3D recovery of urban scenes. Session 2

Guillem Capellera[1], Johnny Núñez[1], and Anna Oliveras[1]

1. Universitat Pompeu Fabra, Barcelona, Spain

The goal of this project is to establish the correlation between two images by identifying key-point correspondences. Once the correlation, known as homography, is established, it is applied to various applications such as creating image mosaics, camera calibration, identifying logos and replacing logos. This document explains the procedures followed, the outcomes obtained, and the challenges faced. The key algorithm used to calculate the homography is the normalized DLT algorithm.

1 Homography estimation with the DLT algorithm

The goal is to compute the homography that relates two images:

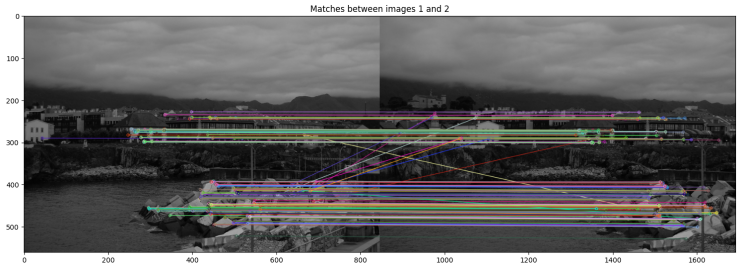
- of the same plane in the 3D scene
- taken with a camera rotating about its centre
- taken with the same static camera varying its focal length
- the whole scene is far away from the camera

1.1 Compute image correspondences

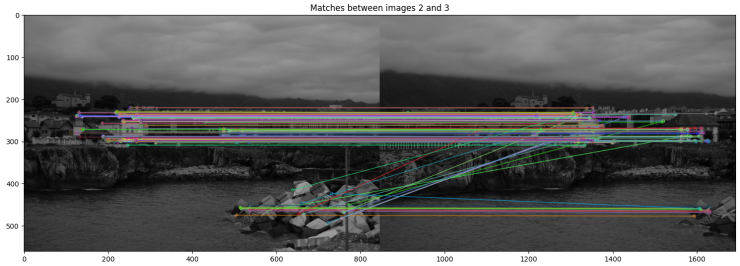
To establish the 2D correspondences between two images, we first detect key-points and compute descriptors using ORB. However, in some cases where ORB did not detect a lot of key-points we also tried to use SIFT. We then match the descriptors using K-NN algorithm and discard those pairs that have a match ratio bigger than a threshold (th_kp_distance=0.85). This last thresholding helps to ensure that the key-point matches are accurate. See Fig.1 to see an example of filtered matches between key-points of two images. However, we see that even with this first filtration, the matches are not all correct, for example, in Fig.1 some features from the rocks are matched to the buildings.

1.2 Compute the homography (DLT algorithm) between image pairs

The homography, referred as H, can be estimated using RANSAC and DLT algorithms. The normalized DLT algorithm allows us to compute the homography given 4 key-point correspondences. RANSAC algorithm iteratively applies the normalized DLT algorithm given 4 random correspondences. Therefore for each step, a homography is obtained and the number of inliers with respect to a given threshold (th_inliers=3) is computed. The RANSAC algorithm returns the homography H with the largest number of inliers. In Fig. 2 we see the same



(a) Matches between image 1 and 2



(b) Matches between images 2 and 3

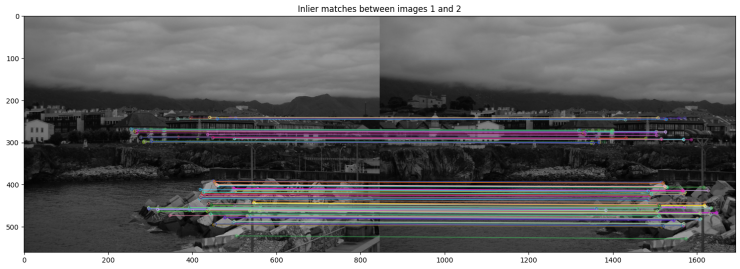
Fig. 1: Image correspondences of the Llanes landscape images

example as before but with the filtrated matches. It is interesting to see that now the matches are more accurate, and in fact, the wrong correspondences that we commented on before are gone.

