python
q = np.cross(Q1, Q2)
VP1 = np.cross(l, m) ; VP1 = VP1 / VP1[2]
VP2 = np.cross(p, q) ; VP2 = VP2 / VP2[2]
VL = np.cross(VP1, VP2) ; VL = VL / VL[2]
H_p = np.array([[1,0,0],[0,1,0],[VL[0],VL[1],VL[2]]])
print("(VP1,VP2,VL) = ", VP1,VP2,VL)
print("H_p = ", H_p)
(x_min,y_min,mapped_img1) = Mapped(img1, H_p)
mapped = Image.fromarray(np.uint8(mapped_img1), 'RGB')
mapped.save("img1-m21.jpg")

# Method 2 - affine distortion (mapped_img1 as a start)
draw_img12 = Image.open("img1-m21.jpg")
draw = ImageDraw.Draw(draw_img12)
R1 = np.array([108,546,1])
R2 = np.array([97,579,1])
S1 = np.array([108,546,1])
S2 = np.array([124,527,1])
T1 = np.array([425,318,1])
T2 = np.array([389,358,1])
U1 = np.array([425,318,1])
U2 = np.array([442,275,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
draw.line((T1[0], T1[1], T2[0], T2[1]), fill=(255,0,0), width = 5)
draw.line((U1[0], U1[1], U2[0], U2[1]), fill=(255,0,0), width = 5)
draw_img12.save("img1-m22-annotaton.jpg")
r = np.cross(R1, R2);
s = np.cross(S1, S2);
t = np.cross(T1, T2);
u = np.cross(U1, U2);
print("(r,s,t,u)=", r, s, t, u)
lhs = np.array([[r[0]*s[0], r[0]*s[1]+r[1]*s[0]], [t[0]*u[0], t[0]*u[1]+t
[1]*u[0]]])
rhs = np.array([[-r[1]*s[1]],[-t[1]*u[1]]])
v_tmp = np.matmul(np.linalg.inv(lhs), rhs)
v_tmp2 = np.array([[v_tmp[0][0],v_tmp[1][0]],[v_tmp[1][0],1]])
sigma, U = np.linalg.eig(v_tmp2)
A = np.matmul(U, np.diag(np.sqrt(sigma)))*(-1)
print("Sanity check: a_11 = ", A[0][0],
      ", a_11*a_22-a_12*a_21 = ", A[0][0]*A[1][1]-A[0][1]*A[1][0])
H_a = np.zeros((3,3))
H_a[2][2] = 1
H_a[0:2,0:2] = np.linalg.inv(A)
print("H_a = ", H_a)
(x_min,y_min,mapped2_img1) = Mapped(mapped_img1, H_a)
mapped = Image.fromarray(np.uint8(mapped2_img1), 'RGB')
mapped.save("img1-m22.jpg")
```

```python
img2 = np.array(Image.open("2.jpg"))
L1 = np.array([232,57,1])
L2 = np.array([231,281,1])
M1 = np.array([338,73,1])
```

```
M2 = np.array([336,278,1])
P1 = np.array([232,57,1])
P2 = np.array([338,73,1])
Q1 = np.array([231,281,1])
Q2 = np.array([336,278,1])
l = np.cross(L1, L2)
m = np.cross(M1, M2)
p = np.cross(P1, P2)
q = np.cross(Q1, Q2)
VP1 = np.cross(l, m) ; VP1 = VP1 / VP1[2]
VP2 = np.cross(p, q) ; VP2 = VP2 / VP2[2]
VL = np.cross(VP1, VP2) ; VL = VL / VL[2]
H_p = np.array([[1,0,0],[0,1,0],[VL[0],VL[1],VL[2]]])
print("(VP1,VP2,VL) = ", VP1,VP2,VL)
print("H_p = ", H_p)
(x_min,y_min,mapped_img2) = Mapped(img2, H_p)
mapped = Image.fromarray(np.uint8(mapped_img2), 'RGB')
mapped.save("img2-m21.jpg")

