POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Homework Report

Image Analysis and Computer Vision

Author: **Salvi Niccolò**

Student ID: 10773726
Academic Year: 2024-25

Contents

1 | Image Processing

The objective is to find straight lines, image of the circumference C and the image of the unknown planar curve S in the provided image.

Figure 1.1: Original image

1.1. F1: Straight Line Detection

The **Canny** method can be used to detect edges in the image. The algorithm works by first applying a smoothing Gaussian filter to the image (to remove noise), then computing the gradient of the image, applying nonmax suppression and tracking edges via hysteresis thresholding. Matlab already includes (in the Image Processing Toolbox) an implementation for this method in the edge function. First, we need to convert the image to grayscale. This way, the resulting image matrix will contain doubles in the [0,1] interval, representing the "intensity" of each pixel.

```
img = im2double(imread('look-outCat.jpg'));
imgGrayscale = rgb2gray(img);
```

Then we can simply use the edge function to apply the Canny method. A threshold can be specified to adjust the output image; in our case a value of [0.05, 0.2] seems to yield the best results.

```
BW = edge(Igray, 'canny', [0.05, 0.2]);
```

The result is pretty good, since we are able to identify some of the requested lines.

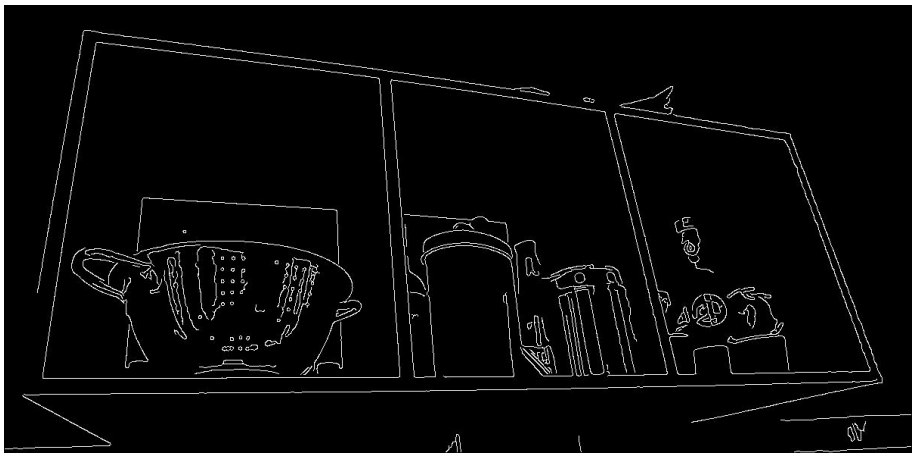


Figure 1.2: Canny

We can use the results of the previous step to detect straight lines in the image. This can be done using the **Hough transform**. A Hough transform of a datum (in this case a point on a detected edge) is a set of models compatible with the datum in the *model space*. For a point in the cartesian plane, the HT will be a sinusoid; collinear points will map to (almost) concurrent HTs.

The Hough transform is also included in Matlab:

```
[H, T, R] = hough(edges);
```

where `T` is θ, `R` is ρ and `H` is a matrix whose rows and columns correspond to ρ and θ values respectively.

We can then use the `houghpeaks` function to detect peaks, and pass the results to `houghlines` to extract lines.

```
P = houghpeaks(H, 100, 'threshold', 0.3*max(H(:)));
hlines = houghlines(edges, T, R, P, 'FillGap', 4, 'MinLength', 14);
```

These functions include various configuration options, namely:

- `numpeaks` is the number of peaks to detect (set to 20 here)
- `threshold` is the minimum value to be considered a peak. By default it is set to half the maximum value in the `H` matrix. For our image, we can get better results by lowering it to 30% of the max value.

- `FillGap` is, quoting the Matlab documentation, *the distance between two line segments associated with the same Hough transform bin*. In practice, this value controls the max distance between two detected segments for them to be merged into one segment.
- `MinLength` controls the minimum length for a segment. Setting this to a higher value allows to remove some bad segments in noisy areas of the image (e.g. the hedge).

The parameters detailed above can be adjusted based on whether we want a cleaner output or more/longer lines in the output.

Figure 1.3: Straight lines detected with the Hough Transform

To conclude, we were able to find precisely `h` lines and some of the `l` lines. The worst result is for `m` lines, but increasing the contrast between dark and light areas, we could find any additional and useful edges.

1.2. F2: Image of C

The goal of this task was to find the image of the circumference `C` from the provided image. The task involved identifying the conic equation of the circumference based on a set of manually selected points and subsequently visualizing the computed conic on the original image.

To achieve this, the following steps were taken:

Point Selection: Instead of using an interactive tool like 'getpts', the points on the circumference were pre-defined for consistency and precision:

```
x = [417.9118, 437.6296, 456.3276, 585.1732, 721.4981, 724.8977]
y = [566.9941, 581.6125, 526.1986, 510.9003, 543.1967, 592.1513]
```

These points were selected manually based on the visible circumference in the image.

Fitting the Conic Equation: The general conic equation is expressed as:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

To determine the coefficients (a, b, c, d, e, f), a design matrix A was constructed from the selected points:

$$A = \begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

The coefficients were estimated by solving $A \cdot c = 0$ using Singular Value Decomposition (SVD). The last column of the V matrix from the SVD provided the solution.

Conic Matrix Representation: The coefficients were used to construct the conic matrix C:

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

The matrix was normalized by dividing all elements by $C(3, 3)$.

Figure 1.4: Detected conic C

1.3. F3: Image of S

The goal of this task was to identify and visualize the image of a unknown curve S in the provided image. This involved detecting edge points, robustly fitting an ellipse to these points using the **RANSAC** (Random Sample Consensus) algorithm, and plotting

the detected curve over the original image. To focus on the elliptical curve S, the edge points were restricted to a predefined region of interest (ROI). This eliminated irrelevant edges outside the region containing the curve.

RANSAC was used to robustly fit an ellipse to the detected edge points in the ROI. The algorithm iteratively selects random subsets of points, fits a candidate ellipse, and identifies inliers based on their proximity to the candidate. The best-fitting ellipse was refined by re-fitting to all inliers identified in the final RANSAC iteration.

The fitted ellipse parameters (center, axes, and orientation) were used to plot the curve S as a partial ellipse. The visualization highlighted the curve on the original image, restricted to a specific segment based on its geometric properties.

