IACV Homework 2020/2021

Image analysis and computer vision

Politecnico di Milano
Dennis Zanutto
gennaio 2021

1. Description of the problem

The observed scene is an image of the Castello di Miramare in Trieste.

Six facades are presents, in the image.

It's known that the yellow line in the picture lies in the same horizontal plane, moreover the facade 1 and 2 are orthogonal. Same constraint is applied between 4 and 5 and also between 5 and 6. This also means that 4 and 6 are parallel.

Additional informations about the camera are provided. It's a digital camera and the skew factor can be assumed to be null. On the other hand no information about aspect ratio, focal distance and principal point have been given.

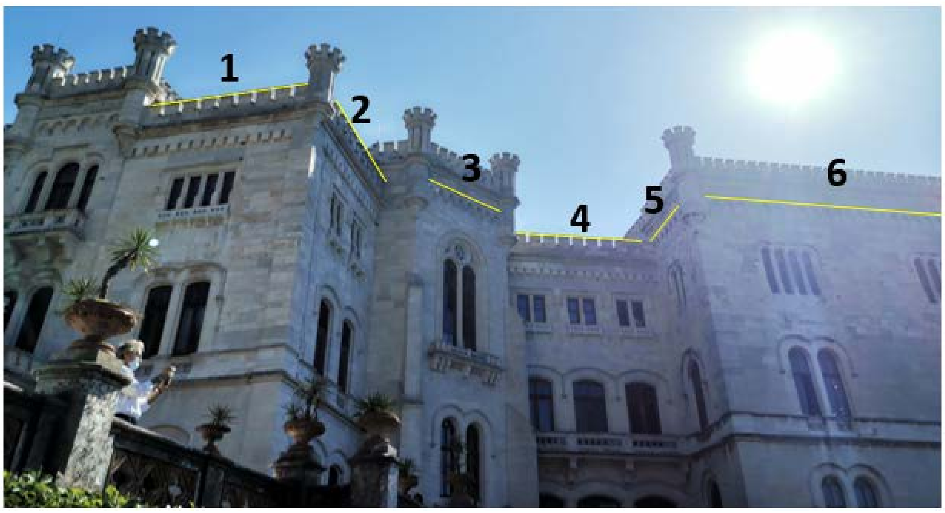

Fig. 1 assigned image

Image processing

2. Feature extraction

First assignment consists in finding the edges, corners and lines that are present in the image.

1. Edges

In order to find the edges of the image canny algorithm has been used.

In MATLAB `edgs( image, 'canny', [thresholds] ).`

This consists in 4 steps, that are:

- Gaussian smoothing

The images are usually corrupted by noise, thus a filtering of the image is needed to remove this component and perform a better computation in the following steps. A convolution between a Gaussian kernel and images is performed to obtain the image for the subsequent steps.

- Gradient calculation

To compute the gradient direction and magnitude, the partial derivatives are needed, these are find using the Sobel operator, defined as:

$$\frac{d}{dx}S = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \frac{d}{dy}S = [\frac{d}{dx}S]^T$$

Gradient magnitude and direction are then:

$$|\Delta G| = \sqrt{\frac{d}{dx}S^{2} + \frac{d}{dy}S^{2}} \qquad \theta = atan\left(\frac{\frac{d}{dx}S}{\frac{d}{dy}S}\right)$$

- Non maximum suppression

This step checks for every pixel if the gradient at that point is a maximum in the gradient direction, if not the gradient at that point is set to 0.

- Hysteresis thresholding

This final step is required to obtain the binary image containing the edges. The edges are the points that survive this operation. The two parameters T_{low} and T_{high} can be set to a required value to suppress more or less edges. For example, this helps in avoiding to consider the color changes as edges.

The results for different value of the thresholds are the following:

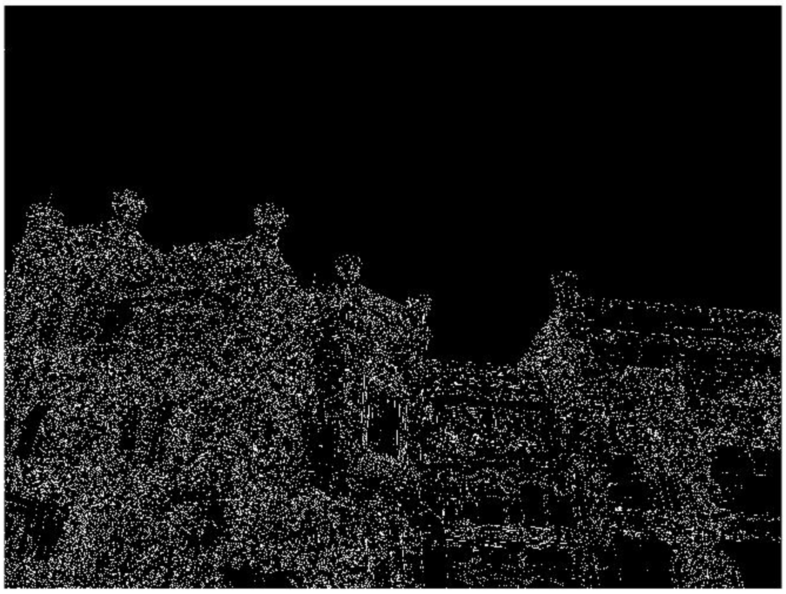

Fig. 2a Canny output

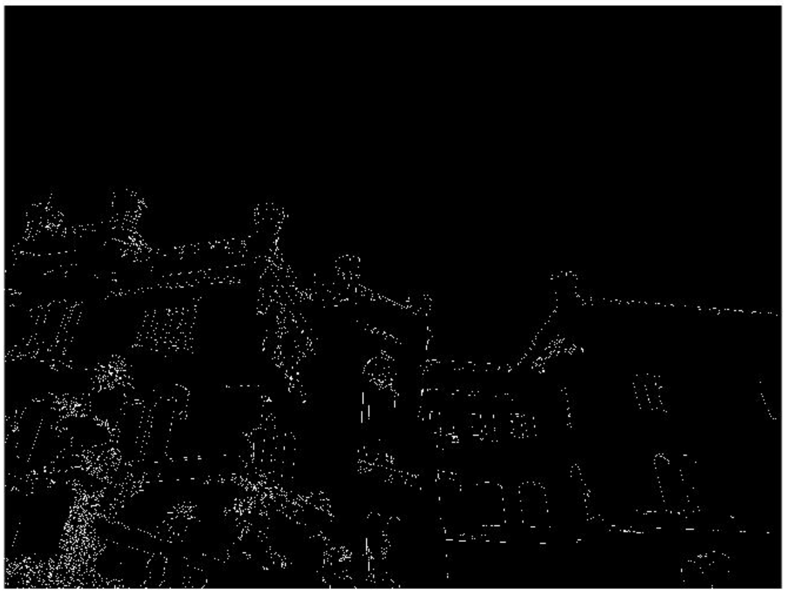

Fig. 2b Canny output with manual threshold

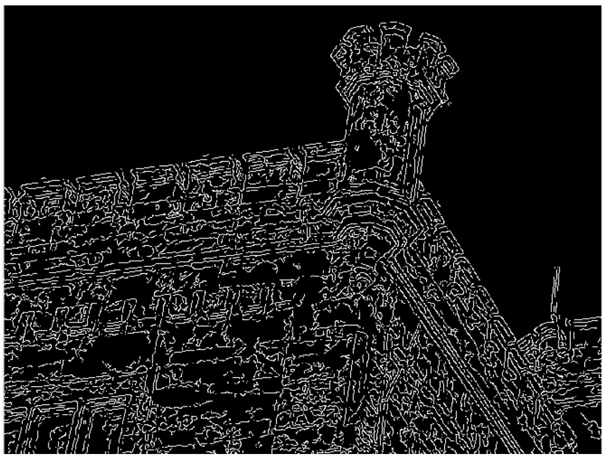
Fig. 2c Canny output zoom

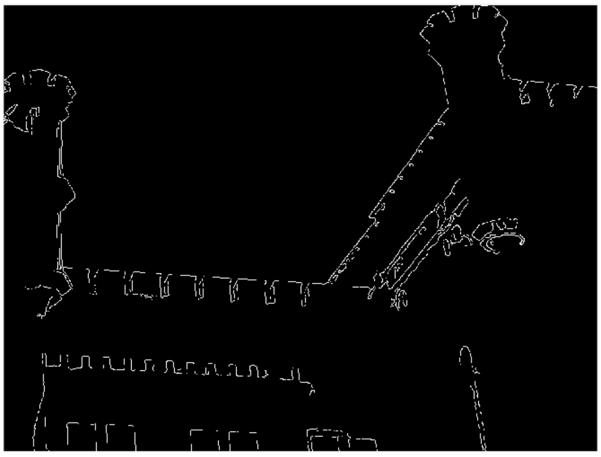
Fig. 2d Canny output 2 zoom

2. Corners

In order to find these features, the Harris algorithm is used.

In MATLAB `detectHarrisFeatures( image )`.

This consists in 4 steps, too:

- Spatial derivative calculation

Like step 1 in Canny algorithm.

- Matrix M

Given the spatial derivatives, the matrix M for every point is calculated. This matrix will be a minimum along any direction when the point is a corner.

- CM value

To understand faster which points are corner, the parameter

$CM = \dfrac{det(M)}{Tr(M)}$ is calculated

- Extract corners

The coordinates where CM is a maximum are the corners.

This is the result for the 100 points that had a maximum.

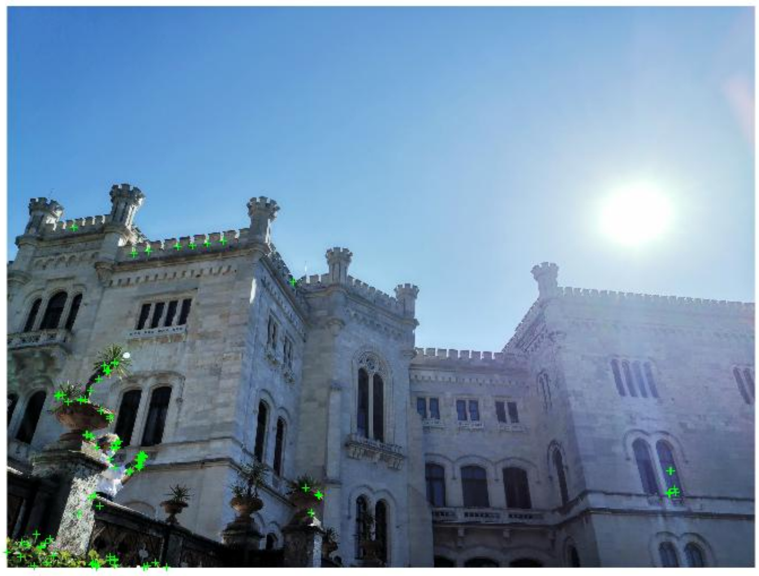

Fig. 3 Harris output

Anyway, with this function is possible to limit the search for a given area, to better extract the local corners.

