Dpto. Ingeniería de Sistemas y Automática
Universidad de Málaga

COMPUTER VISION

EXERCISE 9a: Camera calibration

Concepts: Camera calibration, intrinsic parameters, distortion.

Camera calibration. Since we are almost experts in computer vision, we are going to calibrate the camera of our smartphone or laptop. For that we are going to use the amazing tutorial and calibration application from MATLAB. The tutorial is full of interesting information and directions about how to properly calibrate the camera, as well as how to interpret the obtained results. To calibrate the camera, you only need the provided application, a pattern with a chessboard, and the camera itself! Take different pictures of the chessboard with different perspectives, run the application, and enjoy! **For that, follow the next steps:**

1. Take a look at the tutorial of the *Camera Calibrator application*. With such an application you can calibrate standard or fisheye cameras. We will focus on standard cameras. Tutorial url: https://es.mathworks.com/help/vision/ug/single-camera-calibrator-app.html

2. Following the hints given in the tutorial, take around 15 images of the calibration pattern with your camera, and add them to the application (*Add Images* button). If the camera resolution is too high, resize the images. Include thumbs of the used images in the report.

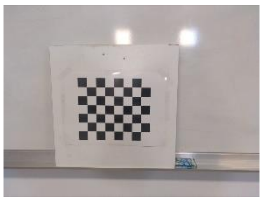


Image 1

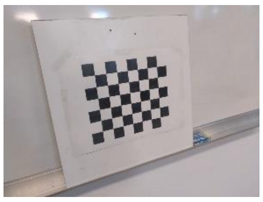


Image 2

Image 3

3. Calibrate the camera (*Calibrate* button). The output should be similar to the one in the image below. Include a similar screenshot with your output, and explain the meaning of the two subfigures on the right side (*Projection errors* and *Extrinsics*).

