POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

Nicolò Tombini
Person Code: 10912627
Course: IACV
Professor: Caglioti Vincenzo

Contents

Chapter 1

Introduction

The image provided as the subject of the homework comes from the Palazzo Te in Mantova: it is a historic architectural marvel, known for its stunning design and rich cultural significance, a remarkable palace that boasts impressive architecture and holds a notable place in the region's history. Specifically, for this homework was given an image depicting a cylindrical vault.

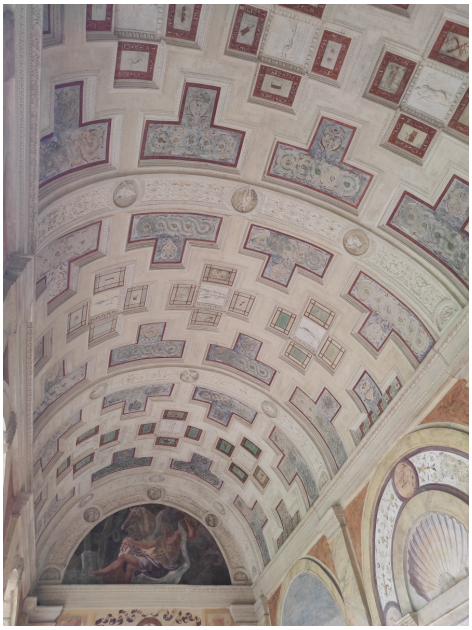


Figure 1.1: Palazzo Te - Mantova

1.1 Image

An uncalibrated, zero-skew camera captures a single image of the aforementioned cylinder. The calibration matrix of the camera relies on four unknown parameters, specifically f_x, f_y, and the two coordinates U_0, V_0 of the principal point. Within the image, two visible circular cross sections yield their respective images $C1$ and $C2$. Additionally, two parallel generatrix lines of the cylinder are observable, and their images $l1$ and $l2$ are extracted.

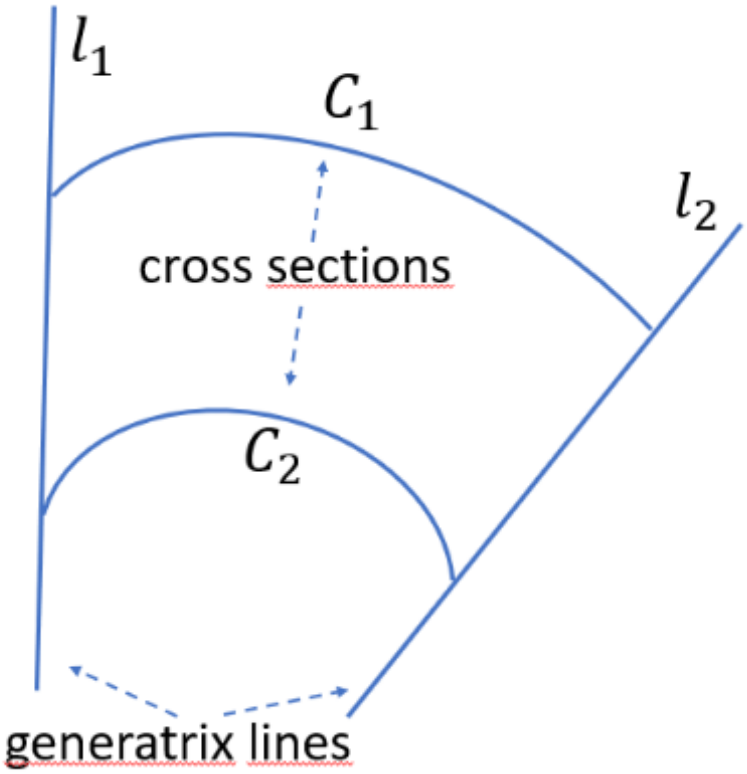


Figure 1.2: Sketch of Palazzo Te - Mantova

Chapter 2

Feature Extraction

Request 1. *Consider the image PalazzoTe.jpg. Using feature extraction techniques (including those implemented in Matlab) plus possible manual intervention, extract both the images* l_1, l_2 *of useful genetrix lines and images* $C1$, $C2$ *of useful circular cross sections.*

2.1 Initial preprocessing

Initially, since colors are unnecessary for the information to be extracted, the image was converted from RGB to grayscale. Subsequently, to enhance contrast and achieve better results in the following stages, it was crucial to apply the adapthisteq function twice to the grayscale image.

```
1 img_gray_scale = rgb2gray(im_rotated);
2 img_gray_norm1 = adapthisteq(img_gray_scale);
3 img_gray_norm2 = adapthisteq(img_gray_norm1);
```

Obtaining this first result:

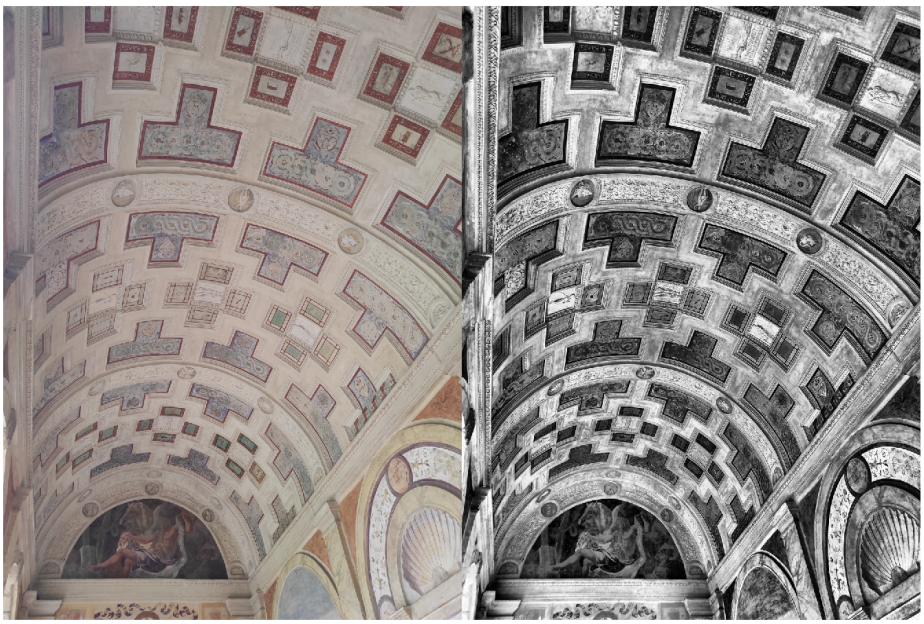


Figure 2.1: Original vs 2 times adapthisteq

2.2 Edge detection

Regarding the approaches used, the decision was made to test several methods for edge detection:

- Canny
- Roberts
- Sobel
- Log

However, the best method that emerged for proceeding was found to be Canny edge detection.