The RANSAC algorithm successfully identified the elliptical curve S, even in the presence of noise and irrelevant edge points. The fitted ellipse was visualized as a partial curve restricted to the ROI, with endpoints refined based on geometric criteria.

I thought also to another way to solve the task, using a similar approach to what we have been used to complete the previous task, using the function `spline()` after selected manually some points on the curve.

Figure 1.5: Detected unknown curve S

2 | Geometry

2.1. G1 - Estimation of vanishing line of the horizontal plane

In projective geometry, a vanishing point is the image of a set of parallel lines in 3D space after projecting onto the 2D image plane. Although truly parallel lines in the scene never meet in Euclidean space, their perspective projection often appears to converge at a single point in the image.

Due to measurement noise and inaccuracies, different pairs of parallel lines will not intersect at a single precise point. Thus, a common strategy is to seek the point that minimizes the sum of squared distances between the measured image lines and a single convergent point, leading to a maximum likelihood estimate under Gaussian noise assumptions.

Although the initial request was to automatically extract straight lines, I opted to manually select all the required lines to achieve a more accurate and precise outcome.

2.1.1. Vanishing Point of l_i Lines

In the following section there will be presented two different approaches to estimate the vanishing point of l_i lines.

Centroid-based Approach

The centroid-based approach is a simple and computationally efficient method for estimating the vanishing point in an image. It involves computing the intersections of all pairs of image lines and taking the centroid (mean position) of these intersection points as the estimated vanishing point.

Despite its simplicity, the centroid method lacks robustness to noise and outliers, as all intersections are treated equally, including those from erroneously detected lines. As such, it is best suited for cases where line measurements are precise and free from significant outliers. For higher accuracy, more advanced methods, such as optimization-based techniques, are preferred.

Levenberg-Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm is a popular technique for solving non-linear least squares problems. It combines the best features of two optimization methods: the Gauss-Newton method and gradient descent.

- **Gauss-Newton:** Efficient for problems that are well approximated by a quadratic near the solution, using the Jacobian and approximate second-order information.
- **Gradient Descent:** Robust for initial steps, but can be slow to converge.

LM smoothly transitions between these two approaches. When the solution is far from the optimum, LM behaves more like gradient descent, ensuring stability. As it gets closer to the optimum, it behaves more like Gauss-Newton, ensuring fast convergence. This makes LM a robust and efficient choice for a variety of parameter estimation problems, including camera calibration, structure-from-motion, and vanishing point estimation.

Comparison and results

In our scenario, the image data and detected lines exhibit relatively low levels of noise, as the parallel structures in the scene are well-defined and the line detections are accurate. As a result, the intersections of the lines cluster closely around the true vanishing point, minimizing the scatter typically caused by noise.

Given these favourable conditions, the centroid-based approach provides a computationally efficient and sufficiently accurate method for estimating the vanishing point. As demonstrated in the accompanying figure, the centroid of the pairwise line intersections aligns closely with the true vanishing point, delivering reliable results without requiring more complex optimization methods like the Levenberg-Marquardt algorithm.

While the LM algorithm offers improved robustness and precision in highly noisy or challenging datasets, in this case, its additional computational overhead is unnecessary. The simplicity and effectiveness of the centroid-based method make it the preferred choice for this specific scenario.

The vanishing points of l lines we found:

$$V_{l_{centroid}} = [3392.3,\ 666.4,\ 1]^T \qquad V_{l_{LM}} = [3416.8,\ 674.4,\ 1]^T$$

The computed values are very similar; therefore, for the subsequent steps, we will use the V_l value obtained through the centroid-based approach.

$$V_l = [3392.3,\ 666.4,\ 1]^T$$

2.1.2. Vanishing Point of m_j Lines

Similarly to the previous section and based on the result's comparison between LM approach and centroid-based one, I estimate the vanishing point of m_j lines using the centroid based approach:

$$V_m = [581.4,\ 883.8,\ 1]^T$$

2.1.3. Estimation of the line at infinity

The task was to construct and visualize the **line at infinity** l'_∞ in the image plane, given the coordinates of two vanishing points.

The line passing through the two vanishing points is computed using **cross product**

$$l_\infty = V_l \times V_m$$

where V_l and V_m are represented in homogeneous coordinates.

The result vector $l'_\infty = [a,\ b,\ c]^T$ represents the line at infinity in homogeneous coordinates, where:

$$ax + by + c = 0$$

The obtained result is:

$$l'_\infty = [-0.0001,\ -0.0011,\ 1]^T$$

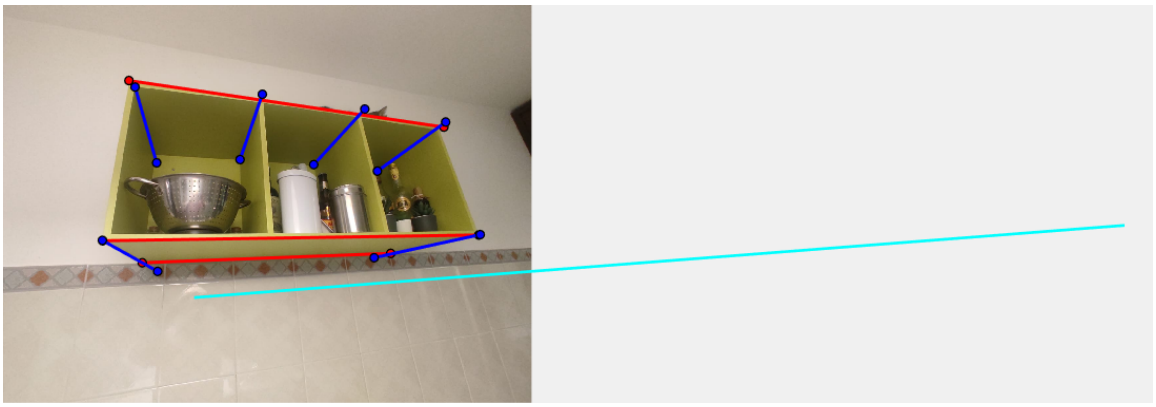


Figure 2.1: Line at Infinity

2.2. G2: 2D reconstruction of the horizontal plane

In the following section we are going to explore several ways to obtain a metric rectification of the original image.

