POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Homework Report

Image Analysis and Computer Vision, 2023-2024
Computer Science and Engineering

Giovanni Versiglioni, 11015537

Contents

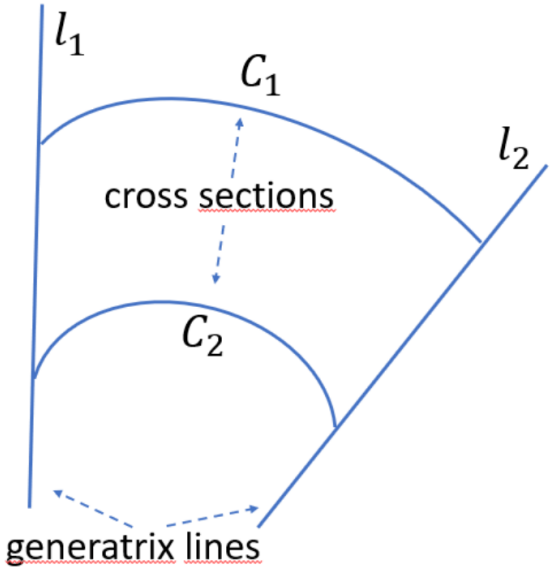
List of Figures

0. Introduction

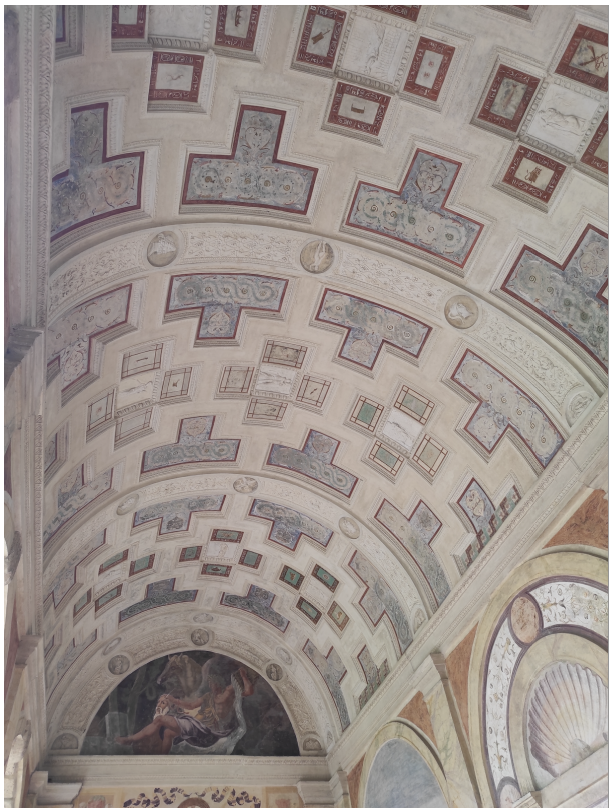
This is the report for the Image Analysis and Computer Vision Homework, A.Y. 2023-2024.

Scene (Figure 0.1a). A right circular cylinder together with two circular cross sections and two generatrix lines. A right circular cylinder is a surface, made of parallel lines, which is symmetric about a symmetry axis. A circular cross section of a right cylinder is a circumference centered on the symmetry axis and perpendicular to the symmetry axis. A generatrix line is a straight line, on the cylinder surface, which is parallel to its axis.

Image (Figure 0.1b). A single image is taken of the described cylinder by an uncalibrated, zero-skew camera. Two circular cross sections are visible, and their images C_1, C_2 are extracted. Two (parallel) generatrix lines of the cylinder are also visible, and their images l_1, l_2 are extracted.



(a) Scene – Right circular cylinder with two cross sections and two generatrix lines.



(b) Image – Cylindrical voult in Palazzo Te, Mantova, Italy.

Figure 0.1: Homework problem formulation and image data.

1. Feature Extraction

The goal is to retrieve geometrical data – such as edges, corners, and lines – from the given image.

1.1. Edge Detection

The **Canny** method can be used to detect edges in the image.

The algorithm comprises the following sequential steps:

(i) Apply Gaussian filter to smooth the image in order to remove noise;
(ii) Find the intensity gradients of the image;
(iii) Apply gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious response to edge detection;
(iv) Apply double threshold to determine potential edges;
(v) Track edges by hysteresis (i.e. finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges).

MATLAB already includes – in the Image Processing Toolbox – an implementation of this method in the `edge` function, which has been used in the `detectEdges.m` script. **Figure 1.1** shows the resulting image.

