Homework Image Analysis and

Computer Vision

Sofisti Giorgio (10500171)

giorgio.sofisti@mail.polimi.it

Politecnico di Milano

A.Y. 2019/2020

January 27, 2020

Contents

1 Introduction

Images provided for this homework are taken inside Sciarra palace's internal courtyard situated in Trevi neighborhood, Rome. The palace built between 1885 and 1888 is a great representation of Liberty architecture with complex, flower inspired, decoration over all the facades. The inside courtyard is covered by a glass roof that let the sunlight penetrate inside this connecting area between two streets. The two images provided shows one internal façade from the bottom of the court. Since the images are taken during a sunny day (probably around noon) there is a high contrast between areas illuminated by sunlight, that penetrates through the glass roof, and the areas that are in the dark, a point to consider during edge detection. Another point that introduce complexity for features detection are for sure the complex decorations that are present on the facades.

Last but not least we have to consider the assumption given in the homework description about metric dimension of windows (fundamental for metric reconstruction step), orthogonality of adjacent façades and skew symmetry assumption (but not natural camera assumption) fundamental when we need to look for K calibration matrix.

In the picture below is a representation of the main façade to be rectified form another view point that can be useful for double check the results.

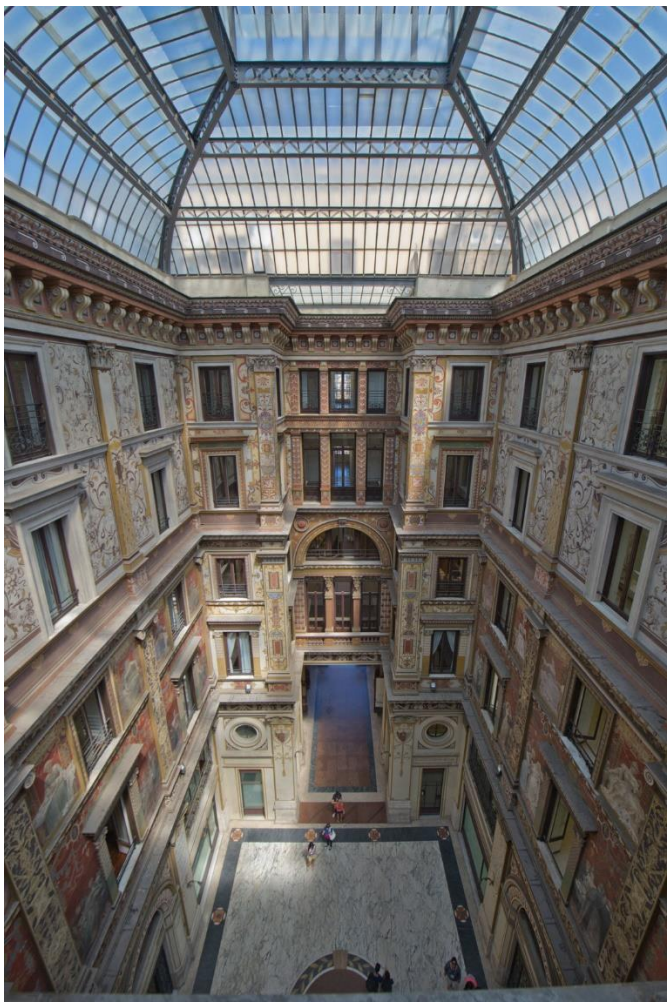


Figure 1 Image of main facade form another view-point.

2 Features Extraction

In order to extract relevant features, form the image I have to perform a series of steps:

1. In features extraction colors are useless information to carry on that complicates and worsen the performance so we can apply `rgb2gray(image)` to transform the image form RGB to Gray Scale.
2. As mentioned in the introduction, since the image presents areas with different exposition, we need to perform a pre-processing phase to improve accuracy and obtain better result in next phases. To do that we tested different normalization algorithms and evaluated that MATLAB function `adapthisteq(image)` applied twice on the image returns the best result.
3. Now I'm ready to apply to the image edge extraction algorithm. After some testing of different option (Sobel, log, Roberts, canny) we opted for Canny method that is the most accurate. Canny is accurate but the threshold imposed is the result of many trial and error and is fundamental for good performance of the next step. After canny I've applied the sliding filter `bwareaopen(BW,P)` to remove small white areas in the image and reduce the complexity and possible error in the next step.