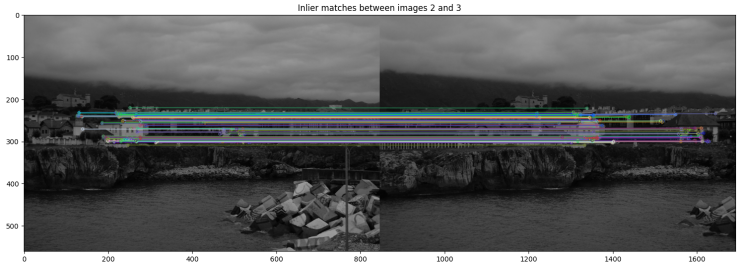
2 Application: Image mosaics

To create different mosaics, we create a function that compares each image with others with techniques that we have seen in the previous section. Therefore, we calculate the DLT algorithm to compute the homographies between pairs of images and apply H (matrix of the corresponding homography).

Continuing with the Llanes example, we can see the created mosaic in Fig. 3. In this particular example, we see that it works well, as there were enough correspondences between the images matching the principal and relevant elements of the image: the buildings and the rocks. However, the other examples are more challenging.



(a) Inlier matches between images 1 and 2



(b) Inlier matches between images 2 and 3

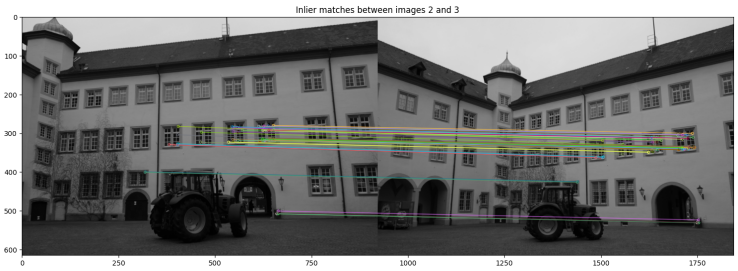
Fig. 2: Image correspondences of the Llanes landscape images after computing the DLT algorithm

Fig. 3: Mosaic of the Llanes landscape

Let's analyze the mosaic of the castle images in Fig.5. We see that the building has been well merged between the three images. However, in the middle of the image, we see a weird effect that seems like a ghost tractor. This is because when superposing the two images, the interpolation is done between the pixel values. So, in this case, as the building is really bright compared to the truck, it feels transparent in some of its parts as it has more power. Notice also that the truck

appears multiple times since there are no inlier matches between the images in that specific region as we can see in Fig.4.

(a) Inlier matches between images 1 and 2

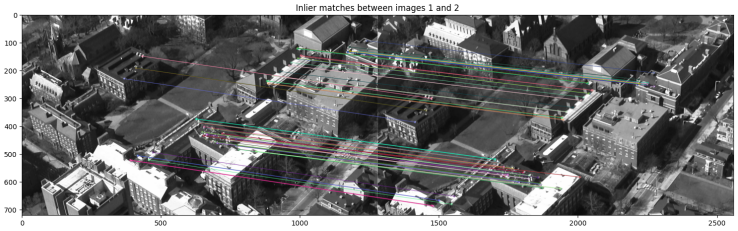


(b) Inlier matches between images 2 and 3

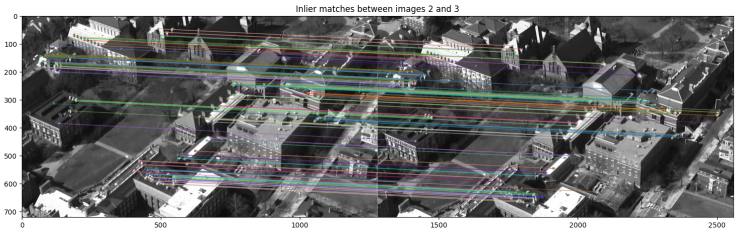
Fig. 4: Image correspondences of the Castle images after computing the DLT algorithm

Fig. 5: Mosaic of the Castle images

In the Aerial13 example, although in general, the panorama is correct, it has some blurry parts like the three in the centre of the image or the bottom right corner as we can see in Fig.7. It is important to highlight that in these two zones, there are no inlier matches either between images 1 and 2 or 2 and 3, as we see in Fig. 6.