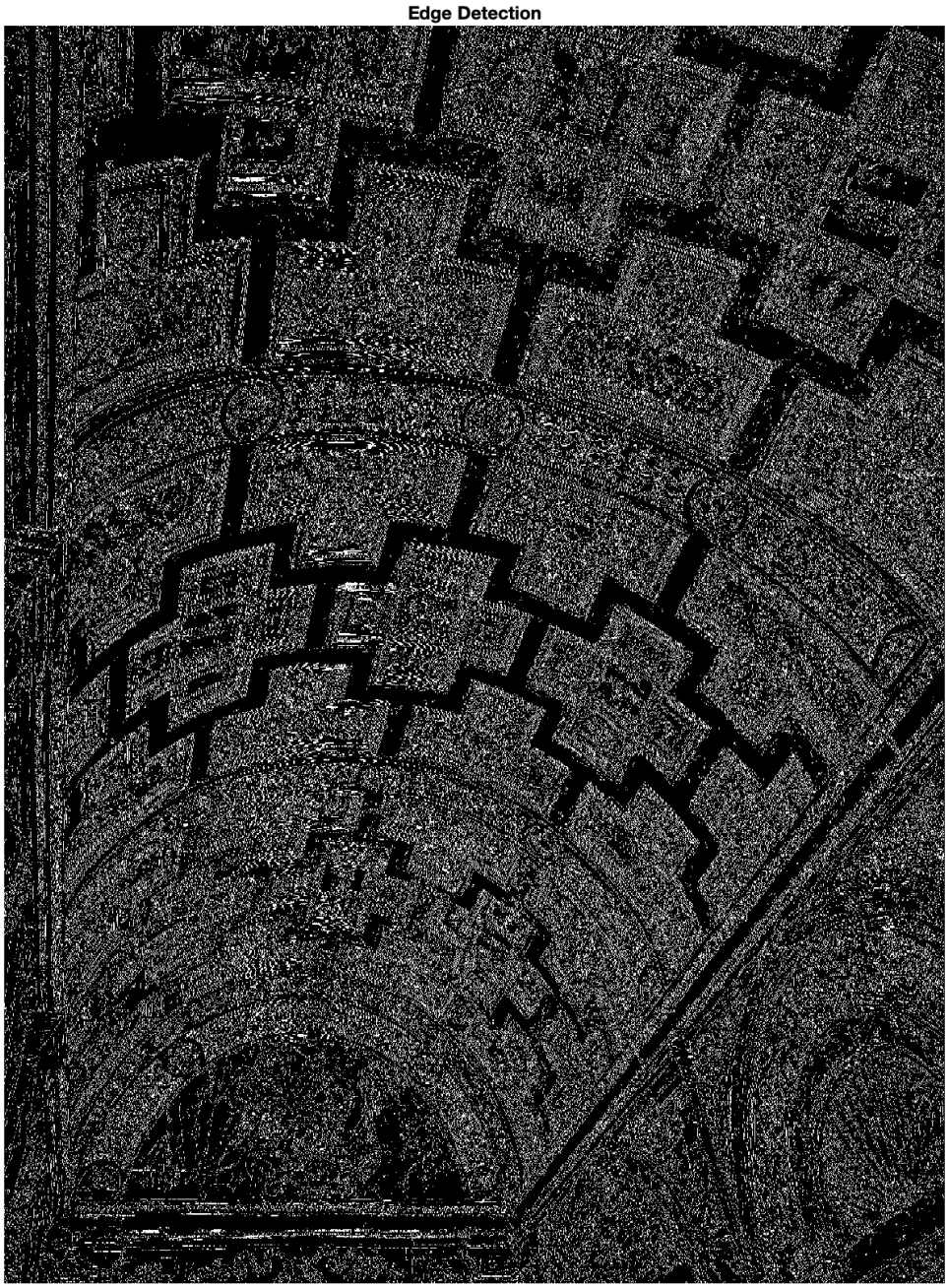


Figure 1.1: Edge Detection – Canny Method.

1.2. Corner Detection

The **Harris** method can be used to detect corners in the image.

The algorithm comprises the following sequential steps:

(i) Color to gray-scale in order to work on a single channel and enhancing speed processing;
(ii) Spatial derivative calculation to highlight pattern changes around each pixel region analyzed and then setting up the tensor structure;
(iii) Harris response calculation by computing the smallest approximated eigenvalue of the structure tensor;
(iv) Non-maximum suppression in order to pick up the optimal values to indicate corners – find the local maxima as corners within the window which is a 3 by 3 filter.

The function `findCorners.m` implements this method and is used in the `detectCorners.m` script to compute corners in the image. **Figure 1.2** shows the resulting image.

Figure 1.2: Corner Detection – Harris Method.

1.3. Line Detection

The **Hough Transform** method can be used to detect lines in the image.

The algorithm comprises the following steps:

(i) Edge detection: using the Canny edge detector;
(ii) Mapping of edge points to the Hough space and storage in an accumulator;
(iii) Interpretation of the accumulator to yield lines of infinite length (interpretation is done by thresholding and possibly other constraints);
(iv) Conversion of infinite lines to finite lines based on the hyper-parameters setting.

The function `findLines.m` implements this method and is used in the `detectLines.m` script to compute lines in the image. **Figure 1.3** shows the resulting image.

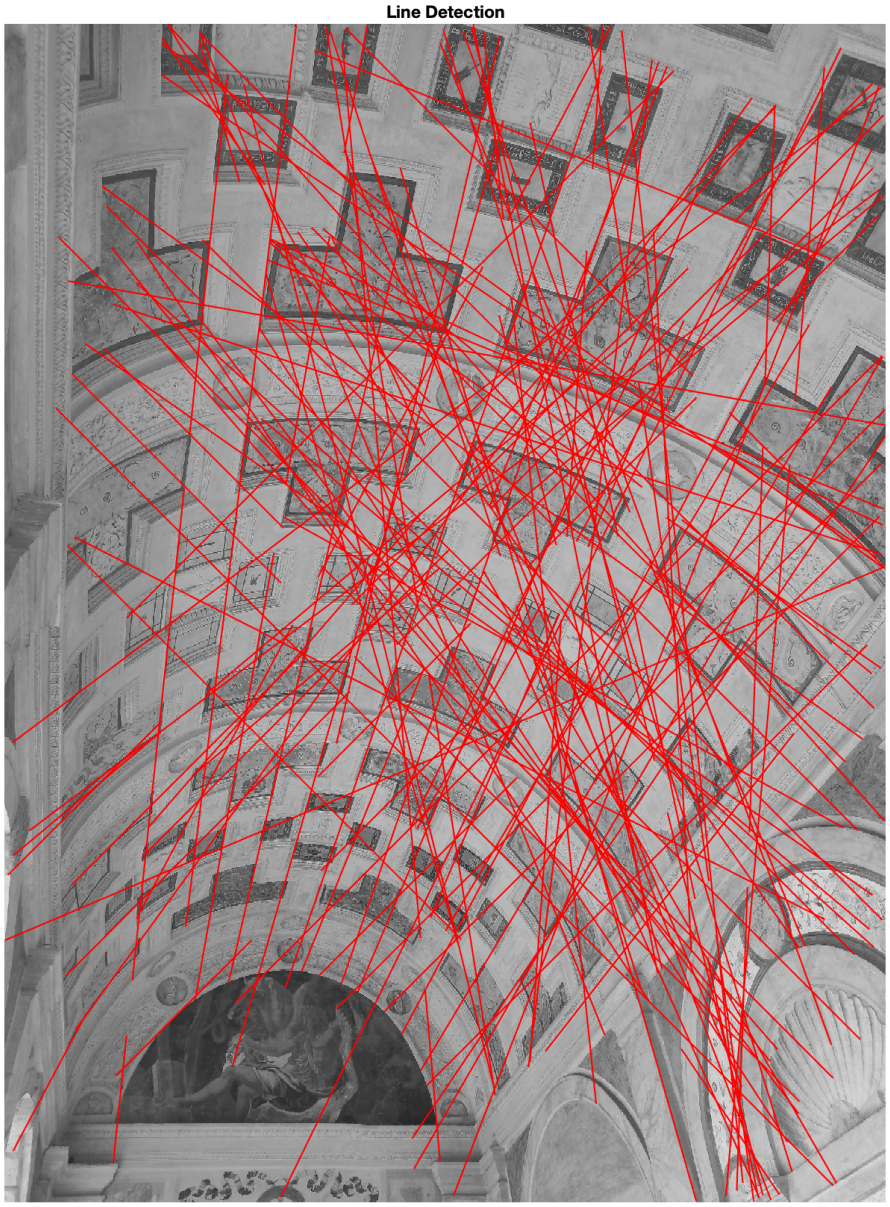


Figure 1.3: Line Detection – Hough Transform.