2.2.1 Canny algorithm

The Canny algorithm finds edges by looking for local maxima of the gradient of the input image. The edge function calculates the gradient using the derivative of a Gaussian filter. This method uses two thresholds to detect strong and weak edges, including weak edges in the output if they are connected to strong edges. By using two thresholds, the Canny method is less likely than the other methods to be fooled by noise, and more likely to detect true weak edges.

Clearly some tuning (in this case a simple grid search) was performed to select the best treshold parameters: this was necessary in order to achieve better results in the next phases, like lines detection.

The best way to vary the threshold is to run **edge** function once, capturing the calculated threshold as the second output argument. Then, starting from the value calculated by **edge**, adjust the threshold higher to detect fewer edge pixels, or lower to detect more edge pixels. Important was also the parameter sigma, that allows us to specify the standard deviation of the Gaussian filter, in order to obtain less noisy edges.

```
1  threshold_values = [0.7, 0.8, 0.9, ..];
2  [BW2, th] = edge(img,'canny');
3  for i = 1:length(threshold_values)
4      for j = 1:length(threshold_values)
5          th2 = th.*[threshold_values(i),
               threshold_values(j)];
6          BW2 = edge(img,'canny', th2);
7          BW2 = bwareaopen(BW2, 80);
8          figure;
9          imshow(BW2);
10     end
11 end
```

Obtaining the following result:

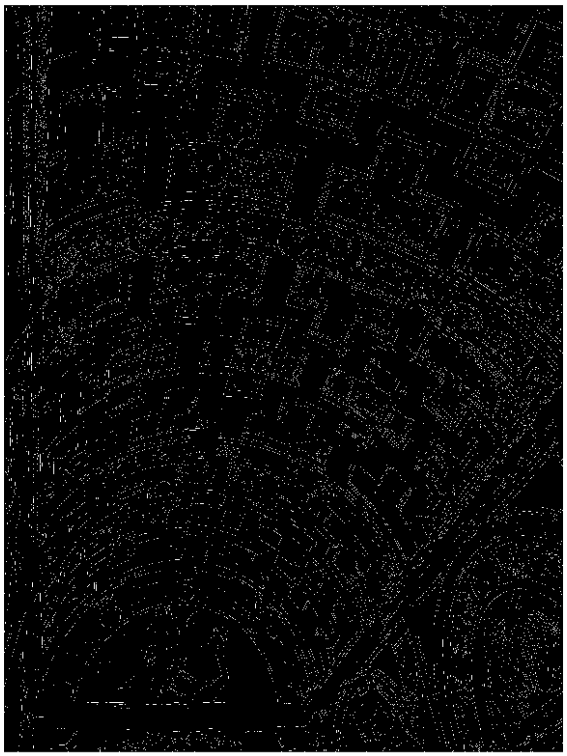


Figure 2.2: Canny edge detection

2.3 Lines detection

For the lines detection task, the Hough transform was employed twice: the first time to extract the two generatrix lines, and the second time to extract a greater number of lines across the entire image. Also here was performed trial and error in terms of selecting parameters in order to identify the best lines in both situations.
The HT uses the parametric representation of a line:

$$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta) \tag{2.1}$$

The function returns ρ, the distance from the origin to the line along a vector perpendicular to the line, and θ, the angle in degrees between the x-axis and this vector. It also returns the H, which is a parameter space matrix whose rows and columns correspond to ρ and θ values respectively.

```
[H, Theta, Rho] = hough("params");
peaks = houghpeaks("params");
lines = houghlines("params");
```

Resulting in:

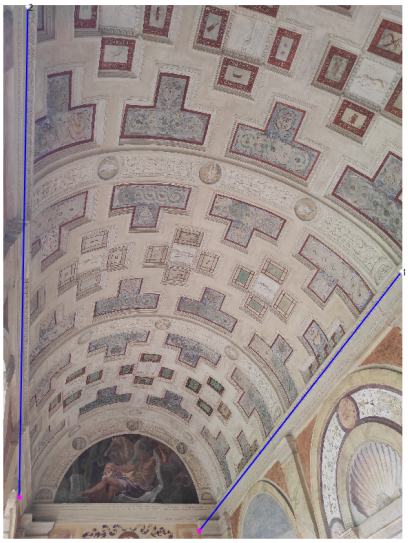


Figure 2.3: Generatrix lines

Figure 2.4: More lines

The generatrix lines l1 and l2 are described by this 2 vectors:

$$\mathbf{l1} = \begin{bmatrix} -0.0002 \\ -0.0001 \\ 1.0000 \end{bmatrix} \quad \mathbf{l2} = \begin{bmatrix} -0.0047 \\ -0.0001 \\ 1.0000 \end{bmatrix}$$

Unfortunately, some noise is present in this lines because they are extracted automatically.

2.4 Ellipses detection

In order to extract the two ellipses of the cylindrical vaults, two approaches were employed: initially, an automatic selection with manual intervention to identify the two correct ellipses, and subsequently, a manual selection.

2.4.1 Regionprops

Initially, was made an attempt to automatically extract the two ellipses. To achieve this goal, the decision was made to use the Matlab **regionprops** function.

The regionprops function measures properties such as area, centroid, and bounding box, for each object in an image finding unique objects using 8-connected neighborhoods for 2-D images and maximal connectivity for higher dimension images.

In order to help regionprops to work properly, the image was converted to black and white with **imbinarize** Matlab function.