(a) Inlier matches between images 1 and 2

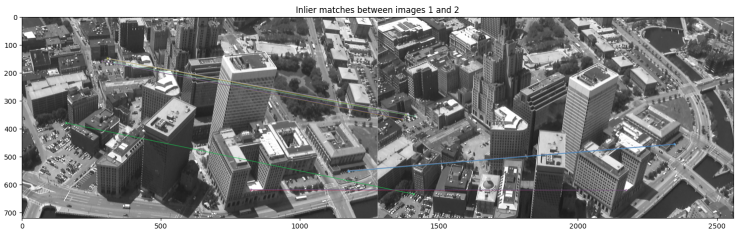


(b) Inlier matches between images 2 and 3

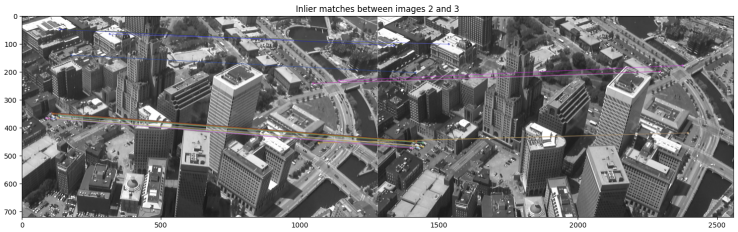
Fig. 6: Image correspondences of the Aerial13 images after computing the DLT algorithm

Finally, Fig.9 shows the mosaic of the Aerial22 images. In this case, the homographies were not estimated correctly. In Fig.8 we see that we don't have matches in any of the buildings. Also, we have very few matches. Thus, we tried to compute the same but using the SIFT descriptor. We got more matches, although a lot of them are wrong, as we can see in Fig. 10. Notice that the mosaic is still wrong in this case, as shows Fig.11. Thus, we realized that this scenario is not possible to do the mosaic because it is not one of the cases when we can apply DLT, as stated at the beginning of section 1. We see, for example, different facades of the same buildings that can't be obtained by just rotating the camera.

Fig. 7: Mosaic of the Aerial13 images



(a) Inlier matches between images 1 and 2



(b) Inlier matches between images 2 and 3

Fig. 8: Image correspondences of the Aerial22 images after computing the DLT algorithm

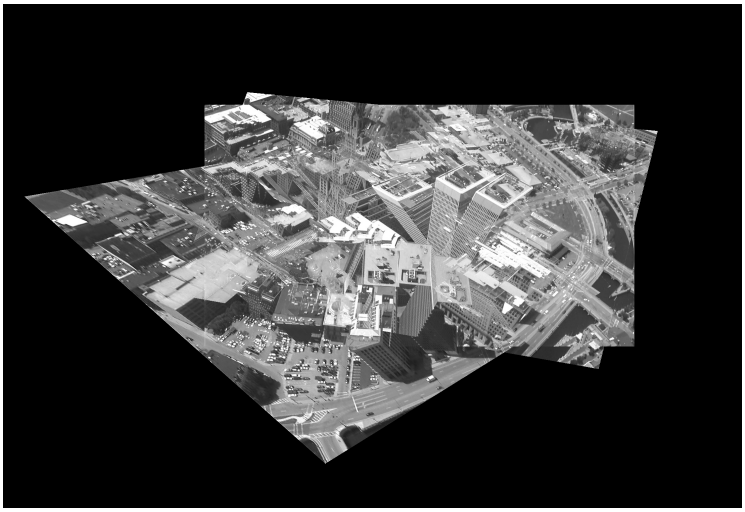
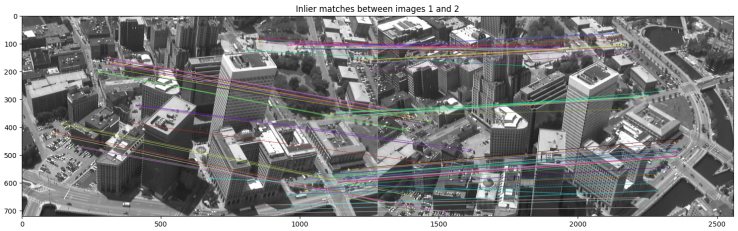
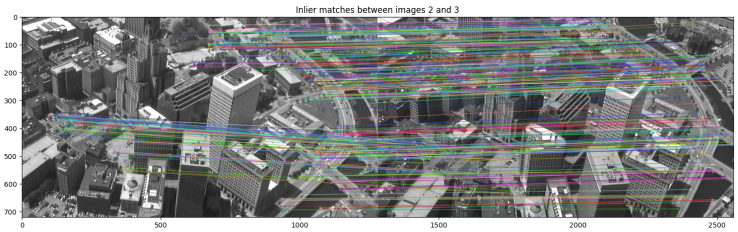