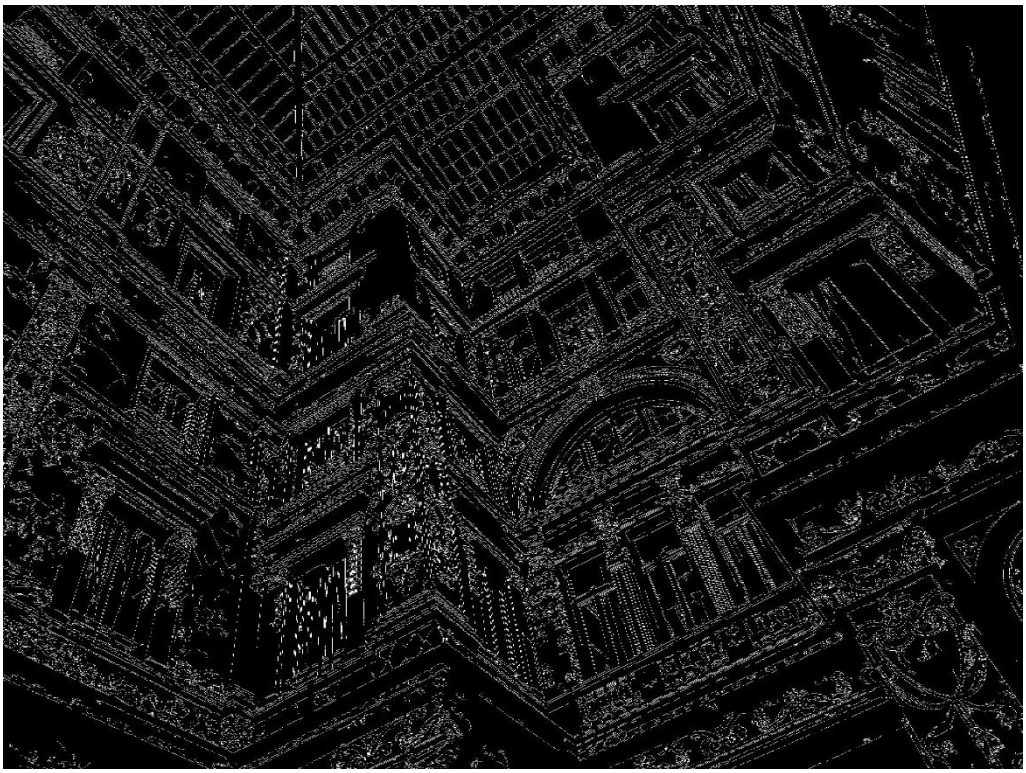


Figure 2 Canny plus filtration

4. Now I’m ready to use the learned method of Hough transform to extrapolate from canny image relevant edges. After some parameter tuning the result in the picture below.

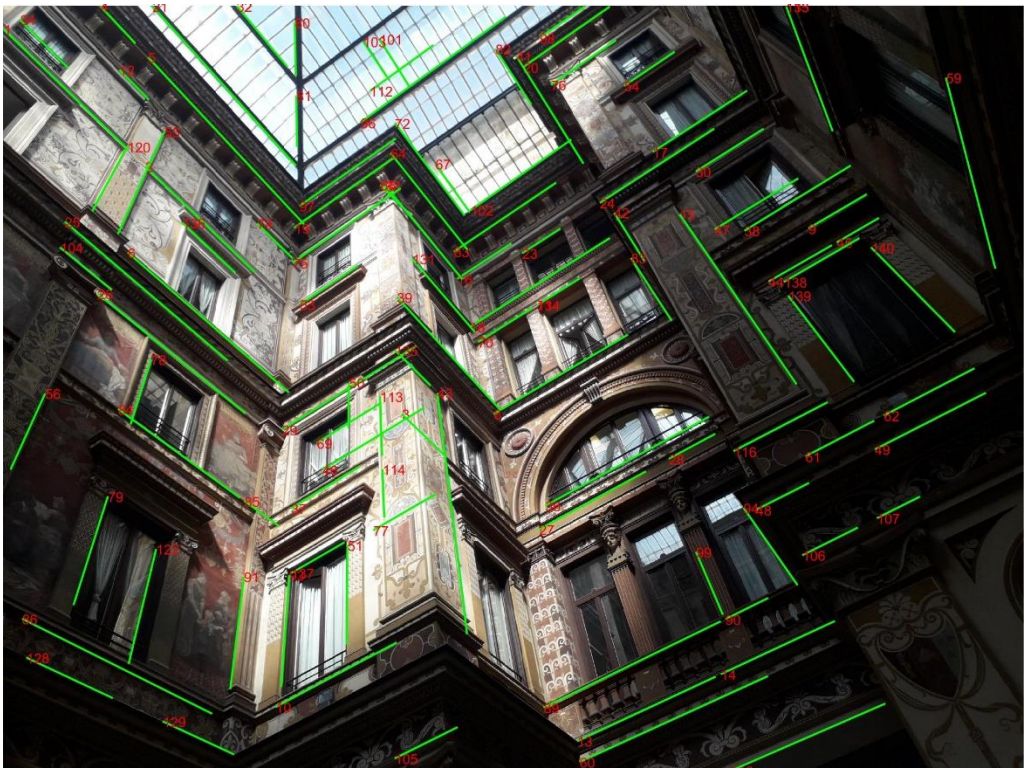


Figure 3 Edges found appling Canny and Hough method.

I’ve also used Harris corner detection algorithm to extract useful information but, as it possible to see in the image below, due to image complex shape and façade decoration, the result is poor and do not add information over the output of edge detection. We can clearly see that Harris, using more than 10000 relevant point, underlines some façade features following them but also a lot of sparse corner are found over the façade in flat decorated areas.

Figure 4 Corner detection with Harris.

For a better but much more longer step by step visualization of all the steps performed enable debug mode by setting it to true in the main.m file.