2. Rectification

This section comprises two main parts. The first part presents theoretical results to Problems 1-5 computed in MATLAB. In the second part will be carried out the metric rectification of a cylindrical surface. Specifically, the unfolding of the part of the surface included between two cross section will be reconstructed onto a plane.

2.1. Problems

The script `Problems.m` contains all the proceedings and results of the following theoretical problems:

1. From C_1, C_2 find the horizon (vanishing) line h of the plane orthogonal to the cylinder axis.
2. From l_1, l_2, C_1, C_2 find the image projection a of the cylinder axis, and its vanishing point V.
3. From l_1, l_2, C_1, C_2 (and possibly h, a, V) find the calibration matrix K.
4. From h, K, V determine the orientation of the cylinder axis w.r.t. the camera reference.
5. Compute the ratio between the radius of the circular cross sections ans their distance.

The very first step is to read the image, convert it to gray-scale, and convert it to double precision – using functions provided by MATLAB such as `imread`, `rgb2gray`, and `im2double`.

2.1.1 Cross Sections and Generatrix Lines Identification, Image Contour

The images of the circular cross sections are conics. Since in Euclidean and projective geometry five points determine a conic, it is possible to scatter five points belonging to any of the cross sections in the image scene and then fit the conic through them. To draw the five points, it is possible to use any of the vaults in the image as a reference.

Using the equation of a conic in Cartesian coordinates (*Equation 1*), it is possible to retrieve the (six) parameters by which it is identified and store them in the homogeneous conic coefficients matrix (*Equation 2*). The MATLAB code is also reported.

$$aX^2 + bXY + cY^2 + dX + eY + f = 0 \tag{1}$$

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \tag{2}$$

```
% conic equation -- cartesian coordinates
A = [x.^2 x.*y y.^2 x y ones(size(x))];
N = null(A);

% conic coefficients
cc = N(:, 1);
[a1, b1, c1, d1, e1, f1] = deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
C1 = [a1 b1/2 d1/2; b1/2 c1 e1/2; d1/2 e1/2 f1]; % conic coefficients matrix
```

To find a generatrix line it suffices to compute the line through any two points laying on the two cross sections identified. In homogeneous coordinates this amounts to the cross product of the two points.

Now – using the parameters identified – it is possible to plot the image contour (i.e. the two cross sections and the two generatrix lines), as shown in **Figure 2.1**. As shown in the following MATLAB code, in order to plot the equations two symbolic variables must be defined.

```
syms 'x';
syms 'y';

% C1,C2 equations -- cartesian coordinates
eq1 = a1*x^2 + b1*x*y + c1*y^2 + d1*x + e1*y + f1;
eq2 = a2*x^2 + b2*x*y + c2*y^2 + d2*x + e2*y + f2;
eqns = [eq1 == 0, eq2 == 0];

% plot cross sections and generatrix lines
fimplicit(eqns, 'LineWidth',2, 'Color','b');
plot([p1(1), p3(1)], [p1(2), p3(2)], 'LineWidth',2, 'Color','r');
plot([p2(1), p4(1)], [p2(2), p4(2)], 'LineWidth',2, 'Color','r');
```

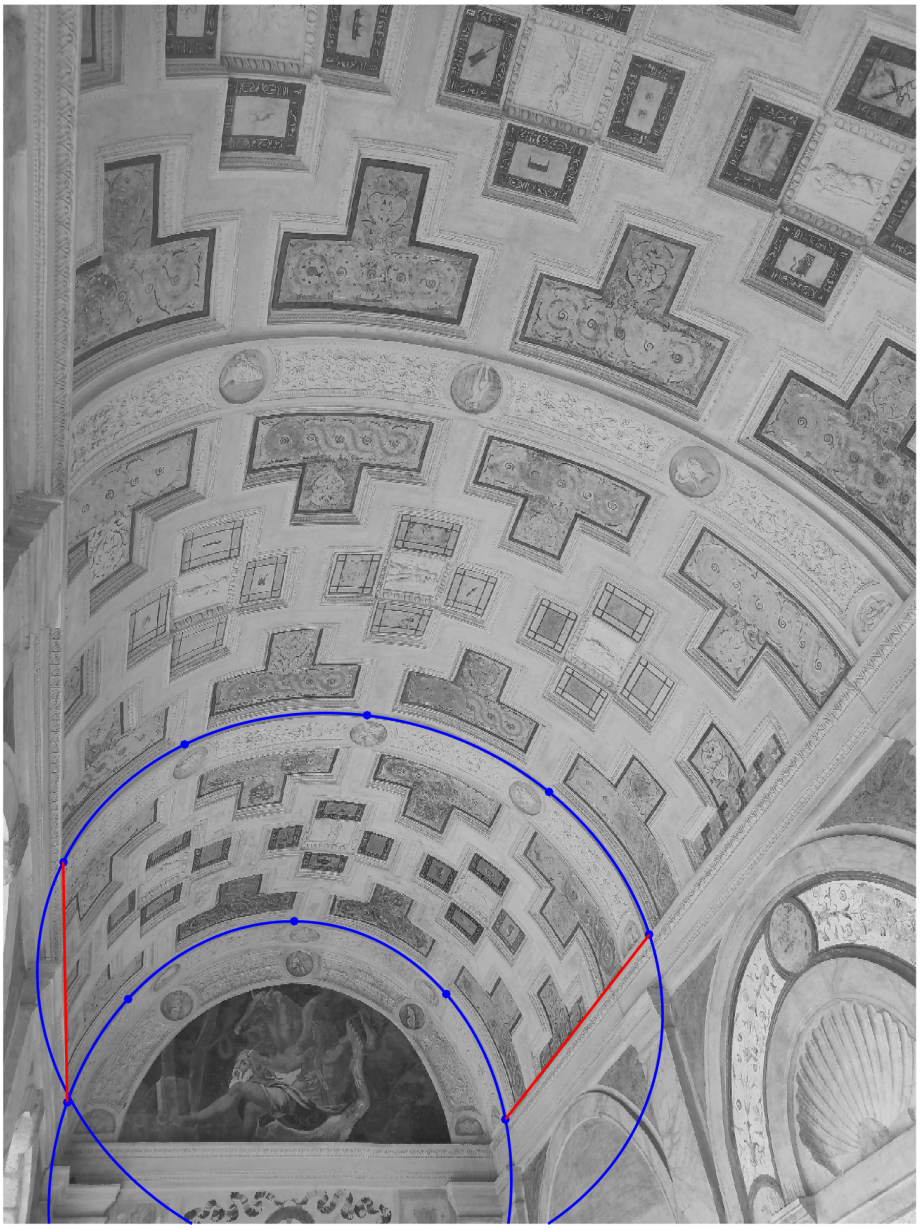


Figure 2.1: Image Contour – Cross Sections (blue) and Generatrix Lines (red).