Initially, as we can see in the following image, a lot ellipses were detected:

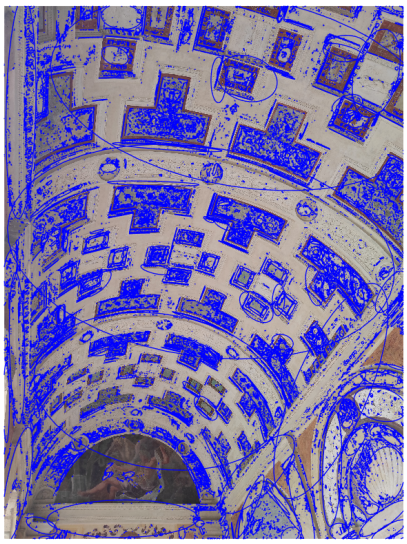


Figure 2.5: Full ellipses detection with regionprops

Secondly, it was decided to perform a manual intervention in order to extract only the meaningful ellipses for the task:

Figure 2.6: Useful ellipses detection with regionprops

The regionprops method employed for this task is shown below:

```
s = regionprops(image,{...
    'Centroid',...
    'MajorAxisLength',...
    'MinorAxisLength',...
    'Orientation'});
```

2.4.2 Manual selection

As it might be seen from the image above, the ellipses are not so well defined: therefore, to achieve better results and accuracy in the subsequent steps, it was decided to manually select the ellipses to minimize the existing noise. It was employed **drawellipse** Matlab function in order to extract the two conics: this function creates an Ellipse object that specifies the shape and position of an elliptical region of interest (ROI).

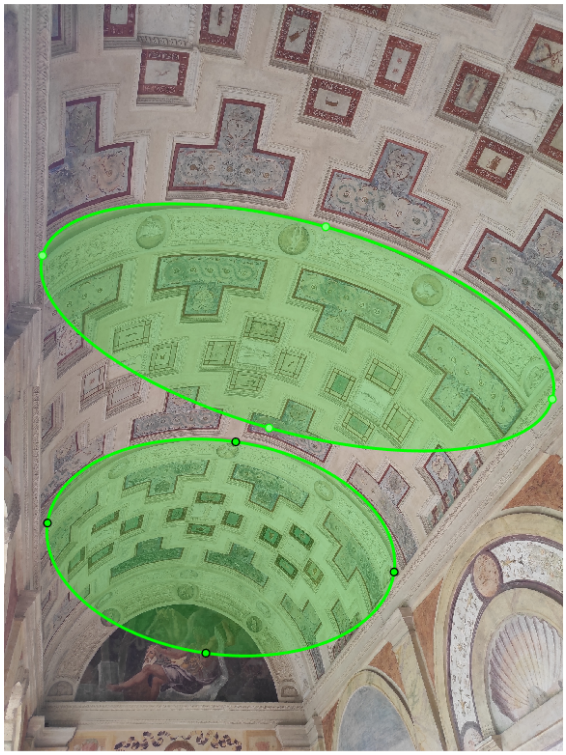


Figure 2.7: Manual ellipse detection

I tried not to intersect the ellipses in the image since, in 3D space, they belong to two parallel planes and, consequently, they would not intersect.

2.5 Corners detection

For corners detection task it was used the well-known Harris algorithm to extract corners and keypoints from the image, that is a gradient based method. Keypoints are expected to be in regions where the image is:

- Well-defined
- Stable

A point is interesting when the image content around there is dissimilar from the neighboring ones.
Resulting in:

Figure 2.8: Harris corners detection

Chapter 3

Geometry

3.1 Horizon line h

Request 2. *From $C1$, $C2$ find the horizon (vanishing) line h of the plane orthogonal to the cylinder axis.*

In order to obtain the horizon we have to rely on the ellipses images, $C1$ and $C2$. We know that every conics intersects the line at infinity l_∞ at two consistent points, I and J.
So, by intersecting $C1$ and $C2$ we will obtain I' and J', the image of circular points: from now on, all what we have to do is the cross between I' and J' to obtain the horizon h.
So from

$$\mathbf{C1} = \begin{bmatrix} a_1 & \frac{b_1}{2} & \frac{d_1}{2} \\ \frac{b_1}{2} & c_1 & \frac{e_1}{2} \\ \frac{d_1}{2} & \frac{e_1}{2} & f_1 \end{bmatrix} \quad \mathbf{C2} = \begin{bmatrix} a_2 & \frac{b_2}{2} & \frac{d_2}{2} \\ \frac{b_2}{2} & c_2 & \frac{e_2}{2} \\ \frac{d_2}{2} & \frac{e_2}{2} & f_2 \end{bmatrix}$$

we can write

$$\begin{cases} a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1 = 0 \\ a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1 = 0 \end{cases} . \tag{3.1}$$

from which we obtain as a solutions:

$$\mathbf{I}' = 1.0e+03 \cdot \begin{bmatrix} 5.1252 + 5.8742i \\ 0.2449 + 2.7322i \\ 0.0010 + 0.0000i \end{bmatrix}$$

$$\mathbf{J}' = 1.0e+03 \cdot \begin{bmatrix} 5.1252 - 5.8742i \\ 0.2449 - 2.7322i \\ 0.0010 + 0.0000i \end{bmatrix}$$

Finally, we are able to compute the horizon h as the cross product between I' and J', since the image of the line at infinity passes thorugh these 2 points:

$$h = I' \times J' \tag{3.2}$$

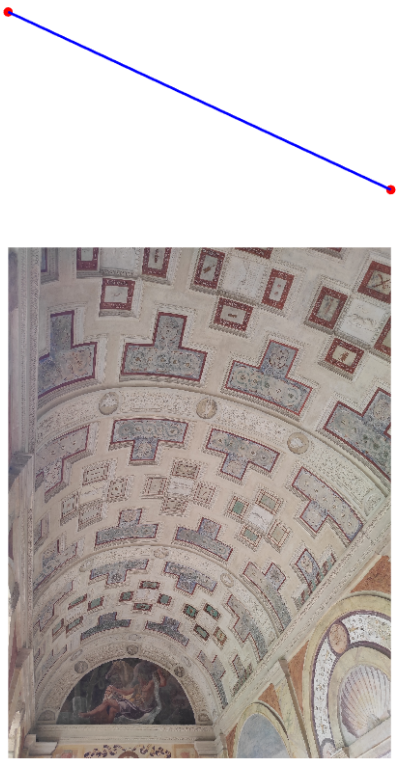


Figure 3.1: Horizon h from C1 and C2

The value of horizon h is shown below:

$$\mathbf{h} = \begin{bmatrix} 0.0002 \\ -0.0005 \\ -1.0000 \end{bmatrix}$$

3.2 Cylinder axis and vanishing point

Request 3. *From $l1$, $l2$, $C1$, $C2$, find the image projection a of the cylinder axis, and its vanishing point V.*