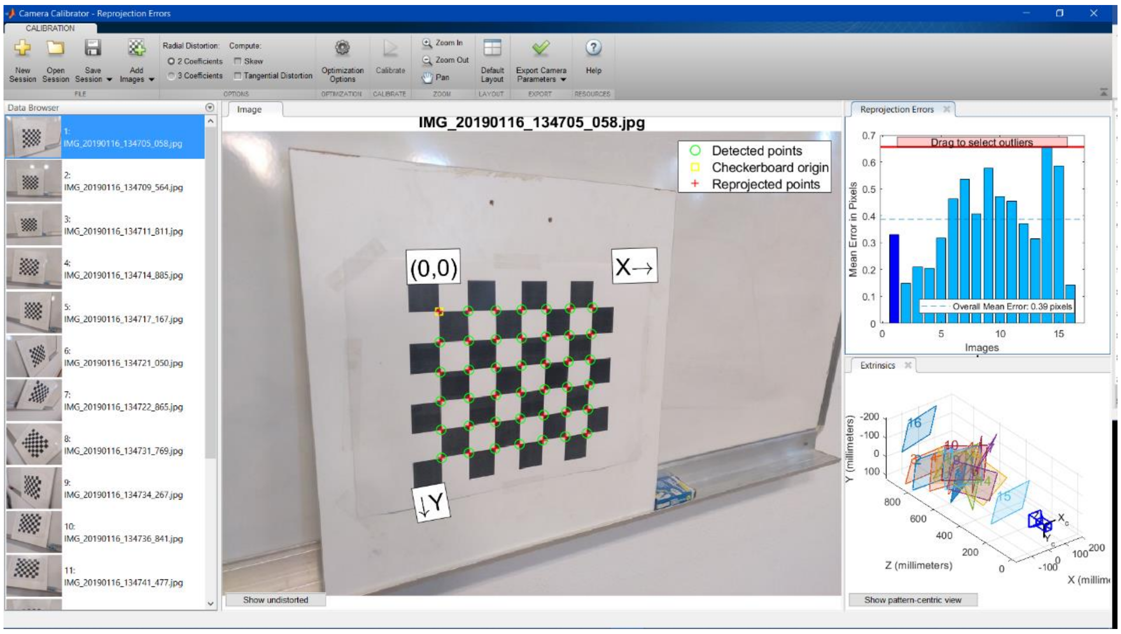


Image 4. Calibration of the camera

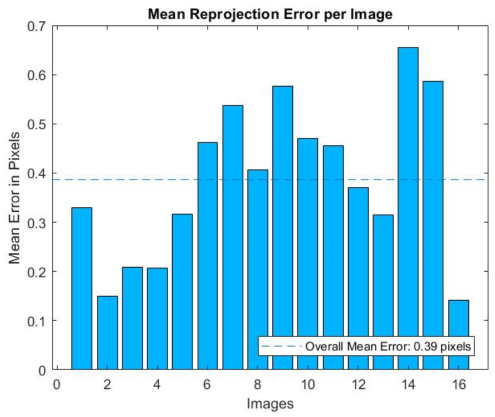


Image 5. Mean error of each image

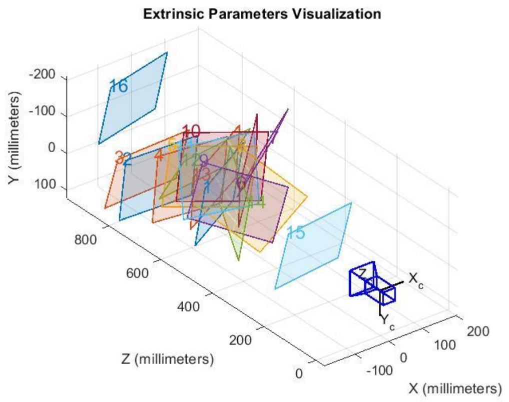


Image 6. Projection of the images

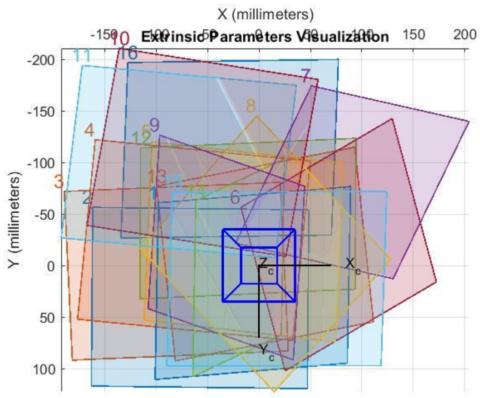


Image 7. Camera centralized

The meanings of each subplot on the right side (Image 5 and Image 6) are:

- The first is the mean of the projection error of each images. This error is produced by the approximation on the calibration and it is the difference between the location of the observed point and the projected one.
- The second is the localization of the camera and its observed planes which helped to calibrate the camera. It is interesting that in Image 7 the camera is in (0,0). This means the calibration has been successful and well made.

4. There are two ways to export the resultant camera parameters:

   i. Export parameters to Workspace: this option will create a structure in your workspace called *cameraParams* with a number of fields, for example: *intrinsicMatrix*, *FocalLength*, *PrincipalPoint*, *RadialDistortion*, etc.
   ii. Generate Matlab script: this way automatically generate a script that performs all the previous steps: loads the images, calibrates the camera, and displays the results, also creating a *cameraParams* struct.

   Use either option to check the camera parameters. Comment in your report as many parameters as you can identify from the calibration lecture (cx, cy, f, etc.) along with their units (meters, pixels, etc.). Also include the content of the computed *intrinscMatrix.*

   In the calibration lectures, it can be identified in the first part the intrinsic values of the camera, which includes:

   - The focal length, which corresponds to f, and his unit, which it is in pixels.

- The principal points, which corresponds to cx and cy respectively, and his unit, which it is in pixels.

