POLITECNICO
MILANO 1863

Image Analysis and Computer Vision Homework

Ali Noshad, ali.noshad@mail.polimi.it, 101108

January 22, 2024

1 Feature Extraction and Line and Ellipse Detection

Consider the image PalazzoTe.jpg Using feature extraction techniques (including those implemented in Matlab) plus possible manual intervention, extract both the images *l1, l2* of useful genetrix lines and images *C1, C2* of useful circular cross sections.

1.1 Image pre-processing

After reading the image in matlab using the function *imread()* it appears that the image is loaded as landscape, so to be coherent with assignment we have to rotate the image. The image is rotated by 270 degrees. Figure 1 demonstrates the image rotation procedure.

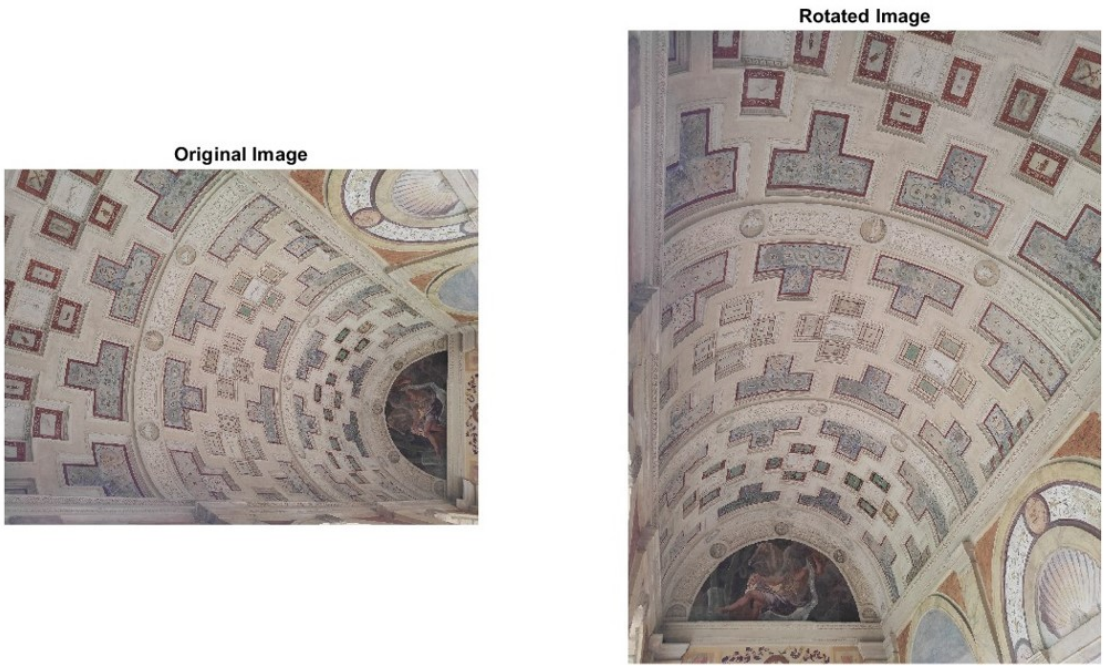

Figure 1: Image Rotation

Transforming images into gray scale considered one of the pre-processing step in various computer vision and image processing tasks. This technique simplifies the image data and reduces the dimensionality, making it easier to process and analyze. Color information can sometimes introduce additional noise in the data. By converting an image to grayscale, you may reduce the impact of color-related noise, making it easier to identify patterns or features in the image. Furthermore, it is essential since methods that we want to use are required a grayscale image. Figure 2 (left) illustrates the grayscale transformation results. We have also tried the negative transformation of the image, by converting the pixel values in of the image to their complements. It transform the image such a way that dark areas become bright, and vice versa. This method can sometimes be useful to enhance the image features and contrast of the image. Figure 2 (right) demonstrate the negative transformation results.

Before we go any further, it is better to do some analyze on the image to better understand our data. By looking at image we can see that image is tend to be more white, so we can see form the histogram analysis the pixel density is more between 100 to 200, so we equalized the image with histogram equalization transformation. This method increases the global contrast of the image specially when the pixel density is represented by a narrow range of intensity values. Through this adjustment, the intensities can be better distributed on the histogram utilizing the full range of intensities evenly. Figure 3 illustrates the results of the histogram analysis of the image.

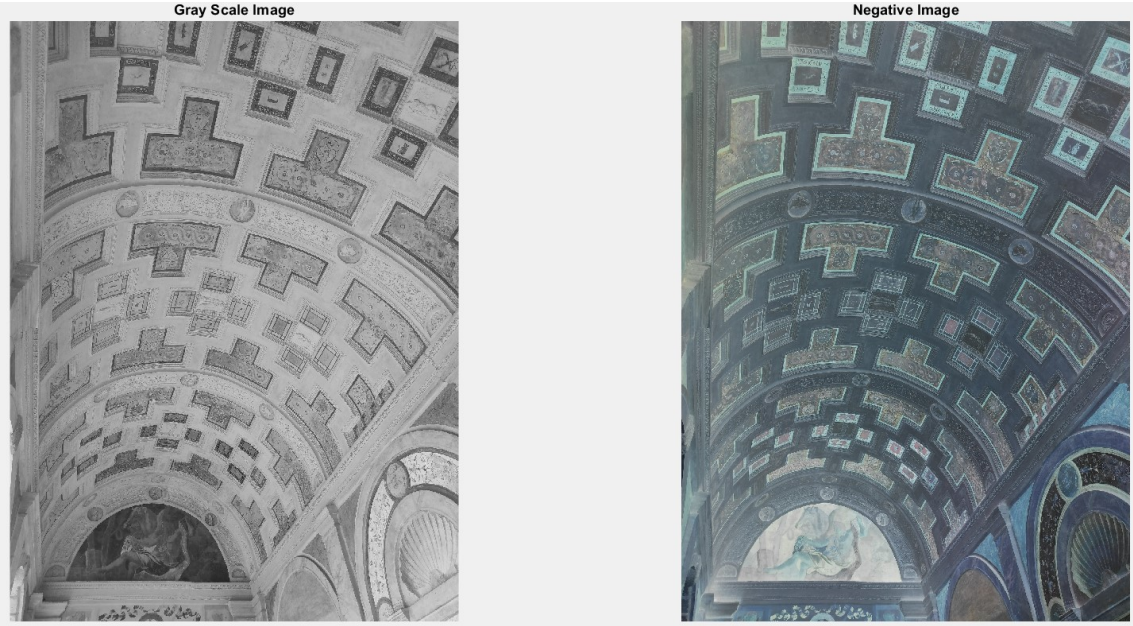

Figure 2: Gray Scale and Negative Transformation

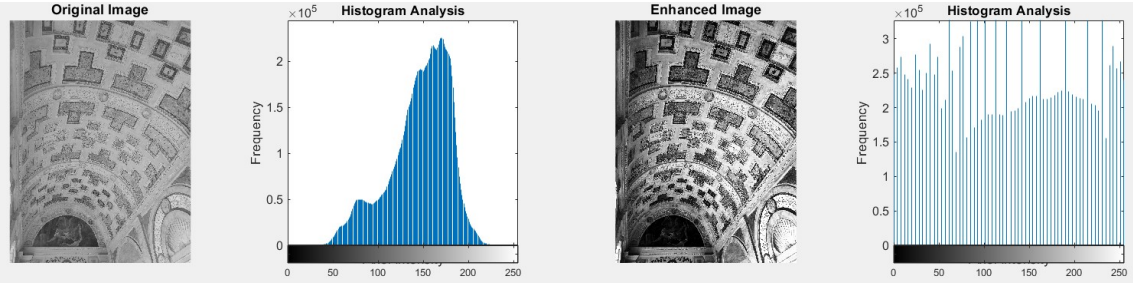

Figure 3: Histogram analysis

1.2 Image Thresholding

After this we tried all the experiments with both the gray scale and also the equalized image. First we performed thresholding approach, in which we converted the gray scale image into the binary image. We used this method to select some important areas in the image such as edges. At first we used the threshold value of 0.5 (of course we tried other values as well) on the gray scale and equalized image, but the resulting image was completely white. Accordingly, we decided to try another approach called Otsu's method, in which it aims to find an optimal threshold value to separate an image into two classes, typically foreground and background, by minimizing the intra-class variance or maximizing the inter-class variance. Figure 4 shows the results of the Thresholded method applied to the gray scale and equalized image.

1.3 Sobel Edge Detector

Now, we tried the edge detection methods implemented in Matlab. We started with **Sobel Edge Detector**, this method performs a 2-D spatial gradient measurement on the image and it emphasizes the regions of high spatial frequency, which corresponds to edges, this method is based on the pair of 3*3 convolution kernel, which one of them is the 90 degree rotation of another kernel, one kernel for each of the two perpendicular orientations. The results of **Sobel Edge Detector** are shown in the Figure 5. In these results we analysis both the gray scale and equalized image to see the difference. An important note on this model or all of the edge detection method that we used is the threshold value. In this case after trying several values, we finally conclude 0.03, which for the Sobel operator,

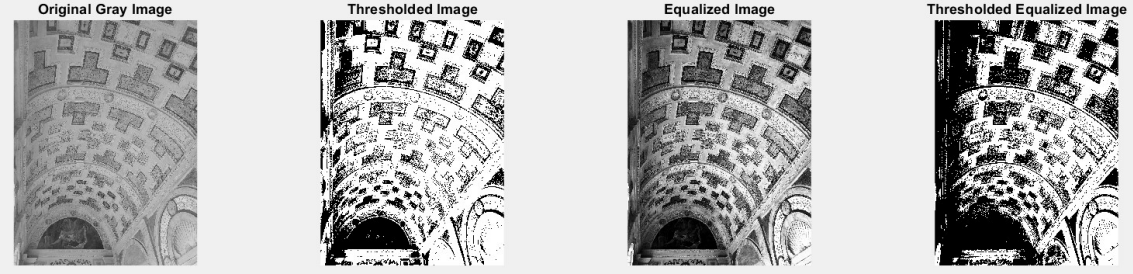

Figure 4: Thresholded analysis

the threshold determines the minimum gradient magnitude required to consider a pixel as an edge.