# Method 2 - affine distortion (mapped_img1 as a start)
draw_img22 = Image.open("img2-m21.jpg")
draw = ImageDraw.Draw(draw_img22)
R1 = np.array([275,68,1])
R2 = np.array([277,343,1])
S1 = np.array([275,68,1])
S2 = np.array([439,91,1])
T1 = np.array([643,330,1])
T2 = np.array([548,319,1])
U1 = np.array([643,330,1])
U2 = np.array([639,225,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
draw.line((T1[0], T1[1], T2[0], T2[1]), fill=(255,0,0), width = 5)
draw.line((U1[0], U1[1], U2[0], U2[1]), fill=(255,0,0), width = 5)
draw_img22.save("img2-m22-annotaton.jpg")
r = np.cross(R1, R2);
s = np.cross(S1, S2);
t = np.cross(T1, T2);
u = np.cross(U1, U2);
print("(r,s,t,u)=", r, s, t, u)
lhs = np.array([[r[0]*s[0], r[0]*s[1]+r[1]*s[0]], [t[0]*u[0], t[0]*u[1]+t
[1]*u[0]]])
rhs = np.array([[-r[1]*s[1]],[-t[1]*u[1]]])
v_tmp = np.matmul(np.linalg.inv(lhs), rhs)
v_tmp2 = np.array([[v_tmp[0][0],v_tmp[1][0]],[v_tmp[1][0],1]])
sigma, U = np.linalg.eig(v_tmp2)
A = np.matmul(U, np.diag(np.sqrt(sigma)))
print("Sanity check: a_11 = ", A[0][0],
      ", a_11*a_22-a_12*a_21 = ", A[0][0]*A[1][1]-A[0][1]*A[1][0])
H_a = np.zeros((3,3))
H_a[2][2] = 1
H_a[0:2,0:2] = np.linalg.inv(A)
print("H_a = ", H_a)
(x_min,y_min,mapped2_img2) = Mapped(mapped_img2, H_a)
```

```python
mapped = Image.fromarray(np.uint8(mapped2_img2), 'RGB')
mapped.save("img2-m22.jpg")
```

```python
img3 = np.array(Image.open("3.jpg"))
draw_img31 = Image.open("3.jpg")
draw = ImageDraw.Draw(draw_img31)
L1 = np.array([1404,855,1])
L2 = np.array([1425,1557,1])
M1 = np.array([3309,855,1])
M2 = np.array([3243,1947,1])
P1 = np.array([2991,456,1])
P2 = np.array([1917,615,1])
Q1 = np.array([2841,2664,1])
Q2 = np.array([1833,2289,1])
draw.line((L1[0], L1[1], L2[0], L2[1]), fill=(255,0,0), width = 20)
draw.line((M1[0], M1[1], M2[0], M2[1]), fill=(255,0,0), width = 20)
draw.line((P1[0], P1[1], P2[0], P2[1]), fill=(0,0,255), width = 20)
draw.line((Q1[0], Q1[1], Q2[0], Q2[1]), fill=(0,0,255), width = 20)
draw_img31.save("img3-m21-annotaton.jpg")
l = np.cross(L1, L2)
m = np.cross(M1, M2)
p = np.cross(P1, P2)
q = np.cross(Q1, Q2)
VP1 = np.cross(l, m) ; VP1 = VP1 / VP1[2]
VP2 = np.cross(p, q) ; VP2 = VP2 / VP2[2]
VL = np.cross(VP1, VP2) ; VL = VL / VL[2]
H_p = np.array([[1,0,0],[0,1,0],[VL[0],VL[1],VL[2]]])
print("(VP1,VP2,VL) = ", VP1,VP2,VL)
print("H_p = ", H_p)
(x_min,y_min,mapped_img3) = Mapped(img3, H_p)
mapped = Image.fromarray(np.uint8(mapped_img3), 'RGB')
mapped.save("img3-m21.jpg")

# Method 2 - affine distortion (mapped_img1 as a start)
draw_img32 = Image.open("img3-m21.jpg")
draw = ImageDraw.Draw(draw_img32)
R1 = np.array([567,516,1])
R2 = np.array([339,684,1])
S1 = np.array([567,516,1])
S2 = np.array([600,968,1])
T1 = np.array([305,1701,1])
T2 = np.array([284,1425,1])
U1 = np.array([305,1701,1])
U2 = np.array([464,1575,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
draw.line((T1[0], T1[1], T2[0], T2[1]), fill=(255,0,0), width = 20)
draw.line((U1[0], U1[1], U2[0], U2[1]), fill=(255,0,0), width = 20)
draw_img32.save("img3-m22-annotaton.jpg")
r = np.cross(R1, R2);
s = np.cross(S1, S2);
t = np.cross(T1, T2);
u = np.cross(U1, U2);
```