3 Shape reconstruction

Now that we have obtained some relevant image features, we are ready for image reconstruction. The process involves the application of two subsequent transformation one after the other that maps the original image to the affine one ad finally to the Euclidean.

$$\boxed{Given\ Image} \xrightarrow{H_{r\,aff}} \boxed{Affine\ Reconstruction} \xrightarrow{H_{a\,e}} \boxed{Euclidean\ Reconstruction}$$

3.0 Vanishing Points and Lines

After the selection of relevant features for each direction (x, y, z) I've computed the vanishing lines and found three best approximation of vanishing points as weighted average of the vanishing points given by the intersection of vanishing lines in the respective directions. An important factor to be consider to obtain reliable and good results is to choose as many features as possible for each direction (at least 3-4 edges, definitely 2 are not enough) so that the approximation is more accurate. Below is shown the result obtained during the computation. In blue you can see vanishing lines, in black all computed vanishing points and red cross are the three selected vanishing point computed as a weightedirection.d average between vanishing point of the same

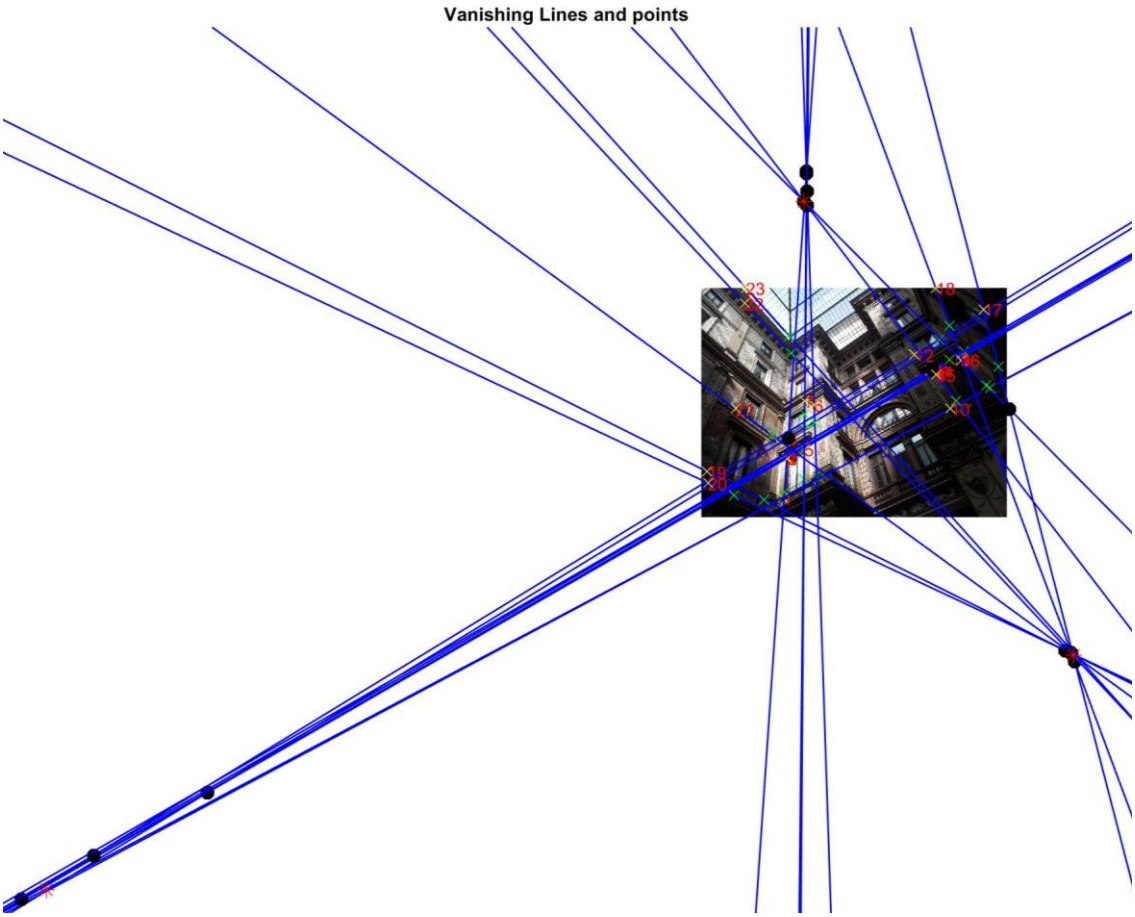


Figure 5 Vanishing line and points.

3.1 Affine property recovery

An affine transformation is a non-singular linear transformation followed by a translation. Since an affine transformation includes non-isotropic scaling, the similarity invariants of length ratios and angles between lines are not preserved under affinity. Affinity invariance instead are parallelism between lines, the length ratio between parallel segments and the consequent ratio between areas.

In order to identify the correct transformation matrix for recovering affinity property I've found the line at the infinity passing through two of the three found vanishing points found at the previous step (I have to choose the two vanishing point based on which façade I want to rectify, in my case I have chosen the two bottom ones to rectify the image in the plane parallel to ground, for better understanding of next steps I'll call it XZ). [pag. 49 Multiple View Geometry in CV]

$$H_{r\,aff} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

Where the last row is built using the tree values of the found line at the infinity.

As you can see now the lines on XZ direction are represented also in the image as parallel lines that intersect in the point at infinity instead of meeting in the vanishing points (main façade, direction Y (vertical direction) is not rectified since we're not interested for now to work on that).