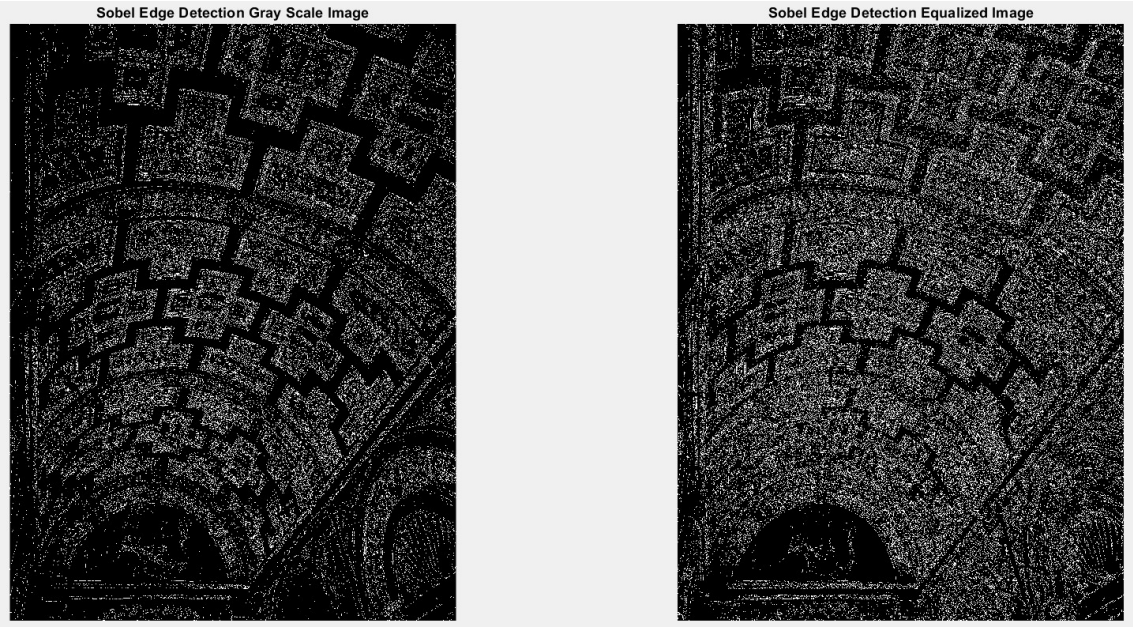

Figure 5: Sobel Edge Detecion on GrayScale and Equalized Image

1.4 Prewitt Edge Detection

Next, we tried the **Prewitt Edge Detection** as above on the both gray scale and equalized Image, and we also set the threshold on 0.02. Like the Sobel Edge Detection method, this operator has the ability to detects two types of edges (horizontal and vertical edges). This method is works on pixel intensities and in case of the sudden change, an edge is detected by the mask, because the edges are detected by changes in pixel intensities, it calculates differentiation. The results of **Prewitt Edge Detection** are shown in the Figure 6. Again, we can see that the results of edge detection on the histogram equalized image is weaker. After researching and analysing more closely the results, we discovered that histogram equalization is a contrast enhancement technique that redistributes the intensities of pixels in an image to cover the entire intensity range more uniformly. While this can improve the overall contrast of an image, it may also alter the distribution of pixel values in a way that affects the effectiveness of edge detection algorithms. Furthermore, we also tried to increase the sensitivity threshold in case of enhanced image, this approach helped the performance, but still there were some noise in the results.

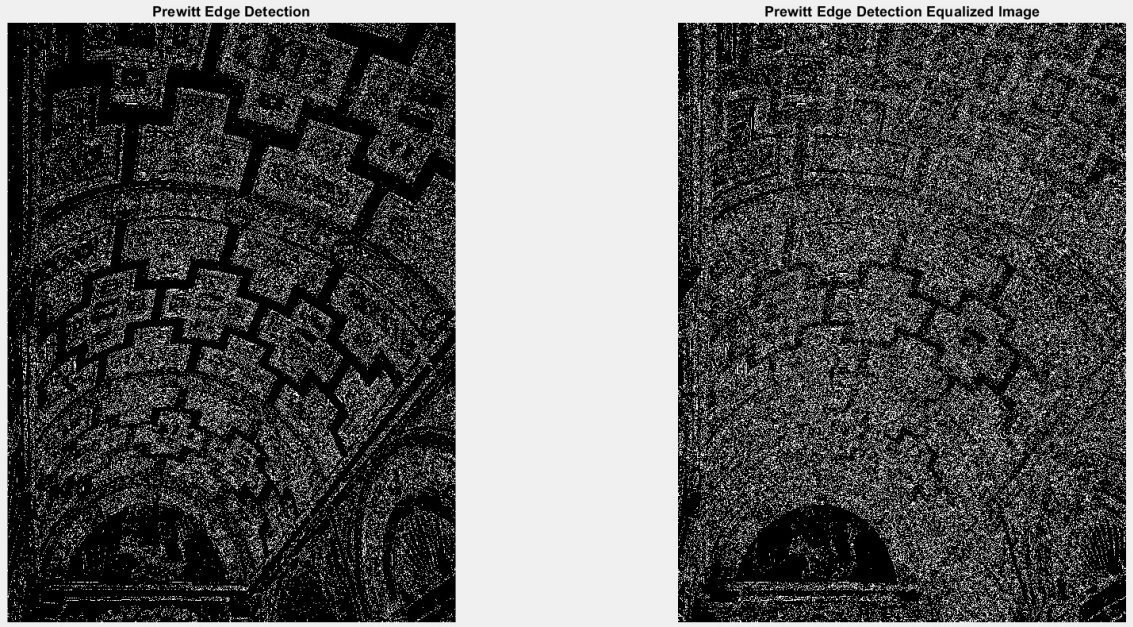

Figure 6: Prewitt Edge Detection on Gray Scale and Equalized Image

1.5 Canny Edge Detection

For the next edge detection method we tried the **Canny Edge Detection**. As a contrast to the **Sobel Edge Detector** edge detection method, **Canny Edge Detection** provides some improvements. First, it applies an noise reduction, since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. Then, like the **Sobel Edge Detector** it finds the intensity gradient of the image, after getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. Finally, it defines two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. The results of **Canny Edge Detection** are shown in the Figure 7. An interesting note, as we can see the results of the gray scale and equalized image, we can see that **Canny Edge Detection** works better on the equalized image, based on the our understanding it can be the results of applying the noise reduction (Gaussian Filter), since as we mentioned previously, equalized image is contains a lost of noise. For this experiment, a lower threshold (0.02 in this case) and an upper threshold (0.3 in this case) were selected, and 15 is the size of the Gaussian filter kernel used in the **Canny Edge Detection** algorithm.

1.6 Laplacian of Gaussian (LoG)

Edge detection algorithms like the **Sobel Edge Detector** work on the first derivative of an image. One limitation with the approach above is that the first derivative of an image might be subject to a lot of noise. Local peaks in the slope of the intensity values might be due to shadows or tiny color changes that are not edges at all. An alternative to using the first derivative of an image is to use the second derivative, which is the slope of the first derivative curve, an edge occurs where the graph of the second derivative crosses zero. This second derivative-based method is called the Laplacian algorithm. The Laplacian algorithm is also subject to noise. In order to solve this problem, a Gaussian smoothing filter is commonly applied to an image to reduce noise before the Laplacian is applied. This method is called the **Laplacian of Gaussian (LoG)**. The results of **Laplacian of Gaussian (LoG)** are shown in the Figure 8. Also, in this approach since we are using the a noise reduction method, the results of the equalized image are better.

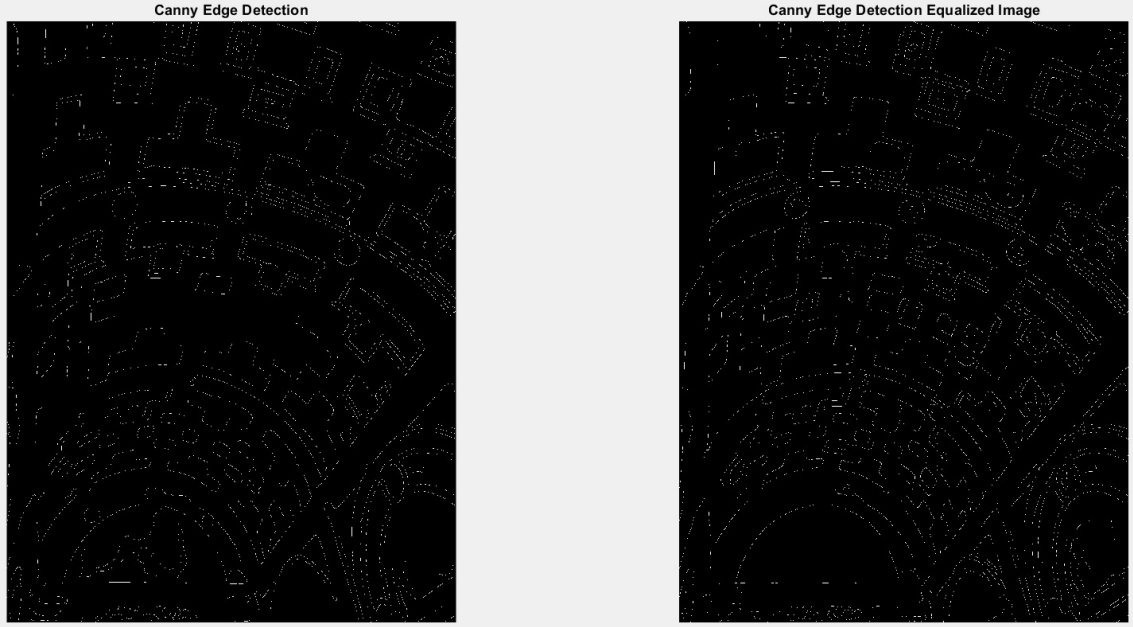

Figure 7: Canny Edge Detection on Gray Scale and Equalized Image

1.7 Roberts operator

The **Roberts operator** performs a simple, 2-D spatial gradient measurement on an image. Like after mentioned methods it highlights regions of high spatial frequency which often correspond to edges. In theory, the operator consists of a pair of 2×2 convolution kernels, One kernel is simply the other rotated by 90°. This is very similar to the **Sobel Edge Detector**. These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The results of **Roberts operator** are shown in the Figure 9.