```python
print("(r,s,t,u)=", r, s, t, u)
lhs = np.array([[r[0]*s[0], r[0]*s[1]+r[1]*s[0]], [t[0]*u[0], t[0]*u[1]+t
[1]*u[0]]])
rhs = np.array([[-r[1]*s[1]],[-t[1]*u[1]]])
v_tmp = np.matmul(np.linalg.inv(lhs), rhs)
v_tmp2 = np.array([[v_tmp[0][0],v_tmp[1][0]],[v_tmp[1][0],1]])
sigma, U = np.linalg.eig(v_tmp2)
A = np.matmul(U, np.diag(np.sqrt(sigma)))*(-1)
print("Sanity check: a_11 = ", A[0][0],
      ", a_11*a_22-a_12*a_21 = ", A[0][0]*A[1][1]-A[0][1]*A[1][0])
H_a = np.zeros((3,3))
H_a[2][2] = 1
H_a[0:2,0:2] = np.linalg.inv(A)
print("H_a = ", H_a)
(x_min,y_min,mapped2_img3) = Mapped(mapped_img3, H_a)
mapped = Image.fromarray(np.uint8(mapped2_img3), 'RGB')
mapped.save("img3-m22.jpg")
```

```python
img4 = np.array(Image.open("4.jpg"))
draw_img41 = Image.open("4.jpg")
draw = ImageDraw.Draw(draw_img41)
L1 = np.array([1254,1353,1])
L2 = np.array([1428,1929,1])
M1 = np.array([1884,825,1])
M2 = np.array([2133,1428,1])
P1 = np.array([1671,681,1])
P2 = np.array([1341,1071,1])
Q1 = np.array([2112,1749,1])
Q2 = np.array([1653,2058,1])
draw.line((L1[0], L1[1], L2[0], L2[1]), fill=(255,0,0), width = 20)
draw.line((M1[0], M1[1], M2[0], M2[1]), fill=(255,0,0), width = 20)
draw.line((P1[0], P1[1], P2[0], P2[1]), fill=(0,0,255), width = 20)
draw.line((Q1[0], Q1[1], Q2[0], Q2[1]), fill=(0,0,255), width = 20)
draw_img41.save("img4-m21-annotaton.jpg")
l = np.cross(L1, L2)
m = np.cross(M1, M2)
p = np.cross(P1, P2)
q = np.cross(Q1, Q2)
VP1 = np.cross(l, m) ; VP1 = VP1 / VP1[2]
VP2 = np.cross(p, q) ; VP2 = VP2 / VP2[2]
VL = np.cross(VP1, VP2) ; VL = VL / VL[2]
H_p = np.array([[1,0,0],[0,1,0],[VL[0],VL[1],VL[2]]])
print("(VP1,VP2,VL) = ", VP1,VP2,VL)
print("H_p = ", H_p)
(x_min,y_min,mapped_img4) = Mapped(img4, H_p)
mapped = Image.fromarray(np.uint8(mapped_img4), 'RGB')
mapped.save("img4-m21.jpg")

# Method 2 - affine distortion (mapped_img1 as a start)
draw_img42 = Image.open("img4-m21.jpg")
draw = ImageDraw.Draw(draw_img42)
R1 = np.array([651,207,1])
R2 = np.array([604,384,1])
```

```
S1 = np.array([651,207,1])
S2 = np.array([680,379,1])
T1 = np.array([558,563,1])
T2 = np.array([575,663,1])
U1 = np.array([558,563,1])
U2 = np.array([582,470,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
draw.line((T1[0], T1[1], T2[0], T2[1]), fill=(255,0,0), width = 20)
draw.line((U1[0], U1[1], U2[0], U2[1]), fill=(255,0,0), width = 20)
draw_img42.save("img4-m22-annotaton.jpg")
r = np.cross(R1, R2);
s = np.cross(S1, S2);
t = np.cross(T1, T2);
u = np.cross(U1, U2);
print("(r,s,t,u)=", r, s, t, u)
lhs = np.array([[r[0]*s[0], r[0]*s[1]+r[1]*s[0]], [t[0]*u[0], t[0]*u[1]+t
[1]*u[0]]])
rhs = np.array([[-r[1]*s[1]],[-t[1]*u[1]]])
v_tmp = np.matmul(np.linalg.inv(lhs), rhs)
v_tmp2 = np.array([[v_tmp[0][0],v_tmp[1][0]],[v_tmp[1][0],1]])
sigma, U = np.linalg.eig(v_tmp2)
A = np.matmul(U, np.diag(np.sqrt(sigma)))*(-1)
print("Sanity check: a_11 = ", A[0][0],
      ", a_11*a_22-a_12*a_21 = ", A[0][0]*A[1][1]-A[0][1]*A[1][0])
H_a = np.zeros((3,3))
H_a[2][2] = 1
H_a[0:2,0:2] = np.linalg.inv(A)
print("H_a = ", H_a)
(x_min,y_min,mapped2_img4) = Mapped(mapped_img4, H_a)
mapped = Image.fromarray(np.uint8(mapped2_img4), 'RGB')
mapped.save("img4-m22.jpg")
```