2.1.2 Circular Points Image, Vanishing Line

Two conics always intersect in four distinct points in projective geometry. Looking at the two identified cross sections shown in **Figure 2.1**, it is clear that there are two real intersections and two complex conjugate intersections – that are exactly the image of circular points.

```
% C1,C2 intersections
S = solve(eqns, [x,y]);
s1 = [double(S.x(1));double(S.y(1));1];
s2 = [double(S.x(2));double(S.y(2));1];
s3 = [double(S.x(3));double(S.y(3));1];
s4 = [double(S.x(4));double(S.y(4));1];

% circular points image (complex conjugate pair)
II = s1;
JJ = s2;
```

The vanishing line is the line through the image of circular points, and (again) in homogeneous coordinates this amounts to the cross product of the two points. Since the vanishing line equation is computed in homogeneous coordinates, in order to plot the line in the image it is first necessary to convert it in Cartesian coordinates. This can be accomplished by dividing the line vector (in homogeneous coordinates) by the third component. Finally, the line is plotted in the image, as shown in **Figure 2.2**.

```
% vanishing line
h = cross(II, JJ);
h = h/h(3);
```

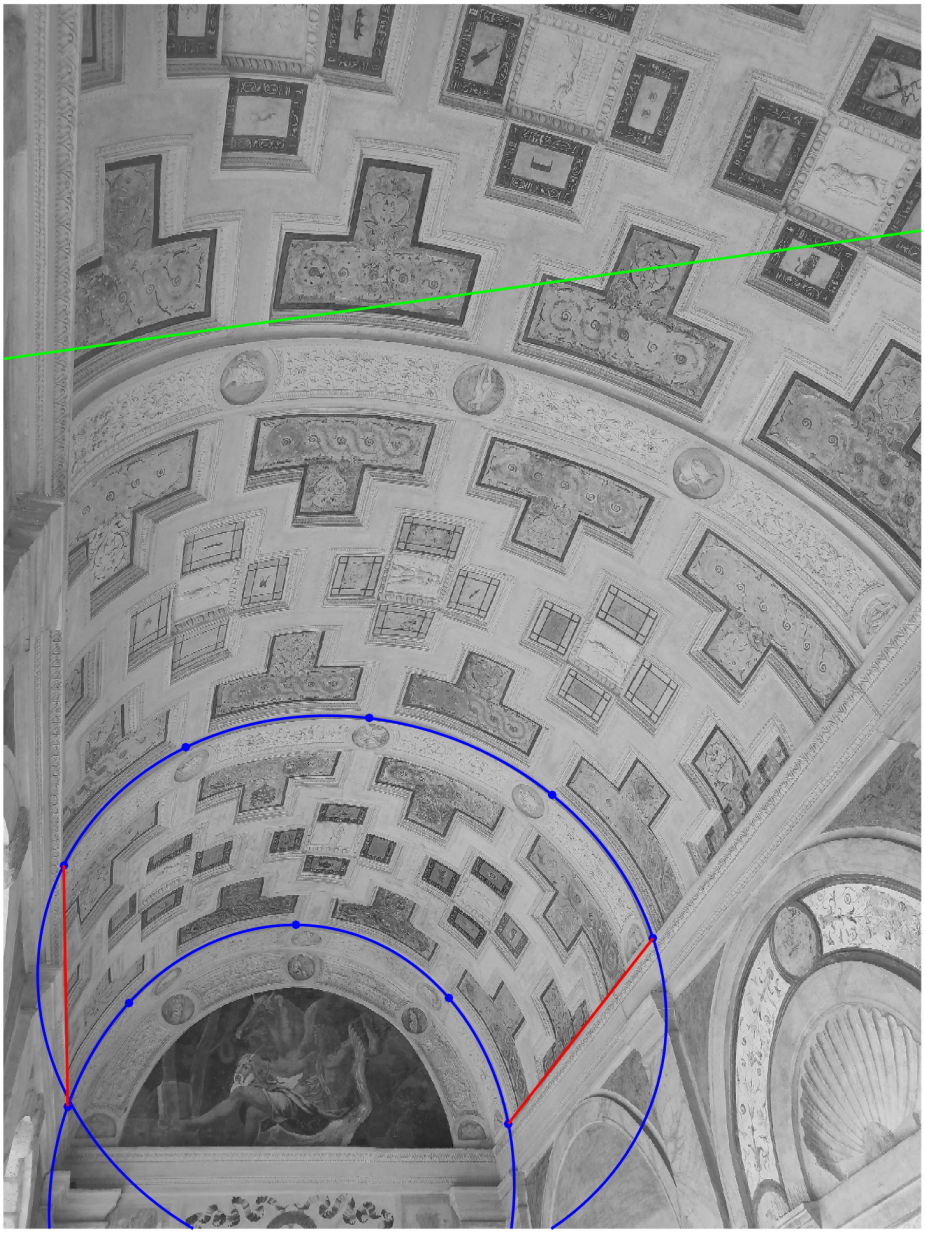


Figure 2.2: Vanishing Line (green).

