ECE 661: Homework 3

Wei Xu
Email: xu1639@purdue.edu
Due date: 1:30 pm, Sept. 20, 2022
(Fall 2022)

SOLVING LOGIC AND STEPS

Point-to-point correspondences

The homogeneous coordinates (HC) representation of a physical point $\mathbf{x} = (x, y)^\top \in \mathbb{R}^2$ can be written as $(u, v, w)^\top \in \mathbb{R}^3$, where $x = \frac{u}{w}$ and $y = \frac{v}{w}$. The homography on homogeneous 3-vectors can be represented by a non-singular 3×3 matrix $\mathbf{H}$, as in

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{1}$$

where

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2}$$

Specifically, let $\mathbf{x}$ be the measurement and $\mathbf{x}'$ be the corresponding point in the undistorted image, which is different from the description in my last homework. Because $\mathbf{H}$ is homogeneous, we can set that $h_{33} = 1$, so that

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \tag{3}$$

Then

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{4}$$

which is

$$\begin{cases} uh_{11} + vh_{12} + wh_{13} = u' \\ uh_{21} + vh_{22} + wh_{23} = v' \\ uh_{31} + vh_{32} + w = w' \end{cases} \tag{5}$$

Then

$$\begin{cases} x' = \frac{u'}{w'} = \frac{uh_{11}+vh_{12}+wh_{13}}{uh_{31}+vh_{32}+w} \\ y' = \frac{v'}{w'} = \frac{uh_{21}+vh_{22}+wh_{23}}{uh_{31}+vh_{32}+w} \end{cases} \tag{6}$$

which is

$$\begin{cases} uh_{11} + vh_{12} + wh_{13} - x'uh_{31} - x'vh_{32} = x'w \\ uh_{21} + vh_{22} + wh_{23} - y'uh_{31} + y'vh_{32} = y'w \end{cases} \tag{7}$$

Divided by w, then

$$\begin{cases} xh_{11} + yh_{12} + h_{13} - x'xh_{31} - x'yh_{32} = x' \\ xh_{21} + yh_{22} + h_{23} - y'xh_{31} + y'yh_{32} = y' \end{cases} \tag{8}$$

Since we are required to use four points,

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2'x_2 & -x_2'y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y_2'x_2 & -y_2'y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3'x_3 & -x_3'y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y_3'x_3 & -y_3'y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4'x_4 & -x_4'y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y_4'x_4 & -y_4'y_4 \end{bmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{pmatrix} \tag{9}$$

There are 8 unknown variables in total, so if the left-most matrix is full rank, there will be a solution. If it is written as $\mathbf{Ah} = \mathbf{b}$, the solution will be $\mathbf{h} = \mathbf{A}^{-1}\mathbf{b}$. Then $\mathbf{H}$ can be obtained through $\mathbf{h}$.

After obtaining $\mathbf{H}$, we can apply it to each pixel from the original image to get the corresponding pixel on the undistorted image. To get better performance, $\mathbf{x} = \mathbf{H}^{-1}\mathbf{x}'$ can be used to map the pixels back to the original image, so that all pixels will be evaluated.

The algorithm is shown in Algorithm 1. The inputs are recorded image I, detected points P_{dt}, and desired points P_{ds}. The output is the desired image I_d.

Algorithm 1: Point-to-point correspondences(I, P_{dt}, P_{ds})

```
calculate H based on the method mentioned above (h = A^{-1} b);
calculate the image dimension of the desired image, and create an empty image, I_d;
For each pixel p' ∈ I_d
    p = H^{-1} p';
    If p ∈ I :
        I_d[p'] = I[p];
Return I_d;
```

Two-step approach

(a) Eliminate projective distortion with the vanishing line method

The distortion in an image that results in the formation of one or more vanishing points and vanishing lines in the plane of the image is specifically projective, meaning that it is over and above the distortion introduced by affine part of the overall transformation. If a homography is applied to an image that sends the vanishing line back to $\mathbf{l}_\infty$, the remaining distortion in the image will be purely affine.

The homography for removing the projective distortion can be estimated from the parameters of the vanishing line. If the vanishing line is $\mathbf{l} = (l_1, l_2, l_3)^\top$, the homography that takes the vanishing line back to $\mathbf{l}_\infty$ is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} \tag{10}$$

To prove this, recall that when points are transformed by $\mathbf{H}$, lines are transformed by $\mathbf{H}^{-\top}$, which is

$$\mathbf{H}^{-\top} = \begin{bmatrix} 1 & 0 & -\frac{l_1}{l_3} \\ 0 & 1 & -\frac{l_2}{l_3} \\ 0 & 0 & \frac{1}{l_3} \end{bmatrix} \tag{11}$$

Notice that

$$\mathbf{H}^{-\top}\mathbf{l} = \begin{bmatrix} 1 & 0 & -\frac{l_1}{l_3} \\ 0 & 1 & -\frac{l_2}{l_3} \\ 0 & 0 & \frac{1}{l_3} \end{bmatrix} \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} \tag{12}$$

$$= \begin{pmatrix} l_1 - \frac{l_1}{l_3} \times l_3 \\ l_2 - \frac{l_2}{l_3} \times l_3 \\ \frac{1}{l_3} \times l_3 \end{pmatrix} \tag{13}$$

$$= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{14}$$

$$= \mathbf{l}_\infty \tag{15}$$

So it is proved. This approach can help get rid of distortion that is specifically projective. After an image is rectified with respect to this distortion, the image will still contain affine distortion — the primary manifestation of which is unequal scaling along two orthogonal directions in the image. So by the vanishing line, $\mathbf{H}$ can be determined, which can help remove the projective distortion. Taking the cross-product of the 3-vectors for two different lines that are parallel in the undistorted scene can obtain the HC representation for the vanishing point (VP) for those two lines. Then taking the cross-product of two such VPs for two different pairs of parallel lines can obtain the vanishing line.

The algorithm is shown in Algorithm 3. The inputs are recorded image I, detected points P_{dt}. The output is the desired image I_{di} without projective distortion.

Algorithm 2: Vanishing line method to eliminate projective distortion(I, P_{dt})

calculate $\mathbf{H}$ based on the method mentioned above ($\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$);
calculate the image dimension of the desired image, and create an empty image, I_{di};
For each pixel $p' \in I_{di}$
  $p = \mathbf{H}^{-1}p'$;
  If $p \in I$ **:**
    $I_{di}[p'] = I[p]$;
Return I_{di};

