ECE 661: Homework 3

Christina Eberhardt (eberharc@purdue.edu)

Part 1: Point-to-point correspondences

Step 1: Determine the points P (left top corner), Q (left bottom corner), R (right bottom corner) and S (right top corner) for the images Img1, Img2 and Img3 using GIMP. Determine those points matching their measurements

Image	**P**	**Q**	**R**	**S**
Img1	(47, 307)	(8, 575)	(1011, 698)	(1000, 523)
Img2	(478, 718)	(481, 874)	(607, 923)	(601, 736)
Img3	(2062, 702)	(2094, 1480)	(2694, 1329)	(2665, 720)
Img4	(549, 420)	(672, 700)	(1228, 595)	(993, 353)
Img5	(600, 167)	(656, 490)	(968, 403)	(926, 129)

Table 1: Coordinates for PQRS in images Img1 – Img3

The target coordinates for PQRS can be determined from the measurements.

Image	**P**	**Q**	**R**	**S**
Img1_m	(0, 0)	(0, 2890)	(470, 2890)	(470, 0)
Img1_m2	(0, 0)	(0, 75)	(85, 75)	(85, 0)
Img2_m	(0, 0)	(0, 84)	(74, 84)	(74, 0)
Img3_m	(0, 0)	(0, 55)	(36, 55)	(36, 0)
Img4_m	(0, 0)	(0, 60)	(60, 60)	(60,0)
Img5_m	(0, 0)	(0, 160)	(120, 160)	(120, 0)

Table 2: Target Coordinates for PQRS in images Img1 – Img3

To verify the coordinates, plot the points PQRS as determined in the images. The results can be found in Figure 1.

For Img1_m we assume the following:

- The width between two windows is 1.5 times the width of a window
- The height between two windows is 0.5 times the height of a window

Based on these results we choose the closest “nice” number. We also use the smaller coordinates of the same part to get a more detailed view.

For Img4 and Img5 pick measurements as described above.

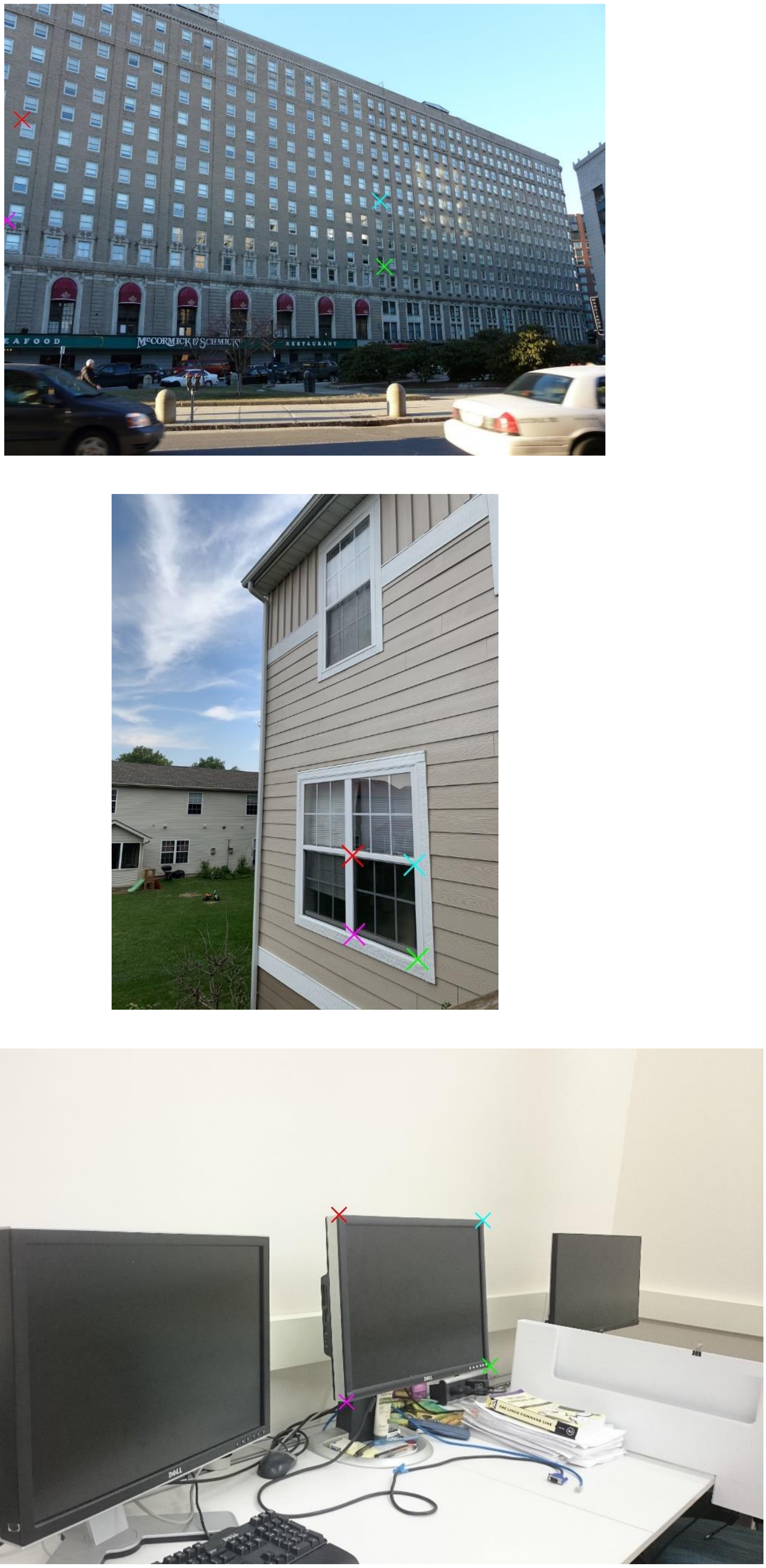


Figure 1: Img1, Img2 and Img3 (from top to bottom) with the chosen coordinates for point-to-point transformation

Figure 2: Img4 (top) and Img4 (bottom) with the chosen coordinates for point-to-point transformation

Step 2: Determine the homographies between the images.

$$\boldsymbol{x}' = \boldsymbol{H}\boldsymbol{x}$$

All coordinates P, Q, R, S need to be translated into homogenous coordinates for this. The homogenous coordinates of a point (x, y) are $\boldsymbol{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$.

Then,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

This yields the following system of equations:

$$x' = h_{11}x + h_{12}y + h_{13}$$

$$y' = h_{21}x + h_{22}y + h_{23}$$

$$1 = h_{31}x + h_{32}y + h_{33}$$

Calculate the coordinates (x', y') from those 3 equations with

$$x' = \frac{x'}{1} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$$

$$y' = \frac{y'}{1} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

Rewrite those equations:

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' - h_{33}x' = 0$$

$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' - h_{33}y' = 0$$

We have four pairs of corresponding points. Hence, we get 8 equations in 9 parameters. No more than 2 of them are on the same straight line. Therefore, we can calculate **H** within a multiplicative constant. Due to H being homogenous it is sufficient to determine **H** within this multiplicative constant – the eight equations are sufficient. Use the indices P, Q, R, S to refer to the respective physical points. Choose $h_{33} = 1$ - this is taking care of the degree of freedom we have with the multiplicative constant - and move its terms to the right side of the equation. Then,

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' = h_{33}x' = x'$$
$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' = h_{33}y' = y'$$

Rewrite this into matrix form:

$$\begin{bmatrix} x_P & y_P & 1 & 0 & 0 & 0 & -x_P x'_P & -y_P x'_P \\ 0 & 0 & 0 & x_P & y_P & 1 & -x_P y'_P & -y_P y'_P \\ x_Q & y_Q & 1 & 0 & 0 & 0 & -x_Q x'_Q & -y_Q x'_Q \\ 0 & 0 & 0 & x_Q & y_Q & 1 & -x_Q y'_Q & -y_Q y'_Q \\ x_R & y_R & 1 & 0 & 0 & 0 & -x_R x'_R & -y_R x'_R \\ 0 & 0 & 0 & x_R & y_R & 1 & -x_R y'_R & -y_R y'_R \\ x_S & y_S & 1 & 0 & 0 & 0 & -x_S x'_S & -y_S x'_S \\ 0 & 0 & 0 & x_S & y_S & 1 & -x_R y'_R & -y_S y'_S \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x'_P \\ y'_P \\ x'_Q \\ y'_Q \\ x'_R \\ y'_R \\ x'_S \\ y'_S \end{bmatrix}$$

This is equivalent to $\boldsymbol{Ah} = \boldsymbol{b}$. Hence, $\boldsymbol{h} = \boldsymbol{A}^{-1}\boldsymbol{b}$. Add the value for $h_{33} = 1$ to the vector and transform it into the 3x3 matrix $\boldsymbol{H}$.

In the next substeps, the homographies between the images and their measurements are calculated using the method outlined above. The results $\boldsymbol{H}$ are reported.

Step 2.1: Determine the homography between the measurement Img1_m and Img1

Let $\boldsymbol{H}_{m1}$ be the homography that maps a point from the measurement to the corresponding point in Img1. We can determine it to be

$$\boldsymbol{H}_{m1} = \begin{bmatrix} 3.24227 & -0.0136501 & 47 \\ 1.09482 & 0.0815696 & 307 \\ 0.00121461 & -1.94156e-05 & 1 \end{bmatrix}$$

Step 2.2: Determine the homography between the measurement Img2_m and Img2

Let $\boldsymbol{H}_{m2}$ be the homography that maps a point from the measurement to the corresponding point in Img2. We can determine it to be

$$\boldsymbol{H}_{m2} = \begin{bmatrix} 0.350095 & -0.0565722 & 478 \\ -1.36355 & 1.68945 & 718 \\ -0.00218314 & -0.000191864 & 1 \end{bmatrix}$$

Step 2.3: Determine the homography between the measurement Img3_m and Img3

Let $\boldsymbol{H}_{m3}$ be the homography that maps a point from the measurement to the corresponding point in Img3. We can determine it to be

$$\boldsymbol{H}_{m3} = \begin{bmatrix} 37.8031 & 0.166517 & 2063 \\ 6.36207 & 13.9739 & 696 \\ 0.00791028 & -0.000189646 & 1 \end{bmatrix}$$

Step 2.4: Determine the homography between the measurement Img4_m and Img4

Let $\boldsymbol{H}_{m4}$ be the homography that maps a point from the measurement to the corresponding point in Img4. We can determine it to be

$$\boldsymbol{H}_{m4} = \begin{bmatrix} 8.37371 & -0.484625 & 549 \\ -0.770525 & 2.02643 & 420 \\ 0.000980571 & -0.00377176 & 1 \end{bmatrix}$$

Step 2.5: Determine the homography between the measurement Img5_m and Img5

Let $\boldsymbol{H}_{m5}$ be the homography that maps a point from the measurement to the corresponding point in Img5. We can determine it to be

$$\boldsymbol{H}_{m5} = \begin{bmatrix} 4.14553 & 0.431777 & 600 \\ -0.117614 & 2.07983 & 167 \\ 0.00154304 & 0.000124661 & 1 \end{bmatrix}$$

Step 3: Apply the homographies to the images

In steps 4.1, 4.2 and 4.3 we apply the inverse homographies of the homographies determined in step 2 to the given images that have projective and affine distortions. To do so, we check for the minimum and maximum x- and y-coordinates when applying the inverse homography to the image. With that information we create an empty image. For all pixels in that empty image we check if the homography maps into the given image. If yes, we find the corresponding nearest pixel in the given image and round its coordinate to the nearest pixel and copy its content into the coordinate in question from the empty image. If not, we leave the pixel in the empty image unchanged.