Fig. 9: Mosaic of the Aerial22 images



(a) Inlier matches between images 1 and 2



(b) Inlier matches between images 2 and 3

Fig. 10: Image correspondences (using SIFT) of the Aerial22 images after computing the DLT algorithm

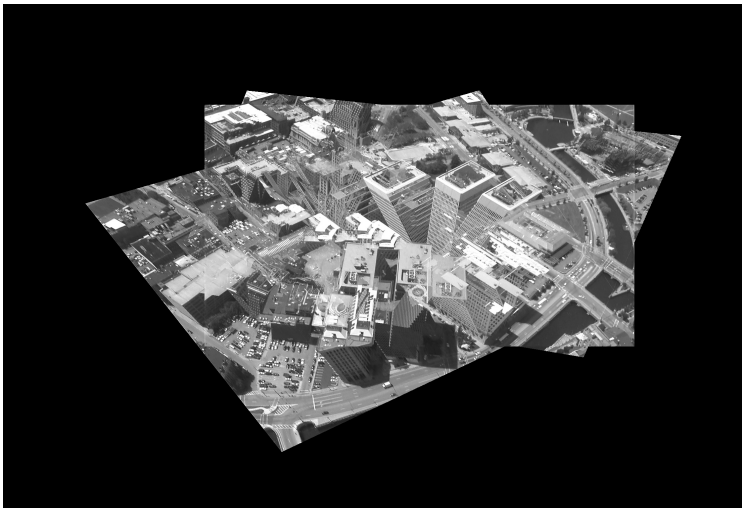


Fig. 11: Mosaic of the Aerial22 images

3 Refinement of the estimated homography with the Gold Standard algorithm

The Homography (H) obtained using the previously explained algorithms can be improved using the Gold Standard algorithm. This is based on solving the following non-constrained minimization problem which minimizes the geometric error:

$$\min_{\hat{H},\hat{x}} \sum_i d([x_i],[\hat{x}_i])^2 + d([x'_i],[\hat{H}\hat{x}'_i])^2 \tag{1}$$

where different correspondences x_i and x'_i are the available data ($x'_i = Hx_i$), [.] is the projection operator to Euclidean coordinates. To solve the Eq.1 we use the function *least_ squares* from SCIPY library with the Levenberg-Marquardt algorithm.

Going back to the Llanes example, using the Gold Standard algorithm we can see that the refined points, after Gold Standard algorithm in Fig.13. To better compare the two homographies (refined and non refined) geometric error before and after the refinement is shown in Table: for each of the examples.

Moreover, the refined mosaics for each of the examples are shown in Fig. 14, 12, 15 and 16. We see that there is not a noticeable difference between the mosaics before and after the refinement. So, the homographies that were working well before now are more accurate but the ones not working are not fixed. In fact, zooming in for example in the case of Llanes, we see that it was improved a lot in some areas like the rocks and the cliffs as we can see in Fig.

Image	Descriptor	num inliers H	Error before	Error after
Llanes H_12	ORB	133	0.22	0.11
Llanes H_23	ORB	154	0.26	0.13
Castle H_12	ORB	44	0.27	0.14
Castle H_23	ORB	43	0.18	0.09
Aerial13 H_12	ORB	81	0.26	0.13
Aerial13 H_23	ORB	193	0.30	0.15
Aerial22 H_12	ORB	14	0.21	0.10
Aerial22 H_23	ORB	34	0.27	0.15
Aerial22 H_12	SIFT	104	0.30	0.13
Aerial22 H_23	SIFT	379	0.31	0.16

Table 1: Mean Reprojected errors before and after the refinement with the Gold Standard algorithm

Fig. 12: Refined mosaic of the Llanes landscape

(a) Image 1 (b) Image 2 for H12

(c) Image 2 for H23 (d) Image 3

Fig. 13: Inlier points (in blue) and refined points (in pink) for the Llanes landscape images