In order to establish which are the possible and navigable ways to compute the metric rectification of our original image, we list the detected object in our image that could help us to achieve our objective:

- 3 independent families of orthogonal lines: l_i, m_j and h_k lines; we know they are orthogonal each other since they represents the three axis in the original `3D` scene
- the horizontal circumference C
- the unknown planar curve S

Based on the available objects, we can compute the desired metric rectification using two different methods:

- metric-stratified method
- metric rectification using one circle

We can't use the method involving 5 pairs of orthogonal lines, because we haven't the necessary pairs of independent orthogonal lines; we have only 3, as reported previously.

2.2.1. Metric stratified method

The metric stratified method consists of first performing an affine rectification and then finding $C^{*\prime}_{\infty}$ through two more constraints.

Affine rectification

By selecting two independent pairs of parallel lines in the original image scene, we can estimate their intersection using the cross product. This allows us to determine the image of the line at infinity that connects the resulting vanishing points.

The goal of affine rectification is to map the image of the line at infinity to its canonical position, thereby preserving and restoring the parallelism of lines. This is a direct result of aligning the line at infinity with its original position.

Since our objective is to compute a 2D reconstruction of the horizontal plane, the lines to be selected are l_i and m_j, for which we have already computed l'_{∞} in the previous section (2.1).

The computed affine-homography matrix is the following:

$$H_{aff} = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \\ -0.0001 & -0.0011 & 1.0000 \end{bmatrix}$$

The affine rectified image is shown below:

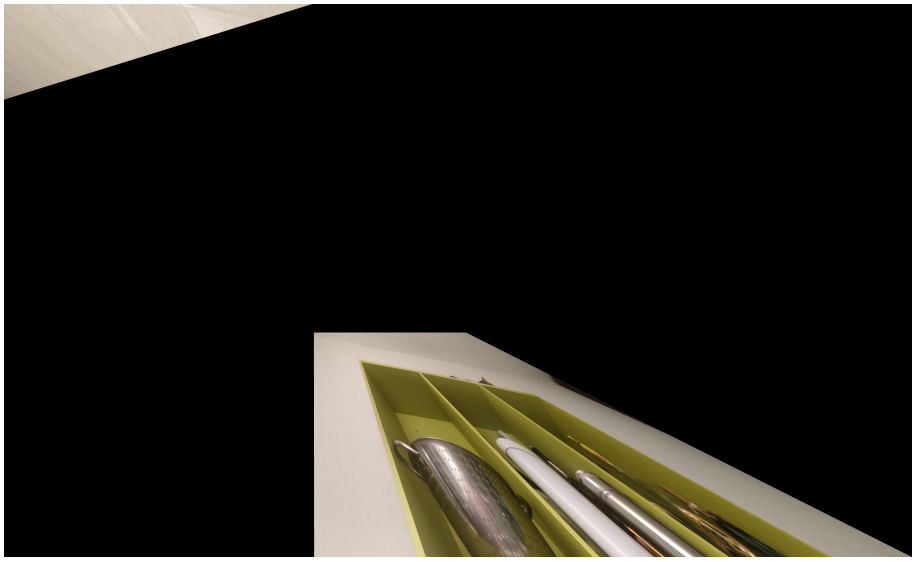


Figure 2.2: Affine rectified image

Metric rectification using imaged right angles

After obtaining the affine rectified image, two additional constraints are necessary to address the two degrees of freedom associated with the circular points. These constraints allow the computation of a metric rectification, which restores the metric properties of the world plane.

One approach to derive these constraints is by identifying two imaged right angles on the world plane. In the affine rectified image, let l' and m' represent an orthogonal line pair l and m on the world plane. The orthogonality condition can then be leveraged, where the cosine of the angle between two lines is given by:

$$\cos\theta = \frac{l'^T C_\infty^{*\prime} m'}{\sqrt{(l'^T C_\infty^{*\prime} l')(m'^T C_\infty^{*\prime} m')}} \tag{2.1}$$

Since l and m are orthogonal, $\cos\theta = 0$, which simplifies to:

$$l'^T C_\infty^{*\prime} m' = 0$$

By substituting the definition of $C_\infty^{*\prime}$, the constraint becomes:

$$\begin{pmatrix} l'_1 & l'_2 & l'_3 \end{pmatrix} \begin{bmatrix} KK^T & 0 \\ 0^T & 0 \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0$$

This equation establishes the necessary relationship to constrain the circular points and achieve metric rectification.

Despite selecting two independent pairs of orthogonal lines, as illustrated in the following image, we were unable to compute the rectified image due to likely numerical errors.

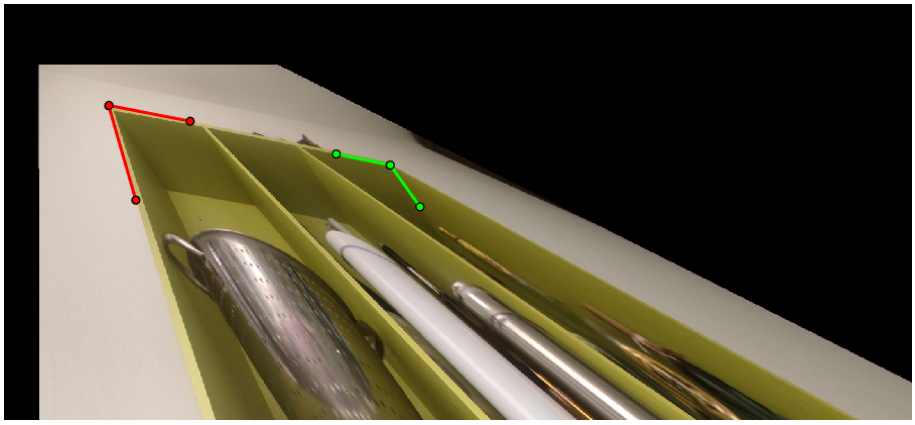


Figure 2.3: Two independent pairs of orthogonal lines

In conclusion, a different approach must be adopted to achieve the metric rectification of the provided image.

2.2.2. Metric rectification using one circle

Having detected the circumference C in section (1.2), we can use this information to compute the metric rectification.

The image of the circular points is determined by the intersection of the conic C and the line at infinity l'_∞. This intersection corresponds to the points on the conic that lie at infinity in the projective plane.

