IMAGE ANALYSIS AND COMPUTER VISION

HOMEWORK, February 2018

BAROZZI SARA – 874392

INTRODUCTION

The objective of the homework is to analyze a given image in order to extract the principal feature and to reconstruct its shape starting from some additional information using the software MATLAB.

The given image represents a bandoneon that is a musical instrument, consisting in two rigid wooden parts connected by a deformable bellow.

INFORMATION GIVEN:

- Image taken by a zero-skew camera (natural camera can NOT be assumed).
- The bandoneon is placed on a horizontal floor.
- Four rectangular faces are visible
- Long side of the horizontal faces is 243 mm, and it is slightly longer than the long side of the vertical face.
- Horizontal floor : groups of parallel lines (mutually orthogonal)
- We can NOT assume square patterns on the floor.
- Long lines in the vertical faces are vertical

QUESTION 1

The first step is to apply different filters to the image to adjust it and to modify the contrast in order to improve the result obtained in the following parts. The MATLAB functions used are:

```
Y= localcontrast(I, 0.4, 0.5);
Y = imadjust(I);
Y= medfilt2(I);
```

Lines extraction

The extraction of the edges with the 'edge' function is done to find the principal lines in the image.

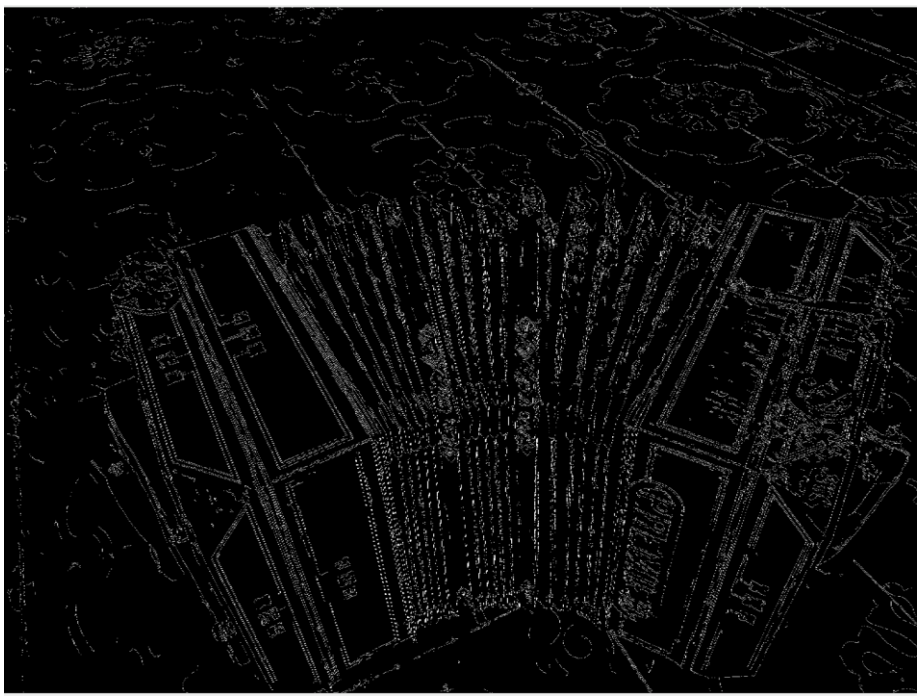
Edge finds edges in intensity image. It takes an intensity or a binary image I as its input, and returns a binary image BW of the same size as I, with 1's where the function finds edges in I and 0's elsewhere. Edge supports six different edge-finding methods. Among all of them it has been chosen 'canny' that is an operator that looks for local maxima of the gradient of I. It uses a multi-stage algorithm to detect a wide range of edges in images. These steps are:

- Apply Gaussian filter to smooth the image in order to remove the noise
- Find the intensity gradients of the image
- Apply non-maximum suppression to get rid of spurious response to edge detection
- Apply double threshold to determine potential edges
- Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Thresholds have been found by doing different experiments to find the best fitting edges.

```
[edges, th] = edge(imm,'canny');
 param = th.*[1.5, 2.5];
 edges = edge(imm,'canny', param);
```

The first line computes the edges and a general threshold 'th' that in the following step have been changed and used to detect the final edges.



