3D recovery of urban scenes. Session 1

Guillem Capellera[1], Johnny Núñez[1], and Anna Oliveras[1]

1. Universitat Pompeu Fabra, Barcelona, Spain

1 Applying image transformations

Linear algebra holds many essential roles in image processing. One of which is the transformation of 2D images through matrix multiplications. In this project, we present an example of such transformation matrix, called homography. Basically, an homography is a 3x3 matrix that represents a linear mapping between two images. Homographies are often used to align images taken from different viewpoints or under different lighting conditions, or to transform an image into a different coordinate space. We will work with three types of planar transformations: similarities, affinities and projectivities (also called homographies). To apply them, the apply_H algorithm will be used (see 1). Given an input image I and an homography H this algorithm uses the inverse warping technique in order to obtain the output transformed image.

Algorithm 1 Apply Homography

- **Inputs:** **I**: The input image (MxNx3 dimensional numpy array).
  H: The 3x3 homography matrix.
  corners: Optional list of four corner points (x, y) in the input image to use for computing the output image size and boundary padding.
- **Outputs:** $\mathbf{I}_{out}$: The output transformed image (MxNx3 dimensional numpy array).
  $\mathbf{axis}_{out}$: A 4-element list containing the max and min values of each x and y axis. It is used to plot the transformed image.
  $\mathbf{corners}_{out}$: Transformed corners (4 dimensional numpy array).

1. Apply the homography matrix **H** to the **corners** of the input image **I** to find the $\mathbf{orners}_{out}$.
2. Find the minimum and maximum x and y coordinates of the four corners_{out}, to construct the $\mathbf{axis}_{out}$
3. Create a meshgrid to represent de coordinates of the output image by using the axis_{out} values.
4. Find the inverse of the homography matrix $\mathbf{H}^{-1}$ and apply it to the meshgrid to obtain the inverse exact mapping between output coordinates and input ones.
5. For each channel, use the SciPy function map_coordinates to interpolate the values of the input image at each output coordinate (obtain the output image $\mathbf{I}_{out}$).

1.1 Similarities

A similarity transformation is an isometry composed with an isotropic scaling. The invariants of this type of transformation are the ratio of lengths, angles and the ratio of areas. The transformation can be expressed as:
$H_S x = \begin{bmatrix} sR & \mathbf{t} \\ 0^T & 1 \end{bmatrix}$ x, where the rotation matrix $R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$.
We applied the transformation with the translation vector t=(100,330), the scaling factor s = 1.5 and with a rotation angle $\theta = \frac{\pi}{6}$.

(a) Original (b) Similarity

Fig. 1: Original image and transformed image after a similarity

Here we realized that to make the correct representation of the image axes (top-left corner not starting always at (0,0)) we had to modify the function plot_img from utils. We added another optional input that corresponds to the argument *extent* of the function pyplot.imshow from MATPLOTLIB. This argument controls the bounding box in data coordinates that the image will fill specified as (left, right, bottom, top) in data coordinates. The used input to fill this argument is the output axis_{out} from apply_H algorithm. This modification is also used in all the following sections.

1.2 Affinities

An affine transformation is a transformation that preserves collinearity, the property of a set of points that remain aligned after the transformation. Thus, these transformations preserve parallel lines, ratios of lengths of parallel lines and ratios of areas. The transformation can be expressed as:
$H_A x = \begin{bmatrix} A & \mathbf{t} \\ 0^T & 1 \end{bmatrix}$ x, where A is a non-singular 2x2 matrix.
We also proved that the affinity can also be expressed as the product of these four transformations: two rotations, a scale, and a translation. We computed the MSE and obtained a very small value $7.19 \cdot 10^{-33}$, showing that the two H matrices are the same.

(a) Original (b) Affinity

Fig. 2: Original image and transformed image after an Affinity

1.3 Projective transformations (Homographies)

A projective transformation, also known as homography, is a type of geometric transformation that maps lines to lines. It is a more general transformation than an affine transformation, as it does not preserve collinearity or parallelism in general. However, it does preserve ratios of distances between points lying on the same line. It can be expressed as $H_P x = \begin{bmatrix} A & \mathbf{t} \\ v^T & \nu \end{bmatrix}$ x. The main difference between a projective and affine transformation is that the vector v is not null for a projectivity, which results in non-linear effects as we can see in 3.

(a) Original (b) Projectivity

Fig. 3: Original image and transformed image after a Projectivity

2 Affine rectification

Affine rectification is a process that applies an affine transformation to an image in order to correct for distortions or align the image with a reference coordinate system. This is often used as a preprocessing step for further image analysis or to correct for geometric distortions produced by a camera lens.

Therefore, Affine rectification is based on the fact that the vanishing line is mapped to its canonical $l_\infty = (0, 0, 1)^T$. To compute the vanishing line, two sets of parallel lines in the real world have been used, since each of them provides a vanishing point. After the rectification, we showed that the selected lines become parallel in the affine rectified image, and the angle between them becomes equal to 0.

(a) Before (b) After

Fig. 4: Original image with the parallel lines used before and after applying the affine rectification

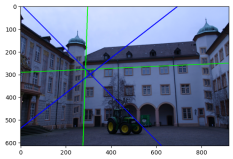
3 Metric rectification

Unlike affine rectification, which only preserves ratios of distances between points lying on the same line, metric rectification preserves both ratios of distances and angles between points. This makes it a more accurate and robust method for correcting distortions in images, but it also requires more information to estimate the transformation.

