# M4 Lab5 - 3D reconstruction from uncalibrated images

Mikel Menta Garde, Sebastián Maya Hernández,

Pedro Luis Trigueros Mondéjar, Alex Vallès Fernández

January 31, 2018

**Abstract**

In this lab we have performed the auto-calibration and 3D stratified reconstruction from a pair of uncalibrated images. First, it was done on synthetic data (for easier debugging purposes) and then on real data. The stratified reconstruction can be divided in three different reconstructions:

1. A **projective reconstruction** that obtains a possible 3D scene (as there exists a projective ambiguity) from the two images and the corresponding projection matrices of each camera.
2. An **affine reconstruction** that makes parallel the 3D lines that are parallel in the real world.
3. A **metric reconstruction** that makes orthogonal the 3D lines that are orthogonal in the real world.

With this we should be able to reconstruct the original scene only with a similarity (or metric) ambiguity.

## 1 Projective reconstruction (synthetic data)

First of all, having at least two views and a set of point correspondences (that are visible in all of them) we need to compute the 3D points and the corresponding projective matrices of each camera.

The recovered matrices and 3D points have not an unique reconstruction, so we will obtain them with a projective ambiguity that we will solve on following sections.

To do reconstruction, we will follow the **factorization method**. Having the corresponding points $x_j^i$, where $i$ represents the image number and the $j$ is the point index, we basically need to obtain the correspondences with the 3D points X

$$x_j^i \equiv P^i X_j \tag{1}$$

which can be rewritten as

$$\lambda_j^i x_j^i = P^i X_j \tag{2}$$

where $\lambda_j^i$ are unknown scalar factors called projective depths.

So we have

$$\begin{bmatrix} \lambda_1^1 & \lambda_2^1 & \lambda_3^1 & ... & \lambda_n^1 \\ \lambda_1^2 & \lambda_2^2 & \lambda_3^2 & ... & \lambda_n^2 \\ \lambda_1^3 & \lambda_2^3 & \lambda_3^3 & ... & \lambda_n^3 \\ ... & ... & ... & ... & ... \\ \lambda_1^m & \lambda_2^m & \lambda_3^m & ... & \lambda_n^m \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ ... \\ P_m \end{bmatrix} \begin{bmatrix} x_1 & x_2 x_3 & ... & x_n \end{bmatrix} \tag{3}$$

where we call measurement matrix $M$ to

$$M = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ ... \\ P_m \end{bmatrix} \begin{bmatrix} x_1 & x_2 x_3 & ... & x_n \end{bmatrix} \tag{4}$$

**Algorithm 1:** Factorization method

1 Determine a subset of scene points and cameras so that the measurement matrix is completely determined.
2 Normalize the set of points in each image using similarity transf Hs.
3 Initialize all $\lambda_j^i = 1$
4 Alternate rescaling the rows of the depth matrix $\wedge$ (formed by $\Lambda_j^i$) to have unit norm and the columns of $\wedge$ to have unit norm until $\wedge$ stops changing significantly (usually two loops).
5 Build the measurement matrix M.
6 Determine the SVD of $M = UDV^T$.
7 Let $P_M = UD_4$ and $X_M = V_4^T$.
8 If $P\sum_i\sum_j d(x_i^j, P^i X_j)^2$converges then stop. Otherwise let $\lambda_j^i = (P^i X_j)_3$ and go to Step 4.
9 Unnormalize the camera matrices $(H_s^i)^{-1}P^i$
10 (Triangulate and resection the non-nucleus scene points and cameras).

Having this notions, the method consists on following the steps explained in algorithm 1.
For performing the normalization, we can apply the following transformation in each image

$$H_s^i = \begin{bmatrix} s^i & 0 & -s^i c_x^i \\ 0 & s^i & -s^i c_y^i \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where $(c_x^i, c_y^i)$ is the position of the centroid of image $i$ and $s^i = \frac{\sqrt{2}}{Dist\_avg(c^i)}$ so that every point is now at an average distance of $sqrt2$ from the new origin.