Hough function

For the extraction of the lines of the image it is used the function Hough that returns a struct variable containing the points through which passes the line, theta and rho (line in parametric form):

```
rho = x *cos(theta) + y * sin(theta)
```

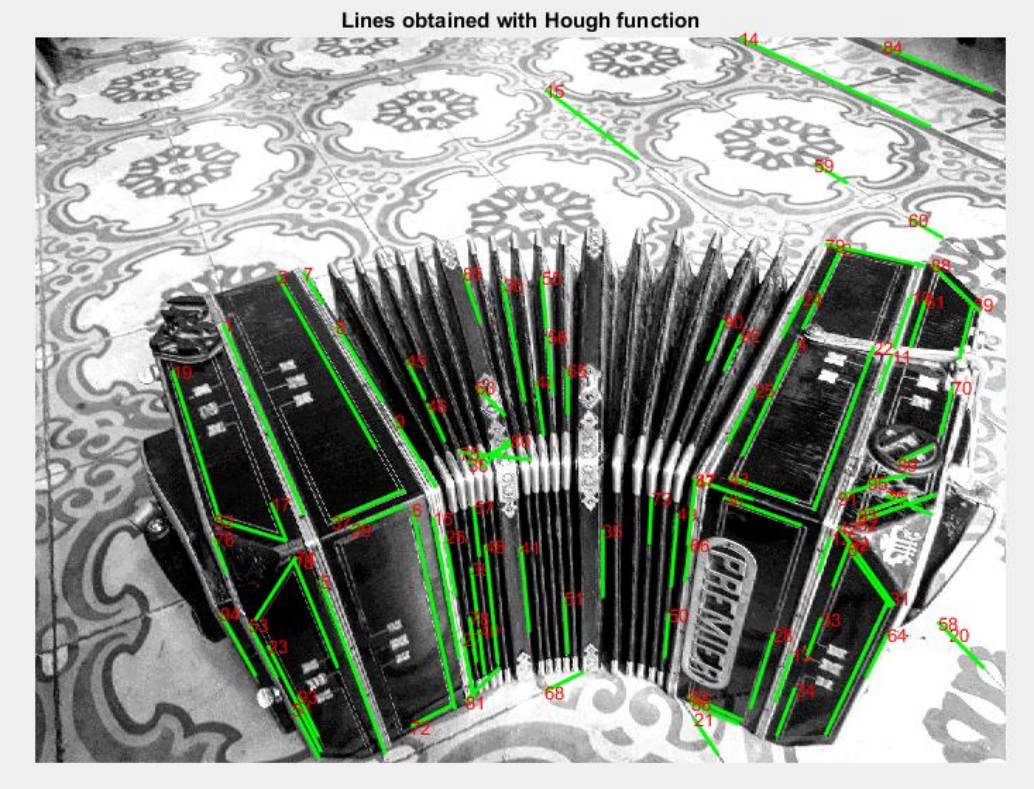
The variable rho is the distance from the origin to the line along a vector perpendicular to the line. Theta is the angle between the x-axis and this vector.

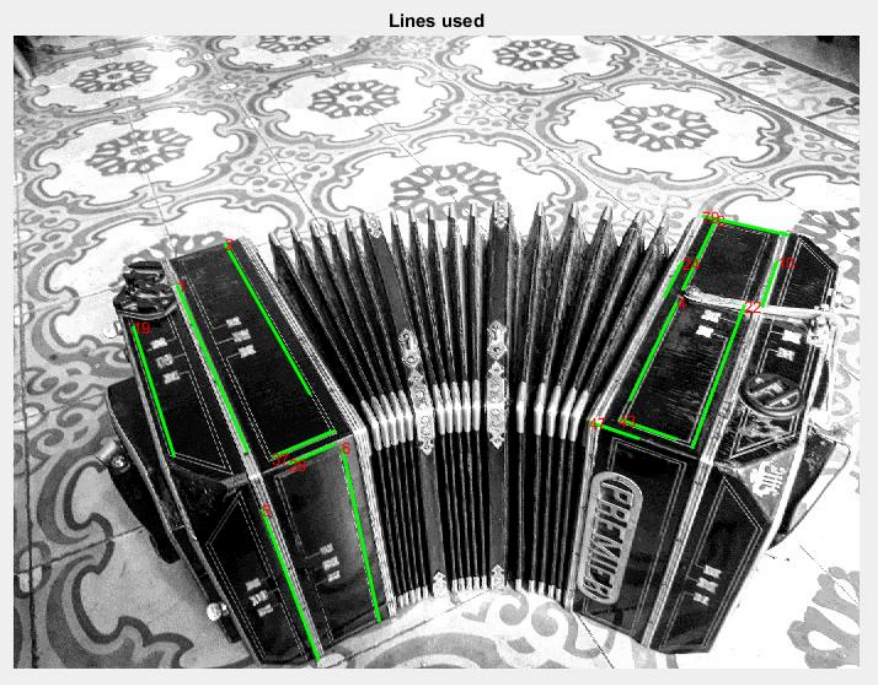
The **Hough** function generates a parameter space matrix whose rows and columns correspond to these rho and theta values, respectively.