3.2.1 Cylinder axis a

Given a point y and a conic C in the plane, the line $l = Cy$ is called the polar line of point y with respect to the conic C.
In this case, the procedure to follow is: point y can be translated first as the center $c1$ of the conic section $C1$, and then as the center $c2$ of the conic section $C2$.
We know that the polar line of the center of a circumference is the line at infinity: so, in order to obtain y we have to convert $l = Cy$ in $y = C^{-1}l$.
For our purpose we have:

$$c1 = C1^{-1} \cdot h \tag{3.3}$$

and

$$c2 = C2^{-1} \cdot h \tag{3.4}$$

In my Matlab code it was:

```
1 c1 = inv(C1) * l_inf_1;
2 c2 = inv(C2) * l_inf_1;
3 a = cross(c1, c2);
```

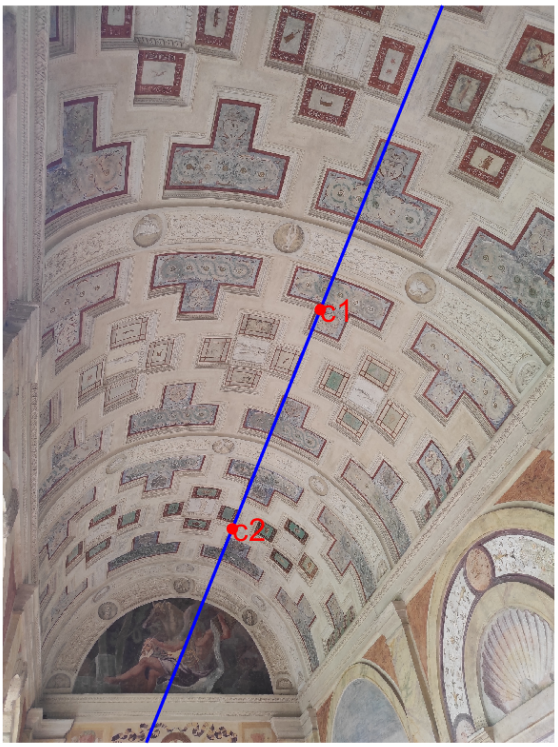


Figure 3.2: Cylinder axis

The value of the cylynder axis a is shown below:

$$\mathbf{a} = \begin{bmatrix} -0.0004 \\ -0.0001 \\ 1.0000 \end{bmatrix}$$

3.2.2 Vanishing point V

To obtain the vanishing point V what we need to do is to take the cross product between the two generatrix lines:

$$V = l1 \times l2 \tag{3.5}$$

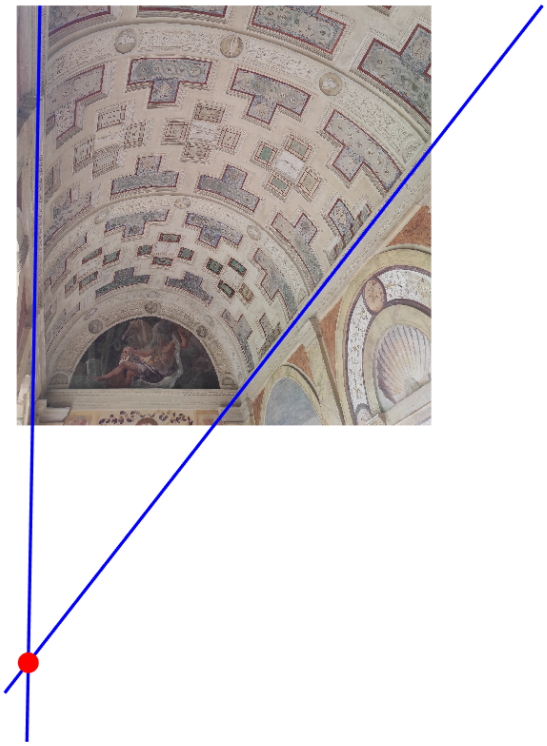


Figure 3.3: Vanishing point V

Our vanishing point V is:

$$\mathbf{V} = 1.0e+03 * \begin{bmatrix} 0.0977 \\ 6.6308 \\ 0.0010 \end{bmatrix}$$

All together we obtained:

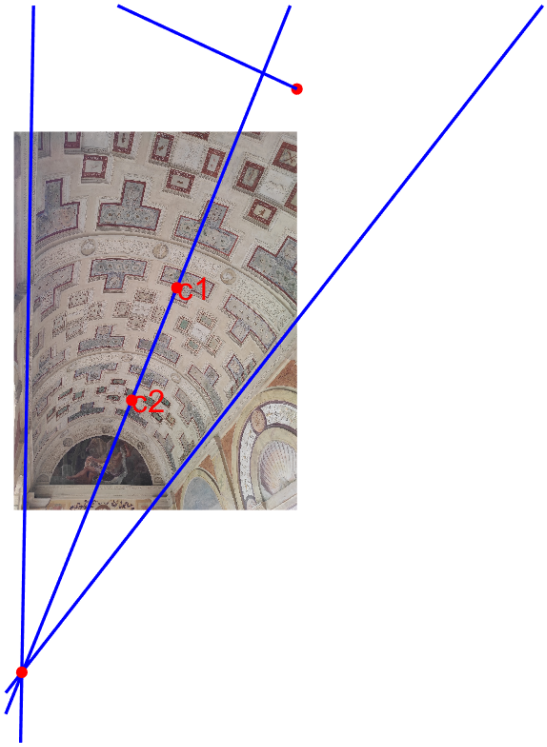


Figure 3.4: l1, l2, V, a, h

In which we can spot the 2 generatrix lines $l1$, $l2$ and the cylinder axis a that intersect together the vanishing point V.