Also, it shows the extrinsic values of the camera, which are the rotation vector of ach images, which is in radians, and the translation vector of each images, which is in millimeter.

$$IntrinsicMatrix = \begin{pmatrix} 750.4334 & 0 & 0 \\ 0 & 750.8098 & 0 \\ 470.6846 & 363.6382 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ x_0' & y_0' & 1 \end{pmatrix}$$

Thus, *IntrisicMatrix* is K^T in the calibration.

EXERCISE 9b: Fundamental matrix

Concepts: Harris operator, fundamental matrix, epipolar lines, stereo.

Once the camera is calibrated, take a pair of pictures from surfaces with visual features, and measure the distance that the camera moved between both shots. It will be our baseline **b**. The idea is to emulate an ideal stereo configuration with two identical and parallel cameras. *Note: 10cm or so is ok.*

Example of three images with a planar surface (the cereal box), where the camera moved 5cm to the right from the image on the left to the one in the middle, and 10 cm to the one on the right.

The following steps will provide a nice 3D reconstruction of the scene in the images. *Note: if you were unable to calibrate your camera use the pepsi_left.tif and pepsi_right.tif images.*

1. **Extracting features**. The first step is to detect the features that we are going to match in order to obtain their 3D coordinates. For that, we rely on the Harris keypoint detector used in the third exercise. Employ it to extract keypoints from both pictures, and show them graphically.

2. **Feature matching**. Once extracted, we need to find the correspondence of each keypoint on the left image in the right one. For that, you can also reuse the code from the previous exercise employing Normalized Cross Correlation (NCC). It is a good idea to add a threshold to remove/ignore matches with a low cross correlation value (e.g. 0.9 or higher). Show graphically the matched features. Note that we are not using any type of constraint for this matching, so this can be improved. Which ones could be used?