The **houghpeaks** function finds peak values in this space, which represent potential lines in the input image.

The **houghlines** function finds the endpoints of the line segments corresponding to peaks in the Hough transform and it automatically fills in small gaps.

Also in this case the input parameters have been chosen though experiments in order to obtain useful lines for the subsequent steps. In the figures are shown the obtained lines and the selected ones.




Lines used

QUESTION 2

Shape reconstruction

The reconstruction of the horizontal face of the image passes through different steps. It is necessary to find the matrix that transforms the initial image to the result.

These transformations are an affine transformation and a Euclidean transformation, followed by a rotation of 180 degree around x axes to obtain the image with the right orientation.

The **Affine transformation is** a function between affine spaces which preserves points, straight lines and planes. Also, sets of parallel lines remain parallel after an affine transformation. It is a non singular transformation followed by a translation in the form:

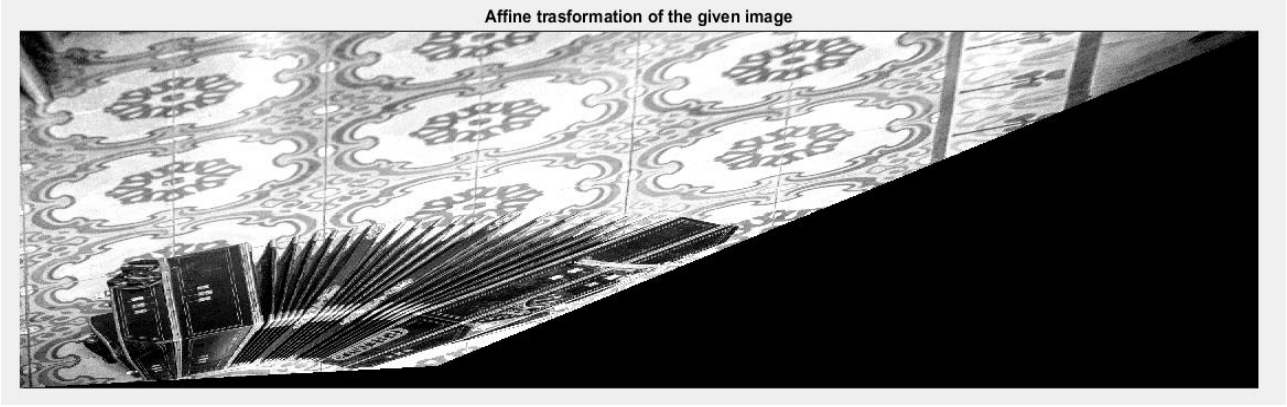
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The affine rectification it is based on the mapping of the line at infinite on to itself. It has been calculated the line at infinite by intersecting vanishing point of the parallel lines of the horizontal face. Once it is found, the transformation matrix called H_aff can be computed easily as:

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

The last row of the matrix contains the imaged line at infinite. In the image we can see that parallel lines are parallel although angles are not the real ones.

```
H_aff =

    1.0000         0         0
         0    1.0000         0
    0.0001    0.0008    1.0000
```



The **Euclidean transformation** that must be applied after the affine one, preserve length and angle measure. Moreover, the shape of a geometric object will not change. It is a necessary passage to obtain the rectification of the image. So it is an isometry, followed by an isotropy, which in this case is a scaling.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The transformation 'Ha' that we want to find can be calculated from the formulas:

$$C_\infty^{*\prime} = H_a C_\infty^* H_a^t$$
$$C_\infty^{*\prime} = \begin{bmatrix} a_{11}^2 & a_{12} * a_{21} & 0 \\ a_{12} * a_{21} & a_{22}^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$C_\infty^* = H_a^{-1} C_\infty^{*\prime} H_a^{-t}$$

C*inf' represent the image of the dual conic. From literature we know that C*inf' can be expressed as a matrix with the upper left side symmetric and homogenous; this means that it has 2 degree of freedom. This means that 2 constrain on the lines are needed, and they can be computed with the following formulas, that linear in case of orthogonality of the lines.

$$\cos(\theta) = \frac{l_1 m_1 + l_2 m_2}{\sqrt{(l_1^2 + l_2^2)(m_1^2 + l_2^2)}}$$
$$\cos(\theta) = \frac{l C_\infty^* m}{\sqrt{(l^t C_\infty^* l)(m^t C_\infty^* m)}}$$
$$\cos(\theta) = \frac{l' C_\infty^{*\prime} m'}{\sqrt{(l'^t C_\infty^{*\prime} l')(m'^t C_\infty^{*\prime} m')}}$$

.