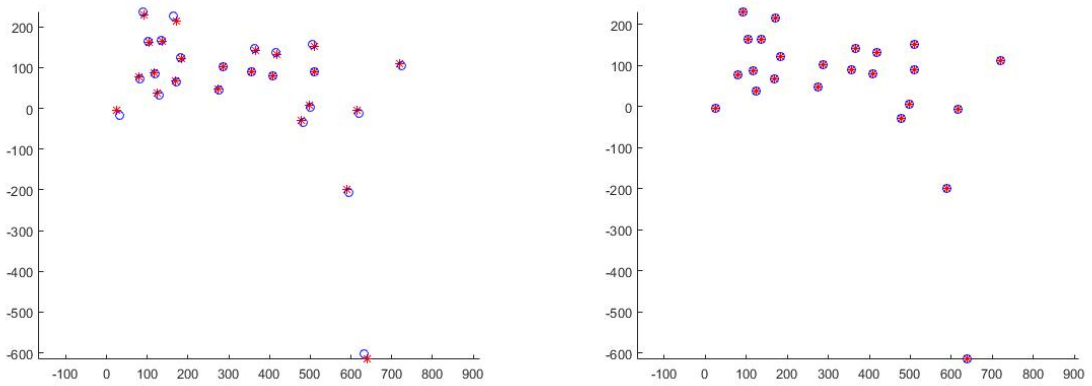
## 1.1 $\lambda_j^i$ initialization

We have tried two different initializations for the $\lambda_j^i$ values of the system: first initializing all of them to value 1 and also with the *Sturm and Triggs* initialization (eq. 6);

$$\lambda_j^i = \frac{(x_j^1)^T F_i 1 (e \times x_j^i)}{\|e \times x_j^i\|^2}\lambda_j^1 \text{ with } \lambda_j^1 = 1 \tag{6}$$

where the epipolar line of $x_j^1$ in image $i$ is the line through the corresponding point $x_j^i$ and the epipole $e$.

The results are shown in figure 1 and a great improvement is clear for the *Sturm and Triggs* initialization.




a) Lambda initialization with ones

b) Lambda initialization using Sturm formula

Figure 1: Comparison between different lambda initialization Image/camera 1

## 2 Affine reconstruction (synthetic data)

After the projective reconstruction, we have to do an affine rectification in order to preserve the parallel lines. The easiest way to proceed is to use prior information, for instance the lines in the scene that are parallel.

For affine rectify the reconstruction we have to find the homography $H_{a\leftarrow p}$.

$$H_{a\leftarrow p} = \begin{bmatrix} I & 0 \\ p^T & 1 \end{bmatrix} \tag{7}$$

For getting this homography first we have to find $p$ and we will do it computing the plane at infinity.

As every plane, the plane at infinity is determined by three points on it. For computing these points we only have to do the cross product over a pair of parallel lines in the projective space. So for three vanishing points $X_1, X_2, X_3$, we have:

$$\begin{bmatrix} X_1^T \\ X_2^T \\ X_3^T \end{bmatrix} p = 0 \tag{8}$$

So we can extract $p$ using SVM and then we use it as the last row of the homography. In the Figure 2 we can see the result that we get of the affine reconstruction.

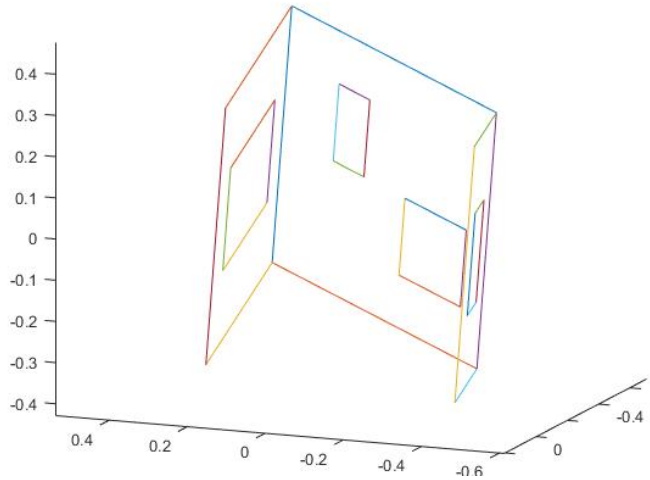



Figure 2: Results of the affine rectification.

## 3 Metric reconstruction (synthetic data)

To obtain a 3D metric reconstruction from an affine reconstruction, we need to find the image of the absolute conic $\omega = K^{-T}K^{-1}$ (and thus the internal parameters) in one of the images and apply an homography

$$H_{e\leftarrow a} = \begin{pmatrix} A^{-1} & 0 \\ 0^T & 1 \end{pmatrix} \tag{9}$$

where $AA^T = (M^T\omega M)^{-1}$ getting $M$ from the projection matrix of the image $P = [M|\vec{m}]$.