2.1.3 Diameters, Centres, Cylinder Axis Image and Vanishing Point

The diameters are computed multiplying the conics' coefficients matrices with the image of the circular points (two diameters computed per conic).

```
% diameters
d1 = C1*II;
d1 = d1/d1(3);
d2 = C1*JJ;
d2 = d2/d2(3);
d3 = C2*II;
d3 = d3/d3(3);
d4 = C2*JJ;
d4 = d4/d4(3);
```

For any conic (i.e. image of circular cross section), the intersection point between the two diameters is the center. In homogeneous coordinates this amounts to the cross product of the two lines (i.e. diameters).

```
% centres computation
O1 = cross(d1,d2);
O1 = O1./O1(3);
O2 = cross(d3,d4);
O2 = O2./O2(3);

% centres plot
scatter(O1(1), O1(2), 40,'filled','o','MarkerFaceColor','y');
scatter(O2(1), O2(2), 40,'filled','o','MarkerFaceColor','y');
```

To retrieve the image of the cylinder axis it suffices to determine the line through the centres of the two cross sections (cross product). The outcome is shown in **Figure 2.5**.

```
% cylinder axis image computation
a = cross(O1,O2);

% cylinder axis image plot
x = linspace(1,100000,1000000);
y=((O2(2)-O1(2))/(O2(1)-O1(1)))*(x-O1(1))+O1(2);
plot(x,y,'linewidth',2,'color','c')
```

The vanishing point of a given line is the direction (i.e. point at infinity) of the line in the plane. Algebraically, given a line in homogeneous coordinates, it is obtained intersecting it with the canonical line at infinity.

```
lInf = [0;0;1];
vp = cross(a,lInf);
vp = vp/vp(3);
```

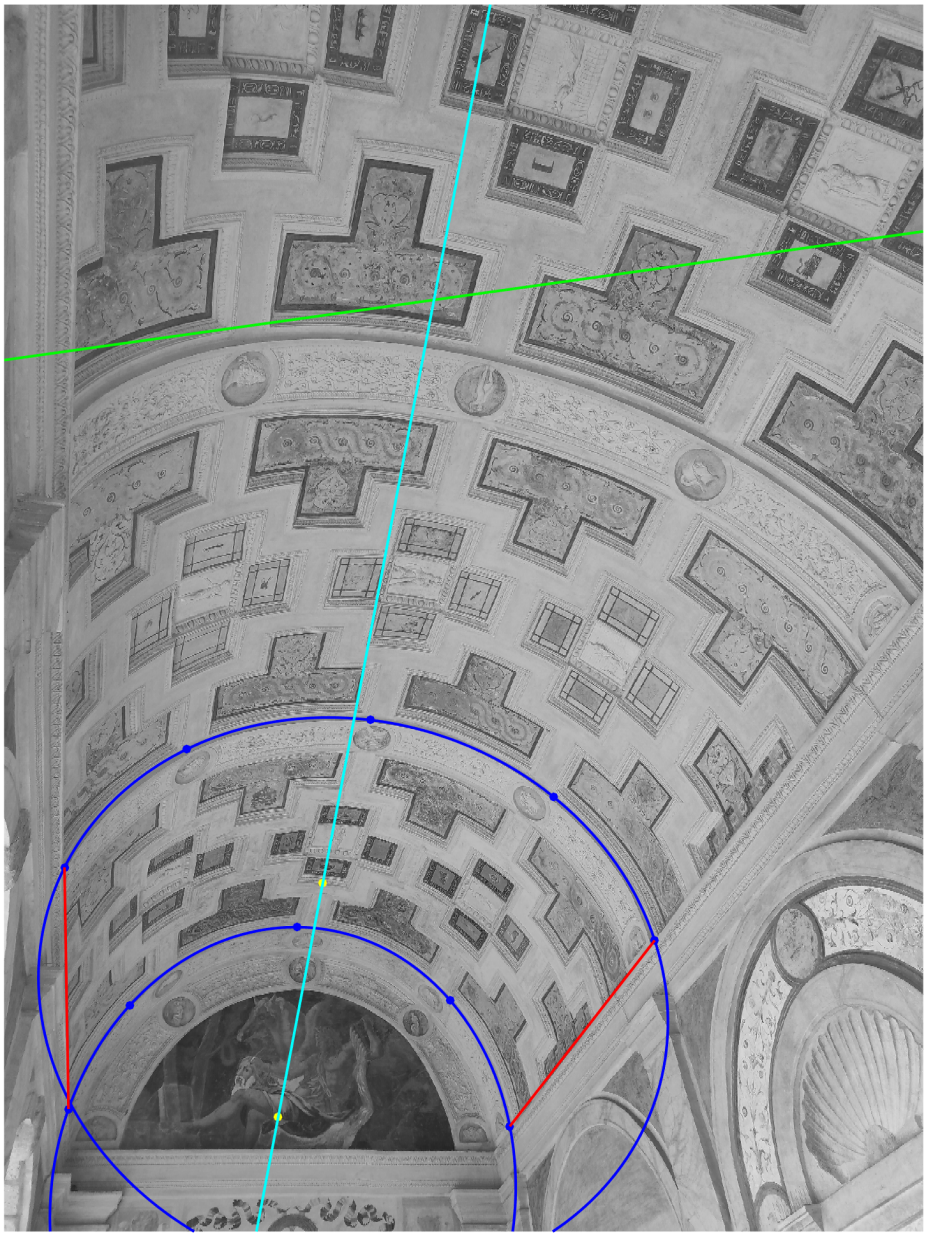


Figure 2.3: Cylinder Axis Image (cyan).

2.1.4 Calibration Matrix

In this section, the calibration matrix K – containing the intrinsic parameters of the camera – will be computed. In the assignment is stated that the skew factor is null, thus there are four unknowns. Therefore, to estimate the calibration matrix – using the Image of the Absolute Conic (IAC) – four constraints (i.e. equations) are necessary. The IAC is strongly related to the camera calibration matrix through the following *Equation 3*.

$$\omega = (KK^T)^{-1} = \begin{bmatrix} \alpha^2 & 0 & -U_0\alpha^2 \\ * & 1 & -V_0 \\ * & * & f_Y^2 + \alpha^2 U_0^2 + V_0^2 \end{bmatrix} \tag{3}$$

In order to compute the IAC – using the function `get_IAC` – three vanishing points were determined, one vertical and two horizontal with respect to the given image.

The vertical vanishing point has been computed intersecting (using the cross product) two vertical parallel lines (green lines in **Figure 2.4**). To find the two horizontal vanishing points, two pairs of parallel lines relying on a horizontal plane were considered (red lines in **Figure 2.4**).

Using the two horizontal vanishing points, one can compute the image of the line at infinity orthogonal to the vertical vanishing point. This allows to have two constraints with the same equation, because it would be completely equivalent to have two separate constraints using orthogonal vanishing points alone. The other two constraints can be easily recovered exploiting the transformation matrix $H = [h_1, h_2, h_3]$ combined with the image of the circular points $h_1 \pm ih_2$.