A clear problem is the plants on the left, that “distract” the algorithm.

3. Lines

In order to extract the lines, Hough lines algorithm has been used, starting from the edges of the image (output of point 1).

This algorithm consists in a voting system that extract the most voted lines. The key points are the following. Every possible model of the line is represented as a Hough point, with two coordinates that are its angle θ and its distance from the center ρ. Thus, a point will belong to the line if

$$X_i cos(\theta) + Y_i sin(\theta) - \rho = 0$$

The result will be a grid with an intensity related to the number of votes, then the local maxima will be extracted and those are the parameters of the lines.

In MATLAB

```
[H,theta,rho] = hough(edgs, 'RhoResolution', rho_res{idx}, 'Theta',
theta_res{idx});
P = houghpeaks(H, how_many, 'threshold', ceil(0.3*max(H(:))));
lines = houghlines(edgs, theta,rho, P, 'FillGap',fill_gap_length,
'MinLength',min_length{idx});
```

To improve the selection of the lines, the range of the parameter theta has been reduced to a neighbourhood of the desired line angle. For example to extract vertical lines we look between -20 degree and +20 degree.

Furthermore, to remove the noise of some part of the images and focus on a given facade, some masks has been applied to the canny output. Then, in order to improve even more, different values of the parameters in the algorithm has been applied based on the lines that needed to be selected. So after I manually discarded the wrong one, this are the select lines, with a different color for every different direction.

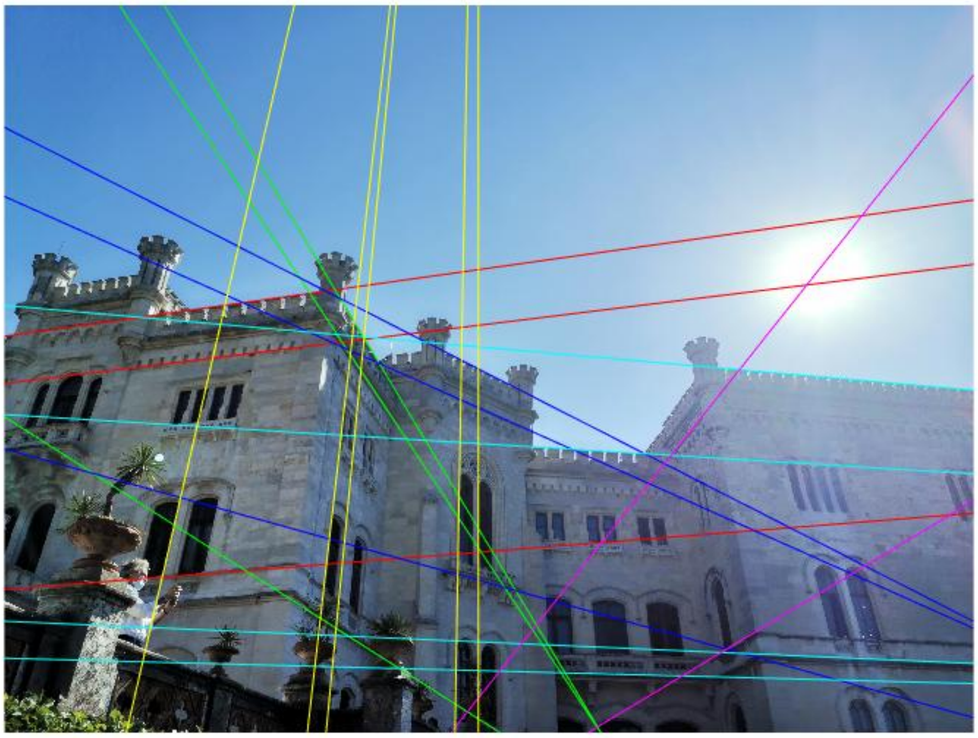

Fig. 4 Selected lines with Hough

3. 2D reconstruct

Once the extraction of the main lines, on the 5 directions that we are interested in on the plane π and the vertical lines, has been terminated, the reconstruction of the plane starts.

To improve the calculations, normalized coordinates has been used through all the following points. The image is rectangular of dimensions 2796x3968, thus I set a *imref2d* object on the image with dimension [0, 1] on the x axis and [0, 2796/3968] on the y. By doing this operation, the transformation of the coordinates is just a similarity, while if I were to choose [0, 1] [0, 1] it would have been an affine. Despite the fact that an affine transformation doesn't move the lines at the infinity, so the result would have worked anyway, the image would have been displayed squared instead of rectangular.