1.8 Harris' corner detector

Compared to the previous one, **Harris' corner detector** takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45-degree angles, and has been proved to be more accurate in distinguishing between edges and corners. The results of **Harris' corner detector** are shown in the Figure 10. Again this methods works better in case of equalized image.

1.9 YCbCr

Next, for a number of reasons, the **YCbCr** color space is frequently utilized in image and video processing. Its ability to separate the chrominance (Cb and Cr) and luminance (Y) components is one of its main advantages. Since the human visual system is more sensitive to changes in brightness than in color, this separation enables more effective compression. Because changes to the chrominance components have a smaller perceived impact than changes to the luminance component, **YCbCr** also makes color modifications and processing simpler. For many image and video applications, such as compression, transmission, and color correction, **YCbCr** is the best option because of this. The resulting Y, Cb, and Cr components are displayed in 11.

1.10 Contrast stretching

In the next step we performed the contrast stretching, which enhance the visibility of details in images by expanding the range of pixel intensities. This way it may help to detect edges better. Equation 1 shows the operation which we performed on the image. Figure 12 shows the resulting image as compared to original image after contrast streching.

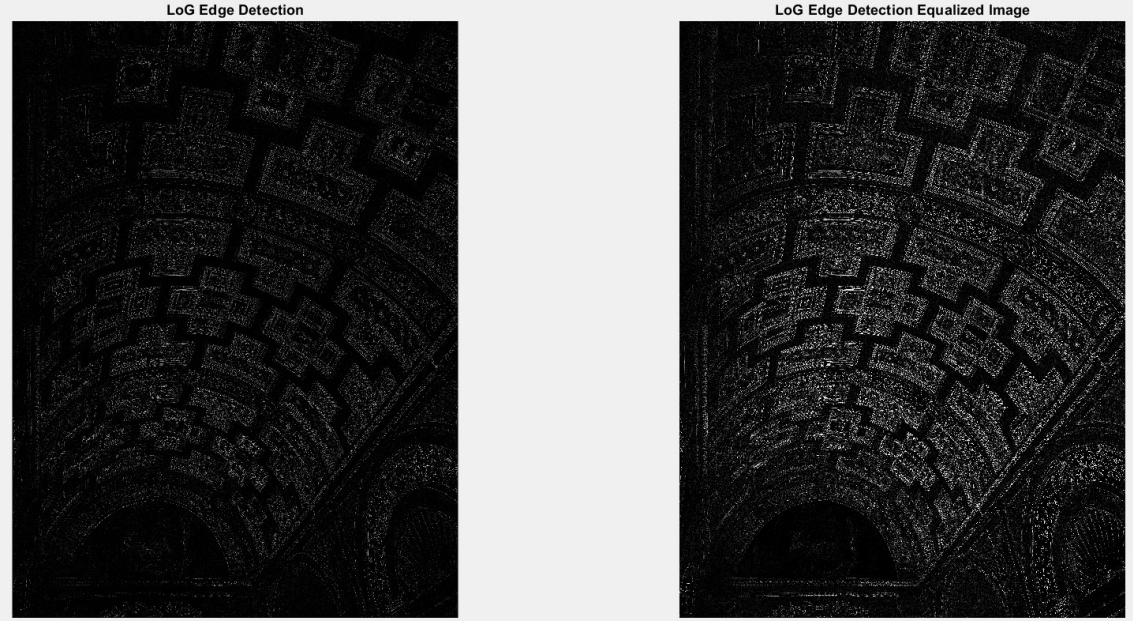

Figure 8: Laplacian of Gaussian (LoG) Detection on Gray Scale and Equalized Image

$$\text{contrastStretchedImage} = \frac{\text{originalImage} - \text{minIntensity}}{\text{maxIntensity} - \text{minIntensity}} \tag{1}$$

1.11 Image masking and color selection

Then, to use the colors in the image for detection of the edges, we created a pipline which at first we select multiple points on the image, the RGB values of these points are stored, and a binary mask is generated by considering pixels with RGB values within a specified tolerance of the selected points. This mask is then applied to the original image, creating a masked image that retains only the regions similar to the selected points. Since our task was to detect the cross sections and generix lines we selected those points on the locations where these two exists. Figure 13 demonstrates the results of the masked image. As we see in the Figure 13, we this approach we better manage to eliminate some unnecessary parts on the image, of course we can use the resulting image for further analysis, and as input of the edge detection methods. We also implemented this approach on the gray scale image, but the results were not satisfying, we assume it may because of the brightness of the image (implementation exists in Matlab file).

1.12 Image convolution

In this section, we describe a MATLAB code for edge detection in a grayscale image (`grayImage`). The code begins by converting the grayscale image to double precision. To enhance the visibility of edges, the we employed a combination of filters. A smoothing filter F is defined as:

$$F = \frac{1}{\|F\|} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

The horizontal edge detection filter (gx) and vertical edge detection filter (gy) are defined as:

$$gx = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} / \sqrt{2} \quad \text{and} \quad gy = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

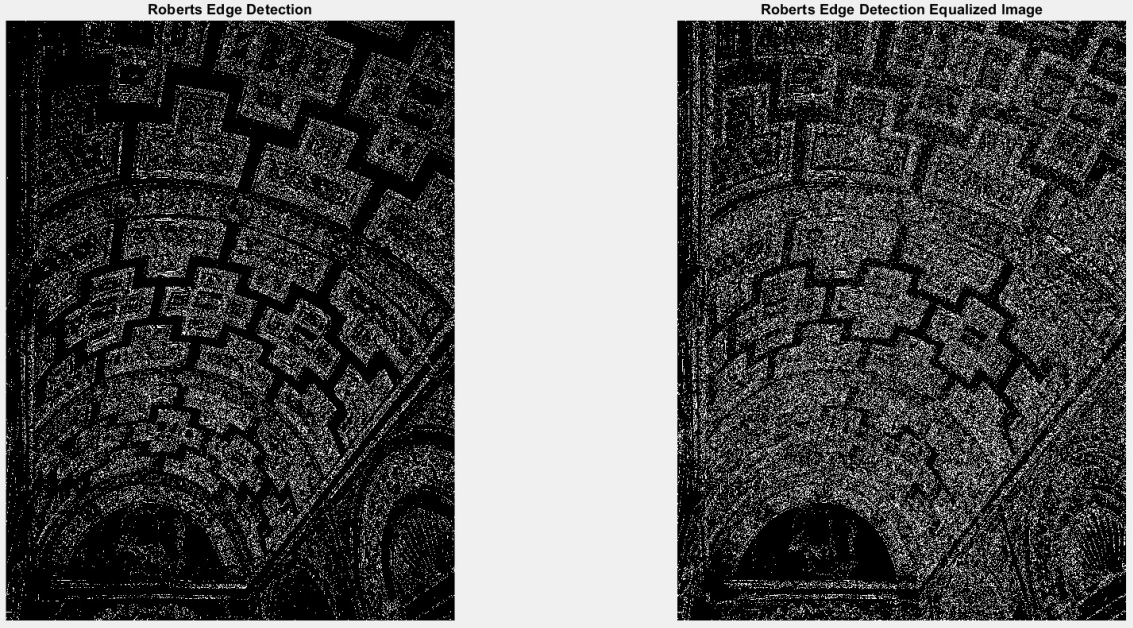

Figure 9: Roberts Detection on Gray Scale and Equalized Image

The grayscale image is then convolved with the smoothing filter, and horizontal (`vert_`) and vertical (`hor_`) gradient images are obtained using gx and gy respectively. The magnitude of the gradients is calculated as:

$$\text{filteredImage} = \sqrt{\text{vert\_}^2 + \text{hor\_}^2}$$

To further refine the edge detection, the Canny edge detector is applied with specified thresholds [0.1, 0.2] and a smoothing parameter of 6:

$$\text{cannyEdge} = \text{edge}(\text{filteredImage}, \text{'canny'}, [0.1, 0.2], 6)$$

Connected components in the resulting binary edge map (cannyEdge) are labeled, and the labeled edges are visualized using a pseudo-color representation. Figure 14 shows the labeled edges (Since the figures may not be really clear I also saved figures with Matlab, which can be seen in the folder).