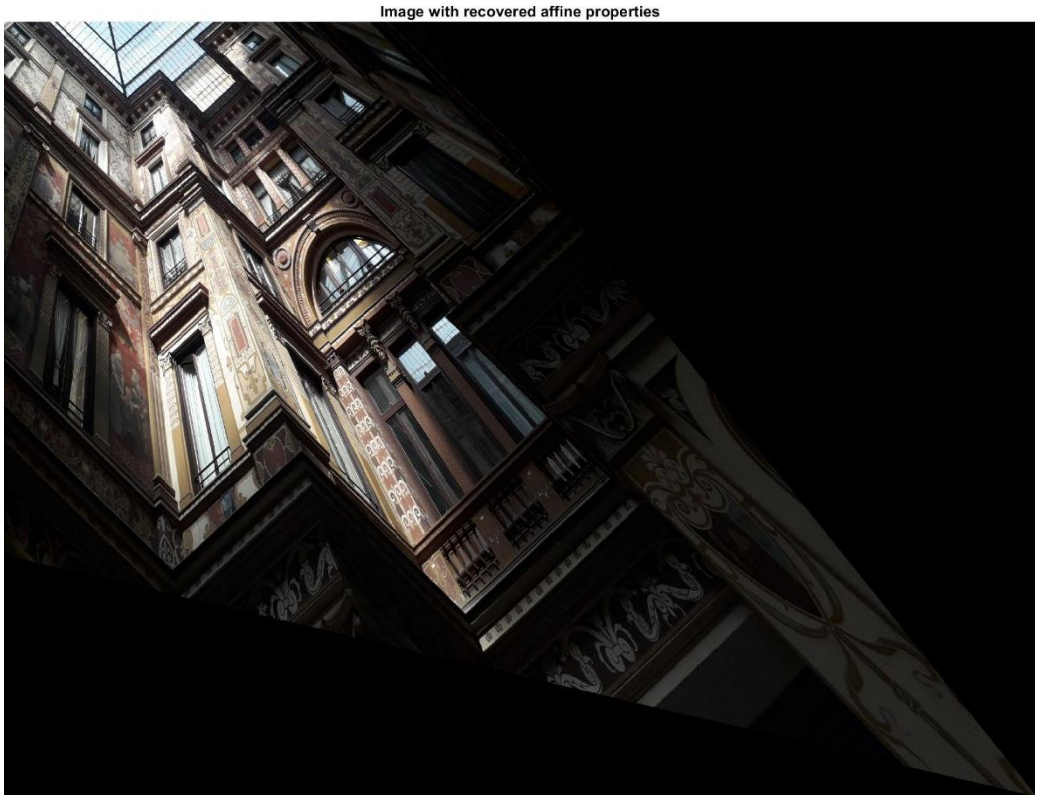


Figure 6 Image with recovered affine properties.

3.2 Euclidean rectification

Now that I've recovered affinity property the next step is to impose orthogonality between façade. In order to perform this operation, we have to build another transformation matrix imposing two conditions [pag. 54 Multiple View Geometry in CV]:

$$\cos(\theta) = \frac{l_1 m_1 + l_2 m_2}{\sqrt{(l_1^2 + l_2^2)(m_1^2 + m_2^2)}}$$

$$\cos(\theta) = \frac{l^T C_\infty^* m}{\sqrt{(l^T C_\infty^* l)(m^T C_\infty^* m)}}$$

$$\cos(\theta) = \frac{l^{T\prime} C_\infty^* m'}{\sqrt{(l^{T\prime} C_\infty^* l')(m^{T\prime} C_\infty^* m')}}$$

I've basically found two couple of lines, chosen form the edge detection step, in X and Z direction that are orthogonal in reality (l_1, m_1, l_2, m_2) and imposed that $\cos\theta = 0$ since should be 90° apart. I've computed the solution of the linear system finding $C_\infty^{*\prime}$ the conic at the infinity and then using single value decomposition I've found the three matrices U, S, V and used them to build $H_{a\ e}$ that maps the affinity to an Euclidean reconstructed image.

$$svd(C_\infty^{*\prime}) = USV^T = H_a C_\infty^* H_a^T$$

Then we decompose S as:

$$S = S_{fact} C_\infty^* S_{fact} = \begin{bmatrix} \sqrt{S_{11}} & 0 & 0 \\ 0 & \sqrt{S_{11}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \sqrt{S_{11}} & 0 & 0 \\ 0 & \sqrt{S_{11}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And we obtain:

$$USV^T = US_{fact} C_\infty^* S_{fact} V^T = H_a C_\infty^* H_a^T$$

Finally, we find:

$$H_a = US_{fact}$$

3.3 Additional transformation applied

For a better visualization I've applied a rotation to the image of 92° in order to have X axis parallel to the image and I've scaled the image to reduce computation time and space requirements that otherwise would have overtaken computer specs.