We substitute in the parameter equations the coefficient of the perpendicular line that intersect on the horizontal face; It is possible to solve the system linearly and to find the value for that matrix C*inf'.

```
C_starprime =

   10.8035   -1.8891         0
   -1.8891    1.0000         0
         0         0         0
```

At this point, it is applied a Chowlesky decomposition. It is equivalent to a SVD function on MATLAB.

```
  svd    Singular value decomposition.
    [U,S,V] = svd(X) produces a diagonal matrix S, of the same
    dimension as X and with nonnegative diagonal elements in
    decreasing order, and unitary matrices U and V so that
    X = U*S*V'.
```

It is possible to obtain:

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

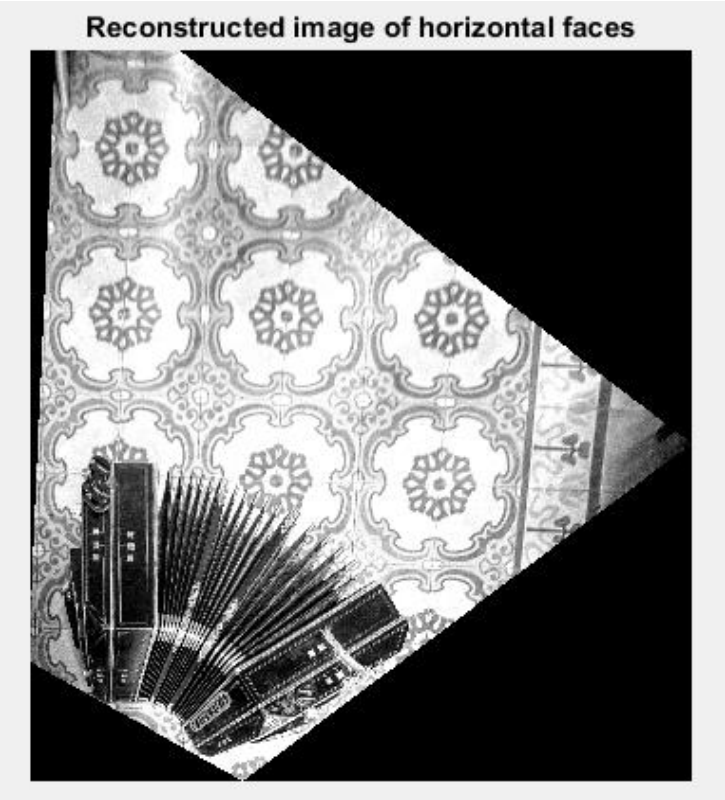
With this equation we know that Ha is equal to U, but having S not in normalized form a further step (a factorization) is needed to obtain Ha.
Knowing this last relationship it is possible to obtain the matrix of the Euclidean transformation called Ha: we compute the square root of the component of S and calculate it as U*S_factorized.

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

At this point it is sufficient to combine the three rotations and to apply the composition to the original image to obtain the rectified result. It is possible to see that lines are now parallel and perpendicular one with respect to the other.

```
                img -> affine -> Euclidean -> rotation
```

```
H_r =

   -0.2944    0.0548         0
   -0.2271   -1.2208         0
   -0.0001   -0.0008   -1.0000
```



Faces orientation

After the reconstruction it is possible to evaluate metric properties. In particular we can evauate the angles that are the invariants for this type of transformations.

The formula used to compute the angle between the two faces is this one.

```
cos_theta_left = (l_1.' * l_4)/(norm(l_1,2)*norm(l_4,2));(product of the lines
divided by the product of the norms of the lines.
```

The line chosen are intersecting lines, taken one from the horizontal face on the left and the other on the right. The angle it has been computed for two couple of lines to better estimate the value.

To compute the relative position of the faces it is necessary to fix the origin in one of the faces, in our case the right one, and calculate the distance from the other face's origin, using a scaling factor. The scaling factor value is obtained with a ratio between length of the long side of the frame and of the short one (calculated as a distance between the lines that define the edge of the bandoneon). It allows to obtain measures in the reference frame of the image.

A rotation matrix that allows passing form the right face to the left face has been computed:

```
relative_pose_from_right_to_left =
R_last*R_from_img_to_right*relative_coordinates;


theta =

   58.1088;

aspect_ratio =

    4.0913;

relative_pose_from_right_to_left =

  178.4689

  -62.8262

         0;
```

QUESTION 3

Camera calibration

Using also the images of vertical lines, calibrate the camera (i.e., determine the calibration matrix K) assuming it is zero-skew (but not assuming it is natural).