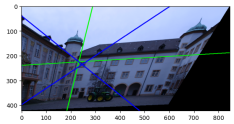
In this case, we need two sets of orthogonal lines which need to be linearly independent. Thus, we need another set of lines, the diagonals of the windows in this case. The idea is that these lines in the affinely rectified image satisfy that $l'C^*_\infty m' = 0$ where l' and m' are one set of lines and $C^*_\infty = \begin{bmatrix} KK^T & \mathbf{0} \\ 0^T & 0 \end{bmatrix}$. This leads to the equation $(l_1m_1;\ l_1m_2 + l_2m_1;\ l_2m_2)s = 0$.

Thus, with the two sets of lines a system of equations can be written and we can find s as the null vector. Then, after finding the matrix S, the matrix K can be obtained up to scale by Cholesky decomposition. Finally, we obtain the matrix H_{s_a} to perform the rectification using the inverse of K. Here is where the input argument *corners* and the output $corners_{out}$ make sense. Until now, all transformations started from the original image, so the corners could be known with the shape of this image. In this case, instead, after the affine rectification, the image may have a background that we don't want to take into account when making the metric rectification. This is why it's necessary to know which are the $corners_{out}$ of the affine rectification and pass them as input to apply the metric rectification.

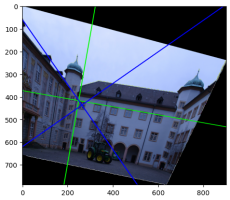
We can see the results and the transformed lines in 5 and we also verified that the angles of the sets of lines after the metric rectification were 90^{o}, since lines became orthogonal after the rectification.



(a) Original



(b) After affine rectification



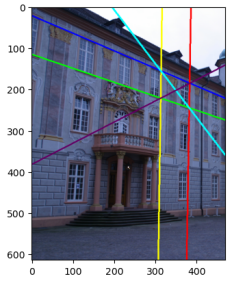
(c) After metric rectification

Fig. 5: Original image and transformed images with the orthogonal lines used

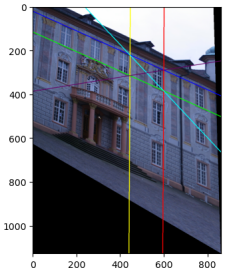
4 Affine and Metric Rectification of the left facade of image 0001

As another example of the rectifications, we used image 0001. However, this time the left facade was cropped for better visualization. We did the same as explained before but applied the transformation to the cropped image. In this case, when we performed the transformation from affine rectified lines to metric rectified lines, we realized that they were not represented at the correct place. The reason is that the function line_draw from utils assumes that the minimum point of the image is the (0,0). Here, unlike the other cases, the image does not start at (0,0), but the minimum y coordinate (y_{min}) is a negative value. For this reason the lines are displaced upwards (see 6). To solve this issue, we have devised a function to rectify these lines in such a way that it translates the line t = -(x_{min}, y_{min}).

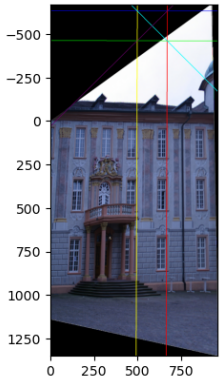
(a) Original



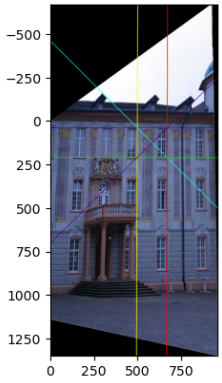
(b) Cropped image



(c) After affine rectification



(d) After metric rectification



(e) After metric and line rectifications

Fig. 6: Original image and transformed images with the parallel and orthogonal lines used

5 OPTIONAL: Metric Rectification in a single step

To perform metric rectification in a single step 5 sets of orthogonal lines are needed, particularly, we used the lines shown in 7 from image 0000. Similarly than in section 3, the conic C^*_∞ is determined on the perspectively imaged plane, this time, using the five orthogonal line pairs. The equation to be solved is: (l1m1,

(l1m2 + l2m1)/2, l2m2, (l1m3 + l3m1)/2, (l2m3 + l3m2)/2, l3m3) c = 0, where c is the conic matrix written as a 6-vector. So, with the five constraints, we can create a system of equations and find c as the null vector. Then we can compute the matrix C and obtain the transformation matrix H with the singular value decomposition. So, $C^*_\infty = UDU^T$ and H = U. However, we were not able to obtain the rectified image. The output we got is shown in the figure 8

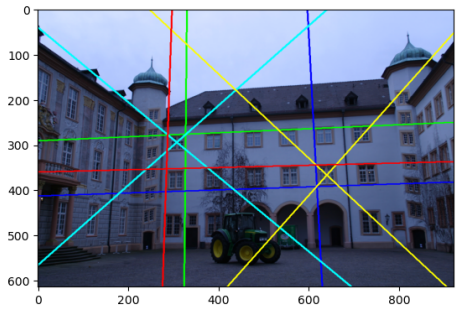


Fig. 7: Image 0000 with the 5 sets of orthogonal lines

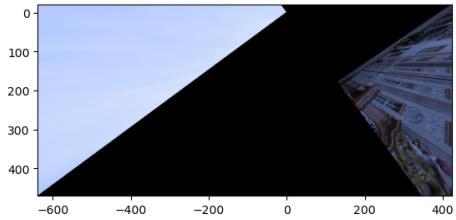


Fig. 8: Wrong metric rectification of Image 0000

6 Conclusions

In this work, we learned how to implement different transformations and rectifications. To quantitatively check the results, the angles at each step have been calculated and compared. During this process, a number of issues have arisen. The first one was based on the need to modify the plot_img function to be able to add an optional argument so that the values of the axes are correct. The other issues are related to the step between affine rectification and metric rectification. Here we needed to modify and create functions in order to use the necessary information correctly. Finally, we had difficulties to perform the metric rectification in one step. Maybe the chosen lines were not the optimal ones.