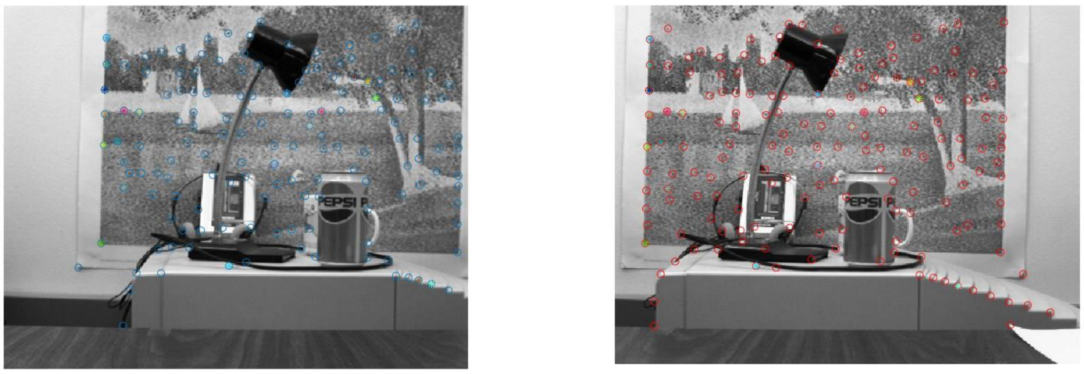


Image 8. NCC matches with high value for uncalibrated camera

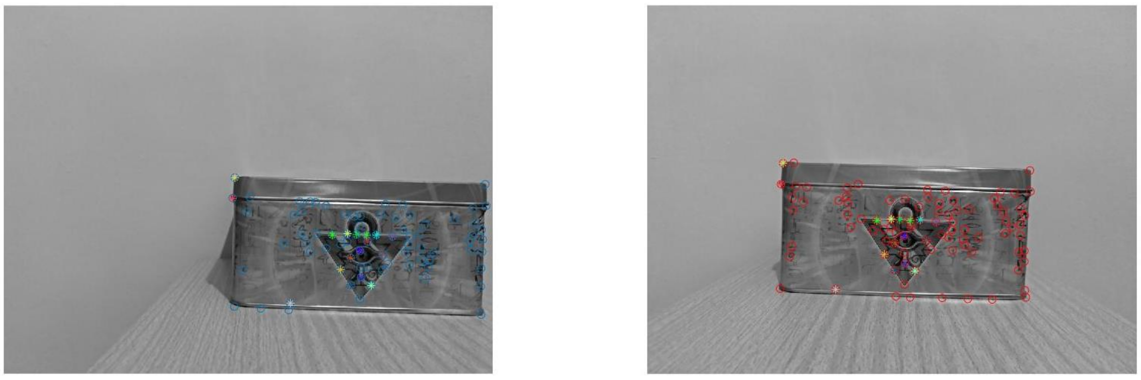


Image 9. NCC matches with high value for calibrated camera with 5cm difference

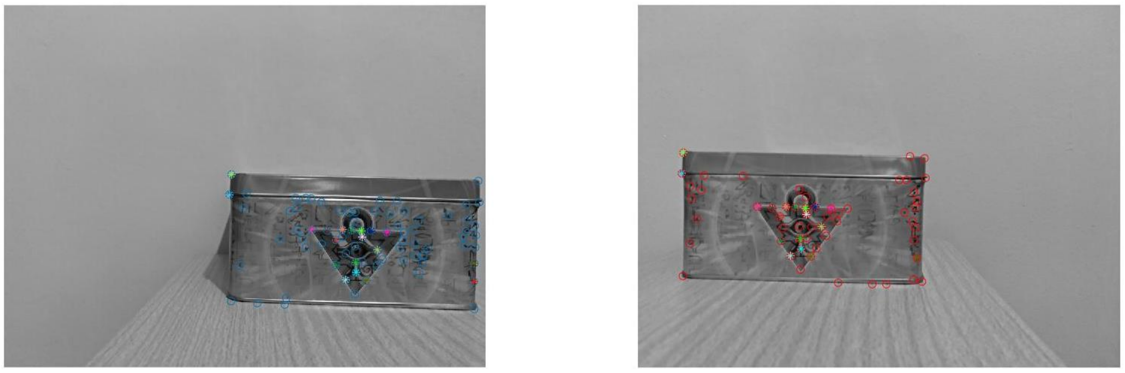


Image 10. NCC matches with high value for uncalibrated camera with 10cm difference

Because we do not do a constraint, some points match with the same one in the other. We could improve this if we use any technique of constrains, such as a dynamic programming approach where we build a graph with all the possible matches and we try to find a minimum cost path from upper left corner to the bottom right one.

3. **Fundamental matrix.** This is not needed for the tridimensional reconstruction, but we are going to compute it using the provided function **ransacfitfundmatrix** in order to further reduce the set of matched features, as well as for drawing epipolar lines. Such a code use RANSAC for computing the fundamental matrix, also returning a vector with the keypoints that agree with the fitted model, removing outliers (mismatched keypoints). Report the obtained fundamental matrix and show the inliers (matches that fulfill the model).

```
fprintf('Fundamental matrix:\n');
disp(F);
fprintf('Inliners: \nLeft image:\n');
disp(h1(1:2,inliers));
fprintf('Right image:\n');
disp(h2(1:2,inliers));
```

4. **Epipolar lines**. Next, using **ginput**, set any pixel on the left image and draw on the right one the corresponding epipolar line. For that, prepare a figure with two subplots. Include in the deliverable some examples. ¿Where are the epipoles in each image?. *Note: use the following code to compute two points of the epipolar line on the right image, and then employ the command line() and such points to draw it.*

```
ln = F'*[x;y;1]; % x,y are the coordinates of the clicked point on the left image
x0 = [0 size(imR,2)];
y0 = [-ln(3)/ln(2) -ln(1)/ln(2)*x0(2)-ln(3)/ln(2)];
subplot(1,2,2);
line(x0,y0);
```