Fig. 14: Refined mosaic of the Castle

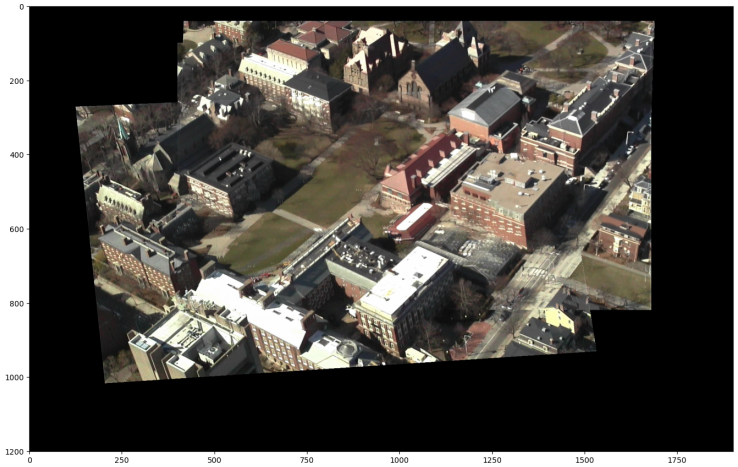


Fig. 15: Refined mosaic of the Aerial13

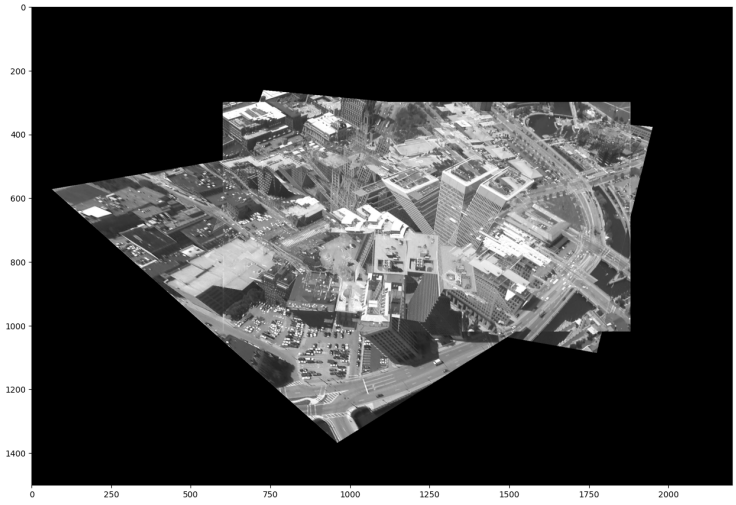


Fig. 16: Refined mosaic of the Aerial22

(a) Llanes mosaic (zoom) (b) Llanes refined mosaic (zoom)

Fig. 17: Zooming of the Llanes mosaic. See that here we can appreciate the improvement of the refinement

4 OPTIONAL. Application: Calibration with a planar pattern and augmented reality

4.1 Camera calibration

Here, we have to find the matrix P of dimension 3x4 to determine the intrinsic and extrinsic parameters of a camera. P (see Eq.2) represents the coordinates of a 3D point in the world, in relation to the camera's coordinate system. This matrix can be composed of K (internal parameters of the camera) which include the focal length and the principal point of the camera, and R matrix that represents the rotation matrix, which describes the orientation of the camera in relation to the world and finally t that represents the translation vector, which describes the position of the camera in relation to the world.

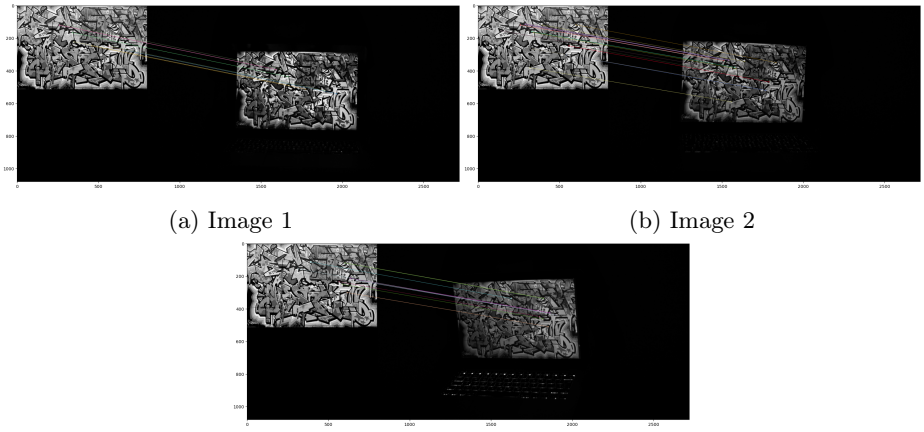
$$P = K[R \mid t] = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{bmatrix} \tag{2}$$

Together, these parameters describe the relationship between the 3D world and the 2D image captured by the camera, allowing for accurate measurements and image rectification.