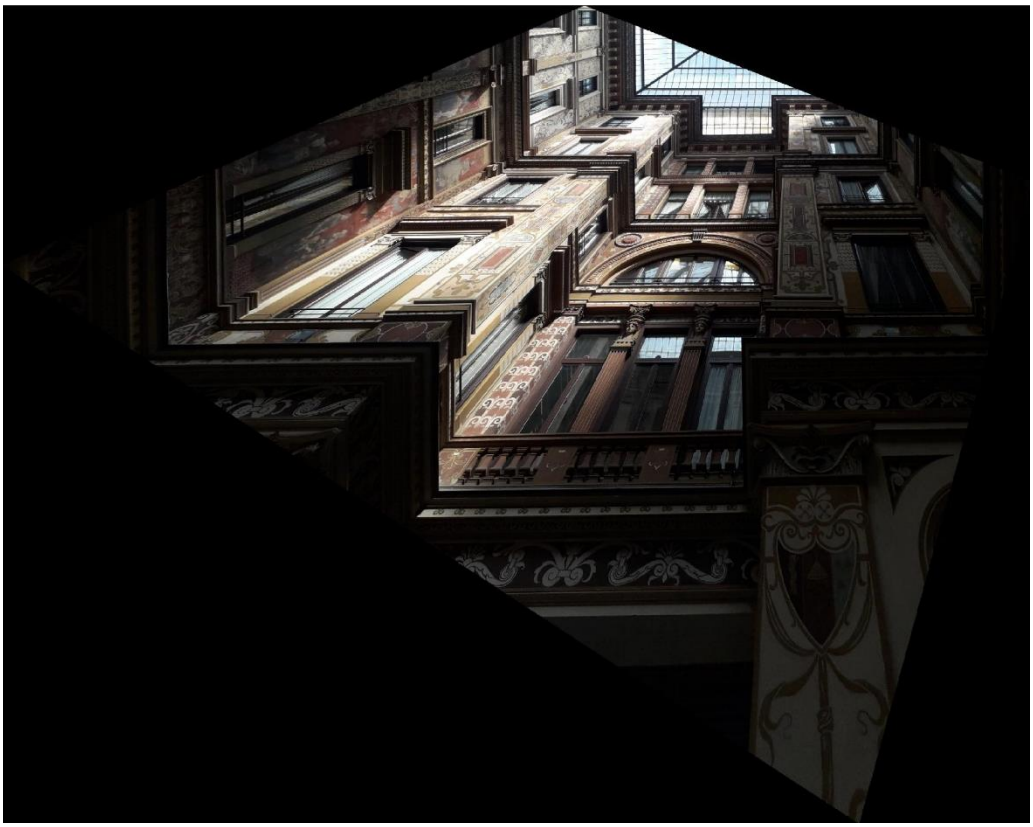


Figure 7 Euclidean reconstruction.

3.4 Metric property recovery

Based on information provided on window size I've imposed that the dimension of coplanar top window size should measure the same length in pixel and applied a non-isomorphic scaling. This basically means that:

$$l' = H_r^{-T} l$$

The composition of all the transformation matrix applied maps the original image to the final metric reconstructed one.

In this final image the window now are represented with the top edge with the same length in pixel. Measuring the resulting façade pixel dimension I computed that the width is around 9.7m.

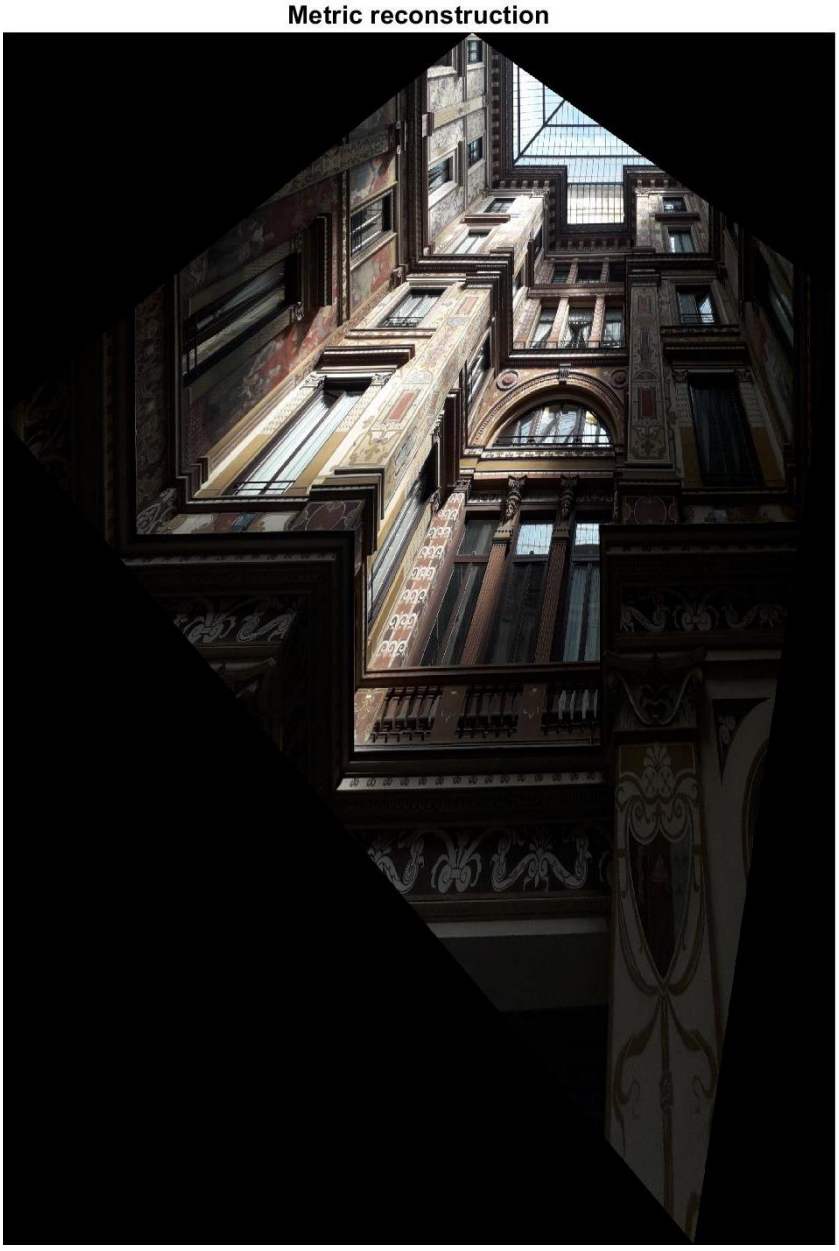


Figure 8 Metric reconstruction.

4 Camera calibration

In this step we want to estimate the camera parameters knowing that the camera is skew-symmetric. Based on the theory [pag. 224 Multiple View Geometry in CV] we need tree vanishing points and at least one more constraint since I cannot assume natural camera. I've used the constraint derived from the H matrix since we are in the case of metric plane with known homography (computed in previous steps).