The image of the frame PQRS int the originally empty image should not have projective or affine distortion because we applied metric retification.

In the following substeps the results for applying the homographies are shown.

Step 3.1: Use the homography $\boldsymbol{H}_{m1}$ to apply metric rectification to Img1

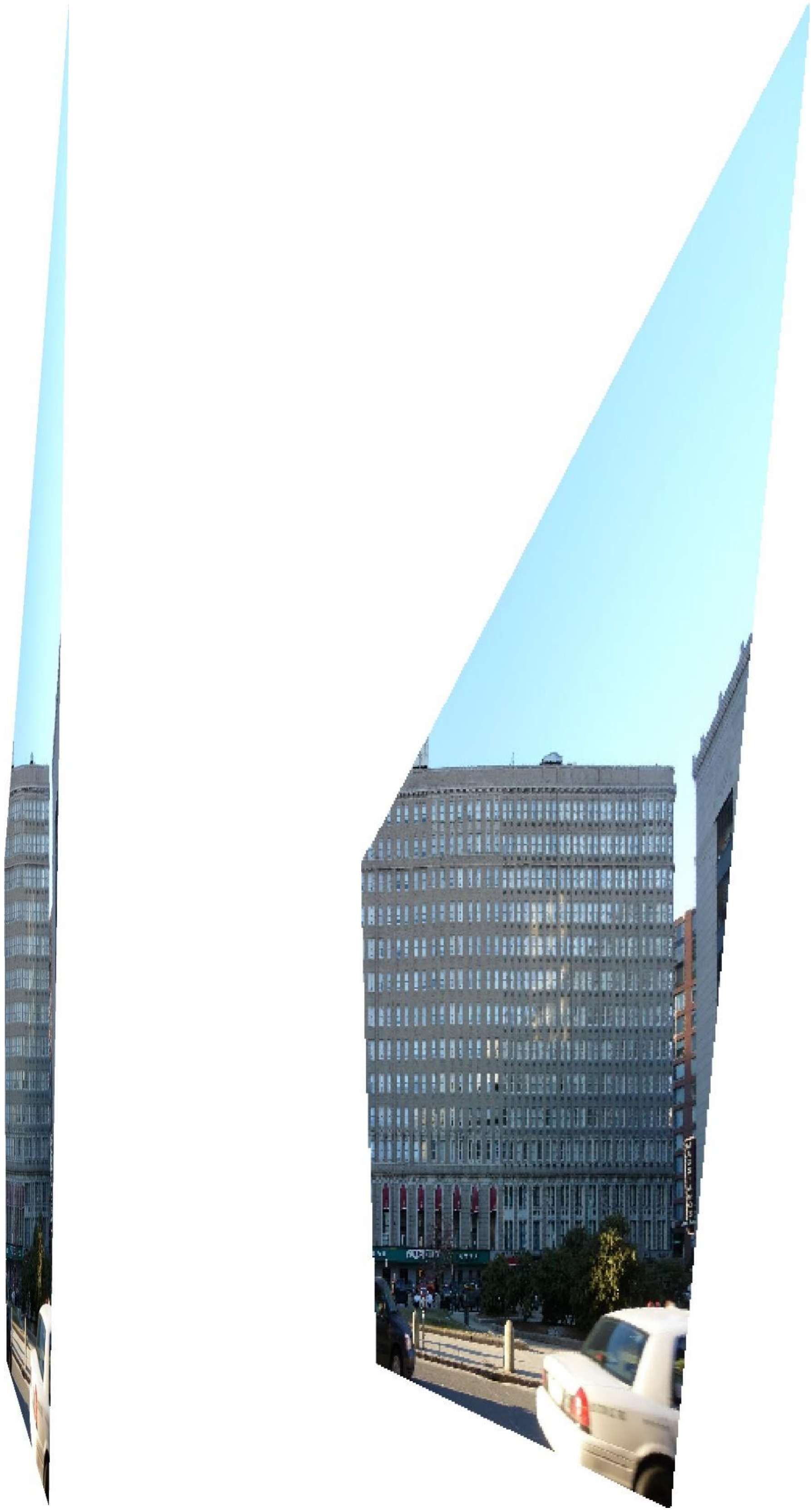


Figure 3: Img1 with metric rectification (left) and metric rectification with unrealistic coordinates (right)

Step 3.2: Use the homography $\boldsymbol{H}_{m2}$ to apply metric rectification to Img2

Figure 4: Img2 with metric rectification

Step 3.3: Use the homography $\boldsymbol{H}_{m3}$ to apply metric rectification to Img3

Figure 5: Img3 with metric rectification

Step 3.4: Use the homography $\boldsymbol{H}_{m3}$ to apply metric rectification to Img4

Figure 6: Img4 with metric rectification

Step 3.5: Use the homography $\boldsymbol{H}_{m3}$ to apply metric rectification to Img5

Figure 7: Img5 with metric rectification

Part 2: Two-step method

Step 1: Remove the projective distortion using the VL method

Let $\boldsymbol{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}$ be the vanishing line in the given image. Then, if we apply a homography that sends the vanishing line back to $\boldsymbol{l}_\infty = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ the remaining distortion will be purely affine. The homography that will do this is

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

While points are transformed by $\boldsymbol{H}$, lines are transformed by $\boldsymbol{H}^{-T}$. To get this, we need $\boldsymbol{H}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -l_1/l_3 & -l_2/l_3 & 1/l_3 \end{bmatrix}$. Then, $\boldsymbol{H}^{-T} = \begin{bmatrix} 1 & 0 & -l_1/l_3 \\ 0 & 1 & -l_2/l_3 \\ 0 & 0 & 1/l_3 \end{bmatrix}$.

Now check that $\boldsymbol{H}^{-T}$ sends the vanishing line $\boldsymbol{l}$ to $\boldsymbol{l}_\infty$.

$$\boldsymbol{H}^{-T}\boldsymbol{l} = \begin{bmatrix} 1 & 0 & -l_1/l_3 \\ 0 & 1 & -l_2/l_3 \\ 0 & 0 & 1/l_3 \end{bmatrix} \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} = \begin{pmatrix} l_1 - l_1 \\ l_2 - l_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \boldsymbol{l}_\infty$$

Hence, when applying H to all points in the image we remove the projective distortion.

We can obtain the vanishing line $\boldsymbol{l}$ doing the following steps:

- Pick two sets of parallel lines in the image. Get their vanishing points (their intersection) by calculating the cross product of the homogenous representations of the lines. We obtain two vanishing points
- Calculate the cross product of the homogenous coordinates of the vanishing points. The result is the homogenous coordinate representation of the vanishing line $\boldsymbol{l}$.

The choice of parallel lines is shown in the following images.

Figure 8: The parallel lines used to obtain the vanishing lines for Img1 (top left), Img2 (top right), Img3 (middle left), Img5 (middle right) and Img4 (bottom)

In the next substeps, the homographies to remove the projective distortions are determined using the method outlined above. The results $\boldsymbol{H}$ are reported. Furthermore, the output images we receive after applying the homographies to the images are attached.

It can be seen, that parallel lines are now parallel in the image.

Step 1.1: Use the homography $\boldsymbol{H}_{VL1}$ to remove the projective distortion from Img1.

$$\boldsymbol{H}_{VL1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.000430656 & -0.000165958 & 1 \end{bmatrix}$$

Figure 9: Img1 after removing the projective distortion

Step 1.2: Use the homography $\boldsymbol{H}_{VL2}$ to remove the projective distortion from Img2.

$$\boldsymbol{H}_{VL2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.00767975 & -0.000370726 & 1 \end{bmatrix}$$

Figure 10: Img2 after removing the projective distortion

Step 1.3: Use the homography $\boldsymbol{H}_{VL3}$ to remove the projective distortion from Img3.

$$\boldsymbol{H}_{VL3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.000206031 & -2.32576e-06 & 1 \end{bmatrix}$$

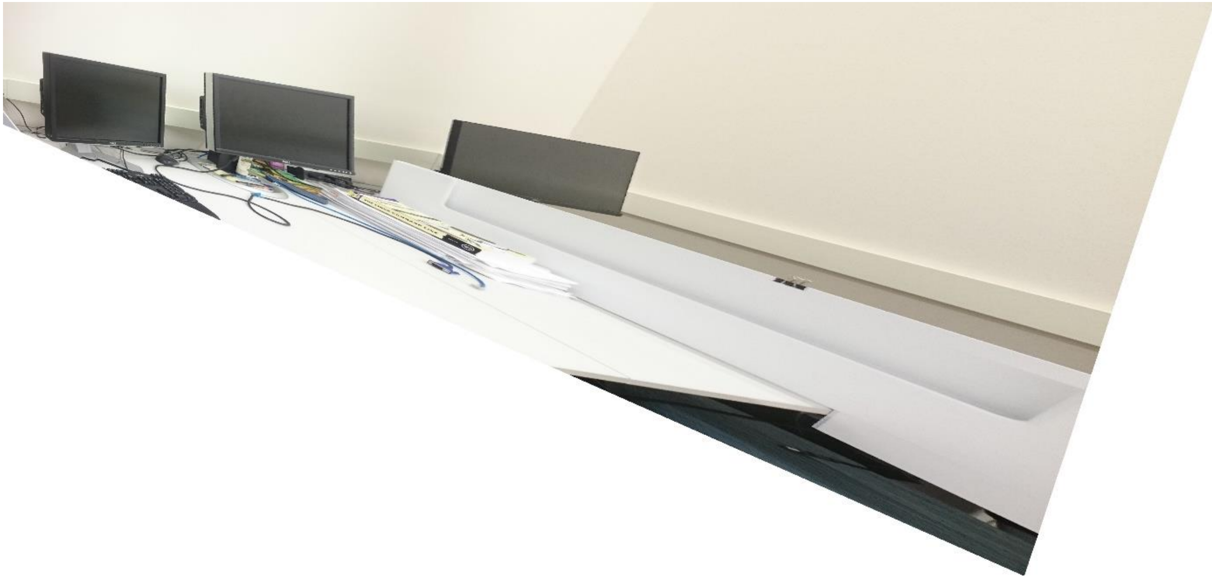


Figure 11: Img3 after removing the projective distortion

Step 1.4: Use the homography $\boldsymbol{H}_{VL4}$ to remove the projective distortion from Img4.

$$\boldsymbol{H}_{VL4} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5.53878e-05 & -0.00187453 & 1 \end{bmatrix}$$