At this point we need to calibrate the camera. This operation is done by finding the parameters (intrinsic parameter of the camera) inside the calibration matrix.

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

The matrix is given by the known relations:

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

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

We have 4 degree of freedom, so four Constraints must be found to compute the matrix 'omega' (that is the image of the absolute conic) to obtain the matrix K. It is possible to find different constraints. Among all of them, we chose four, we compute the matrix omega by linearly solving the system of equations obtained, and finally, with some inverse formulas, we can obtain the inner values of the matrix K.

Two constraints are related with the fact that the line at infinity on the horizontal plane is orthogonal with respect to the vanishing point of the vertical direction on the vertical faces. To obtain these constraints it has been enough to find the line at infinity and put it in relation with the vanishing point of the vertical faces.

$$[l_{inf}]_\times \omega v_p = 0$$

The other two constraints are based on the image of the absolute conic and on the knowledge that the circular points lie on omega. 'H' is a transformation needed to map the corners with their imaged points. A scaling matrix is calculated by knowing the dimensions of the image, and then H is computed by scaling the previously found transformation matrix H_r.

```
H_scaling = diag([1/image_size, 1/image_size, 1]);
H=H_scaling/H_r;
```

From the theory we now that given the homography the imaged circular points are;

H = [h_1; h_2; h_3] → $h_1 \pm i h_2$

```
H =

   -0.0008   -0.0000         0

    0.0001   -0.0002         0

   -0.0003    0.0006   -1.0000
```

$$h_1^T \omega h_2 = 0$$
$$h_1^T \omega h_1 = h_2^T \omega h_2$$

We can create a system with all the constraints found, solve it linearly, find the coefficient in matrix omega, and compute the coefficient for K.

```
The values of the parameter in the K matrix are:

fx= 3.124906e+03;

fy= 3.276910e+03;

u0= 2.054640e+03;

v0= 1.812943e+03;

alfa= 9.536137e-01;

K =

   1.0e+03 *

    3.1249         0    2.0546

         0    3.2769    1.8129

         0         0    0.0010;
```

QUESTION 4

Camera localization

We have to compute the location of the camera with respect to both the faces.

We know the rotation and the angles between the faces, the measure of the horizontal faces (data given), and the calibration matrix.

We fix a reference system on the right face and we compute the coordinate of the vertices of the face(that will be (0,0) for the origin, while for the other side we use the length provided and the aspect ratio between the faces calculated before).

```
x_dl = [0      0]

x_ul = [0    243]

x_dr = [59.3946          0]

x_ur = [59.3946   243.0000]
```

We fix an Homography (with a MATLAB function) that fix the points given in the reference frame of the face with the coordinate of the same points in the image reference frame. We are mapping world points to image point.

```
Corner point of the right horizontal face in the image reference frame:


y_dl =              y_ul =              y_ur =              y_dr =
   1.0e+03 *           1.0e+03 *           1.0e+03 *           1.0e+03 *
    2.8596              3.3762              3.7709              3.3980
    1.9076              0.8862              0.9739              2.0878
    0.0010              0.0010              0.0010              0.0010

H_omog =

   1.0e+03 *

    0.0062    0.0078    2.8596

    0.0013   -0.0027    1.9076

   -0.0000    0.0000    0.0010;
```

At this point with the formulas given by the theory it is possible to compute the rotation of the camera and the position, with respect to the face taken as reference.

$$\lambda = \frac{1}{|K^{-1}h_1|}$$
$$i_\pi = K^{-1}h_1\lambda$$
$$j_\pi = K^{-1}h_2\lambda$$
$$k_\pi = i_\pi \times j_\pi$$
$$o_\pi = K^{-1}h_3\lambda$$

In this formulas i, j, k represent the three rotation of the camera, while ‘o’ is the translation.

```
Lambda = 356.2358;
```

The final rotation `R = [r1, r2, r3]` is:

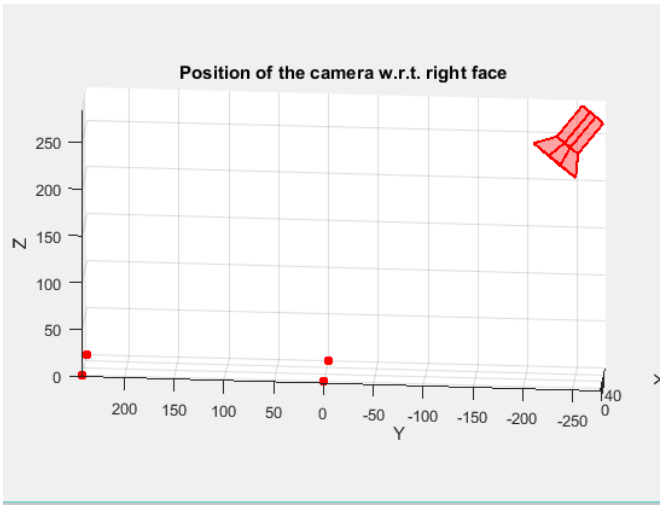
```
R =
    0.9029    0.4978   -0.0067

    0.3044   -0.6277   -0.6963

   -0.3035    0.6039   -0.7183
```

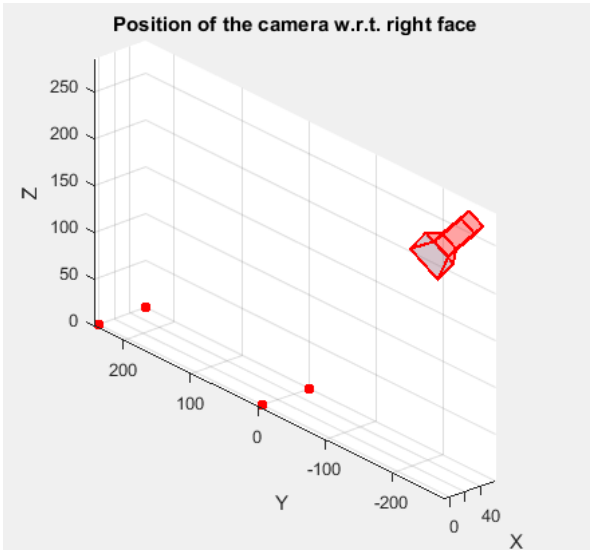
In the code a step is added to find the nearest orthogonal matrix to be sure to have a correct rotation. This is done with the SVD function provided by MATLAB.
To obtain the location of the camera w.r.t. the other face it is sufficient to apply the rotations previously computed to rotate it from the right face to the left.

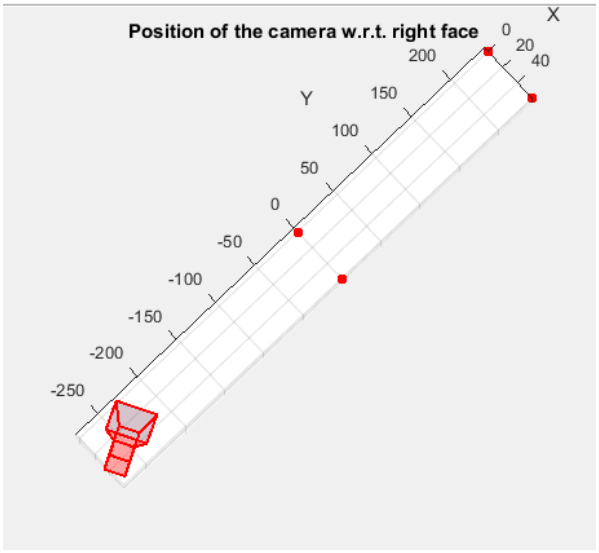
```
cameraRotation =

    0.8862    0.3286   -0.3268

    0.4633   -0.6385    0.6146

   -0.0067   -0.6960   -0.7180;

T =
   91.7597

   10.2888

  356.2358     //translation in the camera reference frame

cameraPosition =

   31.7107

 -254.8794

  263.5537;    //position of the camera in the plane reference frame
```

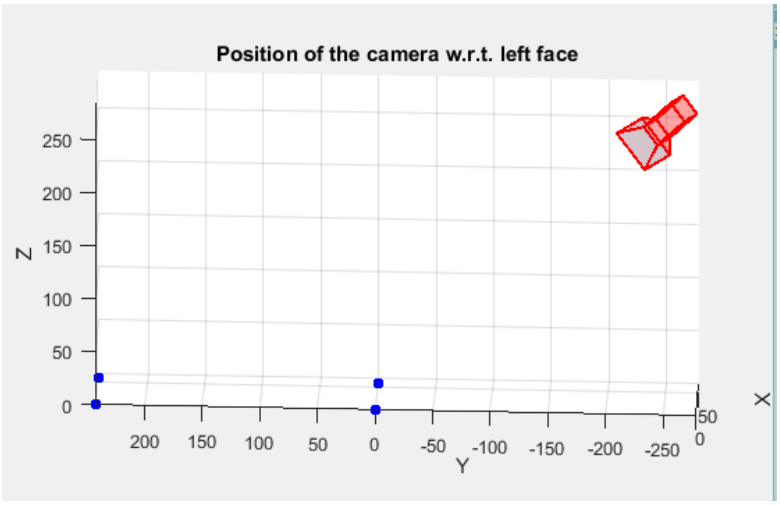
Once the rotation with respect to the left side has been computed, we can calculate also the position w.r.t. the left side.