K matrix assume the form of:

$$K = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

K follow the relation:

$$\omega = (KK^T)^{-1}$$

In order to compute K I've imposed 4 constraint plus skew-symmetry constraint to ω, image of the absolute conic [pag. 226 Multiple View Geometry in CV]. In our case K will assume the form of the symmetric matrix:

$$\omega = \begin{bmatrix} \alpha^2 & 0 & -u_0\alpha^2 \\ 0 & 1 & -v_0 \\ -u_0\alpha^2 & -v_0 & f_y^2 + \alpha^2 u_0^2 + v_0^2 \end{bmatrix}$$

We can now apply the reconstructive transformation [pag. 211 Multiple View Geometry in CV] on the horizontal façade. The three steps are:

(i) For each square compute the homography H that maps its corner points, to their imaged points.

(ii) Compute the imaged circular points for the plane of the square as $H(1, \pm i, 0)^T$. Writing $H = [h_1, h_2, h_3]$, the imaged circular points are $h_1 \pm ih_2$.

(iii) Fit a conic ω to the six imaged circular points. The constraint that the imaged circular points lies on ω ma be rewritten as two real constraints.
If $h_1 \pm ih_2$ lies on ω then $(h_1 \pm ih_2)^T \omega (h_1 \pm ih_2) = 0$ and the imaginary and real parts give respectively:
$$h_1^T \omega h_2 = 0, h_1^T \omega h_1 = h_2^T \omega h_2$$
Which are equation linear in ω. The conic ω is determined up to a scale form five or more such equations.

(iv) Compute the calibration matrix K from $\omega = (KK^T)^{-1}$ using Cholesky factorization.

Just for reference during the computation I've also computed the K_n matrix imposing natural camera and skew-symmetry constraint. We are close to the natural camera case.

The computed calibration parameters are:

$$K = \begin{bmatrix} 3731.6 & 0 & 2654.6 \\ 0 & 3230.9 & 1770.6 \\ 0 & 0 & 1 \end{bmatrix} \qquad K_n = \begin{bmatrix} 3333.7 & 0 & 2051.7 \\ 0 & 3333.7 & 1652.6 \\ 0 & 0 & 1 \end{bmatrix}$$

5 Main façade reconstruction using K

Then the homework requires to use the knowledge of K to reconstruct the main façade. To perform this operation, I've built a matrix from K capable of leveraging on the information fitted inside to map the original image to a rectified one over the XY plane. The matrix $H_{support}$ has the form of:

$$H_{support} = \begin{bmatrix} 0 & 0 \\ K_{11} & 0 \\ 0 & K_{22} \\ K_{11} & K_{22} \end{bmatrix}$$

Using this matrix, I've built the transformation matrix H_{main} that projects the façade points and rectify them. The resulting image is a Euclidean reconstruction of the central façade itself and has been scaled down significantly for reducing computation time. The image reported here is a small crop of the right button corner of the computed image since the original picture shows the façade from ground up and the high inclination of the viewpoint cause the ratification to increase significantly the dimensions of the image and stretches it a lot.