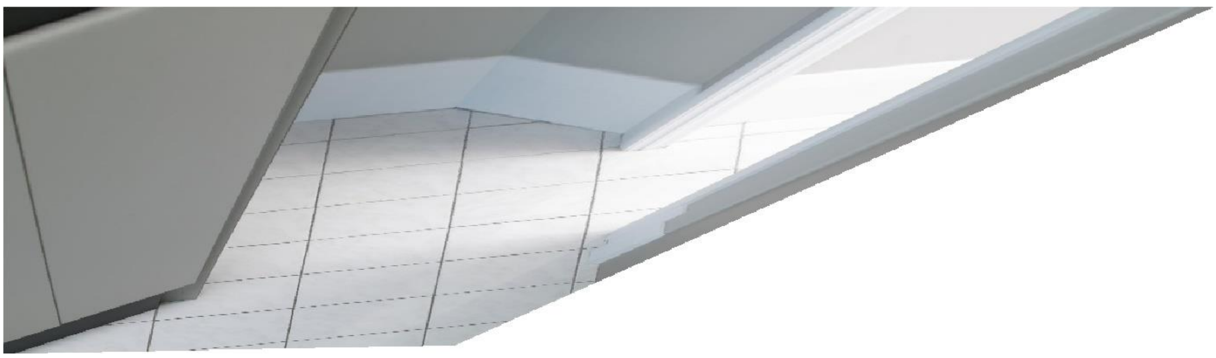


Figure 12: Img4 after removing the projective distortion

Step 1.5: Use the homography $\boldsymbol{H}_{VL5}$ to remove the projective distortion from Img5.

$$\boldsymbol{H}_{VL5} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.00037173 & -1.72341e-05 & 1 \end{bmatrix}$$

Figure 13: Img5 after removing the projective distortion

Step 2: Remove the affine distortion

Identify $\boldsymbol{l}$ and $\boldsymbol{m}$ such that they are orthogonal to each other. Then they form an angle $\theta = 90°$.

$$\cos(\theta) = \frac{\boldsymbol{l}^T \boldsymbol{C}_\infty^* \boldsymbol{m}}{\sqrt{(\boldsymbol{l}^T \boldsymbol{C}_\infty^* \boldsymbol{l})(\boldsymbol{m}^T \boldsymbol{C}_\infty^* \boldsymbol{m})}} = 0$$

Hence, we only care about the nominator and can simplify the formula for this special case to $\cos(\theta) = \boldsymbol{l}^T \boldsymbol{C}_\infty^* \boldsymbol{m} = 0$.

In the recorded image we see the lines $\boldsymbol{l}' = \boldsymbol{H}^{-T}\boldsymbol{l}$ and $\boldsymbol{m} = \boldsymbol{H}^{-T}\boldsymbol{m}$. The dual conic transforms as $\boldsymbol{C}^{*\prime} = \boldsymbol{H}\boldsymbol{C}^*\boldsymbol{H}^T$ Express the formula for $\cos(\theta)$ in terms of those.

$$\begin{aligned} 0 = \cos(\theta) &= (\boldsymbol{H}^T\boldsymbol{l}')^T(\boldsymbol{H}^{-1}\boldsymbol{C}_\infty^{*\prime}\boldsymbol{H}^{-T})(\boldsymbol{H}^T\boldsymbol{m}') \\ &= (\boldsymbol{l}'^T\boldsymbol{H})(\boldsymbol{H}^{-1}\boldsymbol{C}_\infty^{*\prime}\boldsymbol{H}^{-T})(\boldsymbol{H}^T\boldsymbol{m}') \\ &= \boldsymbol{l}'^T\boldsymbol{C}_\infty^{*\prime}\boldsymbol{m}' = \boldsymbol{l}'^T\boldsymbol{H}\boldsymbol{C}_\infty^*\boldsymbol{H}^T\boldsymbol{m}' \end{aligned}$$

We know that **H** is an affine transformation, hence $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$.

Obtain $\boldsymbol{l}' = \begin{pmatrix} l_1' \\ l_2' \\ l_3' \end{pmatrix}$ and $\boldsymbol{m}' = \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix}$ from the image. Then,

$$\begin{aligned} 0 &= (l_1' \quad l_2' \quad l_3') \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{A}^T & \boldsymbol{0} \\ \boldsymbol{t}^T & 1 \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} \\ &= (l_1' \quad l_2' \quad l_3') \begin{bmatrix} \boldsymbol{A}\boldsymbol{A}^T & \boldsymbol{0} \\ \boldsymbol{0}^T & 0 \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} \end{aligned}$$

Consider $\boldsymbol{S} = \boldsymbol{A}\boldsymbol{A}^T = \begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix}$, a symmetric matrix. Then

$$\begin{aligned} 0 &= (l_1' \quad l_2') \begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \end{pmatrix} \\ &= s_{11}l_1'm_1' + s_{12}(l_1'm_2' + l_2'm_1') + s_{22}l_2'm_2' \end{aligned}$$

We do not care about the exact ratios in **S**. Therefore, we only need to determine two of the three unknowns. We need at least two equations of this kind to do so. This means that we need two pairs of orthogonal lines $\boldsymbol{l}$ and $\boldsymbol{m}$.

$$\begin{bmatrix} l'_{a1}m_{a1}' & l_{a1}'m_{a2}' + l_{a2}'m_{a1}' & l_{a2}'m_{a2}' \\ l'_{b1}m_{b1}' & l_{b1}'m_{b2}' + l_{b2}'m_{b1}' & l_{b2}'m_{b2}' \end{bmatrix} \begin{pmatrix} s_{11} \\ s_{12} \\ s_{22} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Set $s_{22} = 1$. Then,

$$\begin{bmatrix} l'_{a1}m_{a1}' & l_{a1}'m_{a2}' + l_{a2}'m_{a1}' \\ l'_{b1}m_{b1}' & l_{b1}'m_{b2}' + l_{b2}'m_{b1}' \end{bmatrix} \begin{pmatrix} s_{11} \\ s_{12} \end{pmatrix} = \begin{pmatrix} -l_{a2}'m_{a2}' \\ -l_{b2}'m_{b2}' \end{pmatrix}$$

We are searching for $\boldsymbol{A}$, non-singular. Assume that $\boldsymbol{A}$ is positive-definite. Then, we can use the singular value decomposition (SVD) to obtain **A** from **S**.

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T$$

$$\boldsymbol{S} = \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^T = \boldsymbol{V}\begin{bmatrix} \lambda_1^2 & 0 \\ 0 & \lambda_2^2 \end{bmatrix}\boldsymbol{V}^T$$

We can read $\boldsymbol{V}$ directly from the SVD and can obtain $\boldsymbol{D} = \begin{bmatrix} \sqrt{\lambda_1^2} & 0 \\ 0 & \sqrt{\lambda_2^2} \end{bmatrix}$.

$$\boldsymbol{C}_\infty^{*\prime} = \boldsymbol{H}\boldsymbol{C}_\infty^{*}\boldsymbol{H}^T = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix}\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0}^T & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{A}^T & \boldsymbol{0} \\ \boldsymbol{t}^T & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}\boldsymbol{A}^T & \boldsymbol{0} \\ \boldsymbol{0}^T & 0 \end{bmatrix}$$

We know that $\boldsymbol{t} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ because we have already removed the projective distortion.

Hence, $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$.

In the next substeps, the homographies to remove the affine distortions are determined using the method outlined above. The results $\boldsymbol{H}$ are reported.

Then we calculate the homography that will remove the projective and affine distortions from our input image. Those resulting homographies $\boldsymbol{H}_{res}$ are reported as well.

Furthermore, the output images we receive after applying the homographies to the images are reported.

It can be seen that parallel lines are now parallel in the image and 90° angles are 90° angles in the image.

Step 2.1: Use the homography $\boldsymbol{H}_{Aff1}$ to remove the affine distortion from Img1 after removing the projective distortions. Use $\boldsymbol{H}_{res1}$ to remove the affine and projective distortion at the same time.

$$\boldsymbol{H}_{Aff1} = \begin{bmatrix} 2.96289 & 0.290226 & 0 \\ 0.290226 & 1.03321 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_{res1} = \begin{bmatrix} 0.347058 & -0.0974873 & 0 \\ -0.0974873 & 0.995237 & 0 \\ 0.000165641 & -0.000207151 & 1 \end{bmatrix}$$

Figure 14: Img1 after removing the projective and affine distortions

Step 2.2: Use the homography $\boldsymbol{H}_{Aff2}$ to remove the affine distortion from Img2 after removing the projective distortions. Use $\boldsymbol{H}_{res2}$ to remove the affine and projective distortion at the same time.

$$\boldsymbol{H}_{Aff2} = \begin{bmatrix} 10.1121 & 0.85296 & 0 \\ 0.85296 & 1.0755 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_{res2} = \begin{bmatrix} 0.105981 & -0.0840516 & 0 \\ -0.0840516 & 0.996461 & 0 \\ -0.000782745 & 0.000276081 & 1 \end{bmatrix}$$

Figure 15: Img2 after removing the projective and affine distortions

Step 2.3: Use the homography $\boldsymbol{H}_{Aff3}$ to remove the affine distortion from Img3 after removing the projective distortions. Use $\boldsymbol{H}_{res3}$ to remove the affine and projective distortion at the same time.

$$\boldsymbol{H}_{Aff3} = \begin{bmatrix} 8.05882 & -0.466874 & 0 \\ -0.466874 & 1.02866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_{res3} = \begin{bmatrix} 0.127439 & 0.0578402 & 0 \\ 0.0578402 & 0.998392 & 0 \\ 2.61218e-05 & 9.59486e-06 & 1 \end{bmatrix}$$

Figure 16: Img3 after removing the projective and affine distortions

Step 2.4: Use the homography $\boldsymbol{H}_{Aff4}$ to remove the affine distortion from Img4 after removing the projective distortions. Use $\boldsymbol{H}_{res4}$ to remove the affine and projective distortion at the same time.

$$\boldsymbol{H}_{Aff4} = \begin{bmatrix} 3.62339 & 0.573721 & 0 \\ 0.573721 & 1.10008 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_{res4} = \begin{bmatrix} 0.300826 & -0.156889 & 0 \\ -0.156889 & 0.990844 & 0 \\ 0.00027743 & -0.00184868 & 1 \end{bmatrix}$$

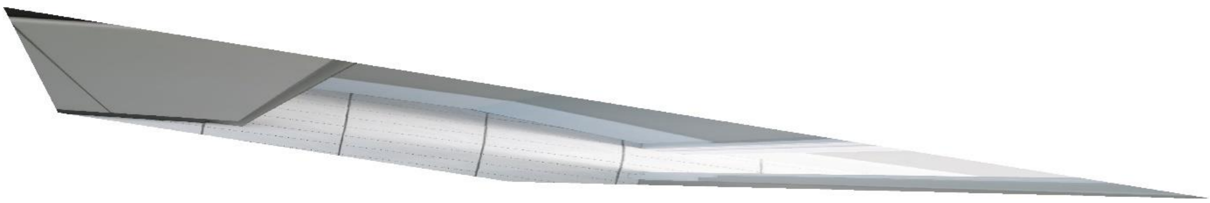


Figure 17: Img4 after removing the projective and affine distortions

Step 2.5: Use the homography $\boldsymbol{H}_{Aff5}$ to remove the affine distortion from Img5 after removing the projective distortions. Use $\boldsymbol{H}_{res5}$ to remove the affine and projective distortion at the same time.

$$\boldsymbol{H}_{Aff5} = \begin{bmatrix} 5.09836 & -0.829504 & 0 \\ -0.829504 & 1.14669 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_{res5} = \begin{bmatrix} 0.222306 & 0.160814 & 0 \\ 0.160814 & 1 & 0 \\ 7.98663e-05 & 4.27451e-05 & 1 \end{bmatrix}$$

Figure 18: Img5 after removing the projective and affine distortions

Part 3: One-step method

Let $\boldsymbol{C}_\infty^{*\prime} = \boldsymbol{H}\boldsymbol{C}_\infty^{*}\boldsymbol{H}^T$ be a projection of $\boldsymbol{C}_\infty^{*}$.