(b) Eliminate affine distortion with the dual degenerate conic method

The formula for $\cos\theta$ can be written in the form of dual degenerate conic $\mathbf{C}^*_\infty$, which is

$$\cos\theta = \frac{\mathbf{l}^\top \mathbf{C}^*_\infty \mathbf{m}}{\sqrt{(\mathbf{l}^\top \mathbf{C}^*_\infty \mathbf{l})\,(\mathbf{m}^\top \mathbf{C}^*_\infty \mathbf{m})}} \tag{16}$$

where

$$\mathbf{C}_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{17}$$

Now express $\mathbf{l}$, $\mathbf{m}$, and $\mathbf{C}_\infty^*$ in the original planar scene in terms of the observed $\mathbf{l}'$, $\mathbf{m}'$, and $\mathbf{C}_\infty^{*\prime}$ in the recorded image. Subtituting $\mathbf{l} = \mathbf{H}^\top \mathbf{l}'$, $\mathbf{m} = \mathbf{H}^\top \mathbf{m}'$, and $\mathbf{C}_\infty^* = \mathbf{H}^{-1}\mathbf{C}_\infty^{*\prime}\mathbf{H}^{-\top}$ in the numerator and setting it to zero, we can write the constraint for estimating $\mathbf{H}$ as

$$\begin{aligned} \cos\theta|_{numerator} &= \left(\mathbf{l}'^\top \mathbf{H}\right)\left(\mathbf{H}^{-1}\mathbf{C}_\infty^{*\prime}\mathbf{H}^{-\top}\right)\left(\mathbf{H}^\top \mathbf{m}'\right) && (18) \\ &= \mathbf{l}'^\top \mathbf{H}\mathbf{C}_\infty^* \mathbf{H}^\top \mathbf{m}' && (19) \\ &= 0 && (20) \end{aligned}$$

So

$$\begin{pmatrix} l_1' & l_2' & l_3' \end{pmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A}^\top & \mathbf{0} \\ \mathbf{t}^\top & 1 \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} = 0 \tag{21}$$

which collapses into

$$\begin{pmatrix} l_1' & l_2' & l_3' \end{pmatrix} \begin{bmatrix} \mathbf{A}\mathbf{A}^\top & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} = 0 \tag{22}$$

Let $\mathbf{S} = \mathbf{A}\mathbf{A}^\top$, then

$$\begin{pmatrix} l_1' & l_2' \end{pmatrix} \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \end{pmatrix} = 0 \tag{23}$$

$\mathbf{A}\mathbf{A}^\top$ is symmetric, so $s_{12} = s_{21}$. Then

$$s_{11} l_1' m_1' + s_{12}\left(l_1' m_2' + l_2' m_1'\right) + s_{22} l_2' m_2' = 0 \tag{24}$$

Although there are three unknowns, only their ratio matters, which means one of them can be set as 1. Let $s_{22} = 1$, then there are only two unknowns.

$$s_{11} l_1' m_1' + s_{12}\left(l_1' m_2' + l_2' m_1'\right) = -l_2' m_2' \tag{25}$$

So two equations should be sufficient to solve for $\mathbf{S}$, meaning two pairs of angle-to-angle correspondences that are orthogonal in the original scene are needed.

To calculate $\mathbf{S}$, assume that $\mathbf{A}$ is positive-definite, then the eigen-decomposition is $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$, where $\mathbf{D} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$ with $\lambda_1, \lambda_2 > 0$ and where the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{A}$. So

$$\begin{aligned} \mathbf{S} &= \mathbf{A}\mathbf{A}^\top && (26) \\ &= \mathbf{V}\mathbf{D}\mathbf{V}^\top \mathbf{V}\mathbf{D}\mathbf{V}^\top && (27) \\ &= \mathbf{V}\mathbf{D}^2\mathbf{V}^\top && (28) \\ &= \mathbf{V}\begin{bmatrix} \lambda_1^2 & 0 \\ 0 & \lambda_2^2 \end{bmatrix}\mathbf{V}^\top && (29) \end{aligned}$$

where the fact that $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$ is used. So by doing an eigen-decomposition of $\mathbf{S}$, the eigenvectors and eigenvalues of $\mathbf{A}$ can be obtained. Then $\mathbf{A}$ can be calculated, which can be used to estimate $\mathbf{H}$ by

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{30}$$

Finally apply $\mathbf{H}$ to the image without the projective distortion, then the affine distortion can be eliminated.

The algorithm is shown in Algorithm 3. The inputs are transformed image from the last step I_{di}, tansformed points by the last step $P_{dt,di}$. The output is the desired image I_d without projective and affine distortions.

Algorithm 3: Dual degenerate conic to eliminate affine distortion(I_{di}, $P_{dt,di}$)

calculate $\mathbf{H}$ based on the method mentioned above ($\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}$);
calculate the image dimension of the desired image, and create an empty image, I_d;
For each pixel $p' \in I_d$
    $p = \mathbf{H}^{-1}p'$;
    If $p \in I_{di}$ **:**
        $I_d[p'] = I_{di}[p]$;
Return I_d;

One-step approach

Let $\mathbf{C}_\infty^{*\prime}$ be a projection of the dual degenerate conic $\mathbf{C}_\infty^*$, then this method eliminates both projective and affine distortions using the homography that maps $\mathbf{C}_\infty^{*\prime}$ back to $\mathbf{C}_\infty^*$. The projection of the dual degenerate conic can be written as $\mathbf{C}_\infty^{*\prime} = \mathbf{H}\mathbf{C}_\infty^*\mathbf{H}^\top$. Substitute it into Equation 19, then

$$\begin{aligned} \cos\theta|_{numerator} &= \mathbf{l}'^\top\mathbf{H}\mathbf{C}_\infty^*\mathbf{H}^\top\mathbf{m}' && (31) \\ &= \mathbf{l}'^\top\mathbf{C}_\infty^{*\prime}\mathbf{m}' && (32) \\ &= 0 && (33) \end{aligned}$$

Notice that the corresponding lines $\mathbf{l}$ and $\mathbf{m}$ of lines $\mathbf{l}'$ and $\mathbf{m}'$ are orthogonal. Let

$$\mathbf{C}_\infty^{*\prime} = \begin{bmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{bmatrix} \tag{34}$$

Then

$$\begin{pmatrix} l_1' & l_2' & l_3' \end{pmatrix} \begin{bmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} = 0 \tag{35}$$

then

$$l_1'm_1'a + \frac{l_2'm_1' + l_1'm_2'}{2}b + l_2'm_2'c + \frac{l_3'm_1' + l_1'm_3'}{2}d + \frac{l_3'm_2' + l_2'm_3'}{2}e + l_3'm_3'f = 0 \tag{36}$$

Although there are six unknowns, only their ratio matters, which means one of them can be set as 1. Let $f = 1$, then there are only five unknowns.

$$l'_1 m'_1 a + \frac{l'_2 m'_1 + l'_1 m'_2}{2} b + l'_2 m'_2 c + \frac{l'_3 m'_1 + l'_1 m'_3}{2} d + \frac{l'_3 m'_2 + l'_2 m'_3}{2} e = -l'_3 m'_3 \tag{37}$$

So five equations should be sufficient to solve for $\mathbf{C}^{*\prime}_{\infty}$, meaning five pairs of angle-to-angle correspondences that are orthogonal in the original scene are needed.