To find the image of the absolute conic $\omega$ we use a combination of the following constraints:

- Constraints of the orthogonality of the scene lines.
- Constraints coming from known internal parameters, such as the zero skew and having square pixels.
- Having the images taken from the same cameras (with same internal parameters).

and using three vanishing points $u$, $v$ and $z$ we construct a system of equations $A\overrightarrow{\omega} = 0$

$$A = \begin{bmatrix} u_1v_1 & u_2v_2 + u_2v_1 & u_1v_3 + u_3v1 & u_2v_2 & u2_v3 + u3_v2 & u_3v_3 \\ u_1z_1 & u_2z_2 + u_2z_1 & u_1z_3 + u_3z1 & u_2z_2 & u2_z3 + u3_z2 & u_3z_3 \\ v_1z_1 & v_2z_2 + v_2z_1 & v_1z_3 + v_3z1 & v_2z_2 & v2_z3 + v3_z2 & v_3z_3 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} \tag{10}$$

doing the SVD decomposition of $A$ we will get the values for $\omega$ in the last vector of matrix $V$.

Once we have obtained $\omega$, we just need to apply a Cholesky factorization on $AA^T = (M^T\omega M)^{-1}$ to obtain the matrix $A$ to insert on $H_{e\leftarrow a}$.

We can see the results in the figure 3.

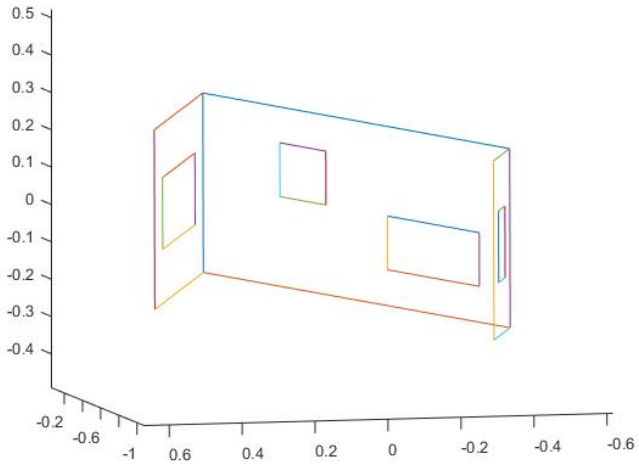



Figure 3: Results of the metric rectification.

# 4 Projective reconstruction (real data)

From this section on, we proved the methodology with real data using the images shown in the following figure 4.

Figure 4: Input images

To compute the projective reconstruction, we need to find point correspondences between the two images. For doing this, we first compute the *SIFT* local features and their matches among both images. Secondly, we compute the fundamental matrix $F$ between them using *RANSAC* and this way we clean the bad correspondences (as they will be outliers for $F$).

Once we have obtained the clean matches, we just apply the methodology used in section 1 to obtain the projective reconstruction.

# 5 Affine reconstruction (real data)

For the affine reconstruction the only difference in comparison with the example of the synthetic data is the computation of the vanishing points. In this case, as we didn't know the exact points of parallel lines, we have to compute it using a provided code that searches all the lines in the image and group the ones that have the same direction.

After applying this code, we get the vanishing points of both of the images and we can compute the Affine reconstruction. In figure 5 we can see the result of this reconstruction.

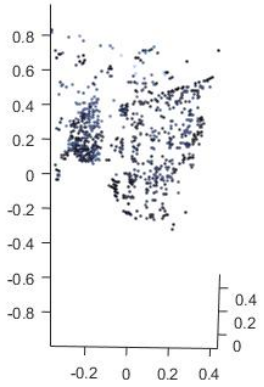



Figure 5: Results of the affine rectification.

# 6 Metric reconstruction (real data)

For the Metric rectification is the same process of the section 3. The results that we got for the metric reconstruction are shown in the Figure 6

a) Up view b) Front view

Figure 6: Comparison between different views of the metric reconstruction

# 7 Optionals from week 4

## 7.1 Plane sweeping

Plane sweeping is a method to obtain the depth map from two (not necessarily rectified) images $I$,$I'$ and their camera matrices $P$, $P'$. It consists on projecting one image $I$ to a frontal plane to the scene at different depths and projecting it back to the first image $I'$. Then, we can check which is the depth that best matches the projection $I_{proj}$ and the image $I'$. Specifically, the steps consist on the following:

For each *depth* until the fixed $max\_depth$ is reached:

1. Get the fronto-parallel plane at the specific *depth* as

$$\Pi = p_3^T - (0, 0, 0, depth) \tag{11}$$

where $p_3^T$ is the last row of camera matrix $P$.