Conic equation A general conic in homogeneous coordinates $[x, y, 1]^T$ is represented as:

$$C(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$$

Where C if the conic matrix:

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} = \begin{bmatrix} -3.20 \times 10^{-7} & \frac{2.12\times10^{-7}}{2} & \frac{2.53\times10^{-4}}{2} \\ \frac{2.12\times10^{-7}}{2} & -3.06 \times 10^{-6} & \frac{3.32\times10^{-3}}{2} \\ \frac{2.53\times10^{-4}}{2} & \frac{3.32\times10^{-3}}{2} & -1.00 \end{bmatrix}$$

Line at infinity The line at infinity l'_∞ in homogeneous coordinates is expressed as:

$$l'_\infty(x, y) = l_1 x + l_2 y + l_3 = 0$$

System of equations The intersection of the conic and the line at infinity is obtained by solving the following system of equations:

- Conic equation:
$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$
- Line at infinity:
$$l_1 x + l_2 y + l_3 = 0$$

Circular points in homogeneous coordinates The resulting points in homogeneous coordinates are:
$$s_1 = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}, \quad s_2 = \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$
These points correspond to the **image of the circular points**.

The computed circular points are complex conjugates, as they lie on the real conic and at infinity. Their representation enables constructing the **dual conic of the circular points**:

$$C'_\infty = s_1 s_2^T + s_2 s_1^T$$

The final metric rectification matrix H_{met} obtained is:
$$H_{met} = \begin{bmatrix} 1.4243 & -6.9489 \times 10^{-1} & 0 \\ -6.9489 \times 10^{-1} & 2.1380 & 0 \\ -1.0000 \times 10^{-4} & -1.1000 \times 10^{-3} & 1 \end{bmatrix}$$

and the final metric rectified image is:

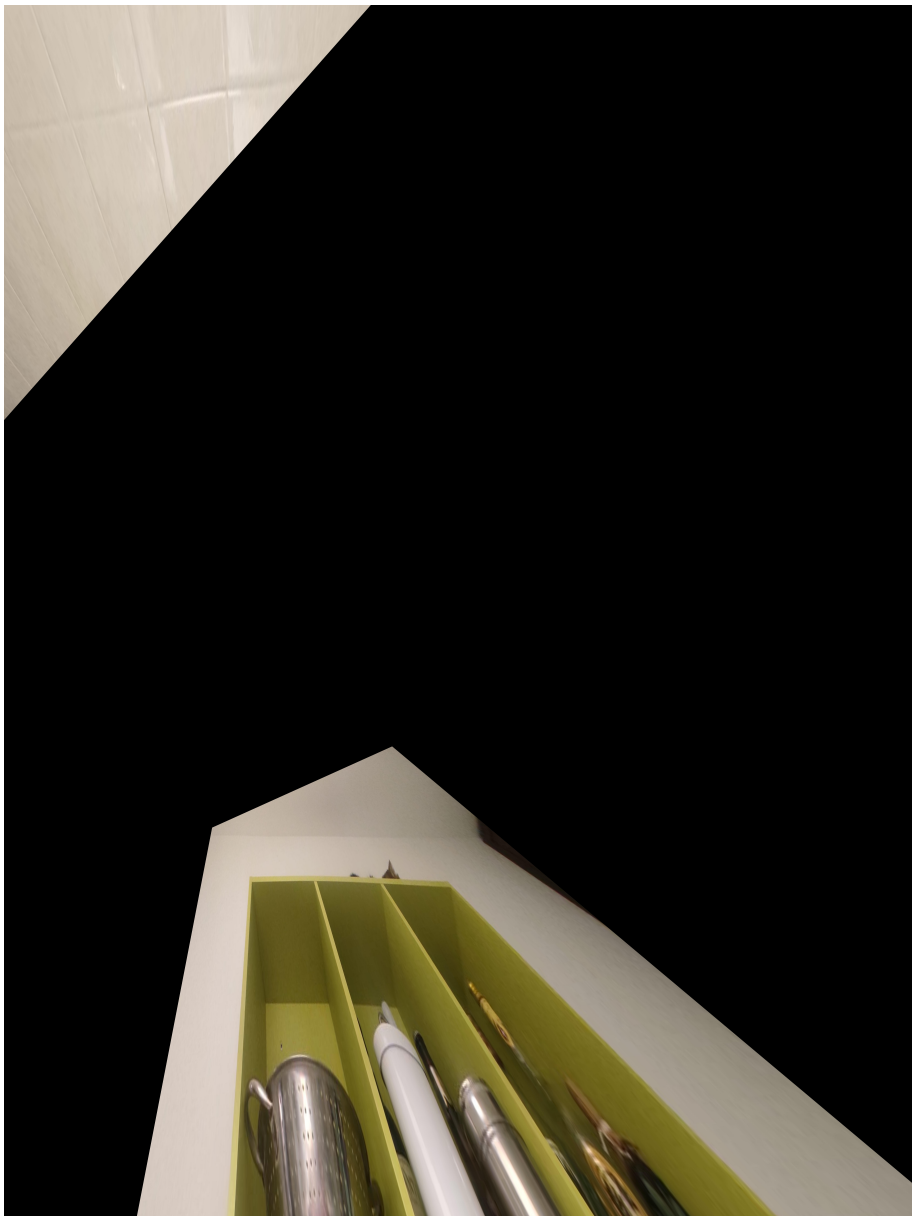


Figure 2.4: Metric rectified image using C

2.2.3. Estimation of m

To compute the depth m of the parallelepiped, we start by considering its known length $l = 1\,m$. Using the lines l and the hypotenuse of the triangle and the angle α between them (refer to the accompanying illustrations for clarity), we calculate $\cos\alpha$ using equation 2.1.