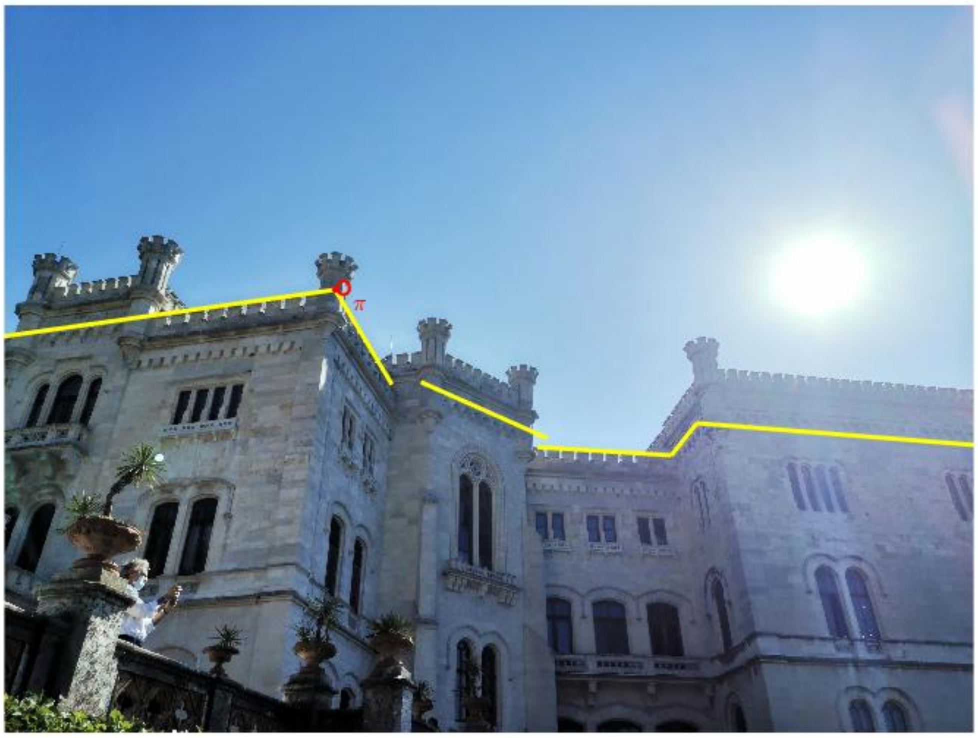

Fig. 5 Plane π

1. Affine rectification

Affine rectification is needed to obtain an image where all the lines that are parallel in the real world are also parallel.

First step is to extract the vanishing point on every direction, that we are interested in. All the parallel lines end up in the same point, so it's possible to use lines that belongs to different planes to extract this feature. The vanishing point between two lines is extracted as the point where the lines cross, thus at least two lines for every direction are needed. If more are available, least square approach can be used to find a point or it can be selected as the centroid of all the possible intersection between those lines.

Then, once all the vanishing points are computed, the image of the line at the infinity of that plane is computed. This line should pass through all those points, but due to small errors in the previous steps it doesn't do it perfectly. Thus, a first approach could be to choose only 2 of them and neglect the others, otherwise a least square approach could extract a more precise line.

Once the line at the infinity has been extracted the following transformation needs to be applied to resort the affine image.

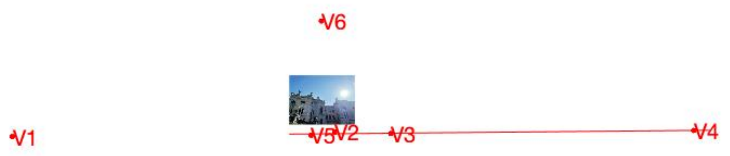

Fig. 6 Vanishing points and line at the infinity for π

$$H_{ar} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l1 & l_2 & l_3 \end{bmatrix} \text{ where } l_\infty = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

The output, using *`imwarp`* is the following one:

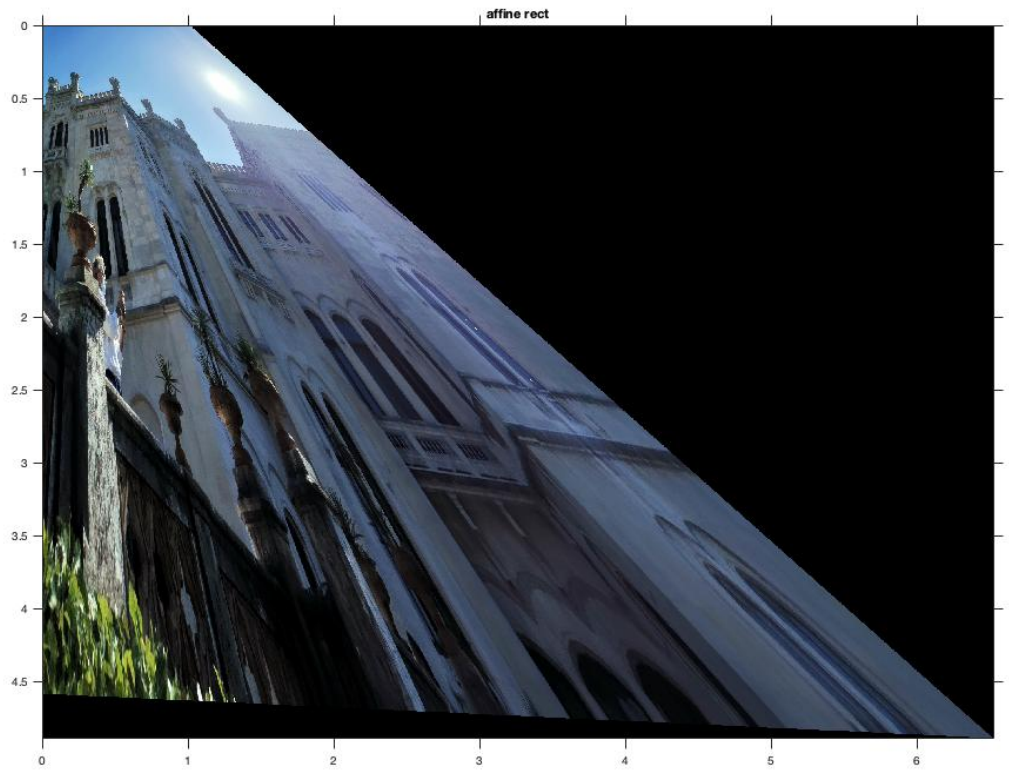

Fig. 7a affine rectification

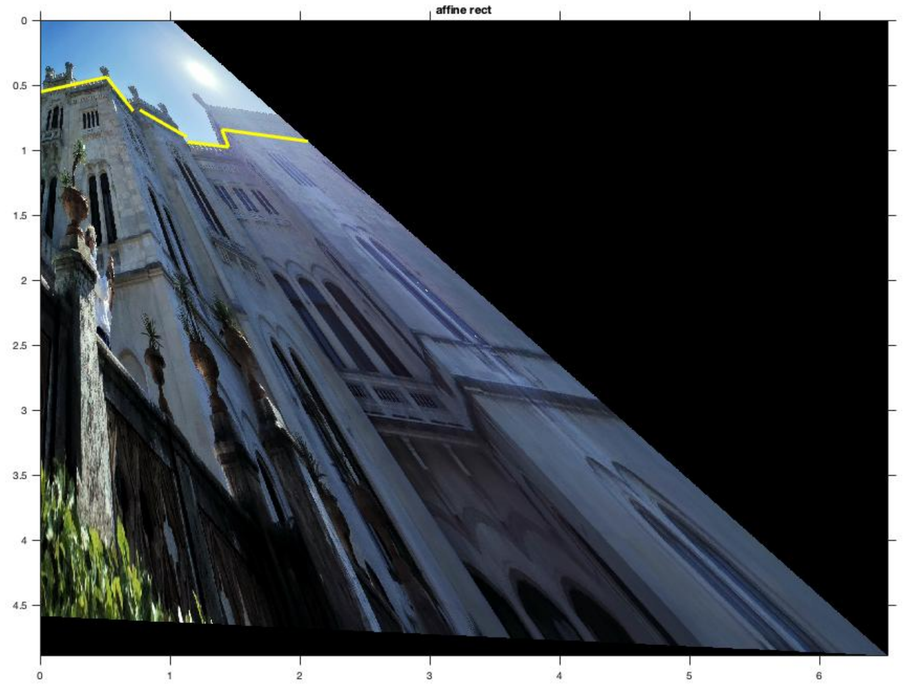

Fig. 7b affine rectification with plane π

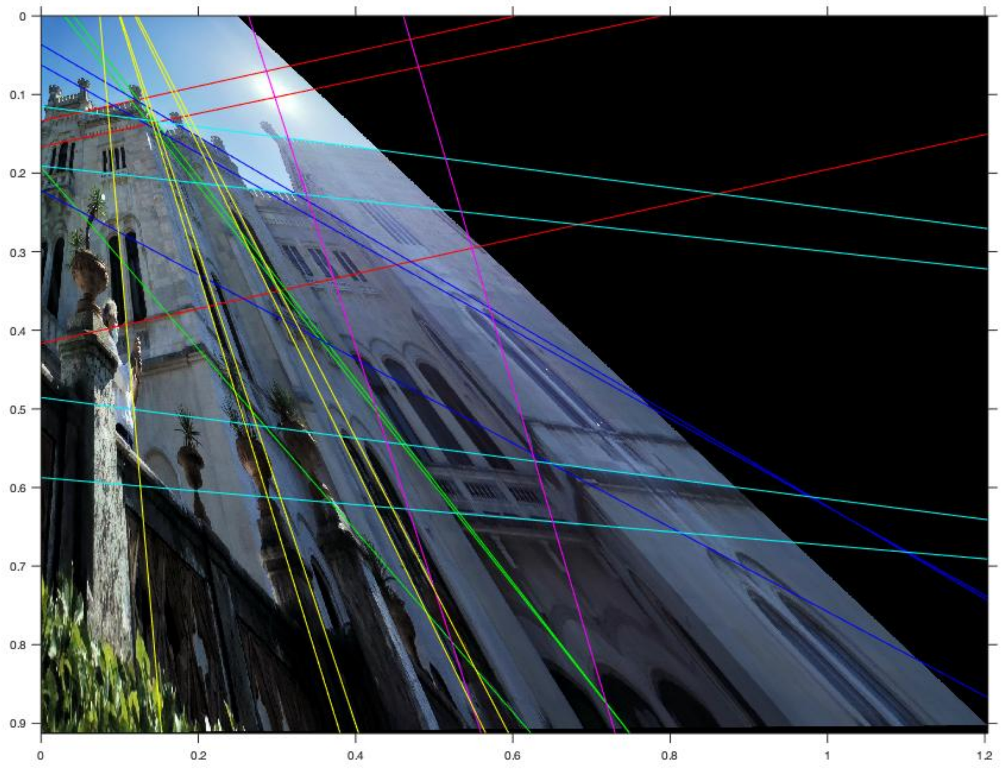

Fig. 7c affine rectification with selected lines

The effect of the transformation is clear, lines that used to be not parallel now are.