$\boldsymbol{H}$ can be written as $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{v}^T & 1 \end{bmatrix}$. Then

$$\boldsymbol{C}_\infty^{*\prime} = \begin{bmatrix} \boldsymbol{A}\boldsymbol{A}^T & \boldsymbol{A}\boldsymbol{v} \\ \boldsymbol{v}^T\boldsymbol{A}^T & \boldsymbol{v}^T\boldsymbol{v} \end{bmatrix} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Identify lines $\boldsymbol{l}' = \begin{pmatrix} {l_1}' \\ {l_2}' \\ 1 \end{pmatrix}$ and $\boldsymbol{m}' = \begin{pmatrix} {m_1}' \\ {m_2}' \\ 1 \end{pmatrix}$ from the image whose corresponding lines $\boldsymbol{l}$ and $\boldsymbol{m}$ are orthogonal in the world coordinates. Note: we can always find such coordinates for the line by dividing the vector by the last coordinate.

$$\begin{aligned}
0 &= \boldsymbol{l}'^T \boldsymbol{C}_\infty^{*\prime} \boldsymbol{m}' \\
&= ({l_1}' \quad {l_2}' \quad 1) \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{pmatrix} {m_1}' \\ {m_2}' \\ 1 \end{pmatrix} \\
&= ({l_1}' \quad {l_2}' \quad 1) \begin{pmatrix} a \cdot m_1' + b/2 \cdot m_2' + d/2 \\ b/2 \cdot m_1' + c \cdot m_2' + e/2 \\ d/2 \cdot m_1' + e/2 \cdot m_2' + f \end{pmatrix} \\
&= a \cdot m_1' l_1' + b/2 \cdot (m_2' l_1' + m_1' l_2') + c \cdot m_2' l_2' + d/2 \cdot (l_1' + m_1') + e/2 \cdot (l_2' \\
&\qquad + m_2') + f
\end{aligned}$$

$$= (m_1' l_1' \quad m_2' l_1' + m_1' l_2' \quad m_2' l_2' \quad l_1' + m_1' \quad l_2' + m_2' \quad 1) \begin{pmatrix} a \\ b/2 \\ c \\ d/2 \\ e/2 \\ f \end{pmatrix}$$

We only care about the ratios. Therefore, we can choose f = 1. Then,

$$-1 = (m_1' l_1' \quad m_2' l_1' + m_1' l_2' \quad m_2' l_2' \quad l_1' + m_1' \quad l_2' + m_2') \begin{pmatrix} a \\ b/2 \\ c \\ d/2 \\ e/2 \end{pmatrix}$$

We have 5 unknowns. Therefore, we have to find (at least) 5 combinations of $\boldsymbol{l}'$ and $\boldsymbol{m}'$.

With those we can then solve the linear system and obtain $\boldsymbol{C}_{\infty}^{*\prime}$. Then we can solve

$$\boldsymbol{A}\boldsymbol{A}^{T} = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

With the SVD as described in part 2. Then we solve for $\boldsymbol{v}$ using

$$\boldsymbol{A}\boldsymbol{v} = \begin{bmatrix} d/2 \\ e/2 \end{bmatrix}$$

$$\boldsymbol{v} = \boldsymbol{A}^{-1} \begin{bmatrix} d/2 \\ e/2 \end{bmatrix}$$

We need additional points to calculate orthogonal lines. Pick the points marked in the images at the end of the report (before the code).

In the next substeps, the homographies to remove the affine and projective distortions at once are determined using the method outlined above. The results $\boldsymbol{H}$ are reported.

Furthermore, the output images we receive after applying the homographies to the images are reported.

It can be seen that parallel lines are now parallel in the image and 90° angles are 90° angles in the image.

Step 1.1: Use the homography $\boldsymbol{H}_{One1}$ to remove the projective distortion from Img1.

$$\boldsymbol{H}_{One1} = \begin{bmatrix} 0.601458 & -0.457102 & 0 \\ -0.457102 & 0.889414 & 0 \\ -0.00189836 & -0.000597674 & 1 \end{bmatrix}$$

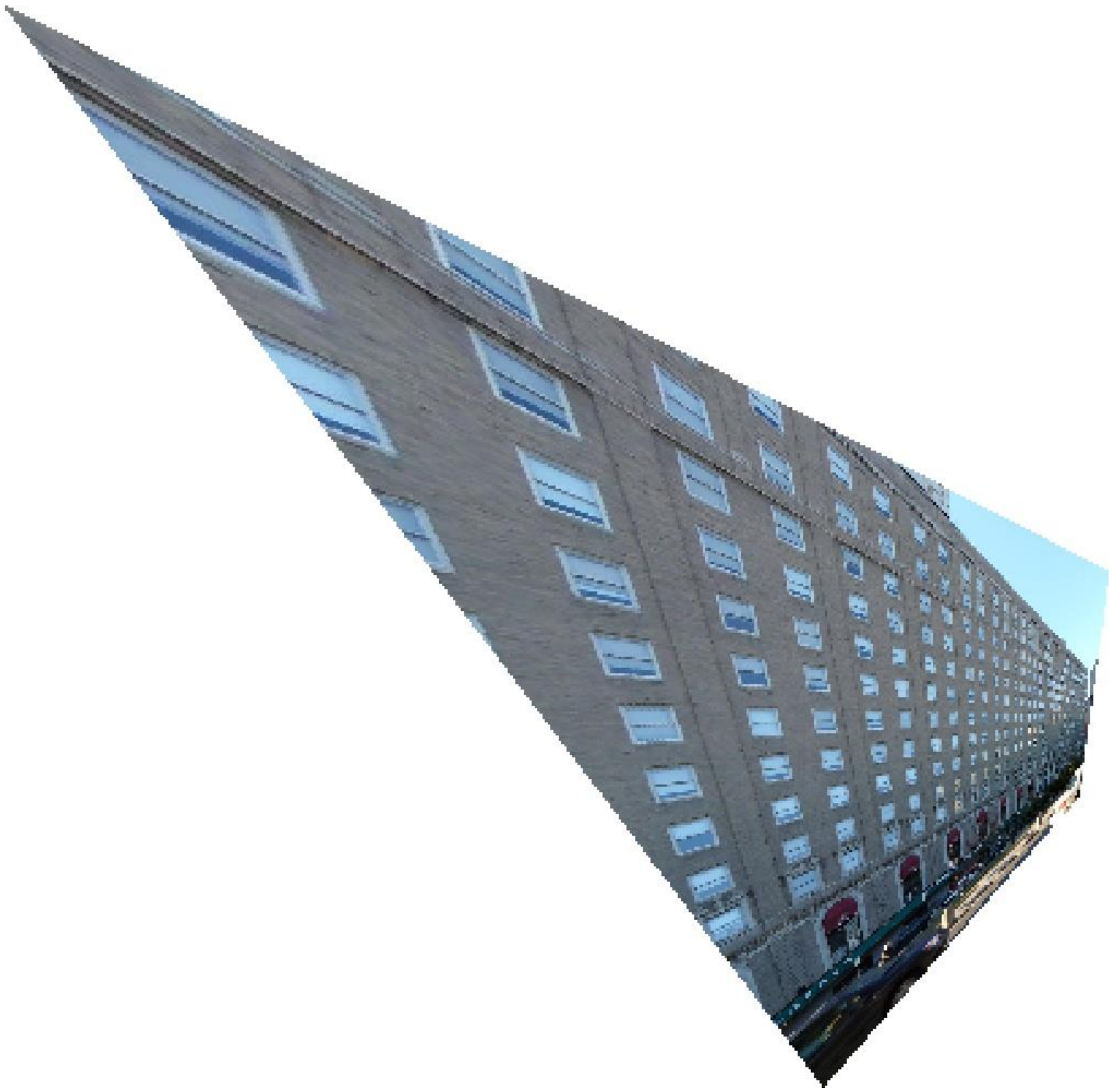


Figure 19: Img1 after removing the projective and affine distortion in one step

Step 1.2: Use the homography $\boldsymbol{H}_{One2}$ to remove the projective distortion from Img2.

$$\boldsymbol{H}_{One2} = \begin{bmatrix} 0.332879 & -0.23387 & 0 \\ -0.23387 & 3.88673 & 0 \\ -0.00534679 & 0.0033178 & 1 \end{bmatrix}$$

Figure 20: Img2 after removing the projective and affine distortion in one step

Step 1.3: Use the homography $\boldsymbol{H}_{One3}$ to remove the projective distortion from Img3.

$$\boldsymbol{H}_{One3} = \begin{bmatrix} 0.998392 & 0.0884925 & 0 \\ 0.0884925 & 0.365611 & 0 \\ 0.000202581 & 3.86118e-05 & 1 \end{bmatrix}$$

Figure 21: Img3 after removing the projective and affine distortion in one step

Step 1.4: Use the homography $\boldsymbol{H}_{One4}$ to remove the projective distortion from Img4.

$$\boldsymbol{H}_{One4} = \begin{bmatrix} 1.04566 & 0.00268478 & 0 \\ 0.00268478 & 1.0046 & 0 \\ 0.000306621 & -0.00185924 & 1 \end{bmatrix}$$