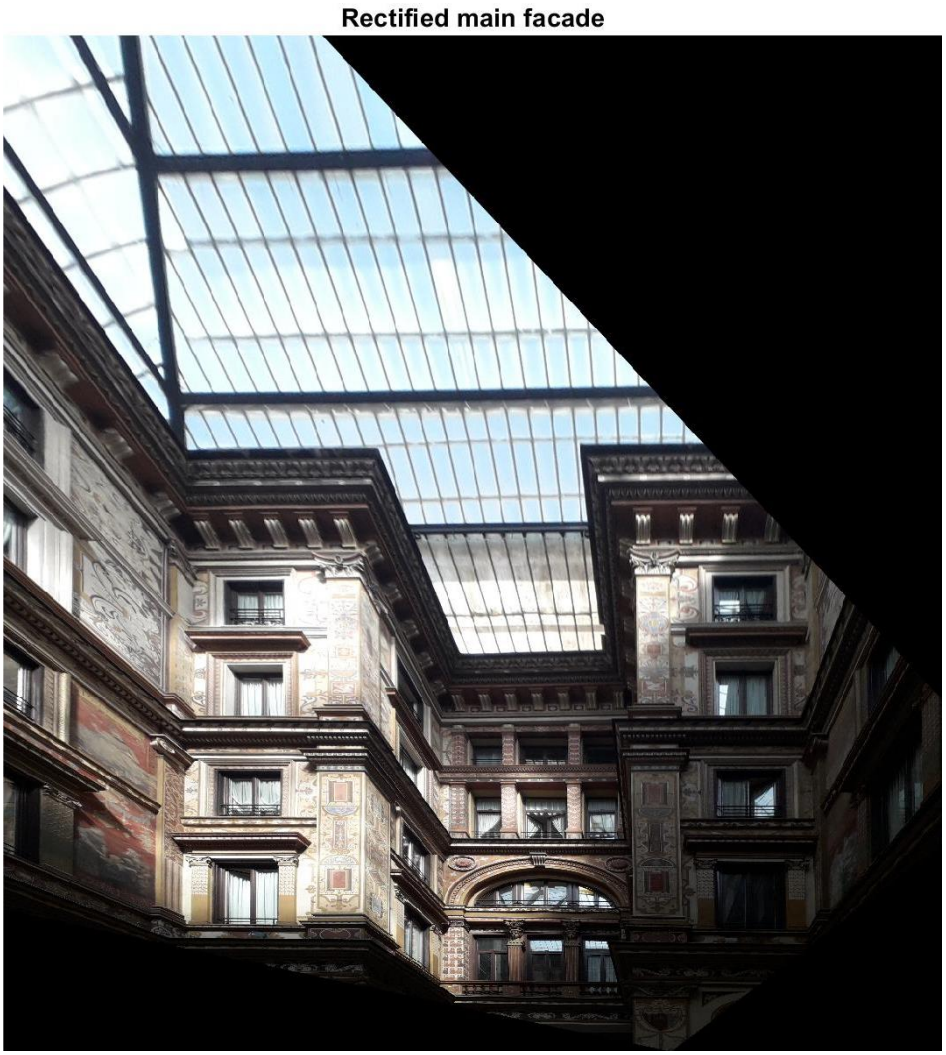
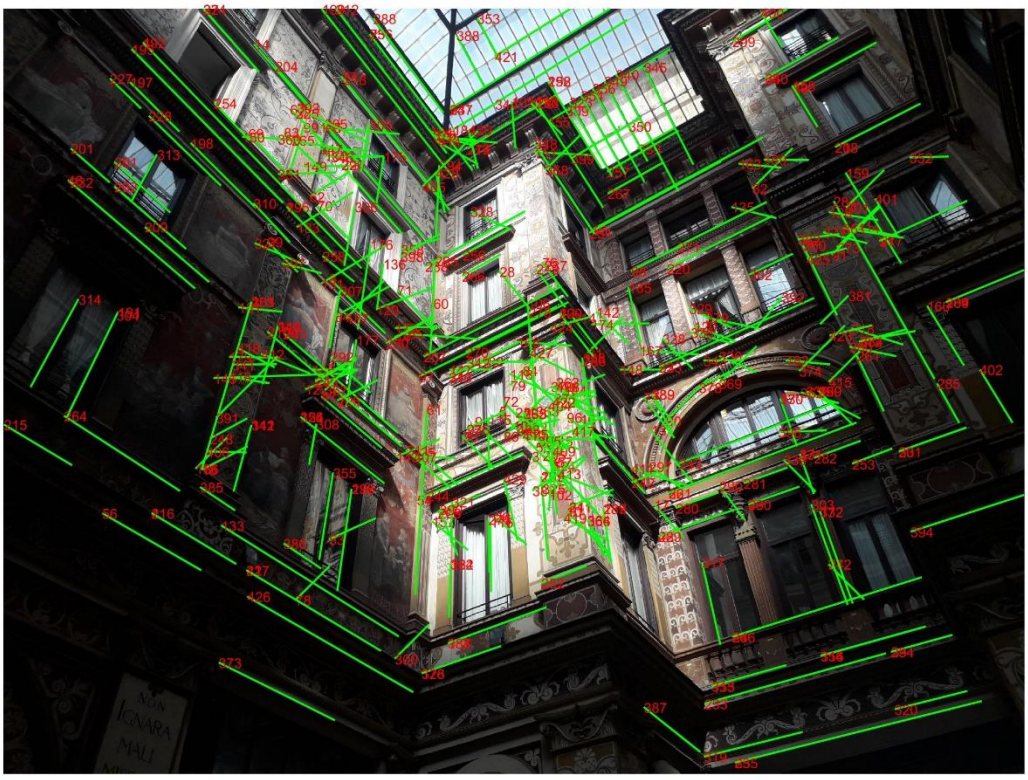
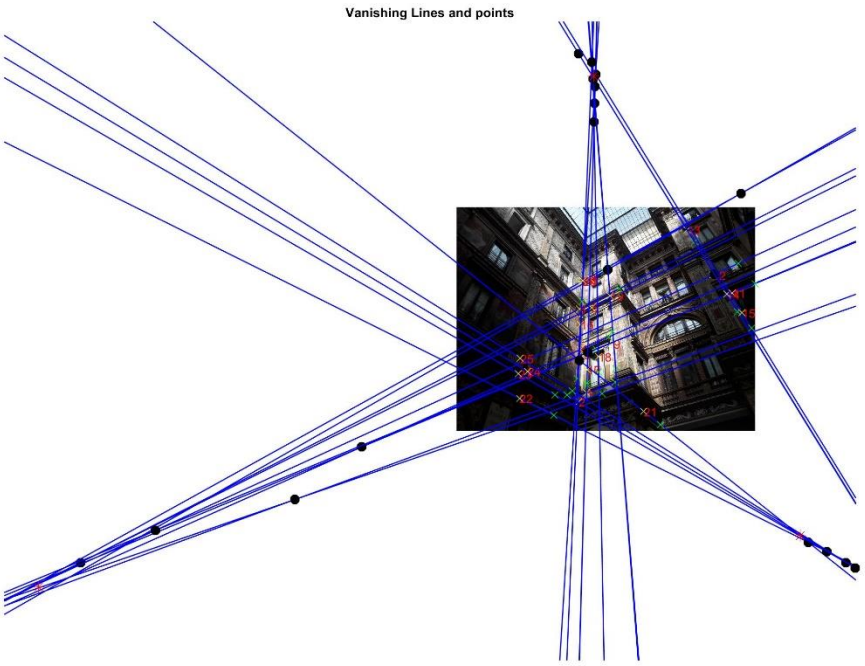


Figure 9 Rectified image using K.

6 Comparison with second image

Setting `ceck_with_second_image= True` the algorithm will run on the second image. The result are reported below. The algorithm is not optimize so edge detection shows more edges than the first case.