From the metric rectified image, we observe that l and m are orthogonal, forming a right triangle. According to the properties of a right triangle, the depth m can be expressed as:

$$m = l \sin\alpha$$

Given $\cos\alpha$, we compute $\sin\alpha$ using the trigonometry identity:

$$\sin\alpha = \sqrt{1 - \cos^2\alpha}$$

Substituting the calculated values, we find:

$$m \approx \frac{1}{3}\, m$$

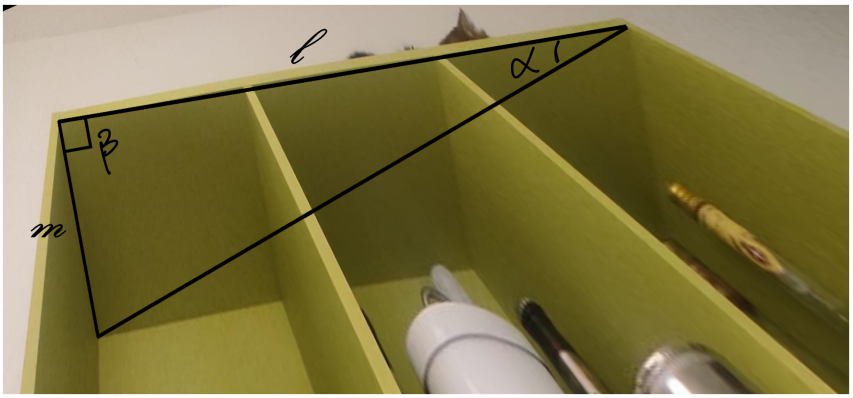


Figure 2.5: Triangle in metric rectified image

Double check of the result

To double check the obtained result, we start doing an hypothesis: the parallelepiped is divide into three parts of equals length.

Compute cross ratio To prove it, we need to compute the cross ratio (CR) of every divisor, between points A, B, C, D shown in the following image.

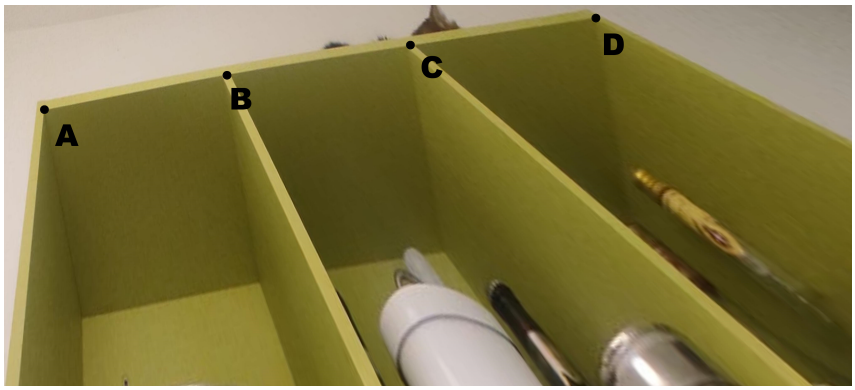


Figure 2.6: Selected points to compute CR

In real world, based on our hypothesis we know:

- $A = 0\,m$
- $B = \frac{1}{3}\,n$
- $C = \frac{2}{3}\,m$
- $D = 1\,m$

Remembering the cross ratio of a 4-tuple of collinear points formula:

$$CR_{Y,Z,X_1,X_2} = \frac{c-a}{c-b} / \frac{d-a}{d-b}$$

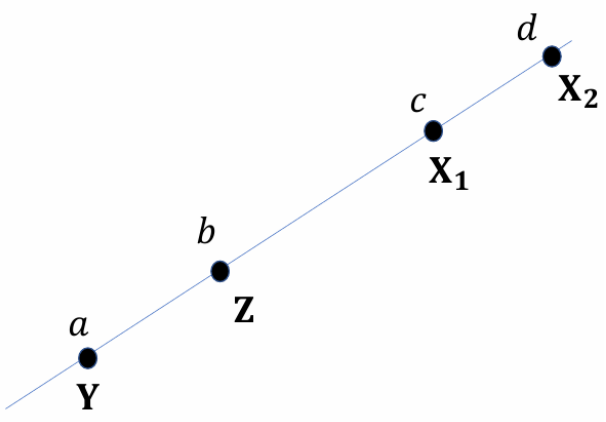


Figure 2.7: Cross ratio formula

The real-world aspect ratio is computed as:

$$CR_{real} = \frac{\frac{2}{3} - 0}{\frac{2}{3} - \frac{1}{3}} / \frac{1-0}{1-\frac{1}{3}} = \frac{4}{3}$$

By executing the code available in `G2\cr.m`, which involves manually selecting and obtaining the pixel coordinates of the points shown in Figure 2.6, the image aspect ratio was determined to be:

$$CR_{image} = 1.3484 \approx \frac{4}{3}$$

Since the aspect ratio (CR) is preserved from the real world to the image and the computed ratios are approximately equal, we can conclude that the parallelepiped is divided into three blocks of equal length.

Estimation of m From the following image, which is a zoom of the metric rectified image, we can see:

- lines l_x and l_y are parallel, as direct consequence of metric rectification, since parallel lines in the real world remain parallel in the metric rectified image
- lines m_1 and m_2 are parallel, for the same reason above
- lines m_1 and l_x are orthogonal, as direct consequence of the metric rectification computed
- lines m_1 and l_y are orthogonal, since m_1 is orthogonal to l_x and l_x is parallel to l_y

So, we can clearly see that all the internal angles are right angles and we can hypothesize that the figure is a square. To confirm our hypothesis, we just measure the angle between between the intersection of the two diagonals and check if is a square one, again using the formula 2.1.

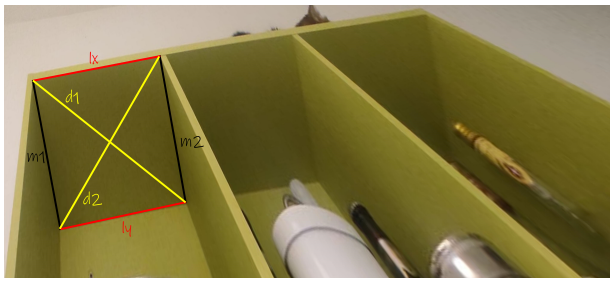


Figure 2.8: Square

The angle computed is $\approx 98°$, similar to a right one. We can conclude that the figure is a square and finally to conclude that the depth m of the parallelepiped is equal to the length of a single block, about $\frac{1}{3}$ meter, which is a similar value to the one computed previously.

The values are not precisely the same, but they are very similar and we can conclude the depth of the parallelepiped is $\frac{1}{3}$ of the length of it.