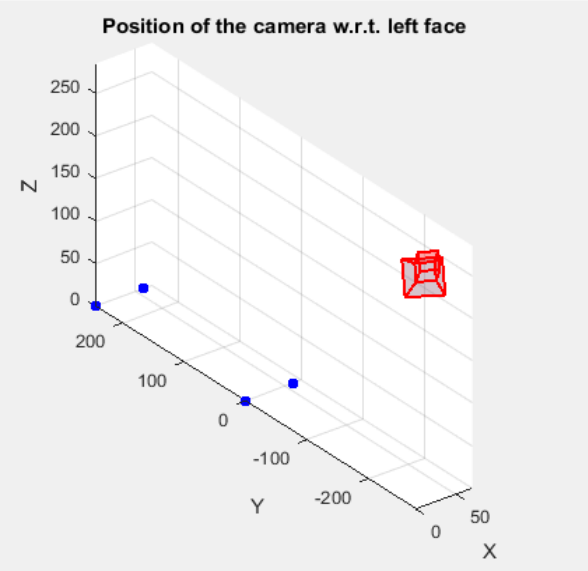
```
R_camera =

     0.8616   -0.5076   -0.0067

    -0.3685   -0.6163   -0.6960

    0.3492    0.6021   -0.7180;
```

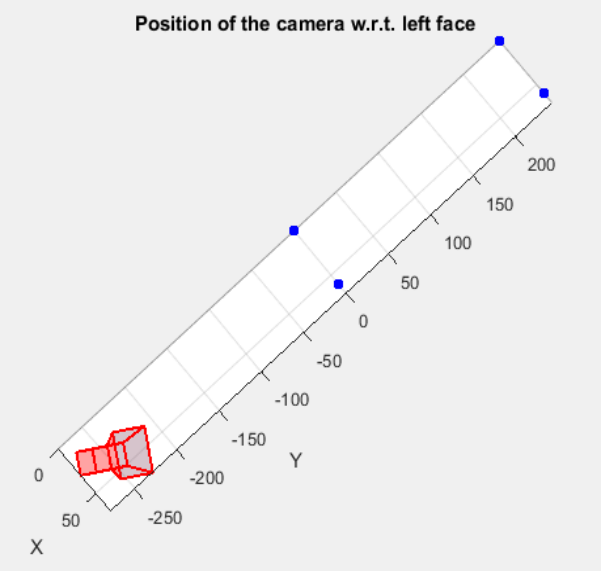

Position of the camera w.r.t. right face
Z
Y


Position of the camera w.r.t. right face
Z
Y
X

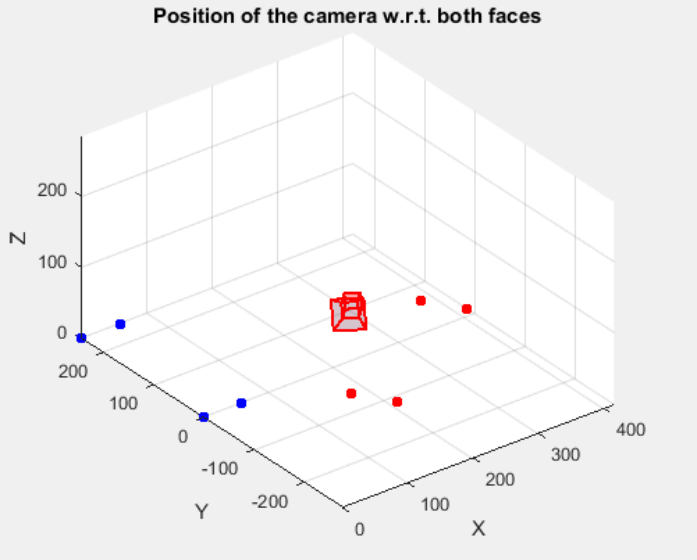

Position of the camera w.r.t. right face
X
Y


Position of the camera w.r.t. left face
Z
Y
X


Position of the camera w.r.t. left face
Z
250
200
150
100
50
0
200
100
0
-100
-200
Y
0
50
X


Position of the camera w.r.t. left face
200
150
100
50
0
-50
-100
-150
-200
-250
Y
0
50
X