1.13 Image convolution (Gaussian and Scharr filters)

As another approach, we enhanced edges in a grayscale image (`grayImage`) through a two-step filter process. First, a 3x3 Gaussian filter is applied to the image for smoothing, defined as:

$$f_{\text{Gaussian}} = \begin{bmatrix} a & a & a \\ a & b & a \\ a & a & a \end{bmatrix}$$

where $a = 0.003744$ and $b = 0.970049$. The image is then convolved with this Gaussian filter:

$$\text{smooth\_image} = \text{conv2}(\text{im2single}(\text{grayImage}), f_{\text{Gaussian}}, \text{'same'})$$

Next, the Scharr filter is applied for edge detection:

$$f_{\text{Scharr}} = \frac{1}{sum(|f_{\text{Scharr}}|)} \begin{bmatrix} -3-3i & 0-10i & 3-3i \\ -10 & 0 & 10 \\ -3+3i & 0+10i & 3+3i \end{bmatrix}$$

The vertical and horizontal derivatives are computed by convolving the smoothed image with the transpose and original Scharr filter, respectively:

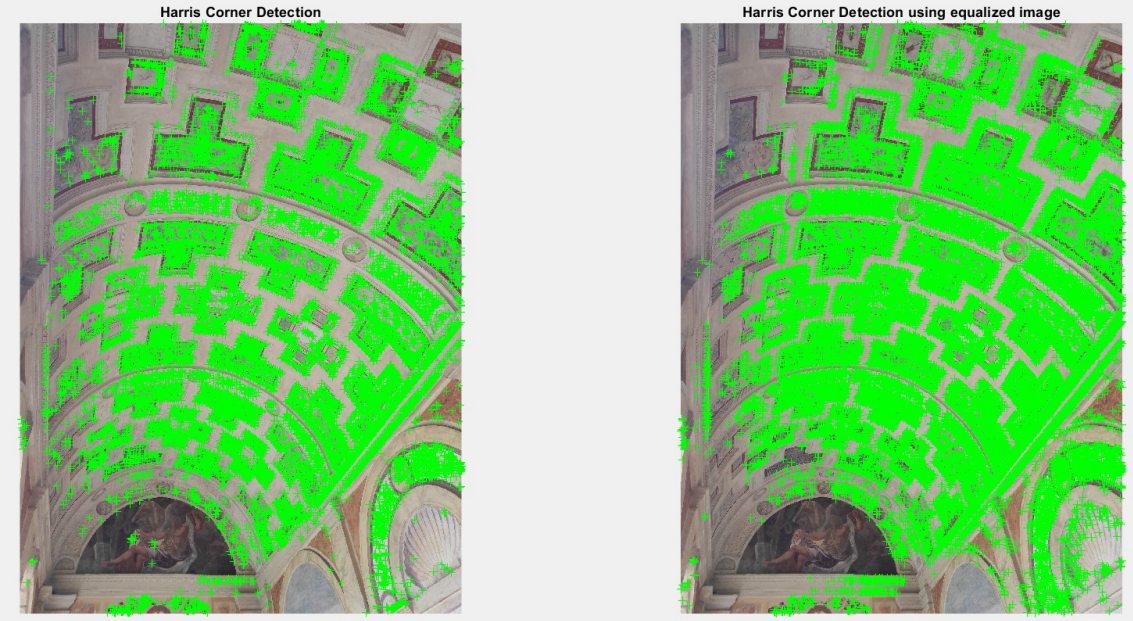

Figure 10: Harris' Corner Detection on Gray Scale and Equalized Image

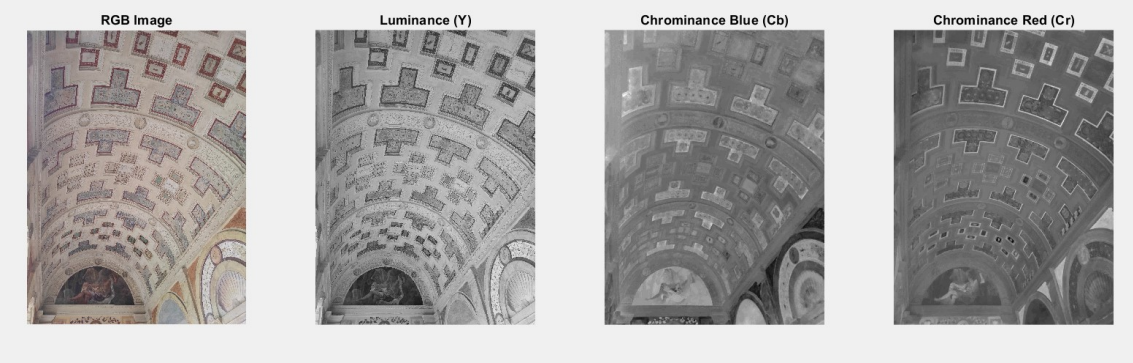

Figure 11: Luminance (Y) and Chrominance (Cb and Cr) components separation on Image

$$\text{vert\_} = \text{conv2}(\text{smooth\_image}, f_{\text{Scharr}}^T, \text{'same'}) \quad \text{and} \quad \text{hor\_} = \text{conv2}(\text{smooth\_image}, f_{\text{Scharr}}, \text{'same'})$$

Finally, the gradient magnitude is calculated:

$$\text{scharrfilteredImage} = \sqrt{\text{vert\_}^2 + \text{hor\_}^2}$$

The original image and the Scharr-filtered image are displayed in Figure 15 for visual comparison.

1.14 Line detection with Canny edge detector and Hough transform

As another in another experiment we performed line detection on the previously equalized image using the Canny edge detector and the Hough transform. First, the Canny edge detection algorithm is applied with adjusted thresholds [0.01, 0.1] and a smoothing parameter of 30:

$$\texttt{edges} = \texttt{edge}(\texttt{enhancedImage}, \texttt{'canny'}, [0.01, 0.1], 30)$$

The resulting binary edge map is then processed using the Hough transform, yielding a Hough transform matrix (`H`) with corresponding angles (`theta`) and distances (`rho`). Peaks in this matrix are detected using an adjusted threshold (in this case, 10% of the maximum value):

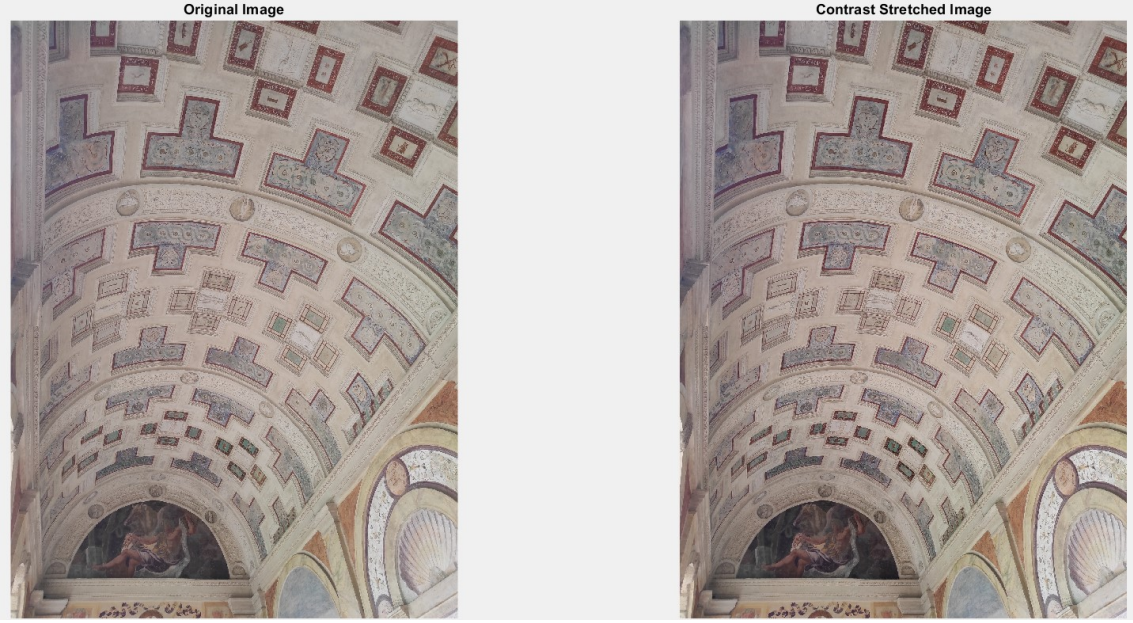

Figure 12: Contrast Streching on Image

$$\texttt{peaks} = \texttt{houghpeaks}(\texttt{H}, 50000, \texttt{'threshold'}, \texttt{ceil(0.1 * max(H(:)))})$$

Line segments are then extracted from the Hough transform matrix and peaks, considering gap-filling tolerance and minimum length parameters:

$$\texttt{lines} = \texttt{houghlines}(\texttt{edges}, \texttt{theta}, \texttt{rho}, \texttt{peaks}, \texttt{'FillGap'}, 30, \texttt{'MinLength'}, 10)$$

Finally, the detected lines are overlaid on the original image (`Image`) for visualization, as it can be seen in Figure 16.