```
# For bonus, "bonus_image.jpg"
L1 = np.array([90,678,1])    # Black pairs
L2 = np.array([1842,309,1])
M1 = np.array([459,1443,1])
M2 = np.array([2331,930,1])
P1 = np.array([726,2472,1])  # Blue pairs
P2 = np.array([1491,735,1])
Q1 = np.array([2619,2679,1])
Q2 = np.array([2697,906,1])
l = np.cross(L1, L2)
m = np.cross(M1, M2)
p = np.cross(P1, P2)
q = np.cross(Q1, Q2)
VP1 = np.cross(l, m) ; VP1 = VP1 / VP1[2]
VP2 = np.cross(p, q) ; VP2 = VP2 / VP2[2]
VL = np.cross(VP1, VP2) ; VL = VL / VL[2]
print("(l,m,p,q) = ", l,m,p,q)
print("(VP1,VP2,VL) = ", VP1,VP2,VL)

L1 = np.array([765,1842,1])  # Red pairs
```

```
L2 = np.array([273,402,1])
M1 = np.array([2058,1305,1])
M2 = np.array([1305,294,1])
P1 = np.array([1131,2397,1])   # Green pairs
P2 = np.array([129,1398,1])
Q1 = np.array([1929,1287,1])
Q2 = np.array([543,378,1])
l = np.cross(L1, L2)
m = np.cross(M1, M2)
p = np.cross(P1, P2)
q = np.cross(Q1, Q2)
VP1 = np.cross(l, m) ; VP1 = VP1 / VP1[2]
VP2 = np.cross(p, q) ; VP2 = VP2 / VP2[2]
VL = np.cross(VP1, VP2) ; VL = VL / VL[2]
print("(l,m,p,q) = ", l,m,p,q)
print("(VP1,VP2,VL) = ", VP1,VP2,VL)
```

5.4 1-step method

```
import numpy as np
from PIL import Image, ImageDraw
from homography_util import *

# Method 3 - 1-step distortion
img1 = np.array(Image.open("1.jpg"))
draw_img1 = Image.open("1.jpg")
draw = ImageDraw.Draw(draw_img1)
lhs = np.zeros((5,5))
rhs = np.zeros((5,1))

# 1st pair of parallel lines
R1 = np.array([1142,817,1])
R2 = np.array([1127,993,1])
S1 = np.array([1142,817,1])
S2 = np.array([1258,763,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 10)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 10)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[0,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[0][0] = - r[2]*s[2]

# 2nd pair of parallel lines
S1 = np.array([1524,637,1])
S2 = np.array([1387,702,1])
R1 = np.array([1524,637,1])
R2 = np.array([1517,835,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 10)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 10)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
```

```
lhs[1,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[1][0] = - r[2]*s[2]

# 3rd pair of parallel lines
R1 = np.array([1507,1122,1])
R2 = np.array([1511,1001,1])
S1 = np.array([1507,1122,1])
S2 = np.array([1359,1167,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 10)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 10)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[2,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[2][0] = - r[2]*s[2]

# 4th pair of parallel lines
S1 = np.array([1103,1245,1])
S2 = np.array([1112,1140,1])
R1 = np.array([1103,1245,1])
R2 = np.array([1225,1209,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 10)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 10)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[3,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[3][0] = - r[2]*s[2]

# 5th pair of parallel lines
R1 = np.array([1244,946,1])
R2 = np.array([1375,894,1])
S1 = np.array([1244,946,1])
S2 = np.array([1232,1098,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 10)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 10)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[4,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[4][0] = - r[2]*s[2]
draw_img1.save("img1-m3-annotaton.jpg")

c_tmp = np.matmul(np.linalg.inv(lhs), rhs)
c_infinity = np.array([[c_tmp[0][0], c_tmp[1][0], c_tmp[3][0]],
                       [c_tmp[1][0], c_tmp[2][0], c_tmp[4][0]],
                       [c_tmp[3][0], c_tmp[4][0], 1]])

sigma, U = np.linalg.eig(c_infinity);
print("c_infinity = ", c_infinity)
print("sigma = ", sigma)
scale = 1e-3
d = np.diag([1/np.sqrt(sigma[0]), 1/np.sqrt(sigma[1]), scale]);
```