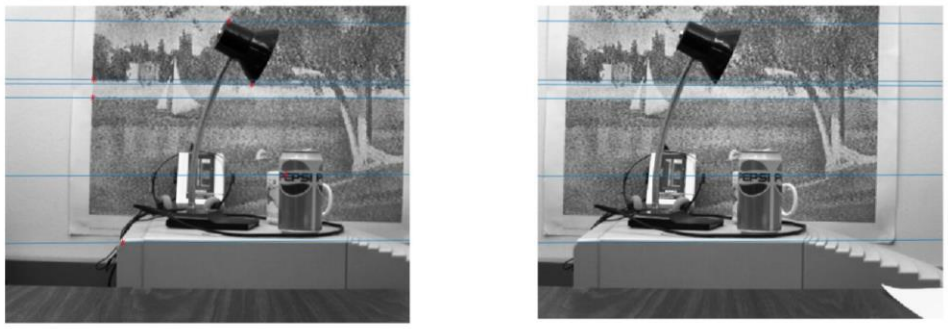


Image 11. Epipolars for an uncalibrated camera

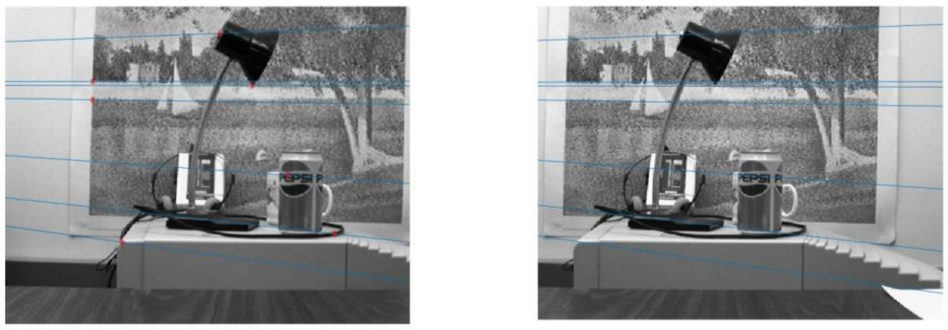


Image 12. Epipolars for an uncalibrated camera

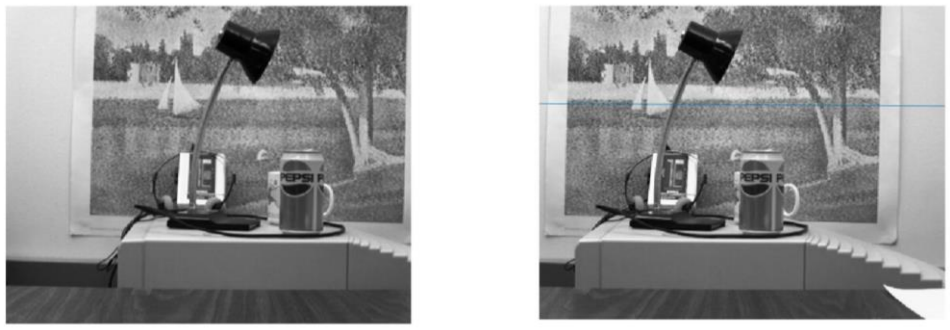


Image 13. Normal function on drawing the epipolar on an uncalibrated camera

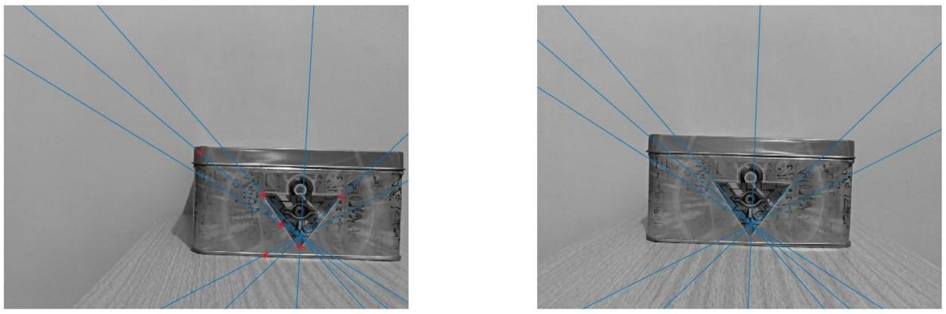


Image 14. Epipolars for a calibrated camera with 5cm of distance

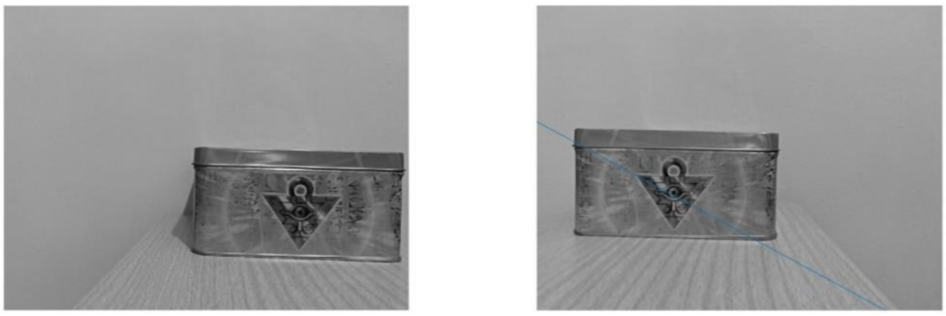


Image 15. Normal function on drawing the epipolar on a calibrated camera with 5cm of distance

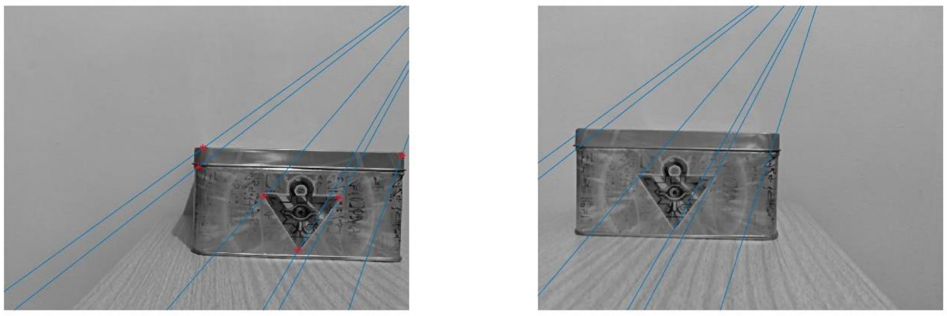