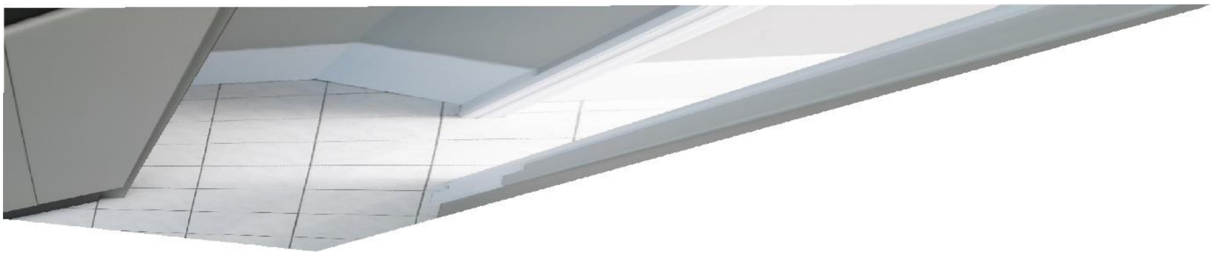


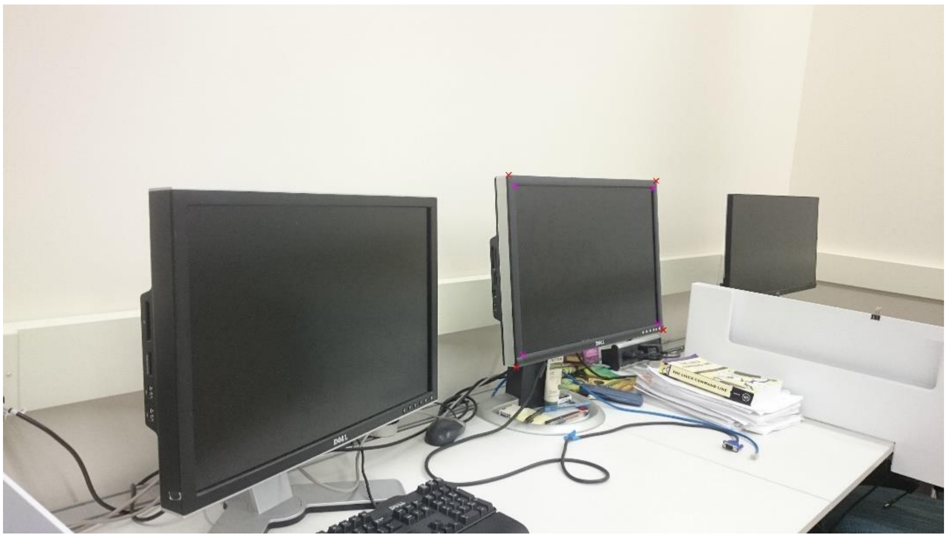
Figure 22: Img4 after removing the projective and affine distortion in one step

Step 1.5: Use the homography $\boldsymbol{H}_{One5}$ to remove the projective distortion from Img5.

$$\boldsymbol{H}_{One5} = \begin{bmatrix} 0.99925 & 0.0387272 & 0 \\ 0.0387272 & 0.666367 & 0 \\ 0.000370784 & 2.91183e-06 & 1 \end{bmatrix}$$

Figure 23: Img5 after removing the projective and affine distortion in one step

Points chosen for the One-step approach:



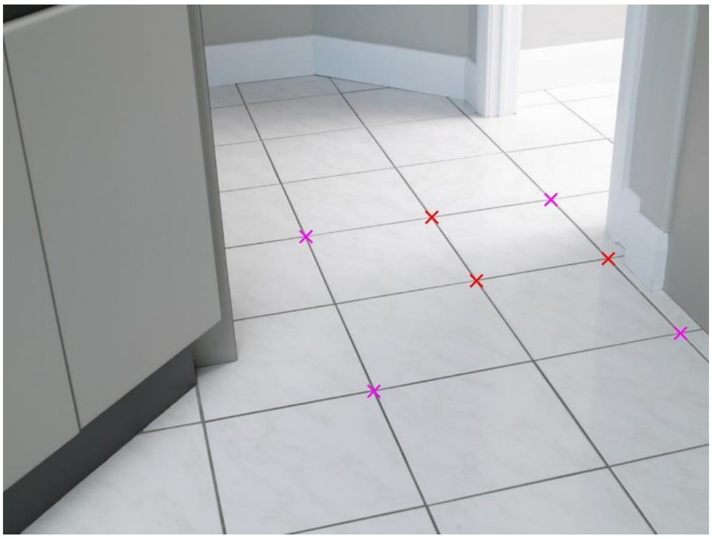


Figure 24: Points chosen to use for transformation. Img1 to Img5 from top to bottom.