```python
H_D = np.matmul(d, np.transpose(U));
H_D[0,:] = H_D[0,:]*(-1)
H_D = H_D/H_D[2,2]
print("inverse(H_D) = ", H_D)

(x_min,y_min,mapped_img1) = Mapped(img1, H_D)
mapped = Image.fromarray(np.uint8(mapped_img1), 'RGB')
mapped.save("img1-m3.jpg")
```

```python
# Method 3 - Image 2 - 1-step distortion
img1 = np.array(Image.open("2.jpg"))
draw_img1 = Image.open("2.jpg")
draw = ImageDraw.Draw(draw_img1)
lhs = np.zeros((5,5))
rhs = np.zeros((5,1))

# 1st pair of parallel lines
R1 = np.array([237,57,1])
R2 = np.array([284,65,1])
S1 = np.array([237,57,1])
S2 = np.array([237,146,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[0,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[0][0] = - r[2]*s[2]

# 2nd pair of parallel lines
S1 = np.array([236,283,1])
S2 = np.array([238,227,1])
R1 = np.array([236,283,1])
R2 = np.array([275,281,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[1,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[1][0] = - r[2]*s[2]

# 3rd pair of parallel lines
R1 = np.array([335,277,1])
R2 = np.array([303,278,1])
S1 = np.array([335,277,1])
S2 = np.array([336,230,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[2,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
```

```
rhs[2][0] = - r[2]*s[2]

# 4th pair of parallel lines
S1 = np.array([337,73,1])
S2 = np.array([308,69,1])
R1 = np.array([337,73,1])
R2 = np.array([337,128,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[3,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[3][0] = - r[2]*s[2]

# 5th pair of parallel lines
R1 = np.array([399,154,1])
R2 = np.array([398,231,1])
S1 = np.array([399,154,1])
S2 = np.array([443,157,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 5)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 5)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[4,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[4][0] = - r[2]*s[2]
draw_img1.save("img2-m3-annotaton.jpg")

c_tmp = np.matmul(np.linalg.inv(lhs), rhs)
c_infinity = np.array([[c_tmp[0][0], c_tmp[1][0], c_tmp[3][0]],
                       [c_tmp[1][0], c_tmp[2][0], c_tmp[4][0]],
                       [c_tmp[3][0], c_tmp[4][0], 1]])

sigma, U = np.linalg.eig(c_infinity);
print("c_infinity = ", c_infinity)
print("sigma = ", sigma)
scale = 1e-3
d = np.diag([1/np.sqrt(sigma[0]), 1/np.sqrt(sigma[1]), scale]);
H_D = np.matmul(d, np.transpose(U));
H_D[1,:] = H_D[1,:]*(-1)
H_D = H_D/H_D[2,2]
print("inverse(H_D) = ", H_D)

(x_min,y_min,mapped_img1) = Mapped(img1, H_D)
mapped = Image.fromarray(np.uint8(mapped_img1), 'RGB')
mapped.save("img2-m3.jpg")
```

```
# Method 3 - Image 3 - 1-step distortion
img1 = np.array(Image.open("3.jpg"))
draw_img1 = Image.open("3.jpg")
draw = ImageDraw.Draw(draw_img1)
lhs = np.zeros((5,5))
```

```
rhs = np.zeros((5,1))

# 1st pair of parallel lines
R1 = np.array([1407,669,1])
R2 = np.array([1425,1245,1])
S1 = np.array([1407,669,1])
S2 = np.array([2541,525,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[0,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[0][0] = - r[2]*s[2]

# 2nd pair of parallel lines
R1 = np.array([3327,423,1])
R2 = np.array([2478,528,1])
S1 = np.array([3327,423,1])
S2 = np.array([3270,1539,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[1,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[1][0] = - r[2]*s[2]

# 3rd pair of parallel lines
R1 = np.array([3186,2790,1])
R2 = np.array([3240,2067,1])
S1 = np.array([3186,2790,1])
S2 = np.array([2502,2538,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[2,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[2][0] = - r[2]*s[2]

# 4th pair of parallel lines
R1 = np.array([1431,2145,1])
R2 = np.array([1842,2292,1])
S1 = np.array([1431,2145,1])
S2 = np.array([1434,1845,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[3,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[3][0] = - r[2]*s[2]
```