2 Geometry

2.1 Finding horizon

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

2.1.1 Theory

As it can be seen, in the image there are two cross sections, in which we can consider them as circumference *C1, C2*. Based on the theory we can understand that a conic is a curve described by a second-degree equation in the plane.The general form of a conic in Euclidean coordinates is given by:

$$aX^2 + bXY + cY^2 + dX + eY + f = 0$$

Here, a, b, c, d, e, and f are coefficients that determine the specific characteristics of the conic section. The conic can be classified as an ellipse, parabola, or hyperbola. In matrix form can be written as:

$$X^T C X = 0$$

where the conic coefficient matrix C is given by

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

Figure 13: Image Masking and Color Selection

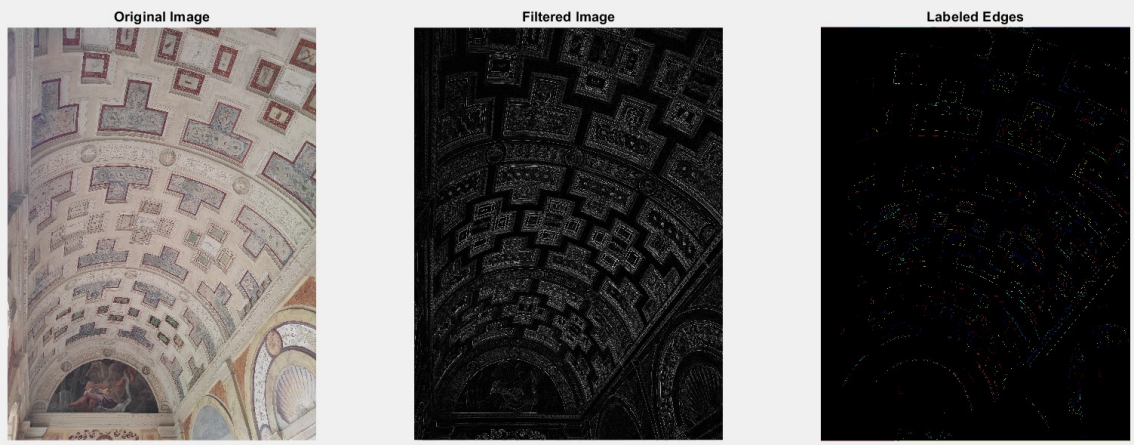

Figure 14: Filtered Image and Convolution

Let's denote the line at infinity as L_∞. The intersection of a conic with L_∞ can be analyzed by homogenizing the equation of the conic.

$$x^2 - 2X_0w + X_0^2w^2 + y^2 - 2Y_0w + Y_0^2w^2 - r^2w^2 = 0$$

$$w = 0$$

$$x^2 + y^2 = 0$$

The two intersection points are the same for all circumferences, all circumferences cross the L_∞ at the same two points I and J. The circular points I and J can be described by:

$$I = \begin{bmatrix} 1 \\ i \\ 0 \end{bmatrix} \; J = \begin{bmatrix} 1 \\ -i \\ 0 \end{bmatrix}$$

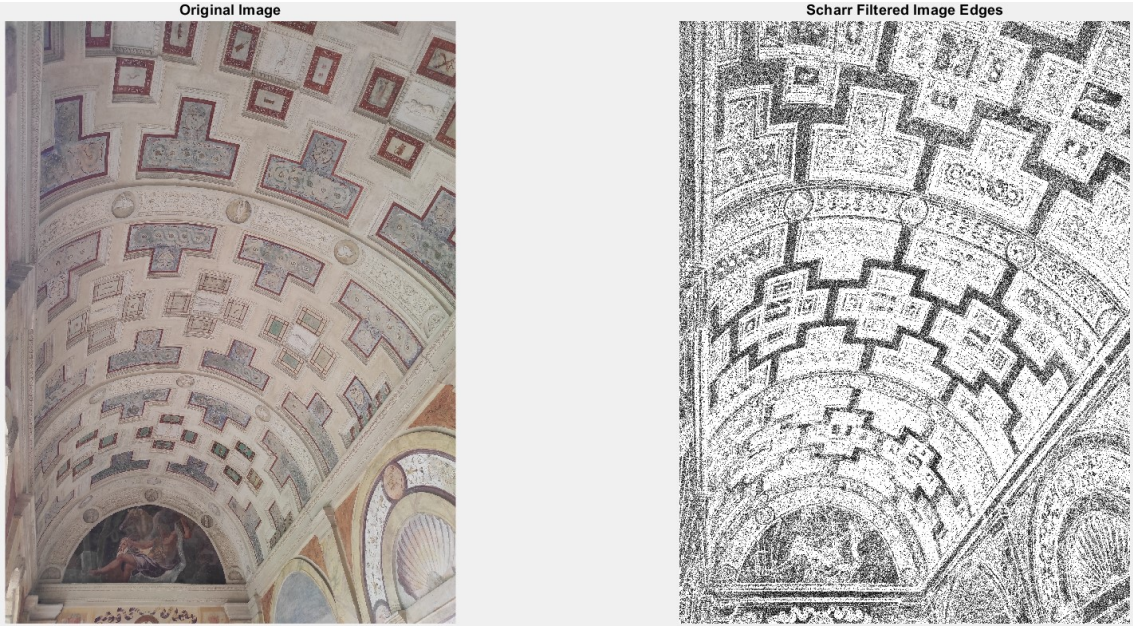

Figure 15: Edge Enhancement with Gaussian and Scharr Filters

and by multiplying these circular points we will get the results the horizon line. So in the image we have two cross section *C1, C2*. We can consider these cross section as image of full circumference and by intersecting the circumference with circular points we get the following equations:

$$I'TC_1I' = 0 \tag{2}$$
$$J'TC_2J' = 0 \tag{3}$$

The equations (2) and (3) are homogeneous quadratic equations. These equations are derived from the fact that, in projective geometry, a circle can be represented by a quadratic equation, and points on a circle satisfy this equation. $I'TC_1I' = 0$ means that the image of the circular point I' lies on the image of circumference C_1, similarly, $J'TC_1J' = 0$ means that the image of the circular point J' lies on the image of circumference C_2. By solving these two equations simultaneously, you determine the coordinates of the points *I'* and *J'*. These coordinates, in turn, define the horizon h, which is the line connecting *I'* and *J'*. So, we can open the above equation as below:

$$C_1 = \begin{bmatrix} a_1 & b_1/2 & d_1/2 \\ b_1/2 & c_1 & e_1/2 \\ d_1/2 & e_1/2 & f_1 \end{bmatrix} C_2 = \begin{bmatrix} a_2 & b_2/2 & d_2/2 \\ b_2/2 & c_2 & e_2/2 \\ d_2/2 & e_2/2 & f \end{bmatrix}$$

$$I' = \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} J' = \begin{bmatrix} x2 \\ y2 \\ 1 \end{bmatrix}$$

the equations become like this after multiplication:

$$a_1x_1^2 + b_1x_1y_1 + c_1y_1^2 + d_1x_1 + e_1y_1 + f_1 = 0$$

$$a_2x_2^2 + b_2x_2y_2 + c_2y_2^2 + d_2x_2 + e_2y_2 + f_2 = 0$$

Solving these equations we will found the *I'* and *J'*. and following this we compute the h as below:

$$h = I' * J' \tag{4}$$

Figure 16: Line Detection with Canny Edge Detector and Hough Transform

2.1.2 Matlab

The script fits a conic to a set of five points, where *C1* is the matrix representing the conic. This is done using the null space (null) of a matrix constructed from the points. The coefficients of the conic are extracted and stored in variables *a1, b1, c1, d1, e1, f1.* Please be note that the first points c and d are at the cross section intersection with generix lines.

```
% Select points interactively
[x, y] = getpts();
% Fit conic to five points
c = [x(1); y(1); 1];
d = [x(5); y(5); 1];
A = [x.^2, x.*y, y.^2, x, y, ones(size(x))];
N = null(A);
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];
```

Fit another conic conic to a set of five points, where *C2* is the matrix representing the conic. This is done using the null space (null) of a matrix constructed from the points. The coefficients of the conic are extracted and stored in variables *a2, b2, c2, d2, e2, f2.* Please be note that the first points a and b are at the cross section intersection with generix lines.

```
% Fit another conic to five more points
[x, y] = getpts();
a = [x(1); y(1); 1];
b = [x(5); y(5); 1];
A = [x.^2, x.*y, y.^2, x, y, ones(size(x))];
```

```
N = null(A);
cc = N(:, 1);
[a2, b2, c2, d2, e2, f2] = deal(cc(1), cc(2), cc(3), cc(4), cc(5), cc(6));
C2 = [a2, b2/2, d2/2; b2/2, c2, e2/2; d2/2, e2/2, f2];
[a1, b1, c1, d1, e1, f1] = deal(C1(1), C1(2)*2, C1(5), C1(3)*2, C1(6)*2, C1(9));
[a2, b2, c2, d2, e2, f2] = deal(C2(1), C2(2)*2, C2(5), C2(3)*2, C2(6)*2, C2(9));
```

Then we set up two equations representing the conics *C1, C2* intersecting with image of circular points *I'* and *J'* It solves the system of equations using the solve function and stores the solutions in the variable *S*. We took the s3 and s4 solutions, since later we can discover these two are acceptable.

```
% Solve the equations for the conics
syms 'x' 'y';
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];
S = solve(eqns, [x, y]);
% Get the intersection points
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];
II = s3;
JJ = s4;
horizon = hcross(II, JJ);
```

The solutions can be seen below, also in the Figure 22 we can see the red line which is perpendicular to the cylinder axies is the horizon.

```
    JJ                II                          Horizon
    1.0e+03 *            1.0e+03 *                    0.0002
                                                     -0.0004
       2.6146 + 3.8525i     2.6146 - 3.8525i          1.0000
      -0.0845 + 1.3897i    -0.0845 - 1.3897i
       0.0010 + 0.0000i     0.0010 + 0.0000i
```

2.2 Finding cylinder axis

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

2.2.1 Theory

According to the question we are required to find the cylinder axies, since we have two cross sections, which in the previous question we considered them as two circumference *C1, C2*, we know that the cylinder axies must pass through the center of these circumference, so theoretically if we manage to find the two centers of the circumferences we know that the line that connect these two centers is the cylinder axies.