Python Code:

```
import numpy as np

import cv2

import copy


def draw_corner_markers_for_Task1(coordinates_1a,coordinates_1b,coordinates_1c, coordinates_1d, coordinates_1e):

    img_1a = cv2.imread("./hw3_Task1_Images/Images/Img1.jpg")

    filename_markers_1a = "./hw3_Task1_Images/Images/with_markers/Img1_markers.jpeg"

    draw_PQRS_on_image(img_1a, coordinates_1a, filename_markers_1a, 40, 4)


    img_1b = cv2.imread("./hw3_Task1_Images/Images/Img2.jpeg")

    filename_markers_1b = "./hw3_Task1_Images/Images/with_markers/Img2_markers.jpeg"

    draw_PQRS_on_image(img_1b, coordinates_1b, filename_markers_1b, 40, 4)


    img_1c = cv2.imread("./hw3_Task1_Images/Images/Img3.jpg")

    filename_markers_1c = "./hw3_Task1_Images/Images/with_markers/Img3_markers.jpeg"

    draw_PQRS_on_image(img_1c, coordinates_1c, filename_markers_1c, 60, 6)


    img_1d = cv2.imread("./hw3_Task1_Images/Images/Img4.jpeg")

    filename_markers_1d = "./hw3_Task1_Images/Images/with_markers/Img4_markers.jpeg"

    draw_PQRS_on_image(img_1d, coordinates_1d, filename_markers_1d, 60, 6)


    img_1e = cv2.imread("./hw3_Task1_Images/Images/Img5.jpg")

    filename_markers_1e = "./hw3_Task1_Images/Images/with_markers/Img5_markers.jpeg"

    draw_PQRS_on_image(img_1e, coordinates_1e, filename_markers_1e, 60, 6)



def draw_lines_connecting_PQRS_on_image(img, filename, line_PQ, line_QR, line_RS, line_PS):

    color = [(255,0,0), (0,255,0),(255,0,0), (0,255,0)]

    i = 0

    for vanishing_line in [line_PQ, line_QR, line_RS, line_PS]:

        x_low = -10

        x_high = np.shape(img)[1] + 10
```

```
        y_low = np.round((vanishing_line[0]*x_low+vanishing_line[2])/(-vanishing_line[1])).astype(int)
        y_high = np.round((vanishing_line[0]*x_high+vanishing_line[2])/(-vanishing_line[1])).astype(int)
        cv2.line(img, (x_low,y_low),(x_high,y_high), color[i],5)
        i= i+1
    cv2.imwrite(filename, img)

def draw_vanishing_lines_for_Task1(vanishing_line_Img1,  vanishing_line_Img2,   vanishing_line_Img3):
    img_1a = cv2.imread("./hw3_Task1_Images/Images/Img1.jpg")
    filename_markers_1a = "./hw3_Task1_Images/Images/with_vanishing_line/Img1_line.jpeg"
    draw_vanishing_lines_on_image(img_1a, vanishing_line_Img1,   filename_markers_1a,3)

    img_1b = cv2.imread("./hw3_Task1_Images/Images/Img2.jpeg")
    filename_markers_1b = "./hw3_Task1_Images/Images/with_vanishing_line/Img2_line.jpeg"
    draw_vanishing_lines_on_image(img_1b, vanishing_line_Img2,  filename_markers_1b,3)

    img_1c = cv2.imread("./hw3_Task1_Images/Images/Img3.jpg")
    filename_markers_1c = "./hw3_Task1_Images/Images/with_vanishing_line/Img3_line.jpeg"
    draw_vanishing_lines_on_image(img_1c, vanishing_line_Img3,  filename_markers_1c,3)


def draw_PQRS_on_image(img,image_coordinates, filename, markersize, marker_thickness):
    colours = [(0,0,250), (255,0,255), (0,255,0) ,(300,300,0)]
    for i in range(4):
        cv2.drawMarker(img,(image_coordinates[i,0],image_coordinates[i,1]), colours[i],1, markersize,
marker_thickness)
    cv2.imwrite(filename, img)

def draw_vanishing_lines_on_image(img, vanishing_line, filename, marker_thickness):
    x_low = -10
    x_high = np.shape(img)[1] + 10
    y_low = np.round((vanishing_line[0]*x_low+vanishing_line[2])/(-vanishing_line[1])).astype(int)
    y_high = np.round((vanishing_line[0]*x_high+vanishing_line[2])/(-vanishing_line[1])).astype(int)
    cv2.line(img, (x_low,y_low),(x_high,y_high), (255,255,255),5)
    cv2.imwrite(filename, img)
```

```
def calculate_lines_between_points(point_coordinates_of_line):
    line_PS = np.cross(point_coordinates_of_line[0], point_coordinates_of_line[3])
    line_PS = line_PS/line_PS[2]
    line_QR = np.cross(point_coordinates_of_line[1], point_coordinates_of_line[2])
    line_QR = line_QR/line_QR[2]
    line_PQ = np.cross(point_coordinates_of_line[0], point_coordinates_of_line[1])
    line_PQ = line_PQ/line_PQ[2]
    line_SR = np.cross(point_coordinates_of_line[3], point_coordinates_of_line[2])
    line_SR = line_SR/line_SR[2]
    return line_PS, line_QR, line_PQ, line_SR


def calculate_diagonal_lines(point_coordinates_of_line):
    line_PR = np.cross(point_coordinates_of_line[0], point_coordinates_of_line[2])
    line_PR = line_PR/line_PR[2]
    line_QS = np.cross(point_coordinates_of_line[1], point_coordinates_of_line[3])
    line_QS = line_QS/line_QS[2]
    return line_PR, line_QS


def calculate_line(point_coordinates_of_line):
    line_PS, line_QR, line_PQ, line_SR = calculate_lines_between_points(point_coordinates_of_line)
    vp_1 = np.cross(line_PS, line_QR)
    vp_2 = np.cross(line_PQ, line_SR)
    vp_1 = vp_1/vp_1[2]
    vp_2 = vp_2/vp_2[2]
    van_line = np.cross(vp_1, vp_2)
    van_line = van_line/van_line[2]
    return van_line, vp_1, vp_2, line_PS, line_QR, line_SR, line_PQ


def bilinear_interpolation(image, coordinates):
```

```
    # get the values of the closest pixels to the coordinates
    pixel_1 = image[np.int(np.floor(coordinates[0]))][np.int(np.floor(coordinates[1]))]
    pixel_2 = image[np.int(np.floor(coordinates[0]))][np.int(np.ceil(coordinates[1]))]
    pixel_3 = image[np.int(np.ceil(coordinates[0]))][np.int(np.floor(coordinates[1]))]
    pixel_4 = image[np.int(np.ceil(coordinates[0]))][np.int(np.ceil(coordinates[1]))]

    dx = coordinates[1]-np.floor(coordinates[1])
    dy = coordinates[0]-np.floor(coordinates[0])

    weight_pixel_1 = 1/np.linalg.norm([dx,dy])
    weight_pixel_2 = 1/np.linalg.norm([1-dx,dy])
    weight_pixel_3 = 1/np.linalg.norm([dx,1-dy])
    weight_pixel_4 = 1/np.linalg.norm([1-dx,1-dy])

    nom = weight_pixel_1*pixel_1 + weight_pixel_2*pixel_2 + weight_pixel_3*pixel_3 + weight_pixel_4*pixel_4
    denom = weight_pixel_1 + weight_pixel_2 + weight_pixel_3 + weight_pixel_4
    res_pixel_val = nom/denom
    return res_pixel_val


def calculate_corners_without_projective_distortion(projective_homography,
coordinates_points_before_transform):
    homography_inverse = np.linalg.inv(projective_homography)
    new_homogenous_coord = homography_inverse.dot(np.transpose(coordinates_points_before_transform))
    new_homogenous_coord = new_homogenous_coord/new_homogenous_coord[2]
    new_homogenous_coord = np.transpose(new_homogenous_coord)
    return new_homogenous_coord


def homography_remove_projective_distortion(vanishing_line_coordinates):
    H = np.array([[1,0,0],[0,1,0],
[vanishing_line_coordinates[0],vanishing_line_coordinates[1],vanishing_line_coordinates[2]]])
    H = np.linalg.inv(H)
```

```
    return H


def homography_remove_affine_distortion(line_PS, line_QR, line_RS, line_PQ):
    W = np.array([[line_PS[0]*line_PQ[0], line_PS[0]*line_PQ[1]+ line_PS[1]*line_PQ[0]],\
               [line_QR[0]*line_PQ[0], line_QR[0]*line_PQ[1]+ line_QR[1]*line_PQ[0]],\
               [line_PS[0]*line_RS[0], line_PS[0]*line_RS[1]+ line_PS[1]*line_RS[0]],\
               [line_QR[0]*line_RS[0], line_QR[0]*line_RS[1]+ line_QR[1]*line_RS[0]]], dtype = np.float64)


    b = np.array([-line_PS[1]*line_PQ[1],-line_QR[1]*line_PQ[1],-line_PS[1]*line_RS[1],-line_QR[1]*line_RS[1]],
dtype = np.float64)
    s = np.linalg.lstsq(W,b)[0]
    s = np.append(s,(s[1],1))
    S = np.reshape(s, (2,2))
    u , d_square, v = np.linalg.svd(S)
    d = np.sqrt(d_square)
    D = np.diag(d)
    A = np.dot(np.dot(u,D),np.transpose(u))
    H = np.append(A[0],(0,A[1][0], A[1][1],0,0,0,1))
    H = np.reshape(H,(3,3))
    H = np.linalg.inv(H)
    return H



def homography_remove_distortions(line_PS, line_QR, line_RS, line_PQ, line_PS_add, line_QR_add,
line_RS_add, line_PQ_add):
    l1 = line_PS
    m1 = line_PQ


    l2 = line_PQ
    m2 = line_QR


    l3 = line_QR
    m3 = line_RS
```

```
    l4 = line_RS
    m4 = line_PS



    l5 = line_PS_add
    m5 = line_PQ_add


    ta =[]
    tb = []


    ta.append([l1[0]*m1[0], l1[0]*m1[1]+l1[1]*m1[0], l1[1]*m1[1], l1[0]*m1[2]+l1[2]*m1[0],
l1[1]*m1[2]+l1[2]*m1[1]])
    tb.append([-l1[2]*m1[2]])


    ta.append([l2[0]*m2[0], l2[0]*m2[1]+l2[1]*m2[0], l2[1]*m2[1], l2[0]*m2[2]+l2[2]*m2[0],
l2[1]*m2[2]+l2[2]*m2[1]])
    tb.append([-l2[2]*m2[2]])


    ta.append([l3[0]*m3[0], l3[0]*m3[1]+l3[1]*m3[0], l3[1]*m3[1], l3[0]*m3[2]+l3[2]*m3[0],
l3[1]*m3[2]+l3[2]*m3[1]])
    tb.append([-l3[2]*m3[2]])


    ta.append([l4[0]*m4[0], l4[0]*m4[1]+l4[1]*m4[0], l4[1]*m4[1], l4[0]*m4[2]+l4[2]*m4[0],
l4[1]*m4[2]+l4[2]*m4[1]])
    tb.append([-l4[2]*m4[2]])


    ta.append([l5[0]*m5[0], l5[0]*m5[1]+l5[1]*m5[0], l5[1]*m5[1], l5[0]*m5[2]+l5[2]*m5[0],
l5[1]*m5[2]+l5[2]*m5[1]])
    tb.append([-l5[2]*m5[2]])


    A = np.asarray(ta)
    b = np.asanyarray(tb)


    x = np.dot(np.linalg.pinv(A),b)
    x = x/np.max(x)
```

```
    S = np.append(x[0], (x[1], x[1], x[2]))
    S = np.reshape(S,(2,2))

    u, d, vh = np.linalg.svd(S)
    D = np.sqrt(np.diag(d))

    A= np.dot(np.dot(u,D),u.transpose())

    d = np.array([x[3], x[4]])

    v = np.dot(np.linalg.pinv(A),d)

    H = np.append(A[0],(0,A[1][0],A[1][1],0,v[0][0],v[1][0],1))
    H = np.reshape(H,(3,3))
    return H


def calculate_Homography(coordinates_A, coordinates_B):
    """
    This function calculates the homography H from an images A to an image B.

    The function is passed the coordinates of the reference points for A and B.
    It returns the 3x3 homography matrix H.
    """
    # coordinates in A
    x_p = coordinates_A[0,0]
    y_p = coordinates_A[0,1]
    x_q = coordinates_A[1,0]
    y_q = coordinates_A[1,1]
    x_r = coordinates_A[2,0]
    y_r = coordinates_A[2,1]
    x_s = coordinates_A[3,0]
    y_s = coordinates_A[3,1]
```

```
    # corresponding coordinates in B
    x_p_prime = coordinates_B[0,0]
    y_p_prime = coordinates_B[0,1]
    x_q_prime = coordinates_B[1,0]
    y_q_prime = coordinates_B[1,1]
    x_r_prime = coordinates_B[2,0]
    y_r_prime = coordinates_B[2,1]
    x_s_prime = coordinates_B[3,0]
    y_s_prime = coordinates_B[3,1]


    A = np.array([[x_p, y_p, 1, 0,0,0, -x_p*x_p_prime, -y_p*x_p_prime],\
                  [0,0,0,x_p, y_p, 1,-x_p*y_p_prime, -y_p*y_p_prime],\
                  [x_q, y_q, 1, 0,0,0, -x_q*x_q_prime, -y_q*x_q_prime],\
                  [0,0,0,x_q, y_q, 1,-x_q*y_q_prime, -y_q*y_q_prime],\
                  [x_r, y_r, 1, 0,0,0, -x_r*x_r_prime, -y_r*x_r_prime],\
                  [0,0,0,x_r, y_r, 1,-x_r*y_r_prime, -y_r*y_r_prime],\
                  [x_s, y_s, 1, 0,0,0, -x_s*x_s_prime, -y_s*x_s_prime],\
                  [0,0,0,x_s, y_s, 1,-x_s*y_s_prime, -y_s*y_s_prime] ])