Image 16. Epipolars for a calibrated camera with 10cm of distance

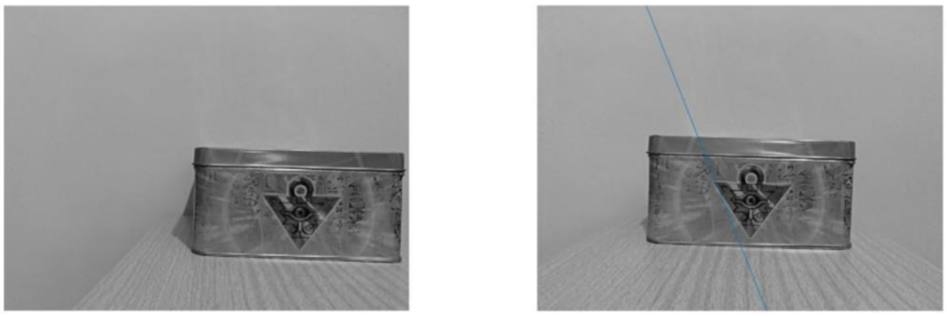


Image 17. Normal function on drawing the epipolar on a calibrated camera with 10cm of distance

As we see in the images, for the uncalibrated camera, normally the epipolars are almost horizontal lines. However, due to how the RAMSAC functions, that lines can converge in a point.
For the tests in the calibrated camera, it seems that always converge in a point, but his epipolar is always well calculated (the point in both images are correlated and pass an epipolar in it).