When $\mathbf{C}^{*\prime}_{\infty}$ is calculated, then observe

$$\mathbf{C}^{*\prime}_{\infty} = \mathbf{H}\mathbf{C}^{*}_{\infty}\mathbf{H}^{\top} \tag{38}$$

$$= \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{v}^{\top} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A}^{\top} & \mathbf{v} \\ \mathbf{0}^{\top} & 1 \end{bmatrix} \tag{39}$$

$$= \begin{bmatrix} \mathbf{A}\mathbf{A}^{\top} & \mathbf{A}\mathbf{v} \\ \mathbf{v}^{\top}\mathbf{A}^{\top} & \mathbf{v}^{\top}\mathbf{v} \end{bmatrix} \tag{40}$$

Compare the matrices, the relationships are

$$\mathbf{A}\mathbf{A}^{\top} = \begin{bmatrix} a & \frac{b}{2} \\ \frac{b}{2} & c \end{bmatrix} \tag{41}$$

$$\mathbf{A}\mathbf{v} = \begin{bmatrix} \frac{d}{2} \\ \frac{e}{2} \end{bmatrix} \tag{42}$$

Then $\mathbf{A}$ can be solved by eigen-decomposition of $\begin{bmatrix} a & \frac{b}{2} \\ \frac{b}{2} & c \end{bmatrix}$ and $\mathbf{v}$ can be solved by $\mathbf{A}^{-1}\begin{bmatrix} \frac{d}{2} \\ \frac{e}{2} \end{bmatrix}$. Then $\mathbf{H}$ is estimated. The matrix $\mathbf{H}^{-1}$ will be the desired homography to rectify both projective and affine distortions.

The algorithm is shown in Algorithm 4. The inputs are recorded image I, detected points P_{dt}. The output is the desired image I_d.

Algorithm 4: Two-step approach(I, P_{dt})

calculate $\mathbf{H}$ based on the method mentioned above ($\mathbf{h} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{v}^{\top} & 1 \end{bmatrix}$);
calculate the image dimension of the desired image, and create an empty image, I_d;
For each pixel $p' \in I_d$
    $p = \mathbf{H}p'$;
    If $p \in I$ **:**
        $I_d[p'] = I[p]$;
Return I_d;

TASK 1

The points used in task 1 are shown in Figure 1 and Table 2. Assume that each set of points can form a square.

(a) Image 1 (b) Image 2

Figure 1: Points used in task 1.

Table 1: Points used in task 1.

Image	P	Q	R	S
Image 1	(156, 320)	(377, 317)	(388, 459)	(203, 458)
Image 2	(195, 383)	(640, 384)	(623, 746)	(216, 745)

Point-to-point correspondences

The points estimated in the undistorted images are shown in Table 2.

Table 2: Points used in task 1.

Image	P	Q	R	S
Image 1	(0, 0)	(120, 0)	(120, 120)	(0, 120)
Image 2	(0, 0)	(170, 0)	(170, 170)	(0, 170)

The calculated homography matrices are shown below, which are mapping from the recorded images to the undistorted scenes ($\mathbf{H}_{p2p}$). And the results are shown in Figure 2. The performances are promising. After the processing, the images are undistorted as desired.

$$\mathbf{H}_{Img1,p2p} = \begin{bmatrix} 0.405085 & -0.137964 & -19.044863 \\ 0.007143 & 0.526227 & -169.506863 \\ 0.000088 & -0.000895 & 1 \end{bmatrix} \tag{43}$$

$$\mathbf{H}_{Img2,p2p} = \begin{bmatrix} 0.350569 & -0.020337 & -60.571859 \\ -0.000886 & 0.394050 & -150.748394 \\ 0.000001 & -0.000216 & 1 \end{bmatrix} \tag{44}$$

(a) Image 1 (b) Image 2

Figure 2: Resulting images with point-to-point approach.

Two-step approach

(a) Eliminate projective distortion with the vanishing line method

In this step, the pairs of lines (PQ, RS) and (SP, QR) are used. The calculated homography matrices are shown below, which are mapping from the recorded images to the images without projective distortion ($\mathbf{H}_{proj}$). And the results are shown in Figure 3. The performances are promising. After the processing, only the affine, similarity, and euclidean distortions exist, because the expectantly parallel lines are parallel now.

$$\mathbf{H}_{Img1,proj} = \begin{bmatrix} 1.000000 & 0 & 0 \\ 0 & 1.000000 & 0 \\ 0.000088 & -0.000895 & 1 \end{bmatrix} \tag{45}$$

$$\mathbf{H}_{Img2,proj} = \begin{bmatrix} 1.000000 & 0 & 0 \\ 0 & 1.000000 & 0 \\ 0.000001 & -0.000216 & 1 \end{bmatrix} \tag{46}$$

(a) Image 1 (b) Image 2

Figure 3: Resulting images with vanishing line method (without projective distortion).

(b) Eliminate affine distortion with the dual degenerate conic method

In this step, the pairs of lines (SP, PQ) and (PR, QS) are used. The calculated homography matrices are shown below, which are mapping from the images without projective distortion to the images without projective and affine distortion ($\mathbf{H}_{aff}$), and mapping from the recorded images to the images without projective and affine distortion ($\mathbf{H}_{comb} = \mathbf{H}_{aff}\mathbf{H}_{proj}$). And the results are shown in Figure 4. The performances are promising. After the processing, only the similarity and euclidean distortions exist, because the expectantly parallel lines are parallel, and the expectantly orthogonal lines are orthogonal now.

$$\mathbf{H}_{Img1,proj} = \begin{bmatrix} 0.980739 & 0.167882 & 0 \\ 0.167882 & 0.985807 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{47}$$

$$\mathbf{H}_{Img1,comb} = \begin{bmatrix} 1.050256 & -0.178858 & 0 \\ -0.178858 & 1.044857 & 0 \\ 0.000088 & -0.000895 & 1 \end{bmatrix} \tag{48}$$

$$\mathbf{H}_{Img2,proj} = \begin{bmatrix} 1.034074 & 0.047980 & 0 \\ 0.047980 & 0.998848 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{49}$$

$$\mathbf{H}_{Img2,comb} = \begin{bmatrix} 0.969209 & -0.046556 & 0 \\ -0.046556 & 1.003389 & 0 \\ 0.000001 & -0.000216 & 1 \end{bmatrix} \tag{50}$$

(a) Image 1 (b) Image 2

Figure 4: Resulting images with dual degenerate conic method (without projective and affine distortions).

One-step approach

In this approach, the pairs of lines (SP, PQ), (PQ, QR), (QR, RS), (RS, SP), and (PR, QS) are used. The calculated homography matrices are shown below, which are mapping from the recorded images to the images without projective and affine distortions ($\mathbf{H}_{1step}$). And the results are shown in Figure 5. The performances are promising. After the processing, only the similarity and euclidean distortions exist, because the expectantly parallel lines are parallel, and the expectantly orthogonal lines are orthogonal now.

$$\mathbf{H}_{Img1,1step} = \begin{bmatrix} 0.001041 & -0.000196 & 0 \\ -0.000196 & 0.000920 & 0 \\ 0.000088 & -0.000895 & 1 \end{bmatrix} \tag{51}$$

$$\mathbf{H}_{Img2,1step} = \begin{bmatrix} 0.000211 & -0.000010 & 0 \\ -0.000010 & 0.000217 & 0 \\ 0.000001 & -0.000216 & 1 \end{bmatrix} \tag{52}$$