    b = np.array([x_p_prime, y_p_prime, x_q_prime, y_q_prime,x_r_prime, y_r_prime,x_s_prime, y_s_prime])

    A_inverse = np.linalg.inv(A)
    h = A_inverse.dot(b)
    h = np.append(h,1)
    H = np.reshape(h, (3,3))
    return H


def apply_homography(source_image, homography, output_filename, change_size, fixed_height = 2000):
    start_image = cv2.imread(source_image)
    start_image = np.transpose(start_image, (1,0,2))

    # Determine the required size of the new frame
    x_shape = start_image.shape[0]
```

```
y_shape = start_image.shape[1]

grid = np.array(np.meshgrid(np.arange(x_shape), np.arange(y_shape),np.arange(1,2)))

combinations = np.transpose(grid.T.reshape(-1, 3))


homography_inverse = np.linalg.inv(homography)

new_homogenous_coord = homography_inverse.dot(combinations)

new_homogenous_coord = new_homogenous_coord/new_homogenous_coord[2]


min_values = np.amin(new_homogenous_coord, axis = 1)

min_values = np.round(min_values).astype(int)

max_values = np.amax(new_homogenous_coord, axis = 1)

max_values = np.round(max_values).astype(int)

required_image_size = max_values - min_values

required_image_width = int(required_image_size[0])

required_image_height = int(required_image_size[1])


if change_size == True:
   aspect_ratio = required_image_width/ required_image_height
   scaled_height = fixed_height
   scaled_width = scaled_height * aspect_ratio
   scaled_width = int(round(scaled_width))
   s = required_image_height/scaled_height
   print(s)
else:
   scaled_width = required_image_width
   scaled_height = required_image_height
   s = 1



# Initialize the new frame

img_replication = np.ones((scaled_width,scaled_height,3),np.uint8)*255


# Look at all pixels in the new frame: does the homography map them into the original image frame?

x_values_in_target = np.arange(scaled_width)
```

```
y_values_in_target = np.arange(scaled_height)

target_grid = np.array(np.meshgrid(x_values_in_target, y_values_in_target, np.arange(1,2)))
combinations_in_target = np.transpose(target_grid.T.reshape(-1, 3))
target_combinations_wo_offset = copy.copy(combinations_in_target)
combinations_in_target[0] = combinations_in_target[0]*s + min_values[0]
combinations_in_target[1] = combinations_in_target[1]*s + min_values[1]

target_new_homogenous_coord = homography.dot(combinations_in_target)
target_new_homogenous_coord = target_new_homogenous_coord/target_new_homogenous_coord[2]

x_coordinates_rounded = np.round(target_new_homogenous_coord[0]).astype(int)
y_coordinates_rounded = np.round(target_new_homogenous_coord[1]).astype(int)
# Check conditions on x
x_coordinates_greater_zero = x_coordinates_rounded > 0
x_coordinates_smaller_x_shape = x_coordinates_rounded < x_shape
x_coordinates_valid = x_coordinates_greater_zero*x_coordinates_smaller_x_shape
# Check conditions on y
y_coordinates_greater_zero = y_coordinates_rounded > 0
y_coordinates_smaller_y_shape = y_coordinates_rounded < y_shape
y_coordinates_valid = y_coordinates_greater_zero*y_coordinates_smaller_y_shape

valid_coordinates = x_coordinates_valid*y_coordinates_valid

target_valid_x = target_combinations_wo_offset[0][valid_coordinates == True]
target_valid_y = target_combinations_wo_offset[1][valid_coordinates == True]

list_of_x_values_target = list(target_valid_x)
list_of_y_values_target = list(target_valid_y)
list_of_valid_coordinate_pairs = list()

for i in range(len(list_of_x_values_target)):
    list_of_valid_coordinate_pairs.append([list_of_x_values_target[i], list_of_y_values_target[i]])
valid_x = x_coordinates_rounded[valid_coordinates == True]
```

```
    valid_y = y_coordinates_rounded[valid_coordinates == True]

    list_of_x_values = list(valid_x)
    list_of_y_values = list(valid_y)

    list_of_original_coords_for_mapping = list()
    for i in range(len(list_of_x_values)):
        list_of_original_coords_for_mapping.append([list_of_x_values[i], list_of_y_values[i]])

    j = 0
    for pair in list_of_valid_coordinate_pairs:
        img_replication[pair[0], pair[1], : ] =
start_image[list_of_original_coords_for_mapping[j][0],list_of_original_coords_for_mapping[j][1], :]
        j = j+1

    img_replication = np.transpose(img_replication, (1,0,2))
    cv2.imwrite(output_filename, img_replication)




def run_code_for_task1_Part1():
    # Use the pixel coordinates corresponding to the measurements as the range
    coordinates_1a =  np.array([[47,307,1],[8,575,1],[1011,698,1],[1000,523,1]])
    coordinates_1b = np.array([[478,718],[481,874],[607,923],[601,736]])
    coordinates_1c = np.array([[2063,696],[2094,1480],[2694,1329],[2665,720]])
    coordinates_1d = np.array([[549,420],[672,700],[1228,595],[993,353]])
    coordinates_1e = np.array([[600,167],[656,490],[968,403],[926,129]])


    # Use the given measurements in the original scene as the domain
    coordinates_1a_target_small = np.array([[0,0],[0,75],[85,75],[85,0]])
    coordinates_1a_target = np.array([[0,0],[0,2890],[470,2890],[470,0]])
    coordinates_1b_target = np.array([[0,0],[0,84],[74,84],[74,0]])
```

```
    coordinates_1c_target = np.array([[0,0],[0,55],[36,55],[36,0]])
    coordinates_1d_target = np.array([[0,0],[0,60],[60,60],[60,0]])
    coordinates_1e_target = np.array([[0,0],[0,160],[120,160],[120,0]])


    draw_corner_markers_for_Task1(coordinates_1a,  coordinates_1b, coordinates_1c, coordinates_1d,
coordinates_1e)

    # Use the
    homography_Img1 = calculate_Homography(coordinates_1a_target, coordinates_1a)
    homography_Img1_small = calculate_Homography(coordinates_1a_target_small, coordinates_1a)
    homography_Img2 = calculate_Homography(coordinates_1b_target, coordinates_1b)
    homography_Img3 = calculate_Homography(coordinates_1c_target, coordinates_1c)
    homography_Img4 = calculate_Homography(coordinates_1d_target, coordinates_1d)
    homography_Img5 = calculate_Homography(coordinates_1e_target, coordinates_1e)

#    Point-by-point transform
    apply_homography("./hw3_Task1_Images/Images/Img1.jpg", homography_Img1,
"./hw3_Task1_Images/Images/point-to-point/Img1-point-to-point.jpeg", True,4000)
    apply_homography("./hw3_Task1_Images/Images/Img1.jpg", homography_Img1_small,
"./hw3_Task1_Images/Images/point-to-point/Img1-point-to-point_small.jpeg", True,4000)
    apply_homography("./hw3_Task1_Images/Images/Img2.jpeg", homography_Img2,
"./hw3_Task1_Images/Images/point-to-point/Img2-point-to-point.jpeg", True, 2000)
    apply_homography("./hw3_Task1_Images/Images/Img3.jpg", homography_Img3,
"./hw3_Task1_Images/Images/point-to-point/Img3-point-to-point.jpeg", True, 2000)
    apply_homography("./hw3_Task1_Images/Images/Img4.jpeg", homography_Img4,
"./hw3_Task1_Images/Images/point-to-point/Img4-point-to-point.jpeg", False)
    apply_homography("./hw3_Task1_Images/Images/Img5.jpg", homography_Img5,
"./hw3_Task1_Images/Images/point-to-point/Img5-point-to-point.jpeg", False)
    return homography_Img1 ,homography_Img2, homography_Img3, homography_Img4, homography_Img5




def run_code_for_task1_Part2():
    coordinates_points_for_VL1 = np.array([[47,307,1],[8,575,1],[1011,698,1],[1000,523,1]])
    coordinates_points_for_VL2 = np.array([[478,718,1],[481,874,1],[607,923,1],[601,736,1]])
```

```
coordinates_points_for_VL3 = np.array([[2090,740,1],[2121,1430,1],[2673,1302,1],[2652,751,1]])
coordinates_points_for_VL4 = np.array([[549,420,1],[672,700,1],[1228,595,1],[993,353,1]])
coordinates_points_for_VL5 = np.array([[600,167,1],[656,490,1],[968,403,1],[926,129,1]])

vl1,  vp_1_1, vp_1_2, line_PS_1, line_QR_1, line_RS_1, line_PQ_1 =
calculate_line(coordinates_points_for_VL1)
vl2,  vp_2_1, vp_2_2, line_PS_2, line_QR_2, line_RS_2, line_PQ_2  =
calculate_line(coordinates_points_for_VL2)
vl3,  vp_3_1, vp_3_2, line_PS_3, line_QR_3, line_RS_3, line_PQ_3  =
calculate_line(coordinates_points_for_VL3)
vl4,  vp_4_1, vp_4_2, line_PS_4, line_QR_4, line_RS_4, line_PQ_4  =
calculate_line(coordinates_points_for_VL4)
vl5,  vp_5_1, vp_5_2, line_PS_5, line_QR_5, line_RS_5, line_PQ_5  =
calculate_line(coordinates_points_for_VL5)

img1 = cv2.imread("./hw3_Task1_Images/Images/Img1.jpg")
filename_1 =  "./hw3_Task1_Images/Images/with_vanishing_line/Img1_helper_lines.jpeg"
draw_lines_connecting_PQRS_on_image(img1, filename_1, line_PQ_1, line_QR_1, line_RS_1, line_PS_1)

img2 = cv2.imread("./hw3_Task1_Images/Images/Img2.jpeg")
filename_2 =  "./hw3_Task1_Images/Images/with_vanishing_line/Img2_helper_lines.jpeg"
draw_lines_connecting_PQRS_on_image(img2, filename_2, line_PQ_2, line_QR_2, line_RS_2, line_PS_2)

img3 = cv2.imread("./hw3_Task1_Images/Images/Img3.jpg")
filename_3 =  "./hw3_Task1_Images/Images/with_vanishing_line/Img3_helper_lines.jpeg"
draw_lines_connecting_PQRS_on_image(img3, filename_3, line_PQ_3, line_QR_3, line_RS_3, line_PS_3)

img4 = cv2.imread("./hw3_Task1_Images/Images/Img4.jpeg")
filename_4 =  "./hw3_Task1_Images/Images/with_vanishing_line/Img4_helper_lines.jpeg"
draw_lines_connecting_PQRS_on_image(img4, filename_4, line_PQ_4, line_QR_4, line_RS_4, line_PS_4)

img5 = cv2.imread("./hw3_Task1_Images/Images/Img5.jpg")
filename_5 =  "./hw3_Task1_Images/Images/with_vanishing_line/Img5_helper_lines.jpeg"
draw_lines_connecting_PQRS_on_image(img5, filename_5, line_PQ_5, line_QR_5, line_RS_5, line_PS_5)
```

```
    homography_Img1_proj, homography_Img2_proj, homography_Img3_proj, homography_Img4_proj, homography_Img5_proj = run_code_for_task1_Part2_1(vl1,vl2,vl3, vl4, vl5)

    homography_Img1_aff ,homography_Img2_aff, homography_Img3_aff, homography_Img4_aff, homography_Img5_aff, res_homography_1, res_homography_2, res_homography_3, res_homography_4, res_homography_5 = run_code_for_task1_Part2_2(homography_Img1_proj, homography_Img2_proj, homography_Img3_proj,  homography_Img4_proj, homography_Img5_proj,coordinates_points_for_VL1, coordinates_points_for_VL2, coordinates_points_for_VL3, coordinates_points_for_VL4, coordinates_points_for_VL5  )


    return homography_Img1_proj, homography_Img2_proj, homography_Img3_proj, homography_Img4_proj, homography_Img5_proj, homography_Img1_aff ,homography_Img2_aff, homography_Img3_aff, homography_Img4_aff, homography_Img5_aff, res_homography_1, res_homography_2, res_homography_3, res_homography_4, res_homography_5



def run_code_for_task1_Part2_1(vl1,vl2,vl3,vl4,vl5):

    homography_Img1 = homography_remove_projective_distortion(vl1)

    homography_Img2 = homography_remove_projective_distortion(vl2)

    homography_Img3 = homography_remove_projective_distortion(vl3)