In this section, we use calibration object (planar pattern) which is a 2D pattern that has a flat, or planar, surface. During the calibration process, images of the planar pattern are captured from different angles and positions using the cameras we want to calibrate. The corners of the squares in the pattern are then detected in the images using image processing algorithms. These detected corners, along with their corresponding 3D coordinates around the world, are used to compute P. Once the intrinsic and extrinsic parameters have been computed, they can be used to rectify images.

In this section, we are asked about the best library to use. The following explanation clarifies our decisions. Scipy works with sparse matrix. What does this mean? A sparse matrix is a matrix that contains many zeros. Scipy saves the positions that contain values different than zero in consideration of saving all values of matrix-like Numpy. This is more efficient because you use

less memory. Another point of view is that Scipy is based on Linear Algebra Package(LAPACK) which is a very optimized calculus on typical factorization algorithms like Cholesky. In the case of inverting a matrix, is good to use *np.linalg.inv* because is parallelized with your CPU. In our case, we use the Scipy library to calculate Cholesky. Finally, the Scipy implementation of Cholesky gives the upper triangular matrix by default, so no need of extra computations like in NumPy, which returns the lower triangular matrix.



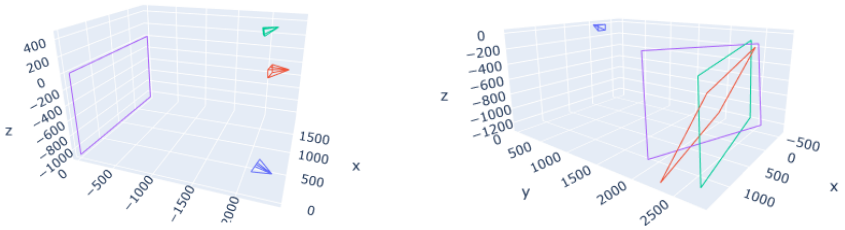
(a) Image 1

(b) Image 2

(c) Image 3

Fig. 18: Inlier matches between the template image (left) and each of the 3 images taken with the cameras we want to calibrate

Finally in Fig. 19 A) we can see the planar pattern and the N camera locations. Note that the optical center computation comes from solving the equation $PX = 0$ for each of the cameras (3 in this case). This is done by using the Singular Value Decomposition (SVD) of P. Then, we know that the optical center (X) is the last column of U^T divided by the last coordinate of the vector. Alternatively, Fig. 19 B) shows a fixed camera and N planar patterns at different relative poses.

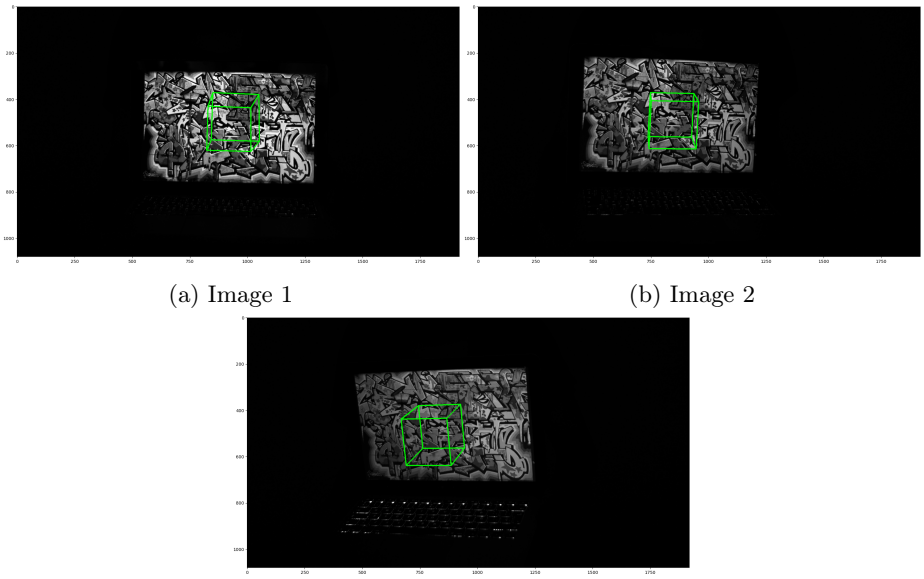


(a) (b)