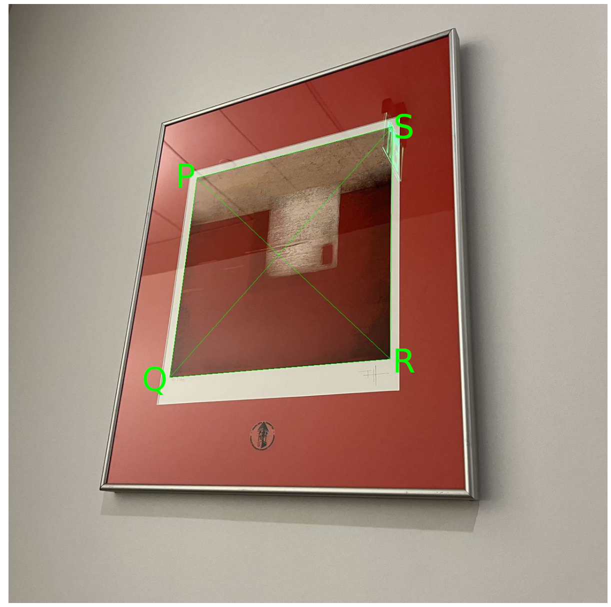
(a) Image 1 (b) Image 2

Figure 5: Resulting images with one-step approach.

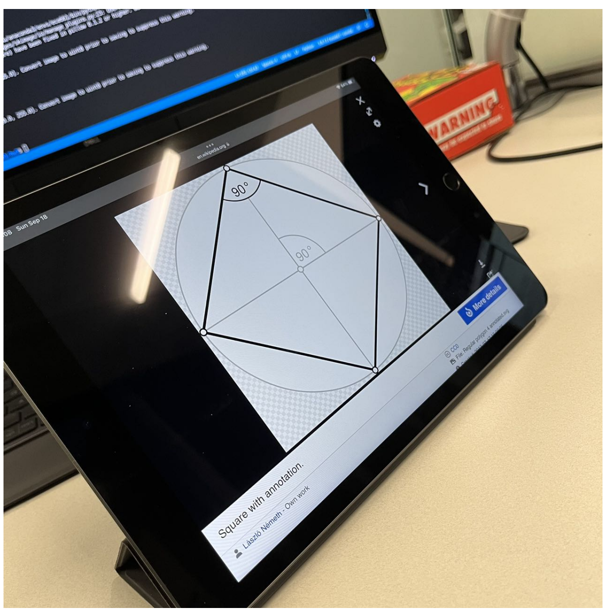
TASK 2

The points used in task 1 are shown in Figure 6 and Table 4. Assume that each set of points can form a square.

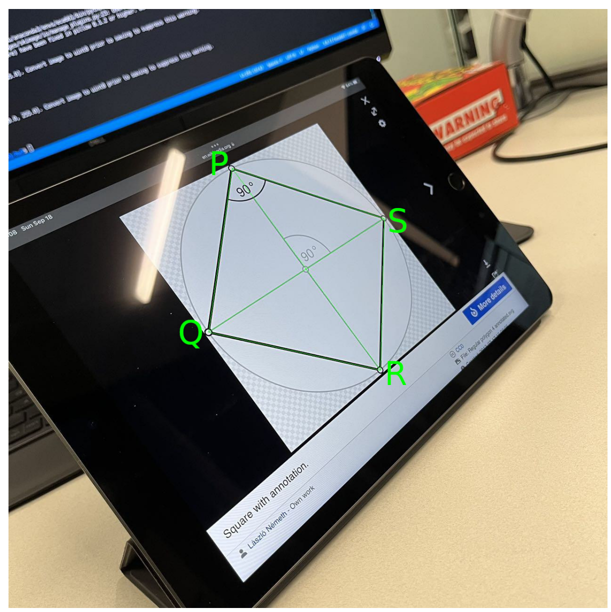
(a) Painting



(b) Painting with annotation



(c) Square



(d) Square with annotation

Figure 6: Images and points used in task 2.

Table 3: Points used in task 2.

Image	P	Q	R	S
Image 1	(370, 401)	(795, 343)	(756, 813)	(261, 817)
Image 2	(339, 476)	(688, 426)	(769, 792)	(446, 800)

Point-to-point correspondences

The points estimated in the undistorted images are shown in Table 4.

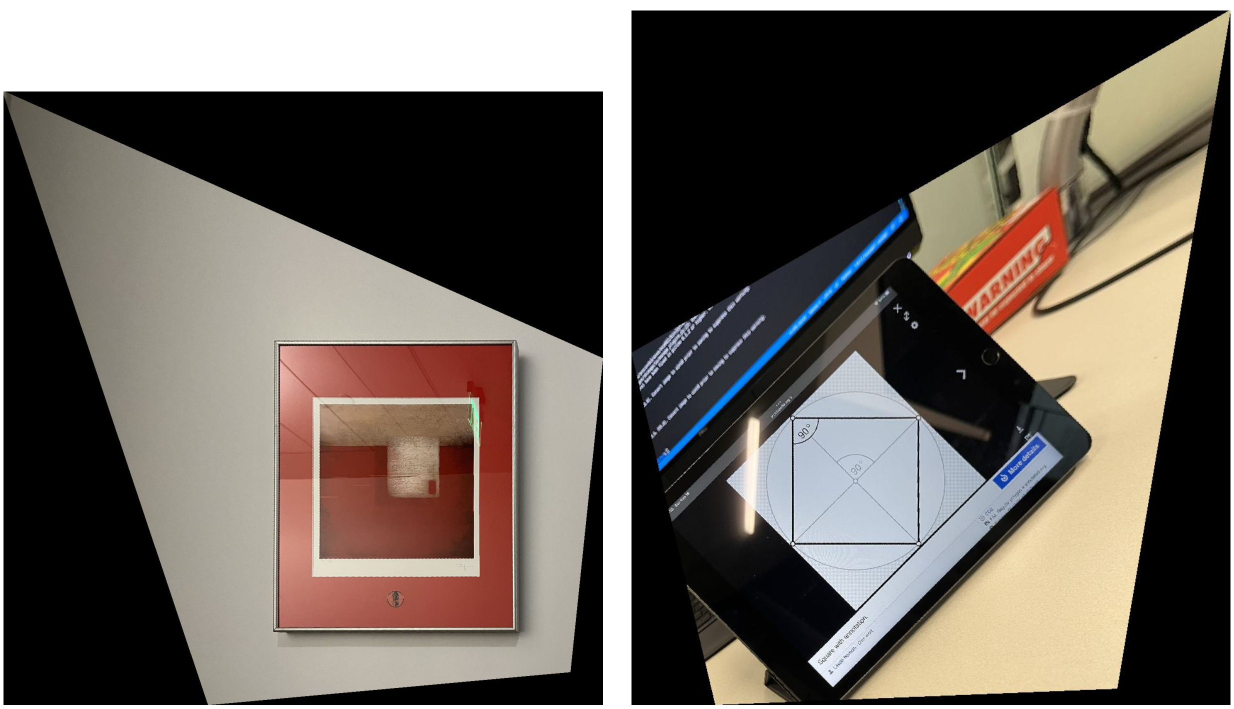
Table 4: Points used in task 1.