    homography_Img4 = homography_remove_projective_distortion(vl4)

    homography_Img5 = homography_remove_projective_distortion(vl5)


#    # Remove projective distortion

    apply_homography("./hw3_Task1_Images/Images/Img1.jpg", homography_Img1, "./hw3_Task1_Images/Images/remove_proj_dist/Img1-proj-rem.jpeg", True, 2000)

    apply_homography("./hw3_Task1_Images/Images/Img2.jpeg", homography_Img2, "./hw3_Task1_Images/Images/remove_proj_dist/Img2-proj-rem.jpeg", False)

    apply_homography("./hw3_Task1_Images/Images/Img3.jpg", homography_Img3, "./hw3_Task1_Images/Images/remove_proj_dist/Img3-proj-rem.jpeg", True, 2000)

    apply_homography("./hw3_Task1_Images/Images/Img4.jpeg", homography_Img4, "./hw3_Task1_Images/Images/remove_proj_dist/Img4-proj-rem.jpeg", True, 2000)

    apply_homography("./hw3_Task1_Images/Images/Img5.jpg", homography_Img5, "./hw3_Task1_Images/Images/remove_proj_dist/Img5-proj-rem.jpeg", True, 2000)

    return homography_Img1 ,homography_Img2, homography_Img3, homography_Img4, homography_Img5



def run_code_for_task1_Part2_2(h1,h2,h3,h4,h5, coordinates_points_for_VL1 ,coordinates_points_for_VL2, coordinates_points_for_VL3, coordinates_points_for_VL4,  coordinates_points_for_VL5):

    new_coordinate_points_for_VL1 = calculate_corners_without_projective_distortion(h1, coordinates_points_for_VL1)
```

```
    new_coordinate_points_for_VL2 = calculate_corners_without_projective_distortion(h2, coordinates_points_for_VL2)

    new_coordinate_points_for_VL3 = calculate_corners_without_projective_distortion(h3, coordinates_points_for_VL3)

    new_coordinate_points_for_VL4 = calculate_corners_without_projective_distortion(h4, coordinates_points_for_VL4)

    new_coordinate_points_for_VL5 = calculate_corners_without_projective_distortion(h5, coordinates_points_for_VL5)


    new_line_PS_1, new_line_QR_1, new_line_PQ_1, new_line_SR_1 = calculate_lines_between_points(new_coordinate_points_for_VL1)

    new_line_PS_2, new_line_QR_2, new_line_PQ_2, new_line_SR_2 = calculate_lines_between_points(new_coordinate_points_for_VL2)

    new_line_PS_3, new_line_QR_3, new_line_PQ_3, new_line_SR_3 = calculate_lines_between_points(new_coordinate_points_for_VL3)

    new_line_PS_4, new_line_QR_4, new_line_PQ_4, new_line_SR_4 = calculate_lines_between_points(new_coordinate_points_for_VL4)

    new_line_PS_5, new_line_QR_5, new_line_PQ_5, new_line_SR_5 = calculate_lines_between_points(new_coordinate_points_for_VL5)


    homography_Img1 = homography_remove_affine_distortion(new_line_PS_1, new_line_QR_1, new_line_PQ_1, new_line_SR_1)

    homography_Img2 = homography_remove_affine_distortion(new_line_PS_2, new_line_QR_2, new_line_PQ_2, new_line_SR_2)

    homography_Img3 = homography_remove_affine_distortion(new_line_PS_3, new_line_QR_3, new_line_PQ_3, new_line_SR_3)

    homography_Img4 = homography_remove_affine_distortion(new_line_PS_4, new_line_QR_4, new_line_PQ_4, new_line_SR_4)

    homography_Img5 = homography_remove_affine_distortion(new_line_PS_5, new_line_QR_5, new_line_PQ_5, new_line_SR_5)


    res_homography_1 = np.dot(h1, np.linalg.pinv(homography_Img1))

    res_homography_2 = np.dot(h2, np.linalg.pinv(homography_Img2))

    res_homography_3 = np.dot(h3, np.linalg.pinv(homography_Img3))

    res_homography_4 = np.dot(h4, np.linalg.inv(homography_Img4))

    res_homography_5 = np.dot(h5, np.linalg.inv(homography_Img5))
```

```
#    # remove affine distortion
    apply_homography("./hw3_Task1_Images/Images/Img1.jpg", res_homography_1,
"./hw3_Task1_Images/Images/remove_aff_dist/Img1-aff-rem.jpeg", True, 1000)
    apply_homography("./hw3_Task1_Images/Images/Img2.jpeg", res_homography_2,
"./hw3_Task1_Images/Images/remove_aff_dist/Img2-aff-rem.jpeg", True, 1000)
    apply_homography("./hw3_Task1_Images/Images/Img3.jpg", res_homography_3,
"./hw3_Task1_Images/Images/remove_aff_dist/Img3-aff-rem.jpeg", True, 1000)
    apply_homography("./hw3_Task1_Images/Images/Img4.jpeg", res_homography_4,
"./hw3_Task1_Images/Images/remove_aff_dist/Img4-aff-rem.jpeg", True, 1000)
    apply_homography("./hw3_Task1_Images/Images/Img5.jpg", res_homography_5,
"./hw3_Task1_Images/Images/remove_aff_dist/Img5-aff-rem.jpeg", True, 1000)
#
    return homography_Img1 ,homography_Img2, homography_Img3, homography_Img4, homography_Img5,
res_homography_1, res_homography_2, res_homography_3, res_homography_4, res_homography_5


def run_code_for_task1_Part3():
    coordinates_points_for_VL1 = np.array([[130,472,1],[125,513,1],[162,521,1],[166,477,1]])
    coordinates_points_for_VL2 = np.array([[382,575,1],[379,835,1],[607,923,1],[594,551,1]])
    coordinates_points_for_VL3 = np.array([[2090,740,1],[2121,1430,1],[2673,1302,1],[2652,751,1]])
    coordinates_points_for_VL4 = np.array([[549,420,1],[672,700,1],[1228,595,1],[993,353,1]])
    coordinates_points_for_VL5 = np.array([[600,167,1],[656,490,1],[968,403,1],[926,129,1]])

    add_coord_points_1 =  np.array([[47,307,1],[8,575,1],[1011,698,1],[1000,523,1]])
    add_coord_points_2 = np.array([[425,231,1],[425,347,1],[515,298,1],[513,167,1]])
    add_coord_points_3 = np.array([[2063,696,1],[2094,1480,1],[2694,1329,1],[2665,720,1]])
    add_coord_points_4 = np.array([[777,385,1],[858,500,1],[1097,460,1],[993,353,1]])
    add_coord_points_5 = np.array([[521,107,1],[605,573,1],[1021,443,1],[963,68,1]])

    img1 = cv2.imread("./hw3_Task1_Images/Images/Img1.jpg")
    img2 = cv2.imread("./hw3_Task1_Images/Images/Img2.jpeg")
    img3 = cv2.imread("./hw3_Task1_Images/Images/Img3.jpg")
    img4 = cv2.imread("./hw3_Task1_Images/Images/Img4.jpeg")
```

```
img5 = cv2.imread("./hw3_Task1_Images/Images/Img5.jpg")


colours = [(0,0,250), (255,0,255), (0,255,0) ,(300,300,0)]
for i in range(4):
    cv2.drawMarker(img1,(add_coord_points_1[i,0],add_coord_points_1[i,1]), colours[0],1, 20, 3)
    cv2.drawMarker(img1,(coordinates_points_for_VL1[i,0],coordinates_points_for_VL1[i,1]), colours[1],1, 20, 
3)
cv2.imwrite("./hw3_Task1_Images/Images/with_markers/part3_Img1_markers.jpeg", img1)


for i in range(4):
    cv2.drawMarker(img2,(add_coord_points_2[i,0],add_coord_points_2[i,1]), colours[0],1, 20, 3)
    cv2.drawMarker(img2,(coordinates_points_for_VL2[i,0],coordinates_points_for_VL2[i,1]), colours[1],1, 20, 
3)
cv2.imwrite("./hw3_Task1_Images/Images/with_markers/part3_Img2_markers.jpeg", img2)


for i in range(4):
    cv2.drawMarker(img3,(add_coord_points_3[i,0],add_coord_points_3[i,1]), colours[0],1, 20, 3)
    cv2.drawMarker(img3,(coordinates_points_for_VL3[i,0],coordinates_points_for_VL3[i,1]), colours[1],1, 20, 
3)
cv2.imwrite("./hw3_Task1_Images/Images/with_markers/part3_Img3_markers.jpeg", img3)


for i in range(4):
    cv2.drawMarker(img4,(add_coord_points_4[i,0],add_coord_points_4[i,1]), colours[0],1, 20, 3)
    cv2.drawMarker(img4,(coordinates_points_for_VL4[i,0],coordinates_points_for_VL4[i,1]), colours[1],1, 20, 
3)
cv2.imwrite("./hw3_Task1_Images/Images/with_markers/part3_Img4_markers.jpeg", img4)


for i in range(4):
    cv2.drawMarker(img5,(add_coord_points_5[i,0],add_coord_points_5[i,1]), colours[0],1, 20, 3)
    cv2.drawMarker(img5,(coordinates_points_for_VL5[i,0],coordinates_points_for_VL5[i,1]), colours[1],1, 20, 
3)
cv2.imwrite("./hw3_Task1_Images/Images/with_markers/part3_Img5_markers.jpeg", img5)




line_PS_1, line_QR_1, line_RS_1, line_PQ_1 = calculate_lines_between_points(coordinates_points_for_VL1)
```

```
line_PS_2, line_QR_2, line_RS_2, line_PQ_2  = calculate_lines_between_points(coordinates_points_for_VL2)
line_PS_3, line_QR_3, line_RS_3, line_PQ_3  = calculate_lines_between_points(coordinates_points_for_VL3)
line_PS_4, line_QR_4, line_RS_4, line_PQ_4  = calculate_lines_between_points(coordinates_points_for_VL4)
line_PS_5, line_QR_5, line_RS_5, line_PQ_5  = calculate_lines_between_points(coordinates_points_for_VL5)

line_PS_1_add, line_QR_1_add, line_RS_1_add, line_PQ_1_add  = calculate_lines_between_points(add_coord_points_1)
line_PS_2_add, line_QR_2_add, line_RS_2_add, line_PQ_2_add  = calculate_lines_between_points(add_coord_points_2)
line_PS_3_add, line_QR_3_add, line_RS_3_add, line_PQ_3_add  = calculate_lines_between_points(add_coord_points_3)
line_PS_4_add, line_QR_4_add, line_RS_4_add, line_PQ_4_add  = calculate_lines_between_points(add_coord_points_4)
line_PS_5_add, line_QR_5_add, line_RS_5_add, line_PQ_5_add  = calculate_lines_between_points(add_coord_points_5)

one_step_hom1 = homography_remove_distortions(line_PS_1, line_QR_1, line_RS_1, line_PQ_1,line_PS_1_add, line_QR_1_add, line_RS_1_add, line_PQ_1_add)
one_step_hom2 = homography_remove_distortions(line_PS_2, line_QR_2, line_RS_2, line_PQ_2,line_PS_2_add, line_QR_2_add, line_RS_2_add, line_PQ_2_add)
one_step_hom3 = homography_remove_distortions(line_PS_3, line_QR_3, line_RS_3, line_PQ_3,line_PS_3_add, line_QR_3_add, line_RS_3_add, line_PQ_3_add)
one_step_hom4 = homography_remove_distortions(line_PS_4, line_QR_4, line_RS_4, line_PQ_4,line_PS_4_add, line_QR_4_add, line_RS_4_add, line_PQ_4_add)
one_step_hom5 = homography_remove_distortions(line_PS_5, line_QR_5, line_RS_5, line_PQ_5,line_PS_5_add, line_QR_5_add, line_RS_5_add, line_PQ_5_add)

apply_homography("./hw3_Task1_Images/Images/Img1.jpg", one_step_hom1, "./hw3_Task1_Images/Images/remove_oneStep/Img1_one_step.jpeg", True,1000)
apply_homography("./hw3_Task1_Images/Images/Img2.jpeg", one_step_hom2, "./hw3_Task1_Images/Images/remove_oneStep/Img2_one_step.jpeg", True,1000)
apply_homography("./hw3_Task1_Images/Images/Img3.jpg", one_step_hom3, "./hw3_Task1_Images/Images/remove_oneStep/Img3_one_step.jpeg", True, 1000)
apply_homography("./hw3_Task1_Images/Images/Img4.jpeg", one_step_hom4, "./hw3_Task1_Images/Images/remove_oneStep/Img4_one_step.jpeg", True, 1000)
apply_homography("./hw3_Task1_Images/Images/Img5.jpg", one_step_hom5, "./hw3_Task1_Images/Images/remove_oneStep/Img5_one_step.jpeg", True, 1000)
```

```
    return one_step_hom1, one_step_hom2, one_step_hom3, one_step_hom4, one_step_hom5



homography_Img1 ,homography_Img2, homography_Img3, homography_Img4, homography_Img5 =
run_code_for_task1_Part1()

homography_Img1_proj, homography_Img2_proj, homography_Img3_proj, homography_Img4_proj,
homography_Img5_proj, homography_Img1_aff ,homography_Img2_aff, homography_Img3_aff,
homography_Img4_aff, homography_Img5_aff, res_homography_1, res_homography_2, res_homography_3,
res_homography_4, res_homography_5 = run_code_for_task1_Part2()

one_step_hom1, one_step_hom2, one_step_hom3, one_step_hom4, one_step_hom5 =
run_code_for_task1_Part3()
```