2.3. G3: Camera calibration

In order to calibrate the camera it's necessary to select at least four constraints, since the zero skew camera was already a constraint itself, to get the **image of the absolute conic** (IAC). The IAC is strongly related to the calibration camera matrix K through the relation $w = (KK^T)^{-1}$:

$$w = \begin{bmatrix} \alpha^2 & 0 & -u_0\alpha^2 \\ 0 & 1 & -v_0 \\ -u_0\alpha^2 & -v_0 & f_y^2 + \alpha^2 u_0^2 + v_0^2 \end{bmatrix} \tag{2.2}$$

2.3.1. Vanishing point of h_k lines

Applying the considerations in section 2.1.1 and selecting the h_k lines from the original image, we obtain the following vanishing point:

$$V_h = \begin{bmatrix} 0.6598 \times 10^3 \\ -1.4098 \times 10^3 \\ 0.0010 \times 10^3 \end{bmatrix} \tag{2.3}$$

2.3.2. System of equations

Two constraints could be easily recovered by exploiting the metric rectify transformation matrix found previously (2.2.2), combined with the image of circular points $h_i \pm i \cdot h_2$:

$$\begin{cases} h_1^T w h_2 = 0 \\ h_1^T w h_1 - h_2^T w h_2 = 0 \end{cases} \tag{2.4}$$

The other two constraints can be obtained from the vanishing point v of direction perpendicular to the vertical plane π:

$$\begin{cases} v^T w h_1 = 0 \\ v^T w h_2 = 0 \end{cases} \tag{2.5}$$

2.3.3. Calibration matrix

Obtaining the calibration matrix K through Cholesky Decomposition of $w = (KK^T)^{-1}$ as:

$$\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.4896 \times 10^3 & 0 & 0.0015 \times 10^3 \\ 0 & 2.1845 \times 10^3 & -0.0002 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Where f_x, f_y, u_0, v_0 are the intrinsic parameters of the camera.

2.4. G4: Estimation of the height of the parallelepiped

To find the estimate value of the height of the parallelepiped, we could use a similar approach to what we have done to find the estimation of the value of the depth of the parallelepiped, explained in section (2.2.3).

In order to follow that approach, we have to rectify the original image in order to have a frontal view of the scene.

2.4.1. Reconstruction of a frontal view

Through the knowledge of the calibration matrix K, section (2.6), we are able to compute the image of the absolute conic:

$$w = (KK^T)^{-1}$$

Then it was enough to compute again the image of the line at infinity, with respect to the vertical plane π we were going to rectify. and intersect with the Image of the Absolute Conic. The result led us to the Image of the two Circular Points as direct result of the equations and finally, through SVD decomposition, we bring them to their canonical position in purpose of obtaining the desired vertical plane rectified.

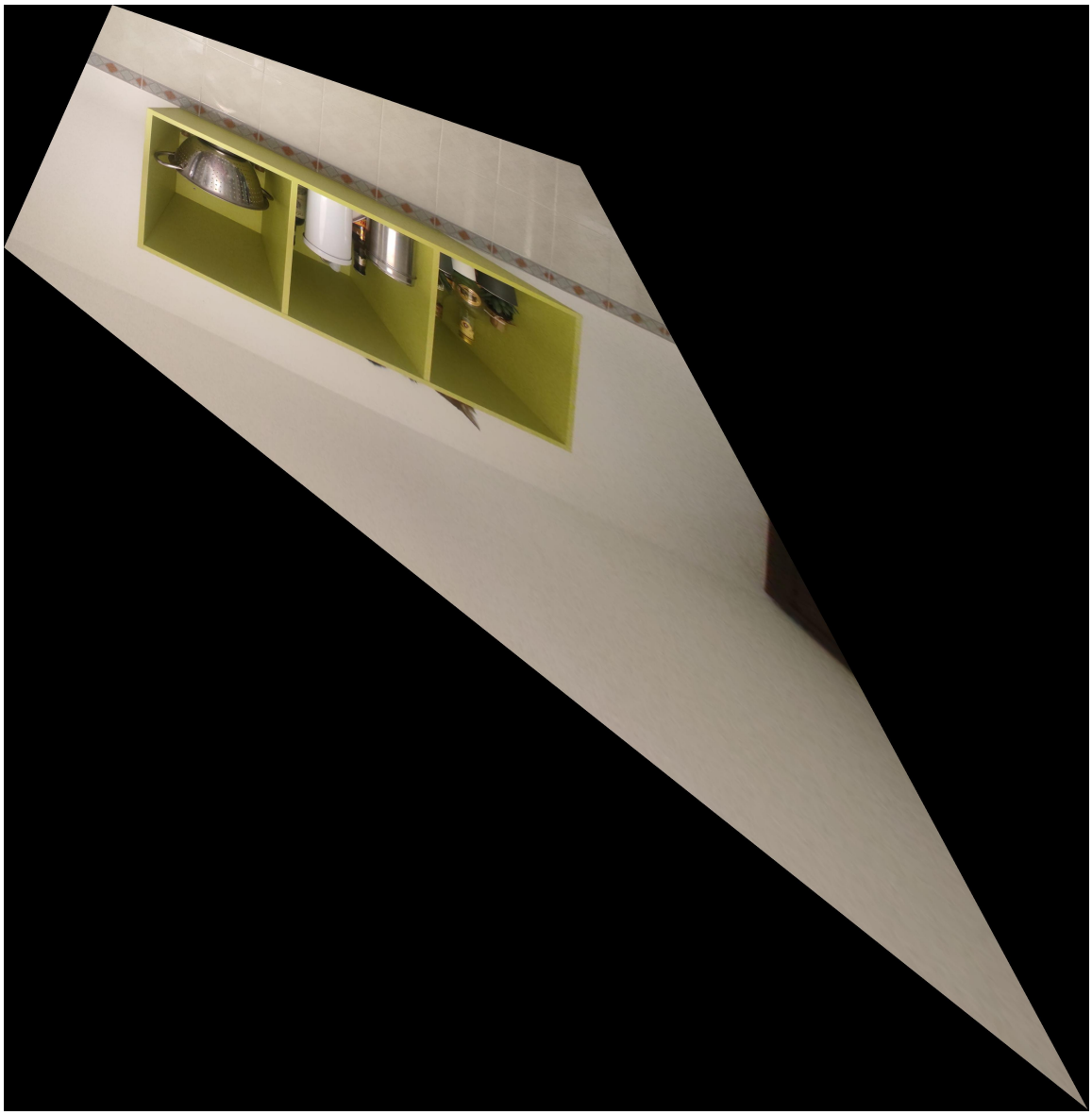


Figure 2.9: Reconstructed vertical plane using K

2.4.2. Estimation of height

Following the same approach used to determine the depth of the parallelepiped, we can measure $\cos\gamma$ using formula 2.1 shown in the figure. Knowing that l_x and h_y should form a right angle and given $l = 1\,\text{m}$, we can compute the numerical value of the height as:

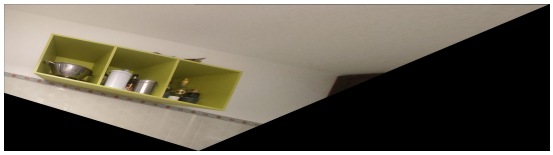
$$h = l\sqrt{1 - \cos^2\gamma}$$

Before proceeding with the numerical computation of h, it is crucial to verify, in the rectified image, whether the angle between l and h is indeed a right angle. Using a simple program to calculate the angle between two lines, we can verify that l and h are

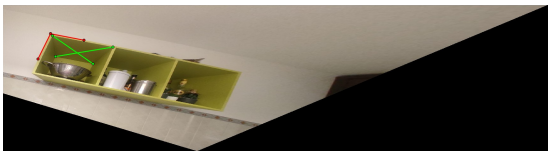
not perfectly perpendicular. This discrepancy arises due to numerical instability and inaccuracies in the matrix K.

2D Reconstruction of the Vertical Plane

In order to obtain a more precise reconstruction of the vertical plane, we follow the typical approach, using metric-stratified method, selecting l and m lines from the provided image in order to build the affine rectified image and finally selecting, from the obtained result, two pairs of orthogonal and independent lines as shown in the following pictures:



(a) Affine Rectification



(b) Pairs of orthogonal and independent lines

Note that the lines in Figure 2.10b can be selected because, as shown in (2.2.3), the green lines are the diagonals of a square and are therefore orthogonal.

The rectified image obtained is:

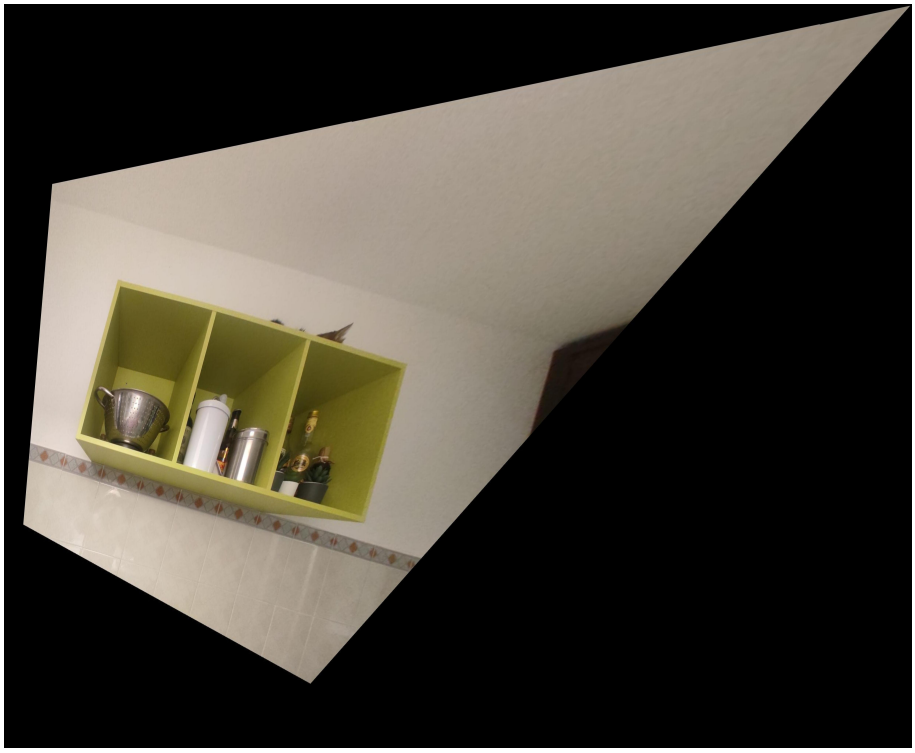


Figure 2.11: Vertical Rectification

Compute h

The obtained result is slightly improved compared to the previous one, as the computed angle between the l and h lines is very close to 90°. This allows us to calculate a more accurate numerical value for the height of the parallelepiped. The same method used to determine its depth, detailed in Section 2.2.3, is applied here. Using the Pythagorean theorem for a right triangle, the l and h lines are defined as the two catheti, we found:

$$h \approx 0.5\, m$$

2.5. G5: X-Y coordinates of a dozen points of S

This section focuses on the computation and visualization of $X-Y$ coordinates for selected points on the curve SS, leveraging a metric-rectified image from previous stages. The objective is to map curve points from their original image space to a rectified coordinate space, ensuring metric accuracy, and to visualize the adjusted curve points on the rectified image.

- **Rectified Image**: The metric-rectified image (2.4) obtained in previous steps was loaded as the base for the analysis.
- **Curve SS Points**: Data points defining the curve S were read from input file. These points are assumed to be in the original image's pixel space.

A set of known reference points in the original image and their corresponding points in the rectified image were defined. These pairs were used to compute the transformation matrix.

A projective transformation (`fitgeotrans`) was computed using the reference points from the original and rectified images. This transformation was applied to map the curve S points from the original to the rectified coordinate space. Using the computed transformation, the curve points were rectified.

The rectified image was displayed with the transformed curve S points overlaid in red:

Figure 2.12: Rectified S

To validate the rectification process and transformation accuracy, a selection of random points from the rectified curve S is displayed below:

X Coordinate (in pixels)	**Y Coordinate (in pixels)**
5.4979	9.5996
5.4950	9.6679
6.1065	9.5406
5.4942	9.6590
5.6008	9.4646
6.1350	9.5641
5.9547	9.4576
5.6890	9.4320
5.5107	9.7314
5.4937	9.6502
6.0864	9.5260
5.4936	9.6415

Table 2.1: Coordinates of random points from the rectified curve S, scaled by 10^3.

2.6. G6: Camera localization

The goal of this chapter is to localize a calibrated camera relative to a parallelepiped using a single image and known correspondences between 3D world points and their 2D projections. The final objective is to determine the camera's extrinsic parameters: the rotation matrix and translation vector, which define the camera's pose in the world coordinate system.