```
# 5th pair of parallel lines
R1 = np.array([825,747,1])
R2 = np.array([840,1092,1])
S1 = np.array([825,747,1])
S2 = np.array([1164,705,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[4,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[4][0] = - r[2]*s[2]
draw_img1.save("img3-m3-annotaton.jpg")

c_tmp = np.matmul(np.linalg.inv(lhs), rhs)
c_infinity = np.array([[c_tmp[0][0], c_tmp[1][0], c_tmp[3][0]],
                       [c_tmp[1][0], c_tmp[2][0], c_tmp[4][0]],
                       [c_tmp[3][0], c_tmp[4][0], 1]])

sigma, U = np.linalg.eig(c_infinity);
print("c_infinity = ", c_infinity)
print("sigma = ", sigma)
scale = 1e-3
d = np.diag([1/np.sqrt(sigma[0]), 1/np.sqrt(sigma[1]), scale]);
H_D = np.matmul(d, np.transpose(U));
H_D[0,:] = H_D[0,:]
H_D[1,:] = H_D[1,:]
H_D = H_D/H_D[2,2]
print("inverse(H_D) = ", H_D)

(x_min,y_min,mapped_img1) = Mapped(img1, H_D)
mapped = Image.fromarray(np.uint8(mapped_img1), 'RGB')
mapped.save("img3-m3.jpg")
```

```
# Method 3 - Image 4 - 1-step distortion
img1 = np.array(Image.open("4.jpg"))
draw_img1 = Image.open("4.jpg")
draw = ImageDraw.Draw(draw_img1)
lhs = np.zeros((5,5))
rhs = np.zeros((5,1))

# 1st pair of parallel lines
R1 = np.array([1215,1227,1])
R2 = np.array([1305,1518,1])
S1 = np.array([1215,1227,1])
S2 = np.array([1341,1071,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[0,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[0][0] = - r[2]*s[2]
```

```
# 2nd pair of parallel lines
R1 = np.array([1782,567,1])
R2 = np.array([1602,765,1])
S1 = np.array([1782,567,1])
S2 = np.array([1914,909,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[1,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[1][0] = - r[2]*s[2]

# 3rd pair of parallel lines
R1 = np.array([2229,1665,1])
R2 = np.array([2115,1386,1])
S1 = np.array([2229,1665,1])
S2 = np.array([1965,1842,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[2,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[2][0] = - r[2]*s[2]

# 4th pair of parallel lines
R1 = np.array([1500,2166,1])
R2 = np.array([1713,2019,1])
S1 = np.array([1500,2166,1])
S2 = np.array([1410,1875,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[3,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[3][0] = - r[2]*s[2]

# 5th pair of parallel lines
R1 = np.array([495,1269,1])
R2 = np.array([576,1638,1])
S1 = np.array([495,1269,1])
S2 = np.array([624,1047,1])
draw.line((R1[0], R1[1], R2[0], R2[1]), fill=(255,0,0), width = 20)
draw.line((S1[0], S1[1], S2[0], S2[1]), fill=(255,0,0), width = 20)
r = np.cross(R1, R2) ;
s = np.cross(S1, S2) ;
lhs[4,:] = np.array([r[0]*s[0], r[1]*s[0]+r[0]*s[1], r[1]*s[1], r[2]*s[0]+
r[0]*s[2], r[2]*s[1]+r[1]*s[2]])
rhs[4][0] = - r[2]*s[2]
draw_img1.save("img4-m3-annotaton.jpg")
```

```python
c_tmp = np.matmul(np.linalg.inv(lhs), rhs)
c_infinity = np.array([[c_tmp[0][0], c_tmp[1][0], c_tmp[3][0]],
                       [c_tmp[1][0], c_tmp[2][0], c_tmp[4][0]],
                       [c_tmp[3][0], c_tmp[4][0], 1]])

sigma, U = np.linalg.eig(c_infinity);
print("c_infinity = ", c_infinity)
print("sigma = ", sigma)
scale = 1e-3
d = np.diag([1/np.sqrt(sigma[0]), 1/np.sqrt(sigma[1]), scale]);
H_D = np.matmul(d, np.transpose(U));
H_D[0,:] = H_D[0,:]*(-1)
H_D[1,:] = H_D[1,:]
H_D = H_D/H_D[2,2]
print("inverse(H_D) = ", H_D)

(x_min,y_min,mapped_img1) = Mapped(img1, H_D)
mapped = Image.fromarray(np.uint8(mapped_img1), 'RGB')
mapped.save("img4-m3.jpg")
```