3.3 Calibration matrix K

Request 4. *From $l1$, $l2$, $C1$, $C2$ (and possibly h, a, and V), find the calibration matrix K.*

The image was taken by an uncalibrated, zero-skew, camera. Its calibration matrix depends on four unknown parameters, namely f_x, f_y and the two coordinates U_0,V_0 of the principal point.
Assuming zero-skew we have:

$$\omega = (\mathbf{KK^T})^{-1} = \begin{bmatrix} a^2 & 0 & -u_0a^2 \\ 0 & 1 & -v_0 \\ -u_0a^2 & -v_0 & f_y^2 + a^2u_0^2 + v_0^2 \end{bmatrix}$$

where $a = \frac{f_x}{f_y}$.
Therefore, in order to obtain K, we have to find firstly the **IAC** w and then, applying the **Cholesky** factorisation, we will be able to find the calibration matrix K.
It was performed the calibration from rectified face plus orthogonal vanishing point method, so we had:

- I' or J' from $C1$ or $C2$.
- the image of the line at the infinity, h.
- the vanishing point obtained with the cross product between $l1$ and $l2$, V.

Therefore, we have:

$$l'_\infty = h = wV \tag{3.6}$$

but only 2 equations since these vectors are homogeneous. In this equation we spot the vanishing point obtained with the 2 generatrix lines $l1$, $l2$ and the image of the line at the infinity, h.

Moreover, from I' we know that:

$$I' = H_R^{-1} I = H_R^{-1} \begin{bmatrix} 1 \\ i \\ 0 \end{bmatrix} = \begin{bmatrix} h1 & h2 & h3 \end{bmatrix} \begin{bmatrix} 1 \\ i \\ 0 \end{bmatrix} = h1 + ih2 \tag{3.7}$$

and

$$I'^T \omega I' = (h1 + ih2)^T \omega (h1 + ih2) = 0 \tag{3.8}$$

obtaining

$$h_1^T \omega h_2 = 0 \tag{3.9}$$

$$h_1^T \omega h_1 - h_2^T \omega h_2 = 0 \tag{3.10}$$

where $h1$ is the real part of I' and $h2$ is the imaginary part of I'.
In this way, we can put all these informations together in order to find the IAC w.

$$\begin{cases} l'_\infty = h = wV \\ h_1^T \omega h_2 = 0 \\ h_1^T \omega h_1 - h_2^T \omega h_2 = 0 \end{cases} . \tag{3.11}$$

We discovered the following IAC w:

$$\omega = \begin{bmatrix} 0.1032 & 0 & -10.0892 \\ 0 & 1 & -1361.0590 \\ -10.0892 & -1361.0590 & 9025926.3031 \end{bmatrix}$$

Since the image was taken from a non natural camera, we don't have square pixels:

- $w(1,1)$ is not equal to $w(2,2)$, so $a_x \neq a_y$

but it is zero-skew:

- $w(1,2) = w(2,1) = 0$

Then, applying the **Cholesky** factorisation we are able to find the calibration matrix K as:

$$K = chol(w)^{-1} \tag{3.12}$$

and at the end our calibration matrix K is the following:

$$\mathbf{K} = 1.0e+03 * \begin{bmatrix} 8.3344 & 0 & 0.0977 \\ 0 & 2.6781 & 1.3611 \\ 0 & 0 & 0.0010 \end{bmatrix}$$

3.4 Orientation of the cylinder axis w.r.t. the camera reference

Request 5. *From h, K, and V determine the orientation of the cylinder axis w.r.t. the camera reference.*

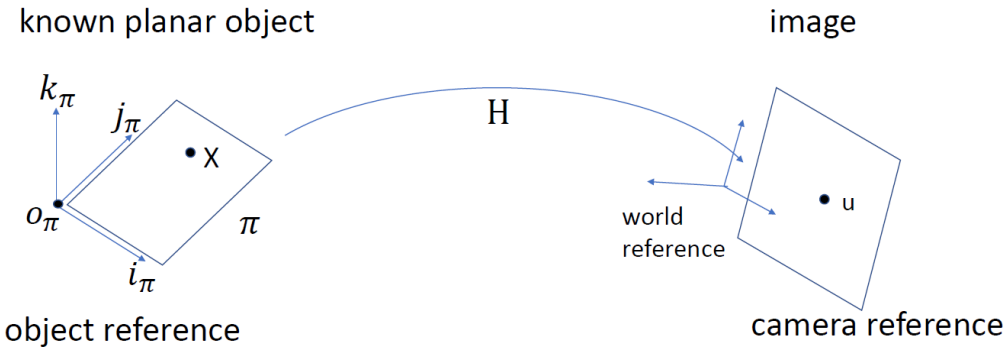


Figure 3.5: Object reference and camera reference

From the theory we know that we need to map the X point in the known planar object into the u point in the image.
We know that world reference is equal to camera reference and that:

$$X_\pi = \begin{bmatrix} x \\ y \\ 0 \\ w \end{bmatrix} \tag{3.13}$$

where X_π is a point on the plane π, and

$$X_w = \begin{bmatrix} i_\pi & j_\pi & k_\pi & o_\pi \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ w \end{bmatrix} = \begin{bmatrix} i_\pi & j_\pi & o_\pi \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} i_\pi & j_\pi & o_\pi \\ 0 & 0 & 1 \end{bmatrix} X_\pi \tag{3.14}$$

We know also that:

$$P = \begin{bmatrix} KR & KRt \end{bmatrix} = \begin{bmatrix} K & 0 \end{bmatrix} \tag{3.15}$$

and finally the point u is equal to:

$$u = PX_w = \begin{bmatrix} K & 0 \end{bmatrix} \begin{bmatrix} i_\pi & j_\pi & o_\pi \\ 0 & 0 & 1 \end{bmatrix} X_\pi = K \begin{bmatrix} i_\pi & j_\pi & o_\pi \end{bmatrix} X_\pi \tag{3.16}$$

So we can write that:

$$H = K \begin{bmatrix} i_\pi & j_\pi & o_\pi \end{bmatrix} \tag{3.17}$$

and it is the same as:

$$K^{-1}H = \begin{bmatrix} i_\pi & j_\pi & o_\pi \end{bmatrix} \tag{3.18}$$

where K is the calibration matrix calculated before and H is the homography from the planar face to its image.
First of all we need to calculate H from the image of the conic dual to the circular points:

$$C^*_\infty = I' J'^T + J' I'^T \tag{3.19}$$

Calc of the SVD:

$$svd(C^*_\infty) = U \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & \epsilon \end{bmatrix} U^T \tag{3.20}$$

Finally, H:

$$H = (U \begin{bmatrix} \sqrt{s_1} & 0 & 0 \\ 0 & \sqrt{s_2} & 0 \\ 0 & 0 & 1 \end{bmatrix})^{-1} \tag{3.21}$$