It's also possible to see small numerical errors in some directions, like green one.

2. Metric from affine

Starting from the affine image.

It's possible to extract the image of the conic dual to the circular points $C_\infty^{*\prime}$. To do so, 2 pairs of orthogonal lines are needed and from their orthogonality relation, the solution is extracted.

In this case the pairs (1, 2) and (4, 5) have been used.

Using SVD, the matrix H_{rect} has been extracted and applied.

The output:

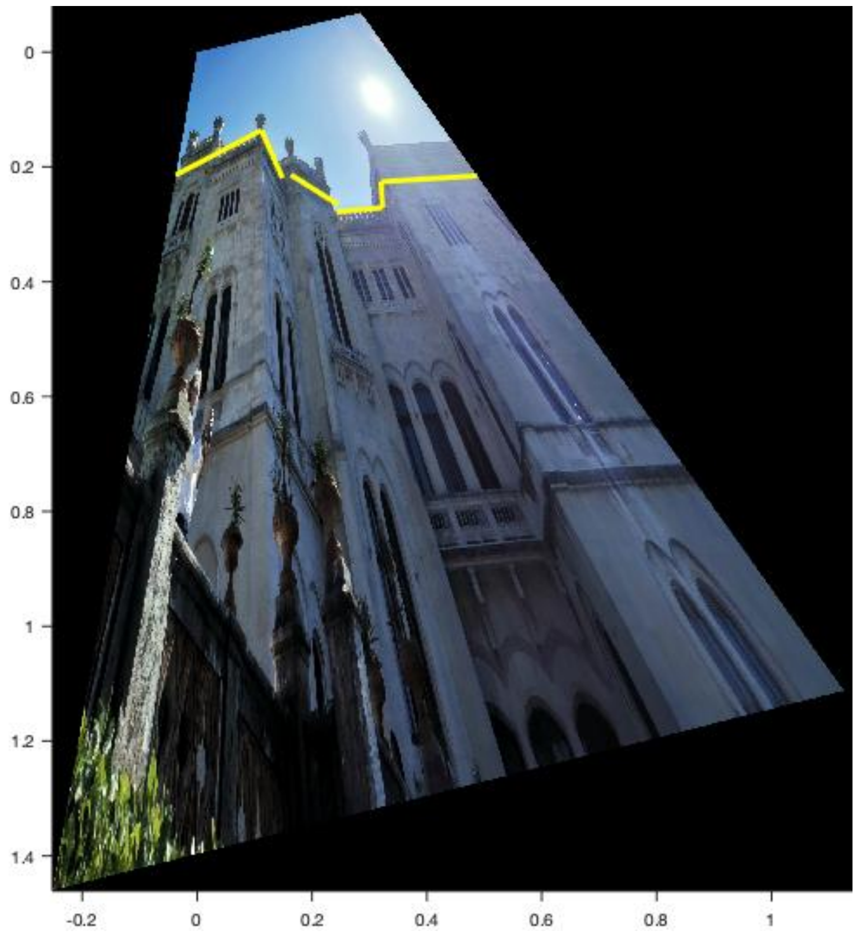

Fig. 8 Metric rectification

After this step, the lines that are perpendicular in the real world are real again.

Another step has been performed between the affine and metric, i.e., the image has been resized, because it was increasing too much in size and the computer couldn't handle it.

4. Calibration

One information is provided at the beginning, i.e., the camera skew factor is 0, but the natural camera assumption cannot be taken ($f_x \neq f_y$). Thus, we can expect the matrix K and the image of the absolute conic to be:

$$K = \begin{bmatrix} f_x & 0 & U_0 \\ 0 & f_y & V_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \omega = \begin{bmatrix} a^2 & 0 & -u_0 a^2 \\ 0 & 1 & -v_0 \\ -u_0 a^2 & -v v_0 & f_y^2 + a^2 u_0^2 + v_0^2 \end{bmatrix}$$

The vertical point of the vanishing point is required at this point.

Using the orthogonality relationship between 4 couples of vanishing point (in the original image) the matrix ω is solved with respect to its parameters.

To do so, the couples of vanishing points used were:

$$v_1 \bar{\omega} v_2 = 0$$
$$v_1 \bar{\omega} v_6 = 0$$
$$v_4 \bar{\omega} v_5 = 0$$
$$v_4 \bar{\omega} v_6 = 0$$

Then, with the relationship holding between K and ω, the parameters are extracted.

$$K = \begin{bmatrix} 0.5774 & 0 & 0.4922 \\ 0 & 0.8239 & -0.2097 \\ 0 & 0 & 1 \end{bmatrix}$$

Notice that coordinates are normalized to 1 by the element `imref2d`.