Image	P	Q	R	S
Image 1	(0, 0)	(400, 0)	(400, 400)	(0, 400)
Image 2	(0, 0)	(200, 0)	(200, 170)	(0, 200)

The calculated homography matrices are shown below, which are mapping from the recorded images to the undistorted scenes ($\mathbf{H}_{p2p}$). And the results are shown in Figure 7. The performances are promising. After the processing, the images are undistorted as desired.

$$\mathbf{H}_{Painting,p2p} = \begin{bmatrix} 1.704014 & 0.446485 & -809.525574 \\ 0.247198 & 1.811366 & -817.820939 \\ 0.000588 & 0.000812 & 1 \end{bmatrix} \tag{53}$$

$$\mathbf{H}_{Square,p2p} = \begin{bmatrix} 0.566301 & -0.187019 & -102.954866 \\ 0.069967 & 0.488371 & -256.183685 \\ 0.000280 & -0.000370 & 1 \end{bmatrix} \tag{54}$$



(a) Painting (b) Square

Figure 7: Resulting images with point-to-point approach.

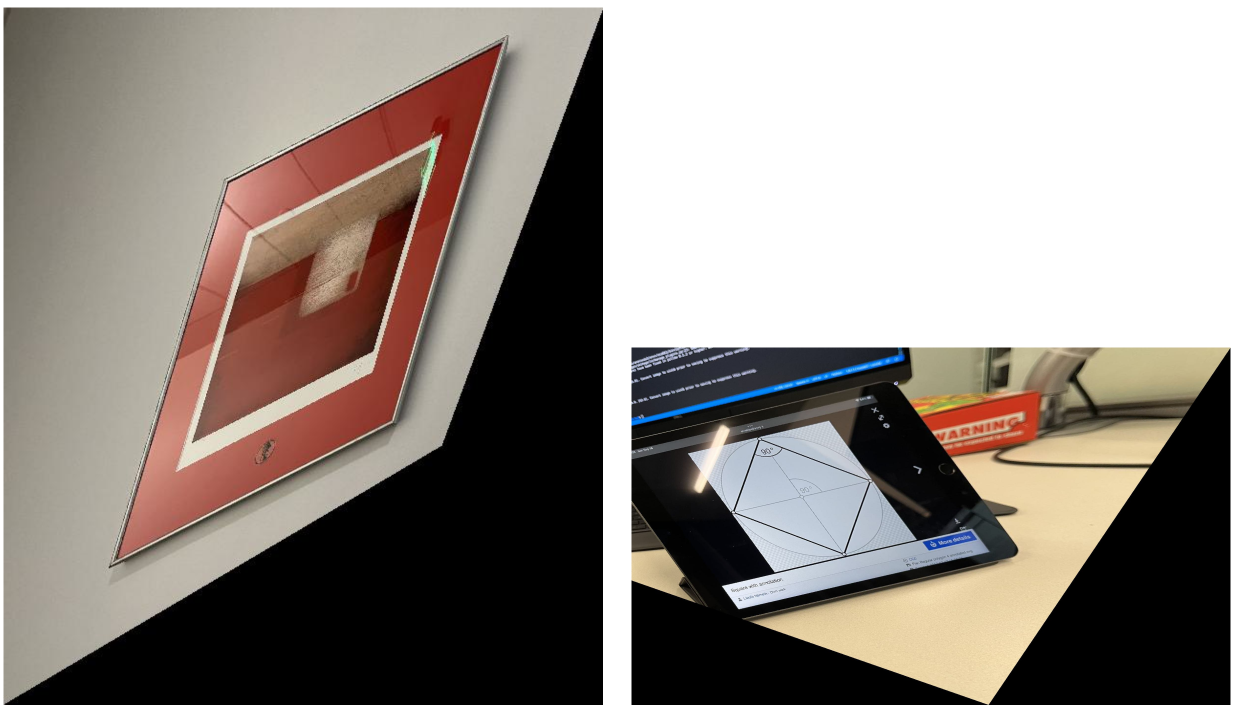
Two-step approach

(a) Eliminate projective distortion with the vanishing line method

In this step, the pairs of lines (PQ, RS) and (SP, QR) are used. The calculated homography matrices are shown below, which are mapping from the recorded images to the images without projective distortion ($\mathbf{H}_{proj}$). And the results are shown in Figure 8. The performances are promising. After the processing, only the affine, similarity, and euclidean distortions exist, because the expectantly parallel lines are parallel now.

$$\mathbf{H}_{Painting,proj} = \begin{bmatrix} 1.000000 & 0 & 0 \\ 0 & 1.000000 & 0 \\ 0.000588 & 0.000812 & 1 \end{bmatrix} \tag{55}$$

$$\mathbf{H}_{Square,proj} = \begin{bmatrix} 1.000000 & 0 & 0 \\ 0 & 1.000000 & 0 \\ 0.000280 & -0.000370 & 1 \end{bmatrix} \tag{56}$$



(a) Painting (b) Square

Figure 8: Resulting images with vanishing line method (without projective distortion).

(b) Eliminate affine distortion with the dual degenerate conic method

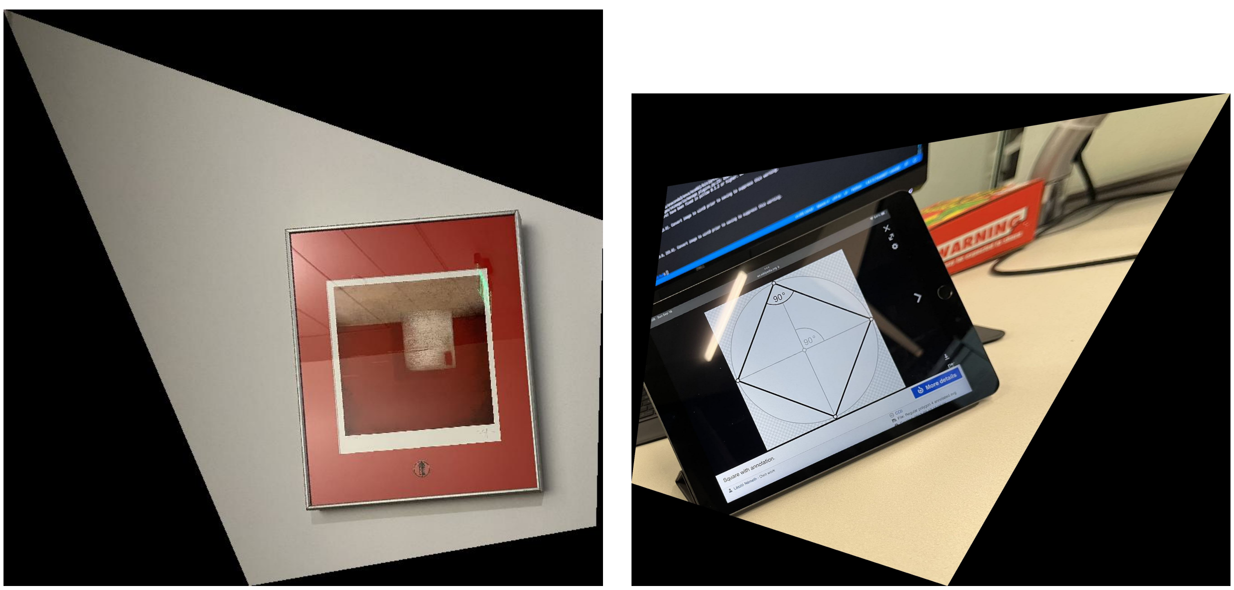
In this step, the pairs of lines (SP, PQ) and (PR, QS) are used. The calculated homography matrices are shown below, which are mapping from the images without projective distortion to the images without projective and affine distortion ($\mathbf{H}_{aff}$), and mapping from the recorded images to the images without projective and affine distortion ($\mathbf{H}_{comb} = \mathbf{H}_{aff}\mathbf{H}_{proj}$). And the results are shown in Figure 9. The performances are promising. After the processing, only the similarity and euclidean distortions exist, because the expectantly parallel lines are parallel, and the expectantly orthogonal lines are orthogonal now.