Once we obtain i_π, j_π and o_π (by inverting K and multiply it for H) we need to compute $k_\pi = i_\pi \times j_\pi$ in order to be able to recover the first part of X_w:

$$\begin{bmatrix} i_\pi & j_\pi & k_\pi & o_\pi \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

So, remembering that:

$$X_w = \begin{bmatrix} i_\pi & j_\pi & k_\pi & o_\pi \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ w \end{bmatrix} \tag{3.23}$$

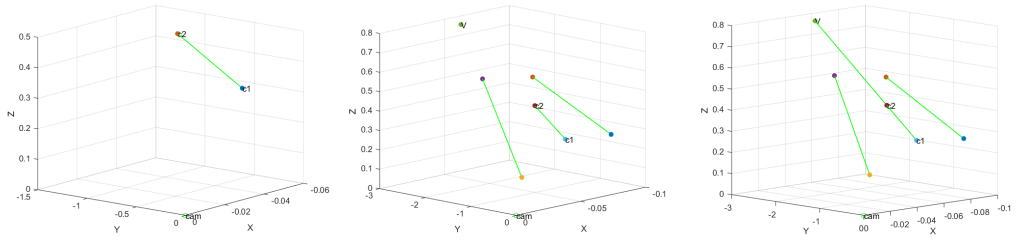
what we have to do is to multiply any point we want to map in camera reference, for the first term of X_w and we are done.
So if we want to convert a generic point in camera reference we could write:

$$P_{3D} = \begin{bmatrix} i_\pi & j_\pi & k_\pi & o_\pi \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P(1) \\ P(2) \\ 0 \\ P(3) \end{bmatrix} \tag{3.24}$$

This is what was done in the below Matlab code:

```
1  res = inv(k) * H;
2
3  i_pi = res(:, 1);
4  j_pi = res(:, 2);
5  O_pi = res(:, 3);
6
7  k_pi = cross(i_pi, j_pi);
8  res = [i_pi j_pi k_pi O_pi; 0 0 0 1];
9
10 c1_3d = res*[c1(1);c1(2);0;c1(3)];
11 c2_3d = res*[c2(1);c2(2);0;c2(3)];
```

Below are shown some results in camera reference:



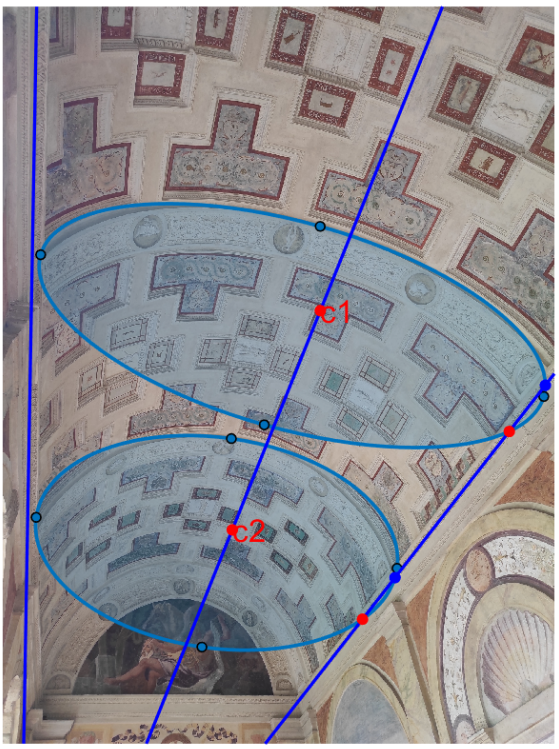
In the first image, it has been generatedo nly the plot of the axis of the cylinder passing through the centers of the two conic sections.
In the second one, the two generatrix lines, $l1$ and $l2$, are depicted along with the vanishing point V related to the plane of the cylinder's axis.
While in the last one, it is shown in three dimensions that the axis of the cylinder passes through the vanishing point V.

3.5 Ratio between the radius of the circular cross sections and their distance

Request 6. *Compute the ratio between the radius of the circular cross sections and their distance.*

3.5.1 Rectification approach

Since we are asked to find a ratio but not on parallel lines, it was necessary to find and apply an euclidean rectification (w.r.t the plane parallel to the cylinder axis, since the distance to be calculated is on that plane) of the image before perform calculations. Below are explained all the steps through the whole process:



First of all, I plotted the two conics $C1$ and $C2$, the two generatrix lines $l1$,

$l2$, the centers of the two conics $c1$, $c2$ and the cylinder axis a. Subsequently, I identified the intersections between the conics and the generatrix line $l1$ (on the right of the image), two for each conic.

First conic:

$$\begin{cases} a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1 = 0 \\ -y - \frac{(l1(1)*x+l1(3))}{l1(2)} = 0 \end{cases} . \tag{3.25}$$

Second conic:

$$\begin{cases} a_2x^2 + b_2xy + c_2y^2 + d_2x + e_2y + f_2 = 0 \\ -y - \frac{(l1(1)*x+l1(3))}{l1(2)} = 0 \end{cases} . \tag{3.26}$$

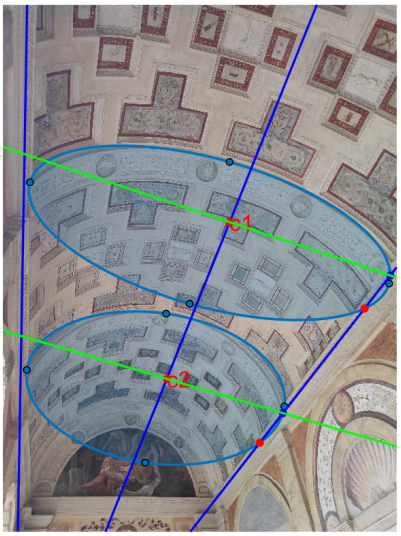
Afterwards, I drew the line passing through the center of the conics and one of the two intersections for each ellipse; I chose the intersection that most closely resembled the true diameter. So I computed:

$$d_1 = c1 \times e1_1 \tag{3.27}$$

and

$$d_2 = c2 \times e2_1 \tag{3.28}$$

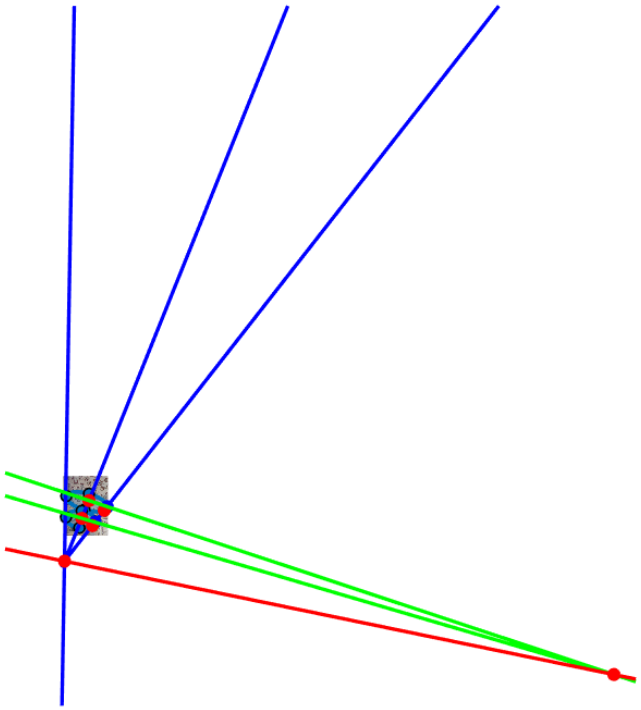
where $e1_1$ and $e2_1$ are respectively the blue intersection of the first and the second conic.



In this way, I have two lines that are parallel in the original scene, which I can use to find the second vanishing point on the plane parallel to the axis of the cylinder; clearly, this could lead to some accuracy issues as the lines are quite parallel even in the image.
Now, we are ready to compute the second vanishing point and the respective vanishing line passing through this vanishing point and the vanishing point V calculated in the previous points.

$$V_2 = d_1 \times d_2 \tag{3.29}$$

$$h_2 = V \times V_2 \tag{3.30}$$



Now that we have the vanishing line relative to the plane parallel to the axis of the cylinder, we can find the images of the circular points I' and J', which we will need to calculate the H_R in question.
From theory we know that

$$I', J' = l'_\infty \cap w \tag{3.31}$$

where l'_∞ is the vanishing line just found, while w is the IAC calculated before.
Once we found I' and J', we are ready and able to calculate H_R as:

$$C^*_\infty = I'J'^T + J'I'^T \tag{3.32}$$

Calc of the SVD:

$$svd(C^*_\infty) = U \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & \epsilon \end{bmatrix} U^T \tag{3.33}$$

Finally, H_R:

$$H_R = (U \begin{bmatrix} \sqrt{s_1} & 0 & 0 \\ 0 & \sqrt{s_2} & 0 \\ 0 & 0 & 1 \end{bmatrix})^{-1} \tag{3.34}$$

As a final step, we need to transform the points of interest, which are the two centers of the conics and the two previously chosen intersections, by multiplying them by H_R. To obtain the measure of the radii, we need to calculate the Euclidean distance from the rectified centers to the rectified intersections. Meanwhile, to obtain the distance between centers, calculate the Euclidean distance between the two rectified centers.

The code below shows what just explained:

```
c1_rect = H_rect * c1;
c1_rect = c1_rect ./ c1_rect(3);

c2_rect = H_rect * c2;
c2_rect = c2_rect ./ c2_rect(3);

i1_rect = H_rect * intersect_1;
i1_rect = i_1_rect ./ i1_rect(3);

i2_rect = H_rect* intersect_2;
i2_rect = i2_rect ./ i2_rect(3);

dist_1 = norm(c1_rect - i1_rect);
dist_2 = norm(c2_rect - i2_rect);
dist_c1_c2 = norm(c1_rect - c2_rect);
```

and finally, the ratio is calculated as:

$$ratio = \frac{radius_1}{dist\_c1\_c2} = \frac{radius_2}{dist\_c1\_c2} \tag{3.35}$$

The results are:

$$ratio = 0.2920$$

$$ratio = 0.2914$$

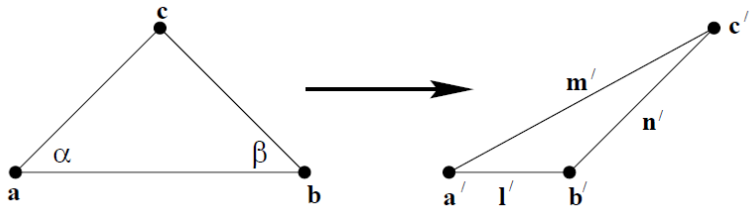
As we can notice from the above results, I would say that the error is relatively small.

3.5.2 Angles approach

From theory we know that exist an espression invariant to projective transformation, that is:

$$cos(\theta) = \frac{l^T C_\infty^* m}{\sqrt{(l^T C_\infty^* l)(m^T C_\infty^* m)}} \tag{3.36}$$

where C_∞^* is the conic dual to the circular points calculated in the previous approach.
Considering these triangles:

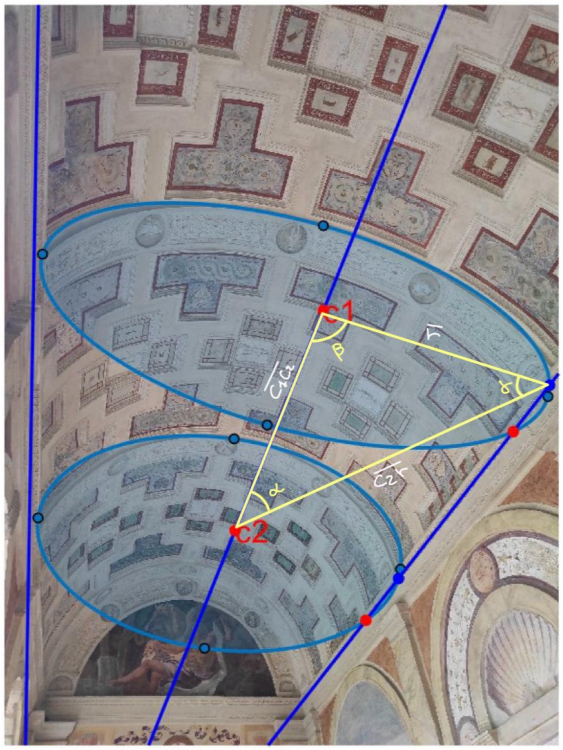


Once C_∞^* is identified, the Euclidean length ratio $\frac{d(b,c)}{d(a,c)}$ may be measured from the projectively distorted figure.
From the standard trigonometric sine rule the ratio of lengths we have:

$$\frac{d(b,c)}{d(a,c)} = \frac{sin(\alpha)}{sin(\beta)} \tag{3.37}$$

where $d(x,y)$ denotes the Euclidean distance between the points x and y. In our problem we have two of these triangles, one for each conics.