Position of the camera w.r.t. both faces
200
150
100
50
0
-50
-100
-150
-200
-250
Y
200
0
Z
0
100
200
300
400
X


Position of the camera w.r.t. both faces
Z
200
100
0
200
100
0
-100
-200
Y
0
100
200
300
400
X

Vertical face reconstruction

We use the result of the localization to compute the vertical face reconstruction. In particular we calculate the projection matrix as the product of K (calibration matrix) an R (rotation matrix of the camera) composed with the translation.

```
P = K * [R,T];
```

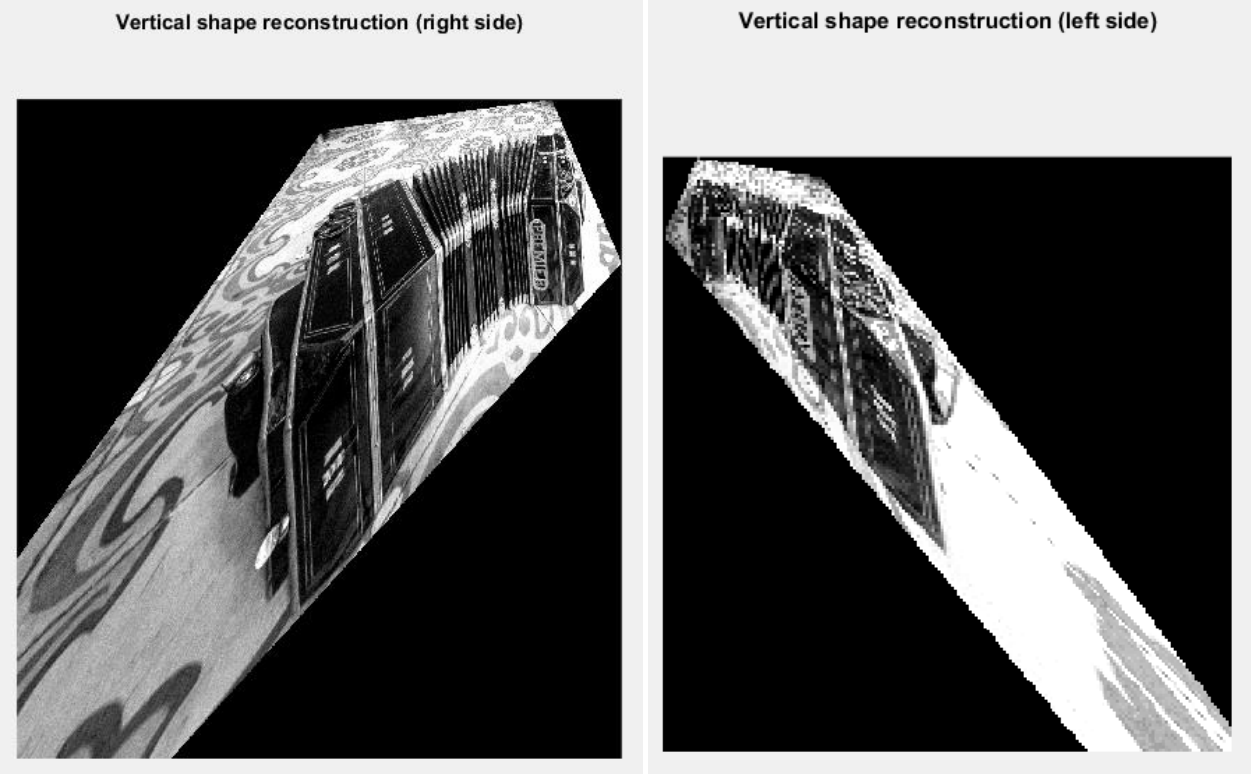
Knowing that points of interest in this case are positioned on the vertical face, their coordinates will always be in the form: $X_0 = [x\ 0\ z\ w]'$.

P maps the point on the vertical face to image point, so we can extract the line of interest from it and directly obtain the transformation matrix needed.

```
H_vert_r_sr =

   -0.0014   -0.0021   13.2610

   -0.0018   -0.0019   14.8696

   -0.0000   -0.0000    0.0552
```

The same operation are done for both faces, utilizing as rotation matrix of the camera the basic one (cameraRotation), in case we are plotting the right reference face, or the rotated one (R_camera) in case we have to use the rotate version.

The results of the reconstruction are shown in the image below.



Probably due to some imprecision, the reconstruction of the left face is less accurate than the reconstruction of the right face.

REFERENCES

"Multiple View Geometry in Computer Vision". second edition. Richard Hartley.

Notes taken in class;

Slides of the course;

Wikipedia and internet pages.