$$\mathbf{H}_{Painting,proj} = \begin{bmatrix} 1.111983 & -0.391983 & 0 \\ -0.391983 & 0.919972 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{57}$$

$$\mathbf{H}_{Painting,comb} = \begin{bmatrix} 1.058240 & 0.450896 & 0 \\ 0.450896 & 1.279108 & 0 \\ 0.000588 & 0.000812 & 1 \end{bmatrix} \tag{58}$$

$$\mathbf{H}_{Square,proj} = \begin{bmatrix} 0.731003 & 0.121263 & 0 \\ 0.121263 & 0.992620 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{59}$$

$$\mathbf{H}_{Square,comb} = \begin{bmatrix} 1.396279 & -0.170575 & 0 \\ -0.170575 & 1.028273 & 0 \\ 0.000280 & -0.000370 & 1 \end{bmatrix} \tag{60}$$



(a) Painting

(b) Square

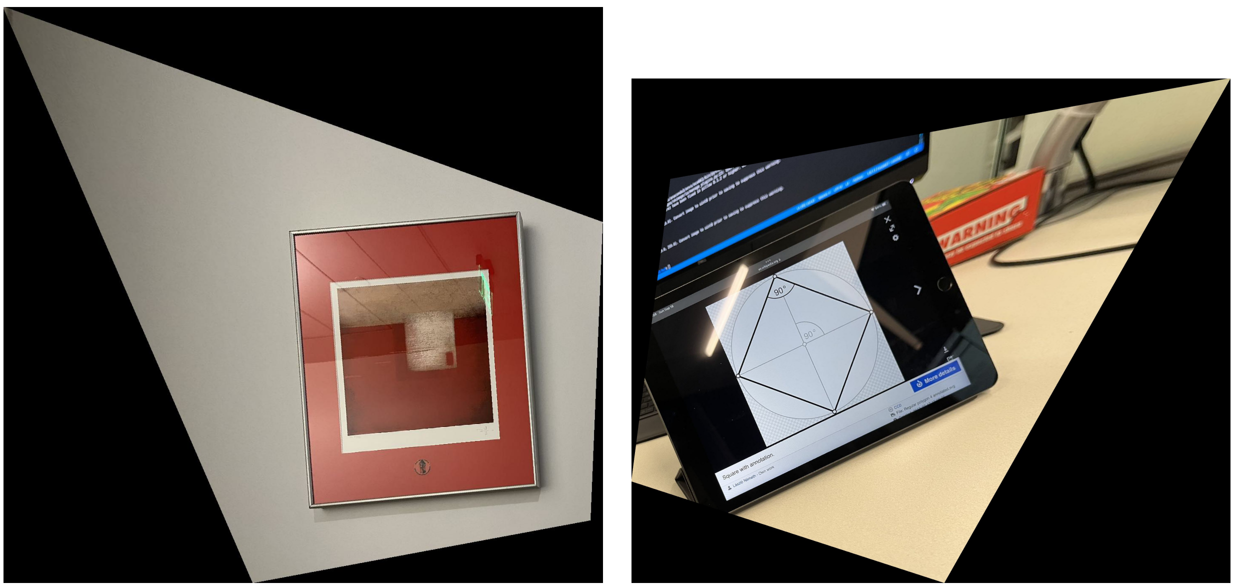
Figure 9: Resulting images with dual degenerate conic method (without projective and affine distortions).

One-step approach

In this approach, the pairs of lines (SP, PQ), (PQ, QR), (QR, RS), (RS, SP), and (PR, QS) are used. The calculated homography matrices are shown below, which are mapping from the recorded images to the images without projective and affine distortions ($\mathbf{H}_{1step}$). And the results are shown in Figure 10. The performances are promising. After the processing, only the similarity and euclidean distortions exist, because the expectantly parallel lines are parallel, and the expectantly orthogonal lines are orthogonal now.

$$\mathbf{H}_{Painting,1step} = \begin{bmatrix} 0.000643 & 0.000278 & 0 \\ 0.000278 & 0.000774 & 0 \\ 0.000639 & 0.000865 & 1 \end{bmatrix} \tag{61}$$

$$\mathbf{H}_{Square,1step} = \begin{bmatrix} 0.000523 & -0.000068 & 0 \\ -0.000067 & 0.000377 & 0 \\ 0.000280 & -0.000370 & 1 \end{bmatrix} \tag{62}$$



(a) Painting

(b) Square

Figure 10: Resulting images with one-step approach.