5. Localization

This step consist in determining the relative pose between the camera and the reconstructed plane.

As a reference point for the plane has been selected, back in step 3, the point at the intersection between line 1 and 2.

To localize the camera with respect to a given plane on the image, the calibration matrix K and the homography between the plane in the image and in the real world are needed.

Both have been already calculated

$H_{omo} = \left(H_{rect} * H_{ar}\right)^{-1}$ (in the MATLAB script also the intermediate rescaling has been considered of course)

Thus, the 2 directions and the origin of the reference of plane π are computed as:

$$\left[i_\pi | \quad j_\pi | \quad o_\pi\right] = K^{-1} H_{omo}$$

Rotation matrix between the camera axis and the plane is

$$R = \left[i_\pi \quad j_\pi \quad k_\pi\right] = \left[i_\pi \quad j_\pi \quad i_\pi \times j_\pi\right] = \begin{bmatrix} 0.9684 & 0.1977 & 0.1525 \\ -0.2686 & 0.8040 & 0.6000 \\ -0.0040 & -0.6220 & 0.8316 \end{bmatrix}$$

While the position of the camera with respect to the reference system of the plane is:

$$T = -R^{-1} o_\pi = \begin{bmatrix} 0.1126 \\ -0.1868 \\ -0.1223 \end{bmatrix}$$

Some small remarks here, the rotation matrix should be a rigid rotation, the components should be linearly independent and with norm 1. Unfortunately it isn't a perfect rigid motion, I guess it's due to some small numerical error that has been brought forward fro the initial step of line selection.

If it were, doing the inverse or the transpose would result in the same matrix, i.e., the inverse rotation, from the plane to the camera.

Plane position with respect to camera system:

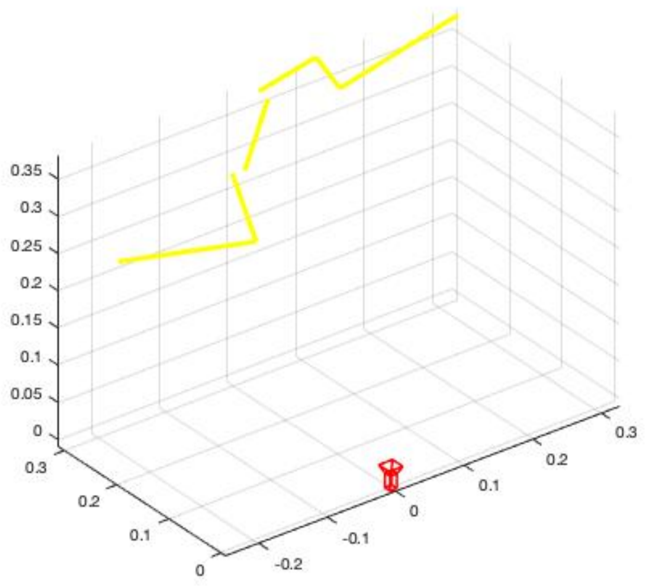

Fig. 9a plane wrt to camera reference

In this figure is represented how the camera sees the plane, so the z axis, represent the distance of a point from the camera center. In this reference system, the point O$_\pi$ is the closest one to the camera.

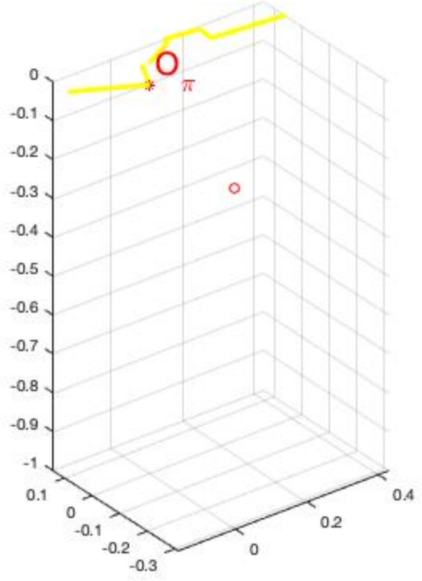

Fig. 9b camera wrt to point Oπ

Figure 9b instead shows where is the camera with respect to the point set as reference point for plane π in step 3. It seems to be correct as the position is below the plane and almost in front of the intersection of lines 1 and 2. More images of the scene:

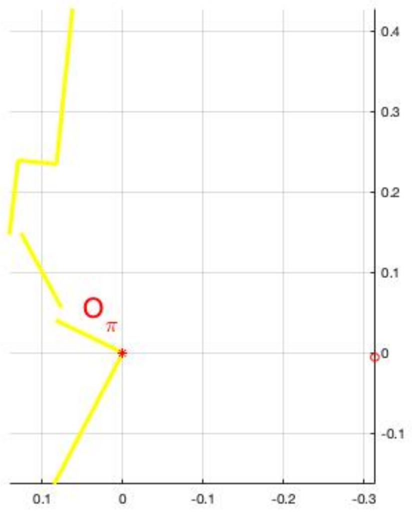

Fig. 9c scene from above

6. Reconstruction

To extract a vertical facade (with line number n of the direction on plane π), the transformation that maps the original circular point of the requested plane to the ones on the image of that plane is needed.