So first we need to describe the polar line. 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. The Figure 17 illustrates this.

Now from the theory we know that the polar line of the point at the center of the circumference is the line at infinity, as it shows in the Figure 18.

This can be proven by the below discussion, we know that the equation of the polar line is:

$$l = C * y \tag{5}$$

if we open this equation and replace the center coordinates of the circumference in y we can see:

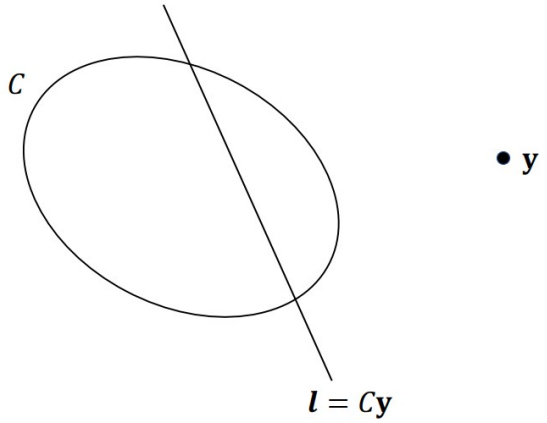

Figure 17: Polar Line w.r.t a Conic

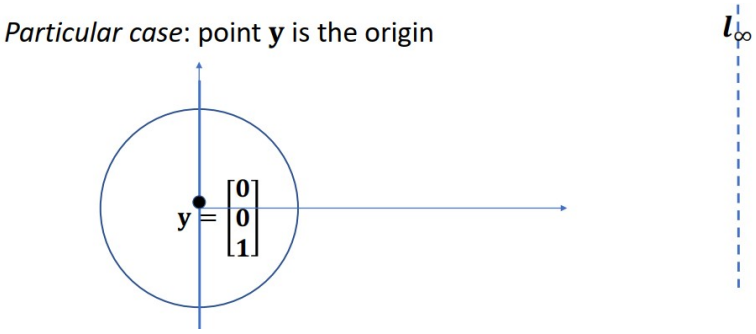

Figure 18: Polar Line of the center of Circumference

$$l = C * y = \begin{bmatrix} 1 & 0 & -X_0 \\ 0 & 1 & -Y_0 \\ -X_0 & -Y_0 & X_0^2 + Y_0^2 - r^2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -r^2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -r^2 \end{bmatrix}$$

So, we can understand that by multiplying the circumference with point at the origin we get the line at the infinity. So based on this theory, we can retrieve the center y by multiplying the inverse of the C (C^{-1}) with line at the infinity. We can extend this theory to our problem in which we want to find the centers of two cross sections (circumferences). Since we have found the horizon from the previous question we can get the centers coordinates based on the below equations:

$$Center_1 = inv(C_1) * h \tag{6}$$

$$Center_2 = inv(C_2) * h \tag{7}$$

where $Center_1$ and $Center_2$ are the centers of the two cross sections, and $inv(C1) = C1^{-1}, inv(C2) = C2^{-1}$. With this approach we can manage to find the center coordinates, and now if we multiply the two centers ($Center_1$ and $Center_2$) together we can get the line which can considered as the cylinder axies. So, in terms of equation it can be written as:

$$a = Center_1 * Center_2 \tag{8}$$

2.2.2 Matlab

Now that we described our procedure in terms of the theory, lets move with implementations. So, for computing the $Center_1$ and $Center_2$, as mentioned in the above paragraph we use the following code:

```
c1_center=inv(C1)*horizon; %compute center of c1
c2_center=inv(C2)*horizon; %compute center of c2
```

and the cylinder axies can be obtained by multiplying these points:

```
a=hcross(c1_center,c2_center); % compute cone axis a
```

the results are the following:

```
 lc1c2
    -0.0003
    -0.0001
     1.0000
```

we can see the results in the Figure 22 the blue line which connects the two centers and it is perpendicular to red line.

2.3 Computing the Calibration Matrix *K*

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

2.3.1 Theory

A scene point X in space is projected to an image point x via a projection matrix P in perspective projection as follows:

$$\lambda \mathbf{x} = P\mathbf{X} = K(R\mathbf{t})\mathbf{X}$$

The calibration matrix, often denoted as K, is a fundamental component in the camera calibration process in computer vision and photogrammetry. The process of calibration involves determining the inherent camera characteristics that make up matrix K. The general form of the calibration matrix K is as follows:

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

The components of the calibration matrix consists of f_x and f_y are the focal lengths of the camera in the x and y directions, respectively. They represent the scaling factor between the camera's image plane and the 3D world. s represents the skew or distortion between the x and y axes. In practice, this term is often zero (as in our case), assuming that the axes are perpendicular. u_0 and v_0 are the coordinates of the principal point, representing the optical center of the camera. They indicate the offset of the principal point from the top-left corner of the image.

From the theory we know that the calibration matrix K can be found using the image of absolut conic. The absolute conic Ω_∞ is a conic on the plane at infinity (Figure 19), defined as:

$$x^2 + y^2 + z^2 = 0$$
$$w = 0$$

and in the matrix for it can be written as:

$$\Omega_\infty = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

Given that a projective transformation H is turned into a conic as:

$$C' = H^{-T} * C * H^{-1} \tag{9}$$

When we apply the projection P to the image absolute conic w, we get:

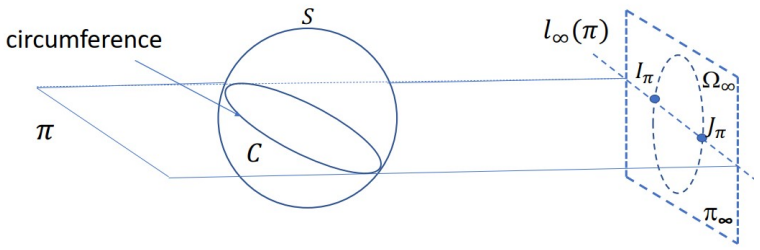

Figure 19: Absolute conic

$$w = P^{-T} I P^{-1} = (KR)^{-T} I (KR)^{-1} = (K^{-T}R) I (R^{-1}K^{-1}) = K^{-T}K^{-1}$$

we can see that the w depends only on the camera calibration matrix K, so if we look at the image absolute dual conic matrix w^* we can see:

$$w^* = w^{-1} = K^{-T}K^{-1} = \begin{bmatrix} f_x^2 + u_0^2 & u_0 v_0 & u_0 \\ u_0 v_0 & f_y^2 + v_0^2 & v_0 \\ u_0 & v_0 & 1 \end{bmatrix}$$

Thus, if we are able to find w, we have determined K as well. The image of the absolute conic is a symmetric matrix and has 4 degrees of freedom, so we need to get 4 equations to fully determine it. This matrix can be describe as:

$$\begin{bmatrix} \frac{1}{{f_x}^2} & 0 & -\frac{u_0}{{f_x}^2} \\ 0 & \frac{1}{{f_y}^2} & -\frac{v_0}{{f_y}^2} \\ -\frac{u_0}{{f_x}^2} & -\frac{v_0}{{f_y}^2} & \frac{{f_x}^2{f_y}^2 + {f_x}^2{v_0}^2 + {f_y}^2{u_0}^2}{{f_x}^2{f_y}^2} \end{bmatrix}$$

For this scope, we can exploit the vanishing points projective transformation H (that must belong to the conic), the system of equations is:

$$\begin{aligned} h_1^T \cdot w \cdot h_2 &= 0 \\ v_2^T \cdot w \cdot h_3 &= 0 \\ v_2^T \cdot w \cdot h_2 &= 0 \\ h_1^T \cdot w \cdot h_1 - h_2^T \cdot w \cdot h_2 &= 0 \end{aligned}$$

where we can obtain the equations from:

$$\mathbf{I'} = \mathbf{H_R^{'}} \mathbf{I} = \mathbf{H_R^{'}} \begin{bmatrix} 1 \\ 0 \\ i \end{bmatrix} = [\mathbf{h}_1 \ \mathbf{h}_2 \ \mathbf{h}_3] \begin{bmatrix} 1 \\ 0 \\ i \end{bmatrix} = \mathbf{h}_1 + i\mathbf{h}_2, (\mathbf{h}_1 + i\mathbf{h}_2)^T \omega (\mathbf{h}_1 + i\mathbf{h}_2) = 0$$

.

and vanishing point can be seen in the Figure **??**.