2. Obtain the homography that maps the image $I$ first, to the fronto-parallel plane $\Pi$ ($\bar{A}$ from eq. 12) and then, back to the image $I'$ ($H$ from eq. 13).

$$A = \begin{pmatrix} P \\ \Pi^T \end{pmatrix}^{-1}$$

$$\bar{A} \text{ sub-matrix } 4 \times 3 \text{ of } A \tag{12}$$

$$H = P'\bar{A} \tag{13}$$

3. Apply the homography $H$ to image $I$ to obtain $I_proj$.

4. Compute a matching score between $I_{proj}$ and $I'$ such as *SSD* (eq. 14) or *NCC* (eq. 15) performing a sliding window.

$$SSD(p) = \sum_{q \in N_p} w(p,q)|I_1(q) - I_2(q)|^2 \tag{14}$$

$$NCC(p,q) = \frac{\sum_{q \in N_p} w(p,q)(I(q) - \bar{I}_1)(I_2(q) - \bar{I}_2)}{\sigma_{I_1}\sigma_{I_2}} \tag{15}$$

$$\bar{I}_1 = \sum_{q \in N_p} w(p,q)I(q) \tag{16}$$

$$\bar{I}_2 = \sum_{q \in N_p} w(p,q)I_2(q+d) \tag{17}$$

$$\sigma_{I_1} = \sqrt{\sum_{q \in N_p} w(p,q)(I_1(q) - \bar{I}_1)^2} \tag{18}$$

$$\sigma_{I_2} = \sqrt{\sum_{q \in N_p} w(p,q)(I_2(q+d) - \bar{I}_2)^2} \tag{19}$$

where $N_p$ is the $n$ size neighborhood of point $p$ defined as

$$N_p = \left\{ q = (q_1, q_2)^T | p_1 - \frac{n}{2} \leq q_1 \leq p_1 + \frac{n}{2}, p_2 - \frac{n}{2} \leq q_2 \leq p_2 + \frac{n}{2} \right\} \tag{20}$$

and $w$ are a set of weights for $N_p$

$$\sum_{q \in N_p} w(p,q) = 1 \tag{21}$$

5. For each pixel, keep the *depth* value with the best matching cost (smallest in the case of *SSD* and biggest in the case of *NCC*).

An example of the results are shown in Fig. 7. As we can see, there appear some noise in areas where textures are similar. This maybe could be cleaned up using some thresholding. However, in the rest of the image seems to perform well.

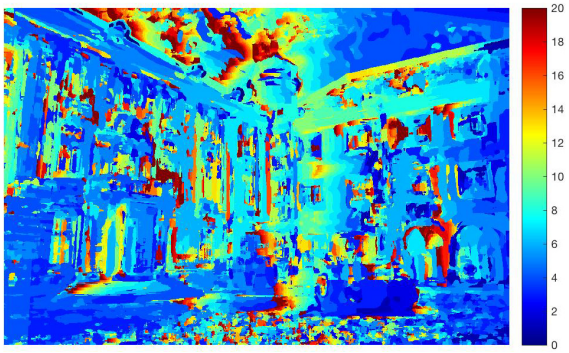



Figure 7: Results with SSD and a window size of 11.

### 7.2 Stereo computation with Belief Propagation

To use belief propagation in order to get better results of the stereo computation first we must define the values of the pairwise potentials and the unary potentials.

For the pairwise potentials we will use the classical Potts Model that tells the graphical model that the disparity should be smooth, in other words, the probability of changing into near disparities is grater than changing to a far disparity. We can see in the equation of the Potts Model below:

$$f_{p,q}(k,k') = \theta_{p,q} 1_{\{y_p \neq y_q\}}(k,k') \tag{22}$$

For the unary potentials we will use the SSD cost converted to a probability ($1/Cost_{SDD}$). To get these potential we must change the stereo_computation function in order to return, for every pixel, the whole array of costs instead of the index of the minimum one.

After defining these potentials, we can create the graphical model with $K = max\_disparity - min\_disparity$. To create the model we used the UGM library for Matlab, that give us all the tools necessary to compute a graphical model. After creating the model, we can solve it using the inference algorithm LBP (Loopy Belief Propagation). After several tests we have not managed to obtain acceptable results. This may be due to some small implementation failure, but we could not figure it out because every time we had to run the whole process it took a lot of time to execute the graphical model code.

## 8 Conclusions

In this lab we have learned how to obtain both the camera internal parameters $K$ or a 3D point reconstruction from a set of uncalibrated images. We have learned to manage different homographies on $\mathbb{P}^3$ too.

Once obtained the first projective reconstruction, the rest of techniques have been quite easy and straightforward to implement. Moreover, the performance is really good when using the proper data.

However, we have seen how to manage point clouds but is still difficult to visualize something more than a general view of the scene when using real data. But, it is something expected while working only with the points extracted from *SIFT*.