Hence, the four constraints (three equations) are reported in the system defined by *Equation 4*.

$$\begin{cases} l'_\infty = \omega v \\ h_1^T \omega h_2 = 0 \\ h_1^T \omega h_1 - h_2^T \omega h_2 = 0 \end{cases} \tag{4}$$

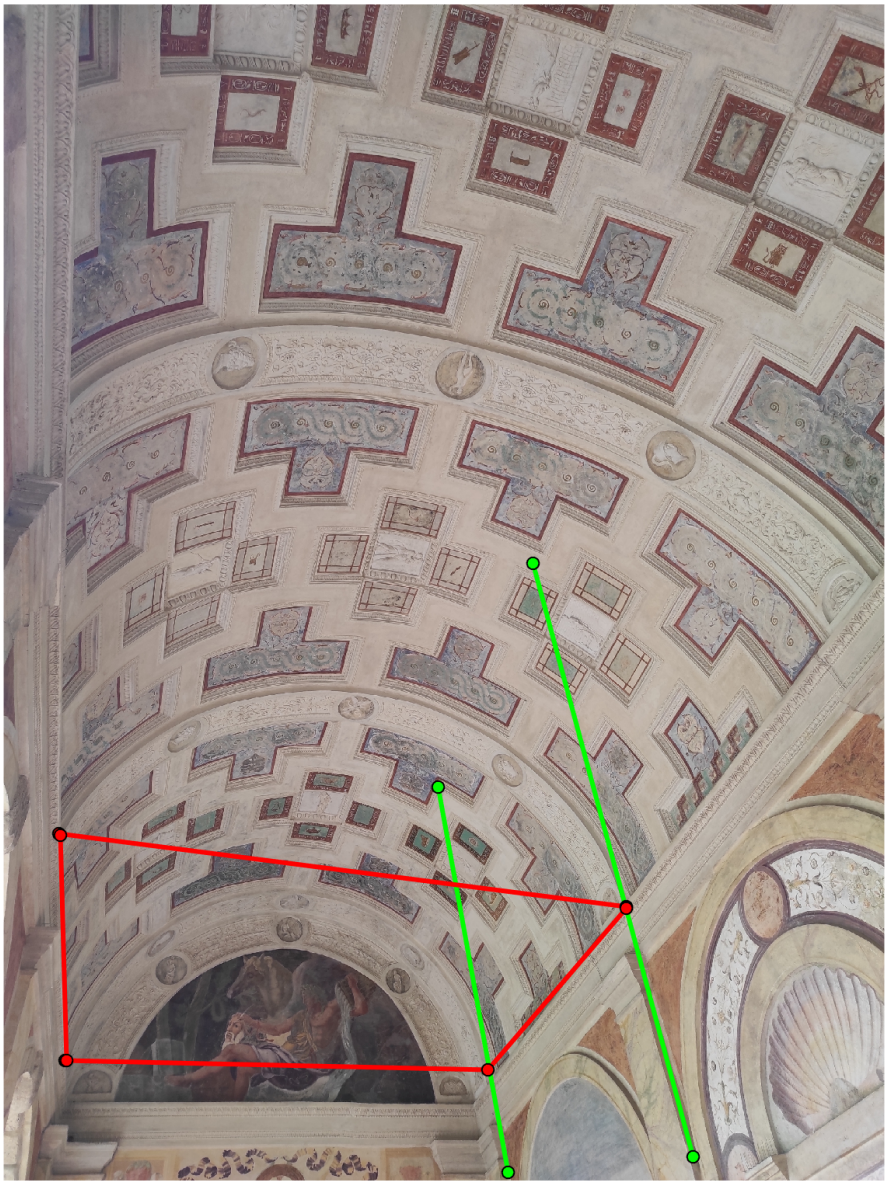


Figure 2.4: Calibration Constraints.

Finally, one can compute the calibration matrix K through Cholesky Decomposition of the IAC. The process is carried out by the following MATLAB code and the obtained calibration matrix is presented in *Equation 5*, where f_X, f_Y, U_0, V_0 are the camera intrinsic parameters.

```
IAC = get_IAC(inf_line, vp1, vp2, vp3, H);
% intrinsic parameters
alfa = sqrt(IAC(1,1));
u0 = -IAC(1,3)/(alfa^2);
v0 = -IAC(2,3);
fy = sqrt(IAC(3,3) - (alfa^2)*(u0^2) - (v0^2));
fx = fy / alfa;
% build K using the parametrization
K = [fx 0 u0; 0 fy v0; 0 0 1];
```

$$K = \begin{bmatrix} f_X & 0 & U_0 \\ 0 & f_Y & V_0 \\ 0 & 0 & 1 \end{bmatrix} \approx 10^3 \begin{bmatrix} 2.2753 & 0 & 1.0161 \\ 0 & 3.8887 & 4.0102 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

2.1.5 Cylinder Axis Orientation, Radius-Distance Ratio

In this part, the (3D) orientation between the cylinder axis and the camera reference is determined using computed information such as the calibration matrix K, the vanishing line h, and the vanishing point of the cylinder axis V.

The process involves transforming the vanishing line h into its corresponding epipolar line h' using the camera calibration matrix K, as stated in *Equation 6*.

$$h' = K^{-T} h \tag{6}$$

Subsequently, the direction vector from the camera center to a selected point on the epipolar line is calculated, representing the orientation of the cylinder axis in 3D space. The vanishing point V is projected into 3D space, and the orientation of the cylinder axis is determined by the cross product of the direction vector and the 3D coordinates of the projected vanishing point. The resulting direction vector is then normalized to represent the unit vector along the axis.

Optional visualizations, such as plotting the vanishing line and the direction vector on the image, provide additional insights into the cylinder axis orientation.

The following MATLAB code summarizes the explained process.

```
% transform vanishing line l to epipolar line l'
h_prime = inv(K)' * h;

% find the intersection point P of l' with the image plane
P = cross(h_prime, [1; 0; 0]);  % assuming x-axis as principal axis

% calculate the direction vector from the camera center to P
C = -inv(K) * h;  % camera center
direction_vector = (P - C) / norm(P - C); % orientation of the cylinder axis
```

```
% display result (3D)
% orientation rotation matrix
R = direction_vector;
[U, ~, V] = svd(R);
R = U * V';
axis_orientation = R.';
poseplot(axis_orientation);
```