From the previously requests, we know:

- the length of the parallelepiped: $l = 1\,meter$
- the depth of the parallelepiped: $m \approx \frac{1}{3}\,meter$
- the height of the parallelepiped: $h \approx \frac{1}{2}\,meter$
- the camera calibration matrix K

Reference System I decide to consider the following reference system:

- X-axis runs along the edge of the length
- Y-axis runs along the edge of the depth
- Z-axis runs vertically along the edge of the height

Defining the 3D real world points and the respective one in the image, I aim to find the 3×4 projection matrix which execute the correct mapping.

This is achieved using the Direct Linear Transformation method. It emphasizes constructing the design matrix A, solving for P using Singular Value Decomposition (SVD), and interpreting P as the key to extracting the camera's extrinsic parameters.

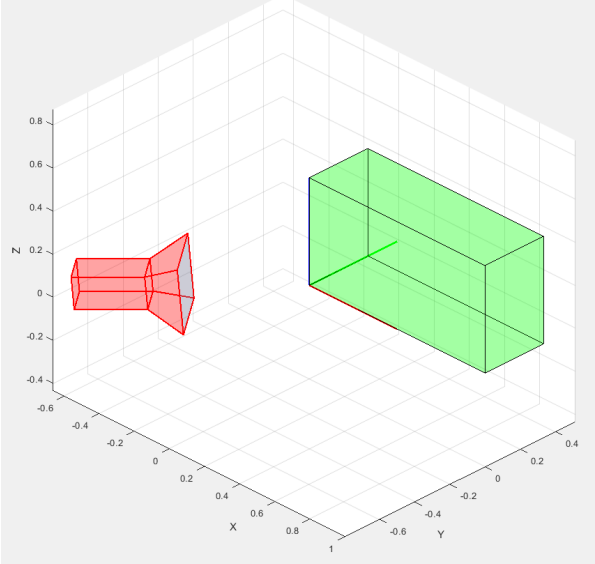
The **camera rotation matrix** found is:

$$R = \begin{bmatrix} -0.9451 & 0.4853 & -0.0503 \\ -0.1257 & -0.3673 & -0.9177 \\ -0.4135 & -0.7935 & 0.3940 \end{bmatrix}$$

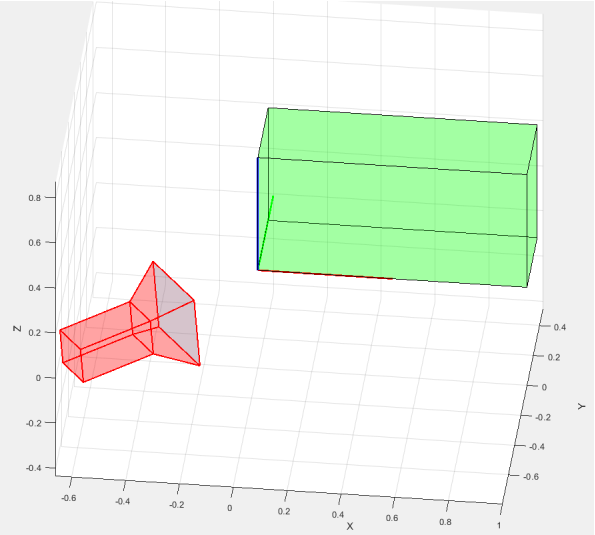
The **camera translation vector** is:

$$t = \begin{bmatrix} -0.1501 \\ -0.2444 \\ -0.7489 \end{bmatrix} \tag{2.7}$$

To enhance the understanding of the obtained result, we visualize the camera within a 3D model:



(a) Camera localization

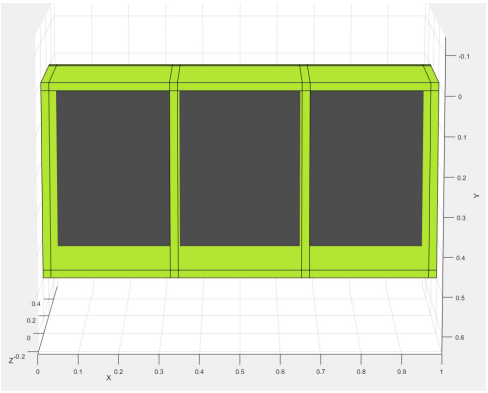


(b) Camera localization from a different view

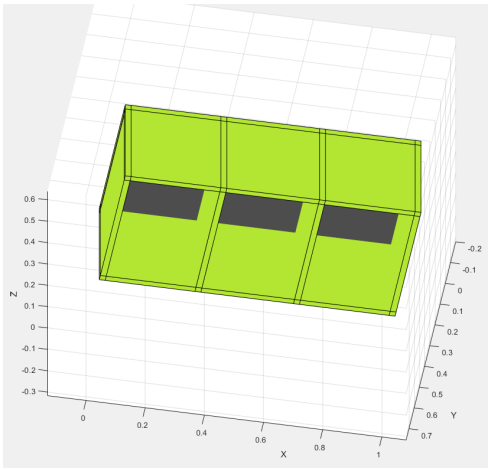
3 | 3D Model of the Parallelepiped

As shown in the images, to enhance the understanding of the scene, I chose to colour the back wall of the parallelepiped black and render it empty. This decision was made to improve the visibility of the depth.

The recovered `3D` model of the rectangular parallelepiped is displayed.



(a) 3D model - View 1



(b) 3D model - View 2

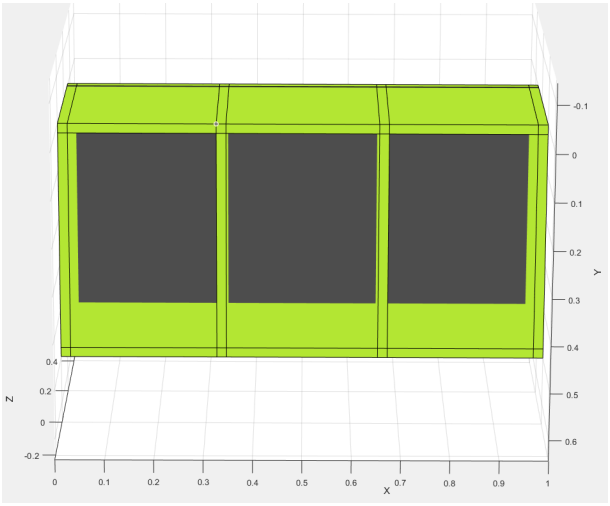


Figure 3.2: 3D model - View 3

List of Figures