SOURCE CODE

```python
import argparse
import os
import numpy as np
from skimage import io, draw

def get_homography(domain_point, range_point):
    # calculate H
    num_points = domain_point.shape[0]
    A = np.zeros((2*num_points, 8), dtype=float)
    b = np.zeros((2*num_points,), dtype=float)
    for i in range(num_points):
        A[2*i, 0] = domain_point[i, 0]
        A[2*i, 1] = domain_point[i, 1]
        A[2*i, 2] = 1
```

```
        A[2*i, 6] = - range_point[i, 0] * domain_point[i, 0]
        A[2*i, 7] = - range_point[i, 0] * domain_point[i, 1]
        A[2*i+1, 3] = domain_point[i, 0]
        A[2*i+1, 4] = domain_point[i, 1]
        A[2*i+1, 5] = 1
        A[2*i+1, 6] = - range_point[i, 1] * domain_point[i, 0]
        A[2*i+1, 7] = - range_point[i, 1] * domain_point[i, 1]
        b[2*i] = range_point[i, 0]
        b[2*i+1] = range_point[i, 1]
    #h = np.linalg.inv(A.T @ A) @ A.T @ b
    h = np.linalg.inv(A) @ b
    H = np.ones((3, 3), dtype=float)
    for i in range(h.shape[0]):
        q, r = np.divmod(i, 3)
        H[q, r] = h[i]
    return H

def get_point(H, p, rescale):
    # get point after transformation
    point_prime = np.array([p[0], p[1], 1])
    point = H @ point_prime
    x = int(point[0]/point[2] * rescale)
    y = int(point[1]/point[2] * rescale)
    return np.array([x, y], dtype=int)

def get_corner_points(H, img, rescale=1):
    # get corner points
    corner_points = np.zeros((4, 2), dtype=int)
    corner_points[0] = get_point(H, (0, 0), rescale)
    corner_points[1] = get_point(H, (img.shape[0], 0), rescale)
    corner_points[2] = get_point(H, (img.shape[0], img.shape[1]),\
                                rescale)
    corner_points[3] = get_point(H, (0, img.shape[1]), rescale)
    return corner_points

def get_img_dim(corner_points):
    # get the dimension of image
    bottom, right = np.max(corner_points, axis=0)
    upper, left = np.min(corner_points, axis=0)
    dim_x = bottom - upper + 1
    dim_y = right - left + 1
    return dim_x, dim_y, upper, left

def plot_transformation(img, H, title, rescale=1):
    # calculate and plot the result
    corner_points = get_corner_points(H, img, rescale)
    dim_x, dim_y, offset_x, offset_y = get_img_dim(corner_points)
    print('Dimension of the image is %s by %s'%(dim_x, dim_y))
    img_undist = np.zeros((dim_x, dim_y, 3), dtype=float)
    for i in range(dim_x):
        for j in range(dim_y):
            point = np.array([(i+offset_x)/rescale,\
                              (j+offset_y)/rescale, 1])
            point_prime = np.linalg.inv(H) @ point
```

```python
            x_prime = int(point_prime[0]/point_prime[2])
            y_prime = int(point_prime[1]/point_prime[2])
            if x_prime >= 0 and x_prime <= img.shape[0]-1 and\
                y_prime >= 0 and y_prime <= img.shape[1]-1:
                img_undist[i, j] = img[x_prime, y_prime]
    io.imsave('./%s.jpg'%title, img_undist)

def point2point(domain_point, range_point, img, title):
    # Range (distorted) = H * Domain (undistorted)
    H = get_homography(domain_point, range_point)

    plot_transformation(img, np.linalg.inv(H), title)

    np.set_printoptions(suppress=True)
    print('H for %s:\n'%title,\
            np.linalg.inv(H)/np.linalg.inv(H)[2,2])
    np.set_printoptions(suppress=False)

def get_line_from_points(p1, p2):
    # calculate line according to two points
    p1_rep = np.array([p1[0], p1[1], 1])
    p2_rep = np.array([p2[0], p2[1], 1])
    l = np.cross(p1_rep, p2_rep).astype(float)
    l /= np.linalg.norm(l)
    return l

def remove_proj_dist(points, img, title):
    # eliminate projective distortion
    l1 = get_line_from_points(points[0], points[3])
    l2 = get_line_from_points(points[1], points[0])
    l3 = get_line_from_points(points[2], points[1])
    l4 = get_line_from_points(points[3], points[2])
    l5 = get_line_from_points(points[1], points[3])
    l6 = get_line_from_points(points[0], points[2])
    vanishing_point1 = np.cross(l1, l3)
    vanishing_point2 = np.cross(l2, l4)
    vanishing_line = np.cross(vanishing_point1, vanishing_point2)
    vanishing_line /= np.linalg.norm(vanishing_line)
    H = np.identity(3, dtype=float)
    H[2] = vanishing_line
    H = H / H[2, 2]
    plot_transformation(img, H, title)

    np.set_printoptions(suppress=True)
    print('H for %s:\n'%title, H)
    np.set_printoptions(suppress=False)
    return [l1, l2, l3, l4, l5, l6], H

def lines_tranformation(lines, H):
    # transform lines
    lines_aff = []
    for line in lines:
        line_aff = np.transpose(np.linalg.inv(H)) @ line
        line_aff /= np.linalg.norm(line_aff)
```

```
        lines_aff.append(line_aff)
    return lines_aff

def get_S(lines_aff):
    # calculate S
    A = np.zeros((2, 2), dtype=float) # A is not that one in S = AA^T
    b = np.zeros((2,), dtype=float)

    A[0, 0] = lines_aff[0][0] * lines_aff[1][0]
    A[0, 1] = lines_aff[0][0] * lines_aff[1][1]\
              + lines_aff[0][1] * lines_aff[1][0]
    A[1, 0] = lines_aff[4][0] * lines_aff[5][0]
    A[1, 1] = lines_aff[4][0] * lines_aff[5][1]\
              + lines_aff[4][1] * lines_aff[5][0]
    b[0] = - lines_aff[0][1] * lines_aff[1][1]
    b[1] = - lines_aff[4][1] * lines_aff[5][1]
    #s = np.linalg.inv(A.T @ A) @ A.T @ b
    s = np.linalg.inv(A) @ b
    S = np.ones((2, 2), dtype=float)
    S[0, 0] = s[0]
    S[0, 1] = s[1]
    S[1, 0] = S[0, 1]
    return S

def get_H_from_S(S):
    # calculate H according to S
    u, s, vh = np.linalg.svd(S)
    eigenvalues = np.sqrt(np.diag(s))
    A = vh @ eigenvalues @ np.transpose(vh)
    H = np.zeros((3, 3), dtype=float)
    H[0:2, 0:2] = A
    H[2, 2] = 1
    return H

def remove_aff_dist(lines_proj, H_aff, img, title):
    # eliminate affine distortion
    lines_aff = lines_tranformation(lines_proj, H_aff)
    S = get_S(lines_aff)
    H_undist = get_H_from_S(S)
    H_combine = np.linalg.inv(H_undist) @ H_aff
    H_combine /= H_combine[2, 2]
    plot_transformation(img, H_combine, title)

    np.set_printoptions(suppress=True)
    print('H for affine distortion removal (%s):\n'%title, H_undist)
    print('H for %s:\n'%title, H_combine)
    np.set_printoptions(suppress=False)

def get_conic(lines):
    # calculate C
    A = np.zeros((5, 5), dtype=float)
    b = np.zeros((5,), dtype=float)
    for i in range(3):
        A[i] = np.array([lines[i][0]*lines[i+1][0],\
```

```
                              lines[i][1]*lines[i+1][0]\
                                  + lines[i][0]*lines[i+1][1],\
                              lines[i][1]*lines[i+1][1],\
                              lines[i][2]*lines[i+1][0]\
                                  + lines[i][0]*lines[i+1][2],\
                              lines[i][2]*lines[i+1][1]\
                                  + lines[i][1]*lines[i+1][2]])
        b[i] = - lines[i][2]*lines[i+1][2]
    A[3] = np.array([lines[3][0]*lines[0][0],\
                          lines[3][1]*lines[0][0]\
                              + lines[3][0]*lines[0][1],\
                          lines[3][1]*lines[0][1],\
                          lines[3][2]*lines[0][0]\
                              + lines[3][0]*lines[0][2],\
                          lines[3][2]*lines[0][1]\
                              + lines[3][1]*lines[0][2]])
    b[3] = - lines[3][2]*lines[0][2]
    A[4] = np.array([lines[4][0]*lines[5][0],\
                          lines[4][1]*lines[5][0]\
                              + lines[4][0]*lines[5][1],\
                          lines[4][1]*lines[5][1],\
                          lines[4][2]*lines[5][0]\
                              + lines[4][0]*lines[5][2],\
                          lines[4][2]*lines[5][1]\
                              + lines[4][1]*lines[5][2]])
    b[3] = - lines[4][2]*lines[5][2]
    #c = np.linalg.inv(A.T @ A) @ A.T @ b
    c = np.linalg.inv(A) @ b # [a, b/2, c, d/2, e/2, f=1]
    return c

def one_step_method(points, img, title, rescale=1):
    # one-step approach
    l1 = get_line_from_points(points[0], points[3])
    l2 = get_line_from_points(points[1], points[0])
    l3 = get_line_from_points(points[2], points[1])
    l4 = get_line_from_points(points[3], points[2])
    l5 = get_line_from_points(points[1], points[3])
    l6 = get_line_from_points(points[0], points[2])
    c = get_conic([l1, l2, l3, l4, l5, l6])
    u, s, vh = np.linalg.svd(np.array([[c[0], c[1]],\
                                          [c[1], c[2]]]))
    eigenvalues = np.sqrt(np.diag(s))
    A = vh @ eigenvalues @ np.transpose(vh)
    v = np.linalg.inv(A) @ np.array([c[3], c[4]])
    H = np.zeros((3, 3), dtype=float)
    H[0:2, 0:2] = A
    H[2, 0:2] = v
    H[2, 2] = 1
    plot_transformation(img, np.linalg.inv(H), title, rescale)

    np.set_printoptions(suppress=True)
    print('H for %s:\n'%title, np.linalg.inv(H)/np.linalg.inv(H)[2,2])
    np.set_printoptions(suppress=False)