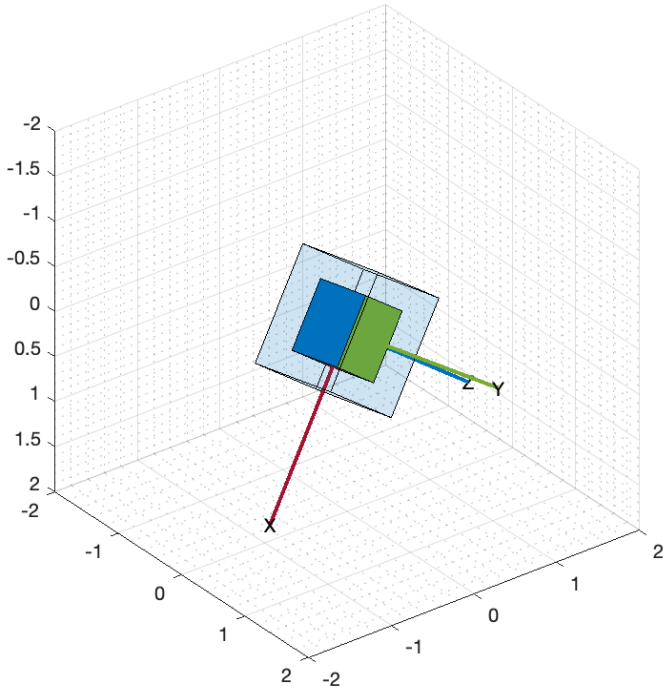


Figure 2.5: Cylinder Axis Orientation (red) and Vanishing Line (green) – plotted in 3D using `poseplot`.

Finally, the radius-distance ratio is computed using known parameters such as generic points on the conics and their centres. Specifically, the Euclidean distance between desired endpoints is determined using MATLAB `pdist` function. The ratio – considering the less distorted conic – is approximately unitary.

```
% compute cross section radius
radiusC1 = pdist([p1(1),p1(2);O1(1),O1(2)], 'euclidean'); % less distorted
radiusC2 = pdist([p3(1),p3(2);O2(1),O2(2)], 'euclidean');

% compute distance between cross sections (centers)
distance = pdist([O1(1),O1(2);O2(1),O2(2)], 'euclidean');

% compute ratio
ratio1 = radiusC1/distance; % 1.0991 (computed result)
ratio2 = radiusC2/distance; % 1.5525 (computed result)
```

2.2. Metric Rectification

The objective is to metric rectify (i.e. 2D reconstruct) the unfolding of the part of the cylinder surface included between the two cross sections onto a plane.

Metric rectification will be accomplished using images of circles [1]. Specifically, three images of circles will be used, as shown in **Figure 2.6**. To determine the parameters of the selected conics, the same method used to estimate the cross sections has been used.

Figure 2.6: Circles Images used for Rectification.

Then, for every pair of conics the intersections are computed. In order to disambiguate between the solutions the image of the circular points is determined as the mean one among the ones computed. Next, the line at infinity is computed using the image of the conic dual to the circular points – that is determined using the definition. The affine rectification matrix is evaluated using the line at infinity, as shown in *Equation 7*.

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ & {l'_\infty}^T & \end{bmatrix} \tag{7}$$

At this point, taken one of the three conics (e.g. C_1) its projective transformation in the rectified plane is obtained applying the rule stated in *Equation 8*.

$$Q = H^{-T} C_1 H^{-1} \tag{8}$$

Using implemented MATLAB function `AtoG` one can extract geometric information about the conic on the rectified plane (i.e. Q) from its (known) parameters.

To compute the affinity that rectifies the cylinder, the idea is to build the one that makes the axis of the ellipses to be equal (i.e. maps the conic in the image onto a circle in the rectified image). To do this, the desired affinity is obtained combining a rotation – using rotation matrix U defined in *Equation 9* – and a scaling – using diagonal matrix S defined in *Equation 10*, which rescales the axis to be unitary.

$$U = \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{9}$$

$$S = \begin{bmatrix} 1 & 0 \\ 0 & a/b \end{bmatrix} \tag{10}$$

The final transformation T is the composition between the homography that maps the image of the line at infinity to its canonical position and the rescaling homography, and is computed as stated in the following *Equations 11, 12, 13*. Eventually, the rectified cylinder surface unfolded onto a plane is shown in **Figure 2.7**.

$$K = USU^T \tag{11}$$

$$A = \begin{bmatrix} K & 0_{2x1} \\ 0_{1x2} & 1 \end{bmatrix} \tag{12}$$

$$T = AH \tag{13}$$

The following MATLAB code carries out the outlined procedure.

```
imDCCP = II*JJ' + JJ*II';
imDCCP = imDCCP./norm(imDCCP);
l_inf = null(imDCCP);
H = [eye(2), zeros(2,1); l_inf(:)'];

% conic projective transformation
Q = inv(H)'*C1*inv(H);
Q = Q./Q(3,3);

% conic geometric information extraction
par_geo = AtoG([Q(1,1),2*Q(1,2),Q(2,2),2*Q(1,3),2*Q(2,3),Q(3,3)]);
center = par_geo(1:2);
axes = par_geo(3:4);
a = axes(1);
b = axes(2);
alpha = par_geo(5);

% rotation and scaling
U = [cos(alpha), -sin(alpha); sin(alpha), cos(alpha)];
S = diag([1, a/b]);

% composition homography
K = U*S*U';
A = [K zeros(2,1); zeros(1,2), 1];
T = A*H;
tform = projective2d(T');
J = imwarp(img, tform);
```