First triangle



Here, we can identify three angles and segments:

- α, the angle between $c1c2$ and $c2r$ segments
- β, the angle between $c1c2$ and r segments
- γ, the angle between r and $c2r$ segments

I calculated these 3 angles following the (3.36) formula and referring to the (3.37), we can write:

$$\frac{d(b,c)}{d(a,c)} = \frac{\overline{r}}{\overline{c1c2}} \tag{3.38}$$

and

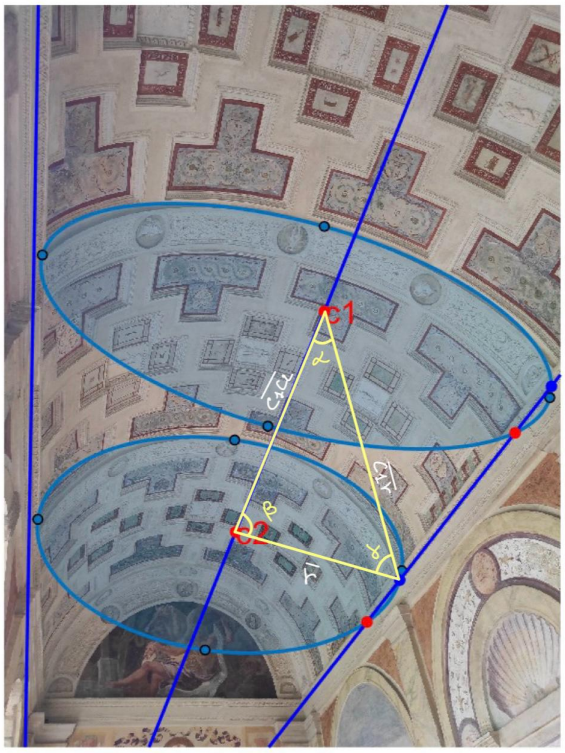
$$\frac{sin(\alpha)}{sin(\beta)} = \frac{sin(\alpha)}{sin(\gamma)} \tag{3.39}$$

Finally, the resulting ratio for this triangle is

$$ratio = 0.2920$$

Second triangle

The situation is similar to before:



Also here, we can identify three angles and segments:

- α, the angle between $c1c2$ and $c1r$ segments
- β, the angle between $c1c2$ and r segments
- γ, the angle between r and $c1r$ segments

I calculated these 3 angles following the (3.36) formula and referring to the (3.37), we can write:

$$\frac{d(b,c)}{d(a,c)} = \frac{\overline{r}}{\overline{c1c2}} \tag{3.40}$$

and

$$\frac{sin(\alpha)}{sin(\beta)} = \frac{sin(\alpha)}{sin(\gamma)} \tag{3.41}$$

Finally, the resulting ratio for this triangle is

$$ratio = 0.2914$$

3.5.3 Disclaimer

Faced with various trials and attempts, even making slight changes to the selections of conics, I noticed quite different results. Therefore, the obtained outcome should be considered with intrinsic noise due to the imprecise selection of various components necessary to carry out the following tasks.

Chapter 4

Unfolding the surface between cross sections

Request 7. *Rectification of a cylindric surface: Plot the unfolding of the part of the surface, included between the two cross sections, onto a plane.*

In this last point I tried to unfold the surface between the 2 cross sections: in particular, what I did, was to crop the surface from the original image:

Figure 4.1: Interested cross section

After tried to rectified it, using the previous calculated H_R, I worked on the conversion from cartesian coordinates to polar coordinates.
Indeed, I tried to map the (x, y) coordinates in polar coordinates, using three Matlab function (explained in the order used):

- **[X, Y] = meshgrid(x, y)**: The function returns the coordinates of the two-dimensional grid based on the coordinates contained in the vectors x and y.
- **[theta, rho] = car2pol(X, Y)**: transforms corresponding elements of the two-dimensional Cartesian coordinate arrays x and y into polar coordinates theta and rho.
- **warp(theta, rho, Z, J)**: displays the indexed image J with colormap map as a texture map on a simple rectangular surface.

So, I took the rectified image and after trying different *meshgrid* parameters through trial and error to find the best combinations that allowed me to achieve the flattest result possible, I called *car2pol* to convert the coordinates from cartesian to polar; finally the plot is shown through *warp*.

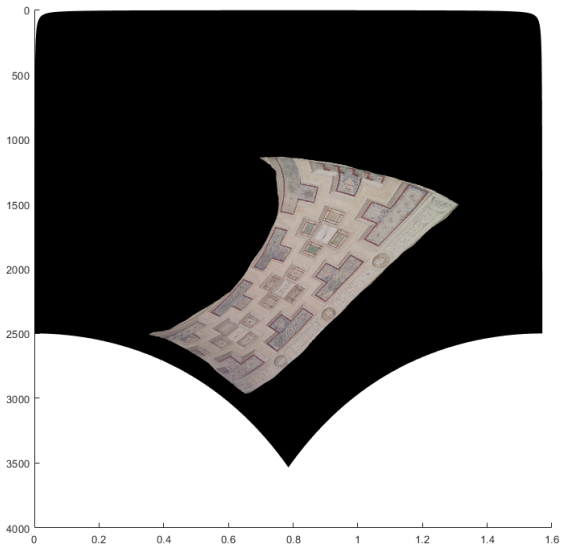


Figure 4.2: Unfolding of the surface between cross sections

Chapter 5

References

- IACV 23-24 slide courses
- how-do-you-choose-optimal-parameters-canny-edge, link
- Richard_Hartley_Andrew_Zisserman-Multiple_View_Geometry, book
- Houghlines, link
- Edge detection, link
- Regionprops, link
- Corner detection Harris, link
- Corner detection SURF, link