5. **Tridimensional reconstruction.** Finally, project in 3D the matched points by cross correlation using the following equations, and represent them in a figure using **plot3**:

$$\boxed{\begin{aligned} X_i &= b*(xl_i-cx)/d_i \\ Y_i &= b*(yl_i-cy)/d_i \\ Z_i &= f*b/d_i \end{aligned}}$$

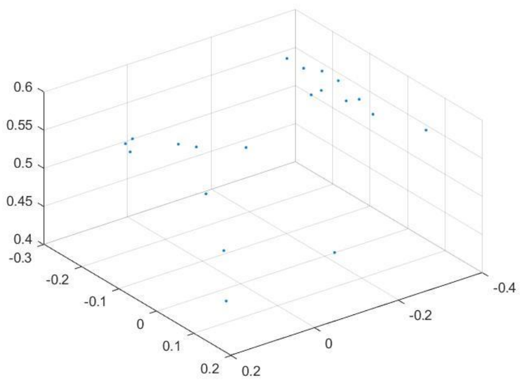
Use the parameters cx, cy, and f from your camera, and the baseline between the two pictures. If you are working with the provided ones (pepsi images), use the following parameters:

$$\boxed{b = 0.119 \text{ m},\ cx = 255.64 \text{ px},\ cy = 201.12 \text{ px},\ f = 351.32 \text{ px},\ d_i\text{= disparity of i-th point}}$$

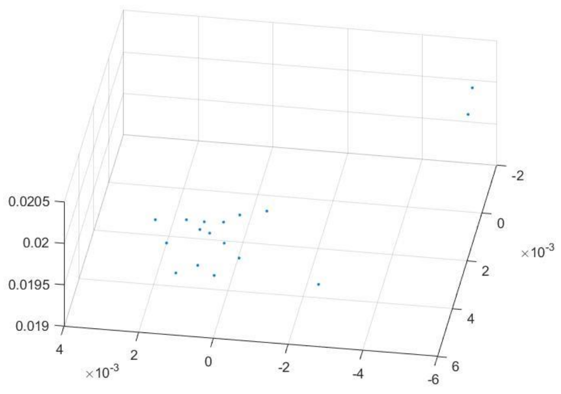
Recall that, as we shown in the lecture, those equations come from:

$$\boldsymbol{X}_l = \frac{k_x b}{d_i}\begin{bmatrix} \frac{1}{k_x}(v_l - v_0) & -\frac{1}{k_y}(u_l - u_0) & f \end{bmatrix}^T = \frac{b}{d_i}\begin{bmatrix} (v_l - v_0) & -\frac{k_x}{k_y}(u_l - u_0) & f \end{bmatrix}^T$$

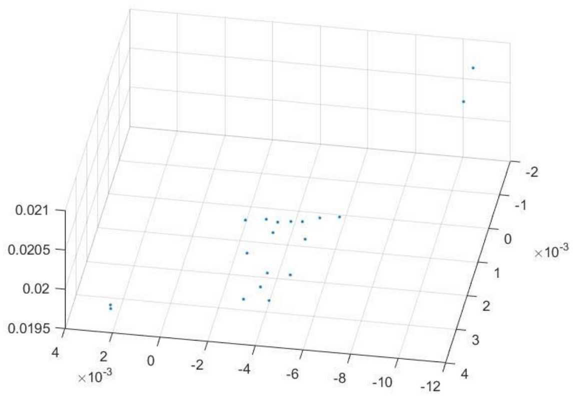
Being v_l = xl, v_o = cx, u_l = yl, u_0=cy. Notice that we are simplifying the problem considering that kx and ky (sx and sy in the calibration provided by Matlab) are roughly 1.



Plot 1. 3D representation of the uncalibrated camera



Plot 2. 3D representation of the calibrated camera with 5cm of difference



Plot 3. 3D representation of the calibrated camera with 10cm of difference

As we see in the plots, it tries to simulate the depth of the composition of the two images, but due to the lack of points, we can represent well enough the object(s).