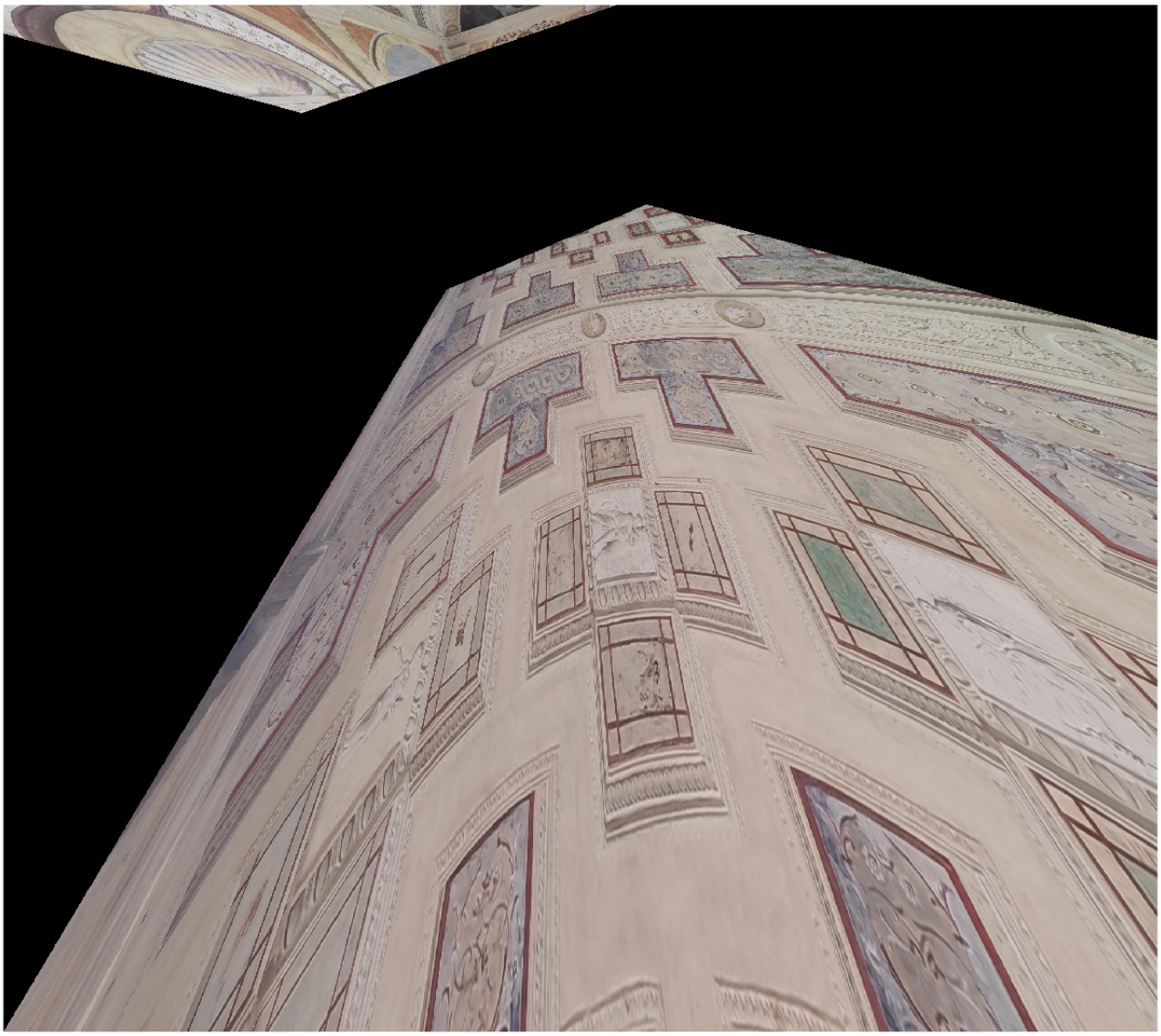
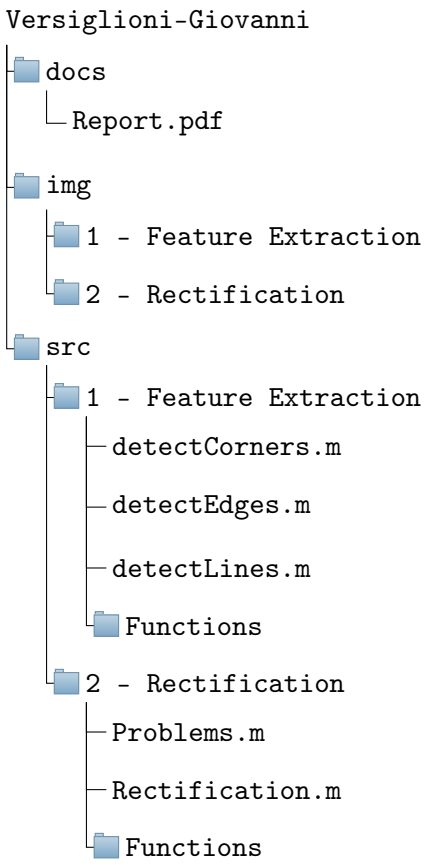


Figure 2.7: Rectified Cylinder Surface between two Cross Sections.

Directory Tree



The homework report is accessible via the `Report.pdf` file.

The used input image and the output images (with respect to both feature extraction and rectification) are inside the `img` folder.

The MATLAB code responsible of extracting features from the image is distributed among the `detectCorners.m`, `detectEdges.m`, and `detectLines.m` scripts. Functions used/implemented are also included.

The MATLAB code containing the detailed solutions to Problems 1-5 is in the `Problems.m` script, while the rectification of the cylinder surface is in the `Rectification.m` script. Functions used/implemented are also included.

Matlab Functions

```
function [corner_x, corner_y] = findCorners(I, sigma, border_margin)
    dx = [-1 0 1;
          -1 0 1;
          -1 0 1];
    dy = dx';
    Ix = conv2(I, dx, 'same');
    Iy = conv2(I, dy, 'same');
    g = fspecial('gaussian', max(1,fix(3*sigma)+1), sigma);
    Ix2 = conv2(Ix.^2, g, 'same');
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');
    cm = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps);
    cm(1:border_margin,:) = 0;
    cm(end-border_margin:end,:) = 0;
    cm(:,end-border_margin:end) = 0;
    cm(:,1:border_margin) = 0;
    T = mean(cm(:));
    CIM = cm;
    CIM(cm<T) = 0;
    support = true(11);
    maxima = ordfilt2(CIM, sum(support(:)), support);
    [corner_x, corner_y] = find((cm==maxima).*(CIM>0));
end

function lines = findLines(I)
    edges = edge(I, 'canny', [0.2 0.25]);
    [H,T,R] = hough(edges);
    P = houghpeaks(H, 100, 'threshold', 0.3*max(H(:)));
    lines = houghlines(edges, T, R, P, 'FillGap', 200, 'MinLength', 500);
end

function [l] = segToLine(pts)
    a = [pts(1,:)';1];
    b = [pts(2,:)';1];
    l = cross(a,b);
    l = l./norm(l);
end

function IAC = get_IAC(l_infs, vps, vp1s, vp2s, H)
% Returns the Image of the absolute conic
% Full code in '2 - Rectification/Functions'
end

function [ParG,code] = AtoG(ParA)
% Conversion of algebraic parameters to geometric parameters
% ParG = [Center(1:2), Axes(1:2), Angle]'
% Full code in '2 - Rectification/Functions'
end
```

References

[1] Yisong Chen, Peng Lu, Wenhang Li, and Shaorong Wang. Metric planar rectification from perspective view via circles. In *2009 IEEE Symposium on Computational Intelligence for Image Processing*, pages 69–75, 2009.

[2] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge books online. Cambridge University Press, 2003.