2.3.2 Matlab

We can estimate the rectifying homography H_R by SVD decomposition of degenerate dual conic to the image of circular points. The Singular Value Decomposition (SVD) of imDCCP is used to decompose the matrix into three matrices: *U, D,* and *V.*

```
imDCCP = II*JJ' + JJ*II'; % image of coinc dual to circular points
imDCCP = imDCCP./norm(imDCCP);

[U,D,V] = svd(imDCCP);
D(3,3) = 1;
```

```
A = U*sqrt(D);
H_R=inv(A); % shape reconstruction homography
H=inv(H_R);
```

The shape reconstruction homography represented by this matrix H_R transforms the captured image into a picture that resembles a planar face. However, the matrix H_R must be inverted in order to use the restrictions formula. Then after defining the 4 equation we use the solve these for equation to retrieve the unknowns.

```
h1=H(:,1);
h2=H(:,2);
h3=H(:,3);
eqs = [
        h1.'*w* h2;
        v2.'*w*h3;
        v2.'*w*h2;
        h1.'*w*h1-h2.'*w*h2;
      ];

res = solve(eqs);
fx = real(double(res.fx)); fx = fx(fx > 0); fx = fx(1);
fy = real(double(res.fy)); fy = fy(fy > 0); fy = fy(1);
u0 = real(double(res.u0)); u0 = u0(1);
v0 = real(double(res.v0)); v0 = v0(1);

K = [fx, 0, u0; ...
     0, fy, v0; ...
     0, 0, 1]
```

Finally, the resulting calibration matrix is as follows:

```
K=
   1.0e+03 *

        0.9657         0   -0.9196
             0    2.2387    3.0071
             0         0    0.0010
```

2.4 Determine the orientation of the cylinder axis w.r.t the camera reference

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

2.4.1 Theory

Imagine about a camera reference on the world reference and a point on a known planar item. These connections can be made (Figure 20):

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

where K is the calibration matrix of the camera, while H is the homography from the planar face to its image. We obtained this information from the previous steps. Finally the Roto-translation matrix can be written as:

$$R_t = \begin{bmatrix} i_\pi & j_\pi & i_\pi * j_\pi & o_\pi \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

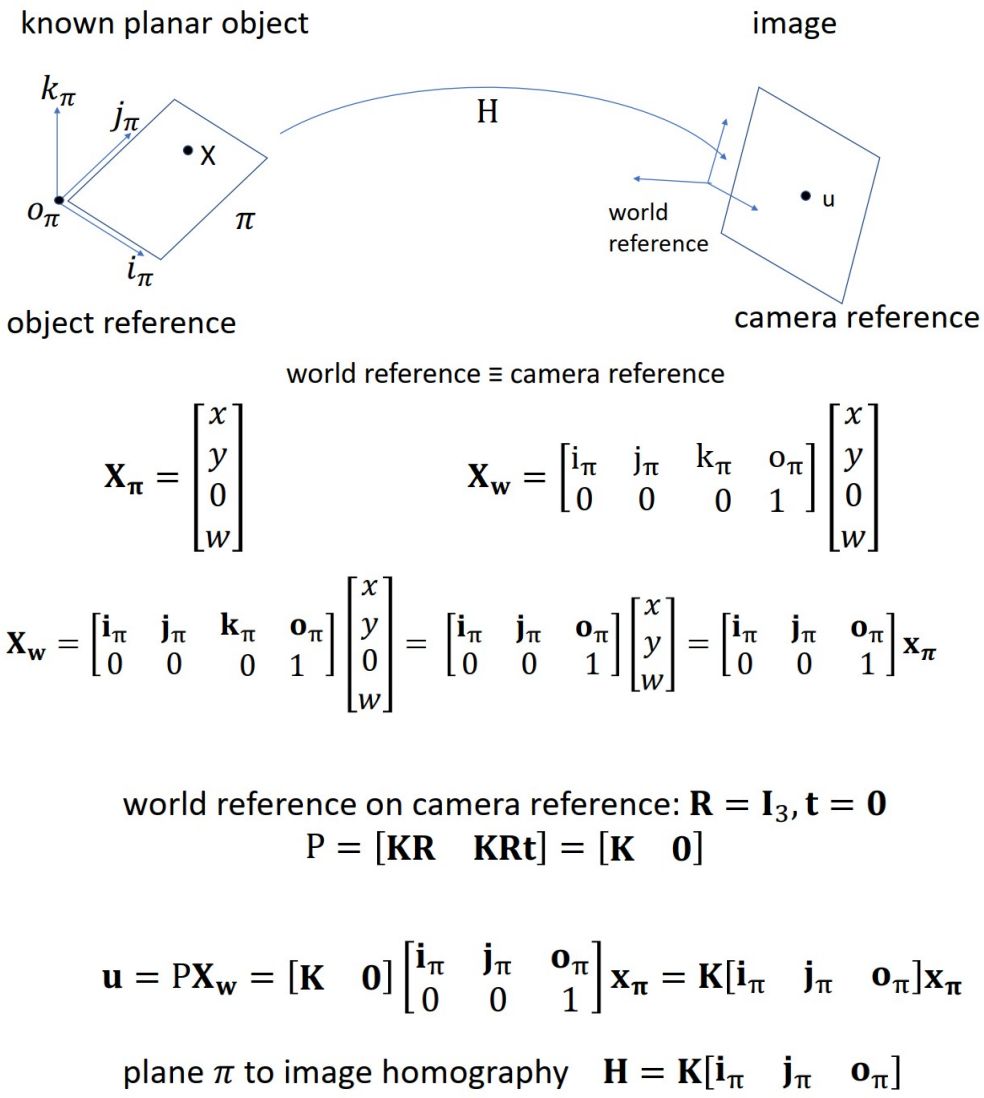

Figure 20: Object reference and camera reference (Image taken from slides)

The two circumferences centers are regarded as known planar positions for the purpose of obtaining the cylinder axis coordinate in the camera reference frame. The fact that the cylinder axis is the line connecting these two locations in the camera reference (or world reference) frame is trivial.

$$Center_1 - 3D = R_t * \begin{bmatrix} Center_1(1) \\ Center_1(2) \\ 0 \\ Center_1(3) \end{bmatrix} \quad Center_2 - 3D = R_t * \begin{bmatrix} Center_2(1) \\ Center_2(2) \\ 0 \\ Center_2(3) \end{bmatrix}$$

2.4.2 Matlab

The provided MATLAB code performs a 3D transformation of the centers of two circumferences, $c1$ and $c2$, from the image plane to the camera reference frame. The key steps of the code are explained below. The transformation matrix Q is computed by multiplying the inverse of the calibration matrix K with the homography matrix H. The basis vectors i_π, j_π, and k_π are extracted from the columns of matrix Q. The third basis vector k_π is computed as the cross product of i_π and j_π.The rotation and translation matrix R_t is constructed using the basis vectors i_π, j_π, k_π and the translation vector O_π (the third column of Q). The 3D coordinates of the centers of circumferences $c1$ and $c2$ in the

camera reference frame are computed by transforming the original 2D centers ($c1_{center}, c2_{center}$) using the rotation and translation matrix R_t.

The overall purpose of this code is to compute a 3D transformation of the centers of two circumferences (*C1, C2*) from the image plane to the camera reference frame. This transformation involves a combination of the calibration matrix, homography matrix, and subsequent normalization of the resulting homogeneous coordinates. Figure 21 illustrates the results concerning this computation.

```
Q=inv(K)*H;
i_pi=Q(:,1);
j_pi=Q(:,2);
O_pi=Q(:,3);
k_pi=cross(i_pi,j_pi);
R_t=[i_pi j_pi k_pi O_pi;
     0     0    0     1]

c1_3d=R_t*[c1_center(1);c1_center(2);0;c1_center(3)]; % compute c1 center in camera reference
c2_3d=R_t*[c2_center(1);c2_center(2);0;c2_center(3)]; % compute c2 center in camera reference
c1_3d=c1_3d./c1_3d(4)
c2_3d=c2_3d./c2_3d(4)
```

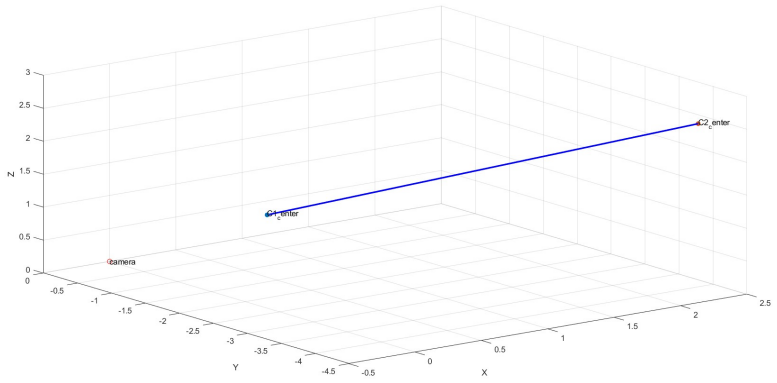

Figure 21: Cylinder axis w.r.t the camera reference

2.5 Computing the ratio

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

2.5.1 Theory

To solve this problem, we first need to find the matrix H such that, when multiplying this matrix with a image point part of a plane, it gives back the rectified version of it. We can apply this matrix to get the rectified version of the points measuring the radius and distance and the computing the ratio. Since our objective is to discover the ratio between two measures, these measurements are sufficient even though they are not the actual ones (since we cannot find the original size with this information alone). We can use the conic dual to the circular points and its image to compute matrix H.

$$C_\infty^* = IJ^T + JI^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and the image is:

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

as previously mentioned the intersection of the circumferences with line at infinity,

$$I = \begin{bmatrix} 1 \\ i \\ 0 \end{bmatrix} J = \begin{bmatrix} 1 \\ -i \\ 0 \end{bmatrix}$$

are the circular points and I' and J' are their images. Now the dual conic can be transformed based on projective transformation:

$$C'^{*} = H^{*}C^{*}H^{T}$$

based on the above equation we can understand that for C'^{*}_{∞}:

$$C'^{*}_{\infty} = H_r^{-1}C^{*}_{\infty}H_r^{-T}$$

According to the dual conic to the circular points which is mapped we can easily retrieve H_r using SVD (Singular Value Decomposition) method. The Singular Value Decomposition (SVD) is a factorization of a matrix into three other matrices. For a given matrix A, the SVD is represented as

$$A = U\Sigma V^{T} \tag{10}$$

Here, U is an orthogonal matrix whose columns are the left singular vectors of A,Σ is a diagonal matrix with singular values of A on its diagonal, and V^T is the transpose of an orthogonal matrix whose columns are the right singular vectors of A. So, if we have A as symmetric matrix, we have $U = V$, this is the cause also for us, since matrix C^{*}_{∞} is symmetric, so the equation is:

$$SVD(C'^{*}_{\infty}) = USU^{T} = H_r^{-1}C^{*}_{\infty}H_r^{-T}$$

and we can derive that $U = H_r^{-1}$. So, as previously mentioned we have already found the image of circular points, by intersecting the the two circumferences. Then we have found the image of dual conic to the circular points as it is mentioned in the equation $C'^{*}_{\infty} = I'J'^{T} + J'I'^{T}$, this was followed by applying the SVD to the to the image of dual conic to the circular points C'^{*}_{∞}, then we get the $H_r = U^{-1}$, after this we apply this matrix to the $Center_1$ and $Center_2$ and the also the two radius of the circumferences $C1, C2$, finally we compute the ratio.