```

```python
def draw_lines(points, img, title):
    # draw annotation lines
    rr, cc = draw.line(points[0][0], points[0][1],\
                       points[3][0], points[3][1])
    draw.set_color(img, [rr, cc], [0, 255, 0])
    rr, cc = draw.line(points[1][0], points[1][1],\
                       points[0][0], points[0][1])
    draw.set_color(img, [rr, cc], [0, 255, 0])
    rr, cc = draw.line(points[2][0], points[2][1],\
                       points[1][0], points[1][1])
    draw.set_color(img, [rr, cc], [0, 255, 0])
    rr, cc = draw.line(points[3][0], points[3][1],\
                       points[2][0], points[2][1])
    draw.set_color(img, [rr, cc], [0, 255, 0])
    rr, cc = draw.line(points[3][0], points[3][1],\
                       points[1][0], points[1][1])
    draw.set_color(img, [rr, cc], [0, 255, 0])
    rr, cc = draw.line(points[2][0], points[2][1],\
                       points[0][0], points[0][1])
    draw.set_color(img, [rr, cc], [0, 255, 0])
    io.imsave('%s_lines.jpg'%title, img)

if __name__ == '__main__':
    # '1.1', '2.1' -- point-to-point in task 1 or 2
    # '1.2', '2.2' -- two-step approach in task 1 or 2
    # '1.3', '2.3' -- one-step approach in task 1 or 2
    parser = argparse.ArgumentParser()
    parser.add_argument('-t', '--task', type=str, default='1.1',\
        help='choose a task', choices=['1.1','1.2','1.3',\
                                       '2.1','2.2','2.3'])
    args = parser.parse_args()
    # P ------l1----- S
    # |  \         /  |
    # |  l6\     /l5  |
    # l2       X      l4
    # |    /     \    |
    # |  /         \  |
    # Q ------l3----- R

    if args.task == '1.1' or '1.2' or '1.3':
        building = io.imread('./hw3images/building.jpg')
        nighthawks = io.imread('./hw3images/nighthawks.jpg')
        building_points = np.array([[156, 320],\
                                    [377, 317],\
                                    [388, 459],\
                                    [203, 458]])
        nighthawks_points = np.array([[195, 383],\
                                      [640, 384],\
                                      [623, 746],\
                                      [216, 745]])
        if not os.path.exists('building_lines.jpg'):
            draw_lines(building_points, building, 'building')
        if not os.path.exists('nighthawks_lines.jpg'):
            draw_lines(nighthawks_points, nighthawks, 'nighthawks')
```

```

    if args.task == '1.1':
        building_points_undist = np.array([[0, 0],\
                                           [120, 0],\
                                           [120, 120],\
                                           [0, 120]])
        nighthawks_points_undist = np.array([[0, 0],\
                                             [170, 0],\
                                             [170, 170],\
                                             [0, 170]])
        point2point(building_points_undist, building_points,\
                    building, 'building_p2p')
        point2point(nighthawks_points_undist, nighthawks_points,\
                    nighthawks, 'nighthawks_p2p')

    if args.task == '1.2':
        lines_proj_bldg, H_aff_bldg = remove_proj_dist(building_points,\
                                      building, 'building_remove_proj')
        lines_proj_nh, H_aff_nh = remove_proj_dist(nighthawks_points,\
                                  nighthawks, 'nighthawks_remove_proj')

        remove_aff_dist(lines_proj_bldg, H_aff_bldg,\
                        building, 'building_remove_proj_aff')
        remove_aff_dist(lines_proj_nh, H_aff_nh,\
                        nighthawks,'nighthawks_remove_proj_aff')

    if args.task == '1.3':
        one_step_method(building_points, building,\
                        'building_1step', 3000)
        one_step_method(nighthawks_points, nighthawks,\
                        'nighthawks_1step', 3000)

    if args.task == '2.1' or '2.2' or '2.3':
        painting = io.imread('./painting.jpg')
        square = io.imread('./square.jpg')
        painting_points = np.array([[370, 401],\
                                    [795, 343],\
                                    [756, 813],\
                                    [261, 817]])
        square_points = np.array([[339, 476],\
                                  [688, 426],\
                                  [769, 792],\
                                  [446, 800]])
        if not os.path.exists('painting_lines.jpg'):
            draw_lines(painting_points, painting, 'painting')
        if not os.path.exists('square_lines.jpg'):
            draw_lines(square_points, square, 'square')

    if args.task == '2.1':
        painting_points_undist = np.array([[0, 0],\
                                           [400, 0],\
                                           [400, 400],\
                                           [0, 400]])
        square_points_undist = np.array([[0, 0],\
```

```
                                              [200, 0],\
                                              [200, 200],\
                                              [0, 200]])
        point2point(painting_points_undist, painting_points,\
                    painting, 'painting_p2p')
        point2point(square_points_undist, square_points,\
                    square, 'square_p2p')

    if args.task == '2.2':
        lines_proj_ptg, H_aff_ptg = remove_proj_dist(painting_points,\
                                    painting, 'painting_remove_proj')
        lines_proj_sq, H_aff_sq = remove_proj_dist(square_points,\
                                  square, 'square_remove_proj')

        remove_aff_dist(lines_proj_ptg, H_aff_ptg,\
                        painting, 'painting_remove_proj_aff')
        remove_aff_dist(lines_proj_sq, H_aff_sq,\
                        square, 'square_remove_proj_aff')

    if args.task == '2.3':
        one_step_method(painting_points, painting,\
                        'painting_1step', 3000)
        one_step_method(square_points, square,\
                        'square_1step', 3000)
```