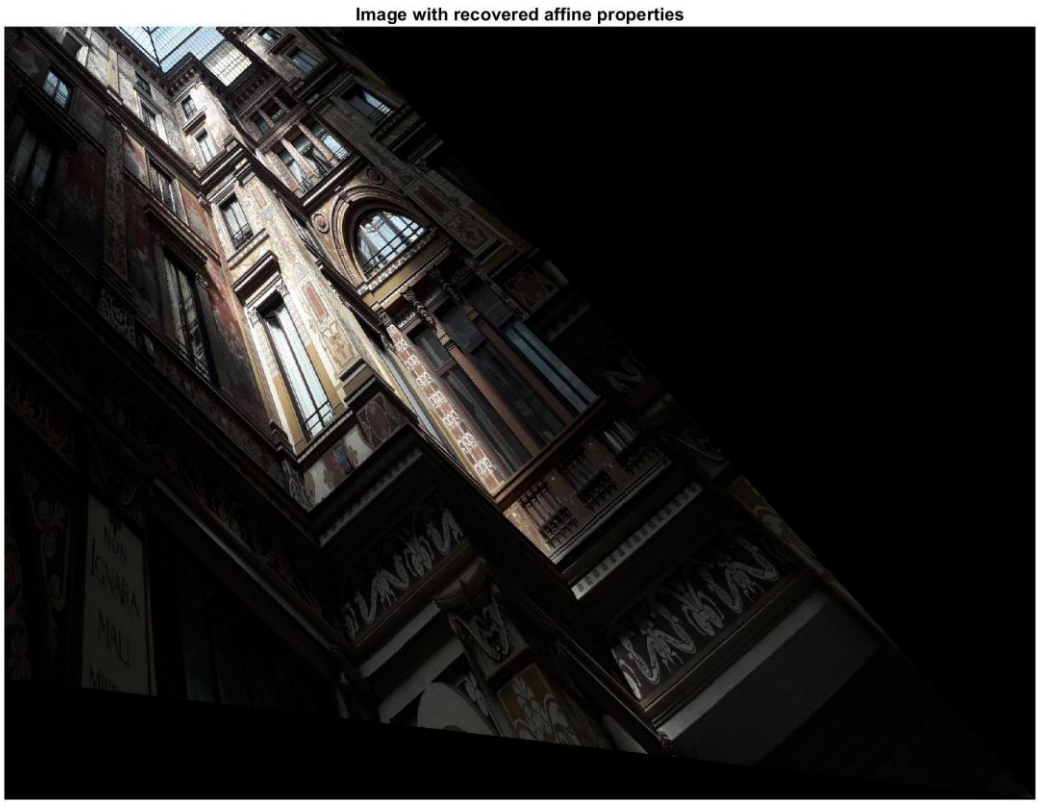


Image with recovered affine properties

8 Camera position estimation

The last point of the homework requires to estimate and show the possible position and orientation of the camera that has taken the picture. Of curse we espect a ground clearance between 1.8 to 2 m and the orientation is from ground up pointing almost orthogonally to the ground. This is possible knowing the shape of the horizontal facade, knowing the size, knowing the image and knowing K. If we identify the plane XZ parallel to ground as π we can write the position of a point in the world reference frame as $X_\omega = [R_\pi|o_\pi]X_\pi$ where X_π is the position of the point in the plane reference frame. The plane reference frame has the x axis identical to the line identified by the lecturer in the homework description (the one passing through the top corner of the window of the main façade), the y axis is the perpendicular to the ground and the z axis is the depth.

In this way a point on the plane can be written (in homogeneous coordinates) as:

$$X_\pi = \begin{bmatrix} x \\ 0 \\ z \\ \omega \end{bmatrix}$$

I can the compute:

$$u = [KR|-KRo][R_\pi|o_\pi]X_\pi$$

Imposing the frame of the camera as world reference we than can compute the $H_{omog} = [h_1; h_2; h_3]$. Then can be computed the rotation matrix R using the equation:

$$\lambda = \frac{1}{|K^{-1}h_1|}$$

$$i_\pi = K^{-1}h_1\lambda$$

$$j_\pi = K^{-1}h_2\lambda$$

$$k_\pi = i_\pi \times j_\pi$$

$$o_\pi = K^{-1}h_3\lambda$$

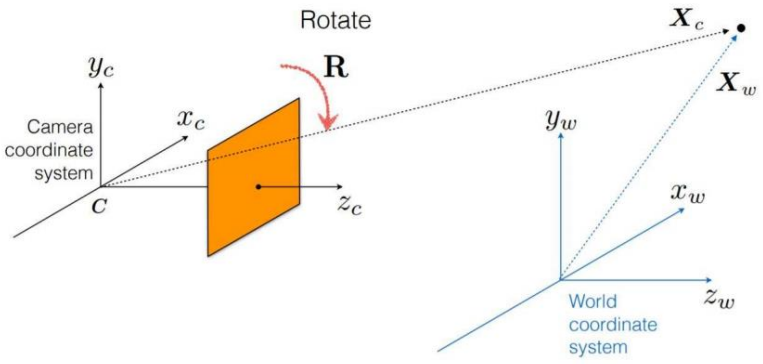


Figure 10 Camera pose estimation

9 References

"Multiple View Geometry in Computer Vision", R. Hartley, and A. Zisserman. Cambridge University Press, New York, NY, USA, 2 edition, (2003).

http://www.cs.cmu.edu/~16385/s17/Slides/11.3_Pose_Estimation.pdf