2.5.2 Matlab

The image of points coincident with the dual to circular points (`imDCCP`) is computed using the circular points II and JJ:

```
imDCCP = II*JJ' + JJ*II'; % image of coinc dual to circular points
```

ormalization is performed by dividing each element of `imDCCP` by its Frobenius norm

```
imDCCP = imDCCP./norm(imDCCP);
```

The SVD of `imDCCP` is computed, decomposing it into matrices U, D, and V,and the third singular value in matrix D is set to 1. Matrix A is computed by multiplying U with the square root of D. The inverse of A is computed to obtain the shape reconstruction homography H_R.

```
[U,D,V] = svd(imDCCP);
D(3,3) = 1;
A = U*sqrt(D);
H_R=inv(A); % shape reconstruction homography
H=inv(H_R)
```

The `AtoG` (the one that used on the exercise session) function converts algebraic parameters of a conic section to geometric parameters. It operates on a parameter vector $[A, B, C, D, E, F]$ representing a conic $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$. The resulting geometric parameters include the center, axes, and orientation angle of the conic. In this code to calculate the ratio cross-sections $(C1, C2)$ radius to their distance we transformed $(C1, C2)$ using a homography matrix. The code calculates the ratio of the sum of radius to distance for both original and rectified cross-sections. These functions are used to compute the ratio and distance, respectively. They encapsulate specific mathematical computations. The `calculate_ratio` function computes a ratio involving the radii and distance, providing a measure of the geometry's proportions. The `calculate_distance` function uses the distance formula to find the Euclidean distance between two points.

```
par_geo_C1 = AtoG([C1(1), C1(2)*2, C1(5), C1(3)*2, C1(6)*2, C1(9)]);
par_geo_C2 = AtoG([C2(1), C2(2)*2, C2(5), C2(3)*2, C2(6)*2, C2(9)]);
c1_center = par_geo_C1(1:2);
c2_center = par_geo_C2(1:2);
radius1 = calculate_radius(c1_center, c);
radius2 = calculate_radius(c2_center, a);
distance = calculate_distance(c1_center, c2_center)
fprintf('The radius of the circle 1 is: %.2f\n', radius1);
fprintf('The radius of the circle 2 is: %.2f\n', radius2);
fprintf('The distance of the two cross section is: %.2f\n', distance);

C1_transformed = H' * C1 * H;
C2_transformed = H' * C2 * H;
c_transformed = H * c;
a_transformed = H * a;

par_geo_C1_rect = AtoG([C1_transformed(1), C1_transformed(2)*2, C1_transformed(5), C1_transformed(3)*
par_geo_C2_rect = AtoG([C2_transformed(1), C2_transformed(2)*2,
C2_transformed(5), C2_transformed(3)*2, C2_transformed(6)*2, C2_transformed(9)]);
c1_center_rec = par_geo_C1_rect(1:2);
c2_center_rec = par_geo_C2_rect(1:2);
radius1_rec = calculate_radius(c1_center_rec, c_transformed);
radius2_rec = calculate_radius(c2_center_rec, a_transformed);
distance_rec = calculate_distance(c1_center_rec, c2_center_rec)
fprintf('The radius of the circle is: %.2f\n', radius1_rec);
fprintf('The radius of the circle is: %.2f\n', radius2_rec);
fprintf('The distance of the two cross section is: %.2f\n', distance_rec);

% Calculate the ratio for original points
ratio_original = calculate_ratio(radius1, radius2, distance);
disp(ratio_original)
% Calculate the ratio for rectified points
ratio_rectified = calculate_ratio(radius1_rec, radius2_rec, distance_rec);
disp('The ratio of the radius to distance for rectified points is:');
disp(ratio_rectified);
disp('The ratio of the radius to distance for original points is:');
disp(ratio_original);
```

Here are the function used for the above code (we did put AutoG since it was used in the practical class):

```
function ratio = calculate_ratio(radius1, radius2, distance)
    % Calculate the ratio between the radius of the circular cross sections and their distance
    ratio = (radius1 + radius2) / distance;
end

function radius = calculate_radius(center, point_on_circle)
```

```
    % center and point_on_circle are 2-element vectors [x, y]
    radius = sqrt((point_on_circle(1) - center(1))^2 + (point_on_circle(2) - center(2))^2);
end


function distance = calculate_distance(point1, point2)
    x1 = point1(1);
    y1 = point1(2);
    x2 = point2(1);
    y2 = point2(2);
    % Calculate the distance between the two points using the distance formula
    distance = sqrt((x2 - x1)^2 + (y2 - y1)^2);
end
```

The ratio of the radius to distance for rectified points is: **3.0200**

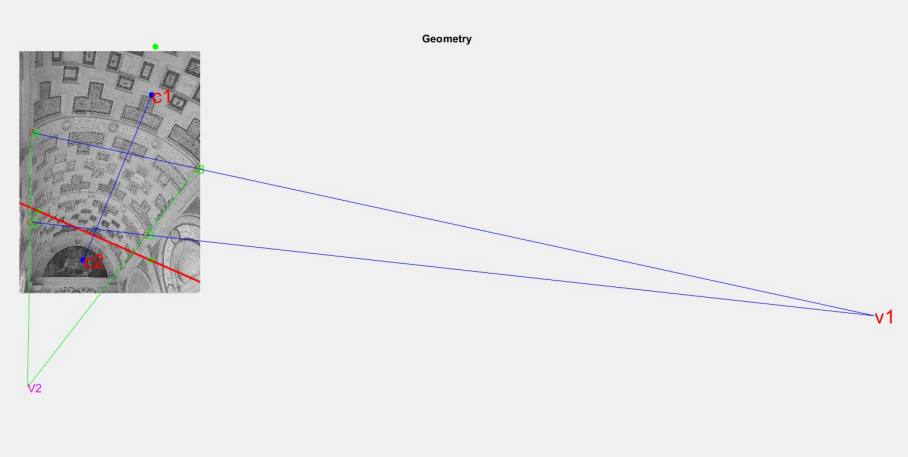

Figure 22: The results of all the geometry operations together on the image

3 Plot the unfolding

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

So, after analyzing the question we can understand that the question ask in a simple form **"Unfolding means unrolling. Imagine that you have a roll of tinfoil or of toilet paper. If you take a section of this and you place it on a plane, you have unfolded (or, unrolled) it. (from Professor)"**, so we need to actually open up the cylinder part which is between the two cross sections *C1, C2*. So, I tried first to implement this approach in a fake data to analyze the results.

Aim is to visualize a cylindrical structure in a 3D space along with its corresponding 2D projection. The cylinder is defined by two circular cross-sections with radii represented by zminRadius and zmaxRadius (the **max** belongs to the close cross section and the **zminRadius** belongs to smaller cross section in the **image**). The axial positions along the cylinder are denoted by **zValues**, while **thetaValues** represent angular positions around the cylinder axis.

Dummy data in the form of a cone is generated using the **rand** function. This cone data is then sorted based on angular positions to prepare for subsequent computations. The Intercept Theorem, specifically Thales' theorem, is employed to calculate the radii at different positions along the cylinder axis.

The next step involves projecting the 3D cone data onto a 2D plane. This is achieved by interpolating the cone data onto a grid corresponding to angular and axial positions. The resulting

planeData matrix represents the 2D projection of the cylindrical structure. This Figure 23 provides a comprehensive understanding of the procedure.

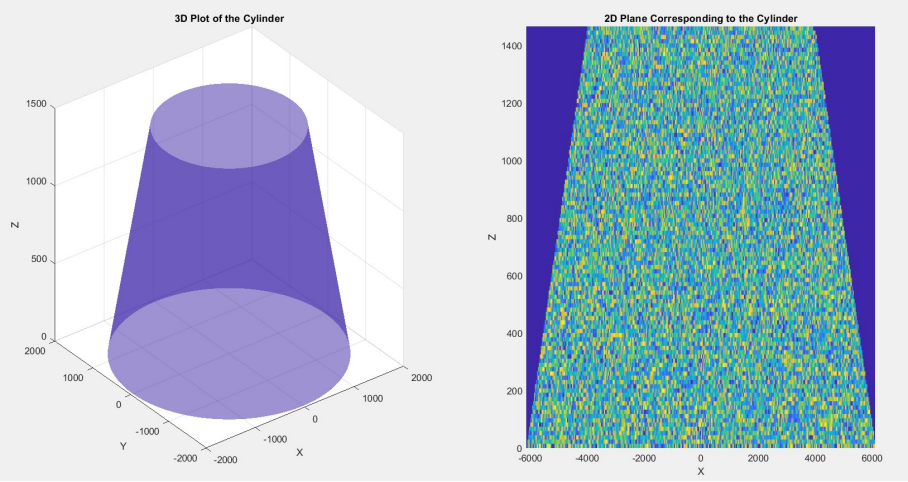

Figure 23: Unrolling a cylinder surface