First step is to find the circular points on the image, these are the ones that solves these two equations:

$$l_{\infty n}{}^T x = 0$$
$$x^T \omega x = 0$$

While ω it's known from step 4, the line at the infinity for that plane has to be computed for the requested plane. This is done by crossing the vanishing point at the infinity and the one on the direction of the plane we are interested in.

$$l_{\infty n} = v_v \times v_n$$

The solutions at these equations are two imaginary points I' and J' .

This points are related to the original I and J with:

$$I' = H_{vert} I$$

Then, when $inv(H_{vert})$ is applied to the original image the vertical facade is obtained.

The result is an image where the vertical lines and the ones parallel to line n are parallel, but this image is rotated and the vertical lines are not vertical.

Then the same transformation $inv(H_{vert})$ is applied to the set of the vertical lines extracted from the original image, the inclination of the vertical transformed line is got as $\theta = atan\left(-\frac{x_1}{x_2}\right)$ and a rotation of $-\theta - \frac{\pi}{2}$ is automatically applied to the image.

The result:

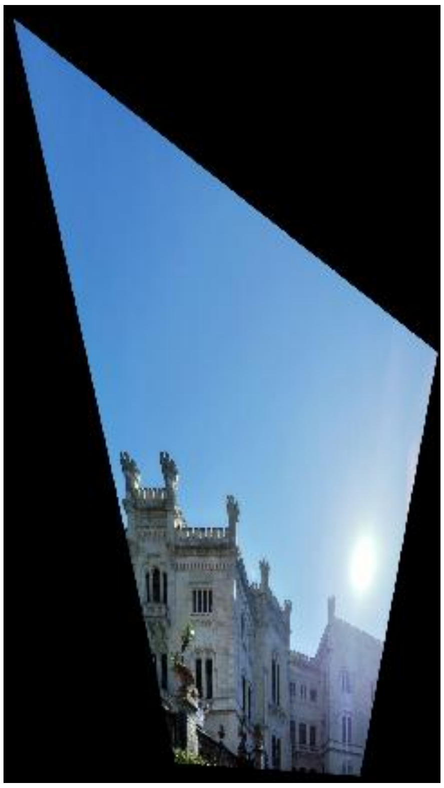

Fig. 10a vertical rectification facade 1

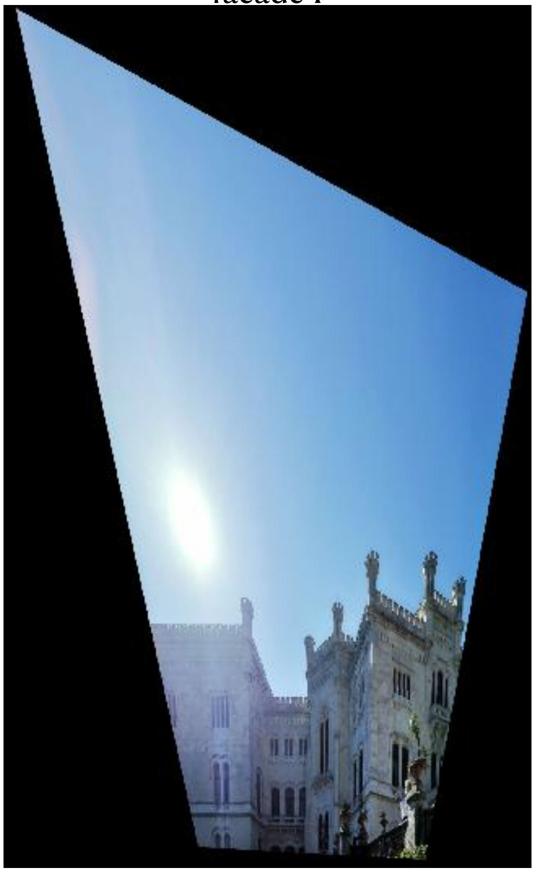

Fig. 10b vertical rectification facade 4-6

7. Final remarks

All the extracted images show an aspect that is quite close to the desired one, small errors are due to imperfection on the selection of lines.

An example is figure 10b, a perfect vertical rectification would have not shown facade 5.

In the end all the results seems to be adequate and can also intermediate steps can be validate, like matrix K, otherwise the subsequent step would have been wrong. Also camera position make sense, it's almost in front of point O_π and also well below the plane.

At any step of the process, there is the possibility to go back to pixel coordinates, indeed it's sufficient to multiply for the inverse of the matrix that has been applied at step 1, at the point definition in the plane. This has to be done before calculations. This matrix is just

$$\begin{bmatrix} WR.PixelExtentInWorldX & 0 & 0 \\ 0 & WR.PixelExtentInWorldX & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Indeed, to better manage this kind of transformation and keep the script ordered I created a couple of classes to manage in a very standard way the points and line.

This helped me in reducing the lines of code needed, with some for cycles and a smart construction of the sets of lines and points I was able to do everything while keeping the script very easy and understandable.

Last remarks goes on the selection of lines, since it is key for all the subsequent steps.

I selected them during the image processing parts, and some small errors are clearly visible. If I were to choose them, for example, starting from the corners detected in Harris algorithm, I could have improved most of the results, but I should have picked them manually and doing some steps at the beginning that I really didn't want to do.