Fig. 19: A) Location of the 3 cameras and the planar pattern. Camera 1 in purple, camera 2 in red, camera 3 in green and image in pink. B) Fixed camera and 3 image planar patterns at different relative poses with respect to the camera. Camera in purple, image 1 in red, image 2 in green and image 3 in pink.

4.2 Augmented reality

We can experiment with augmented reality now that we have calibrated the cameras and know the coordinates of the image in the 3D world for each camera. We can project a cube manually put in the center of the template picture to each of the N cameras image planes using our understanding of the intrinsic and extrinsic matrices of the camera. Calibration is essential required because of this projection.



(a) Image 1

(b) Image 2

(c) Image 3

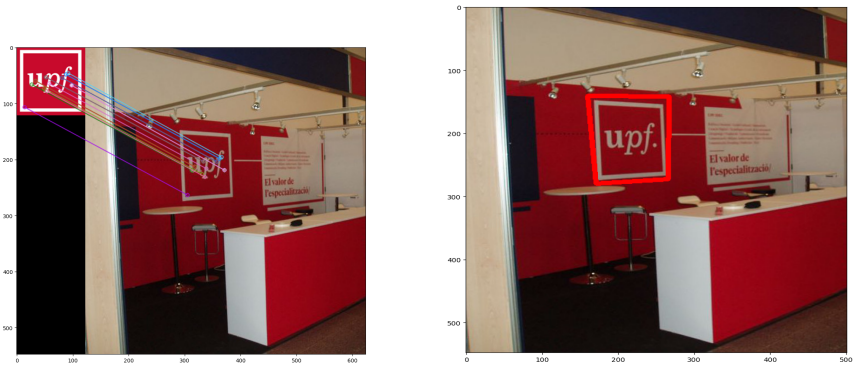
Fig. 20: Inserting a cub on the different images after the camera callibration

5 OPTIONAL. Application: Logo detection

In M1 subject we have learnt how to do masks to provide a detect and crop a painting. In this case, the process involves identifying matches between the logo to be inserted and the target image, using these matches to calculate the homography between the two, and then mapping the corners of the logo using this homography. The new logo is then transformed using the homography to create a mask for replacement.

(a) UPF logo matches (b) UPF Box

Fig. 21: UPF Building

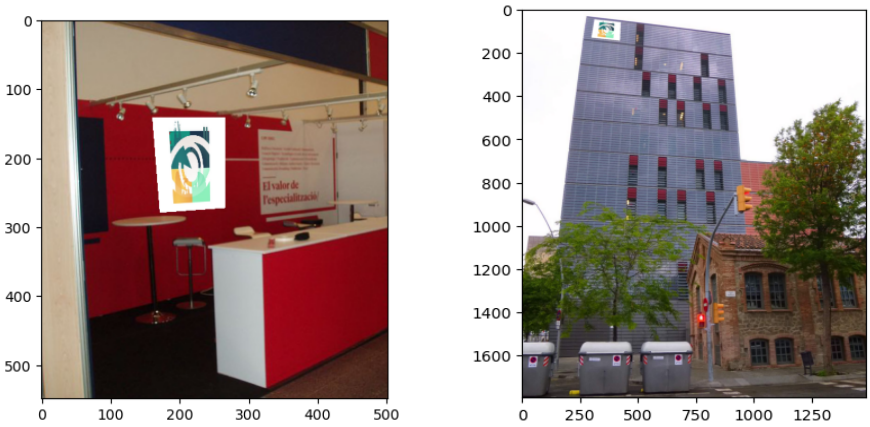


(a) UPF logo matches (b) UPF Box

Fig. 22: UPF Stand

6 OPTIONAL. Application: Logo replacement

In the images 23, the method was found to be sensitive to the specific method used to detect keypoints with some perspective like UPFStand, Therefore we should better algorithm to detect better the corners on the images.



(a) UPF Stand (b) UPF Building

Fig. 23: Replacing Logo UPF by MCV