M4 - 3D Vision

3D recovery of Urban Scenes

Ana Caballero - ana.caballeroc@e-campus.uab.cat
Arnau Vallvé - arnau.vallve@e-campus.uab.cat
Claudia Baca - claudiabaca.perez@e-campus.uab.cat
Joaquim Comas - joaquim.comas@e-campus.uab.cat

INTRODUCTION

The goal of this project is to learn the basic concepts and techniques to reconstruct a real world scene given several images (points of view) of it, not necessarily previously calibrated. In this project we focus on 3D recovery of Urban Scenes using images of different datasets, namely images of facades and aerial images of cities.

WEEK 4

Goal: Reconstruction from two images (known internal parameters)

Mandatory:

- Triangulation with the homogeneous algebraic method (DLT)
- Compute the camera matrices from the Fundamental matrix and K
- Recover the 3D points by triangulation
- Compute the reprojection error
- Depth map computation by a local method (SSD cost)
- Depth map computation by a local method (NCC cost)
- Improve the matching cost by using bilateral weights

Optional:

- New view synthesis

1. Triangulation with the homogeneous algebraic method (DLT)

Given two different matching points x and x' and already having the camera matrices P and P', we aim to use the algebraic DLT method to triangulate the points. For this the function *triangulate.m* was implemented. We aim to solve:

$$\min_{\mathbf{X}} \|\mathbf{AX}\|_2$$
$$\text{such that } \|\mathbf{X}\|_2 = 1.$$

to obtain the estimated 3D point X that intersects the visual rays of two corresponding points x and x' from two images. First, for the method to properly work, we will require of a normalization by H to scale and translate the pixel coordinates of x and x' in an interval of [-1, 1] where *nx* and *ny* correspond to the size of the image im the dimensions x and y.

$$H = \begin{pmatrix} 2/\mathrm{nx} & 0 & -1 \\ 0 & 2/\mathrm{ny} & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

The transformation of the points as well as the camera matrices by H then follows as:

$$\mathbf{x} = H\mathbf{x},\ \mathbf{x}' = H\mathbf{x}',\ P = HP, \text{ and } P' = HP'$$

Then we can build the matrix A with the transformed points and camera matrices as follows:

$$\mathbf{A} = \begin{pmatrix} x\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ x'\mathbf{p}'^{3T} - \mathbf{p}'^{1T} \\ y'\mathbf{p}'^{3T} - \mathbf{p}'^{2T} \end{pmatrix}$$

To solve it, we compute the SVD of the above matrix A and get the singular vector associated to the minimum singular value of A (from the last column of V (null vector) we will get our solution).

To validate the implementation, there is a part of the code that computes an error between our triangulated points and a set of testing points. The error obtained by our implementation (see below) is close to zero which ensures us the implementation seems to properly triangulate.

Error obtained:

```
   1.0e-14 *

  -0.0472   -0.0777   -0.0042   -0.0444         0.0167   -0.0139   -0.0777      0.1776
  -0.0999   -0.0888   -0.0500   -0.0555        -0.0278    0         0.0167      0.0444
  -0.3997   -0.4441   -0.0444   -0.3553         0.0888   -0.3109   -0.4885      0.6217
```

2. Reconstruction from two views

For this exercise, we'll work with the two grayscale facade images shown below:

Image 1: Grayscale facade images *0001_s.png* and *0002_s.png* we'll be using

Similar to the exercises seen in previous weeks, we'll obtain the image keypoints using SIFT and obtain the matches between both of the given images. To refine it, we also remove the outliers by fitting the fundamental matrix F using the RANSAC robust method we've also already explored and explained in previous weeks. We can see those keypoints and matches between images 1 and 2 before and after removing the outliers with the RANSAC fundamental matrix method.

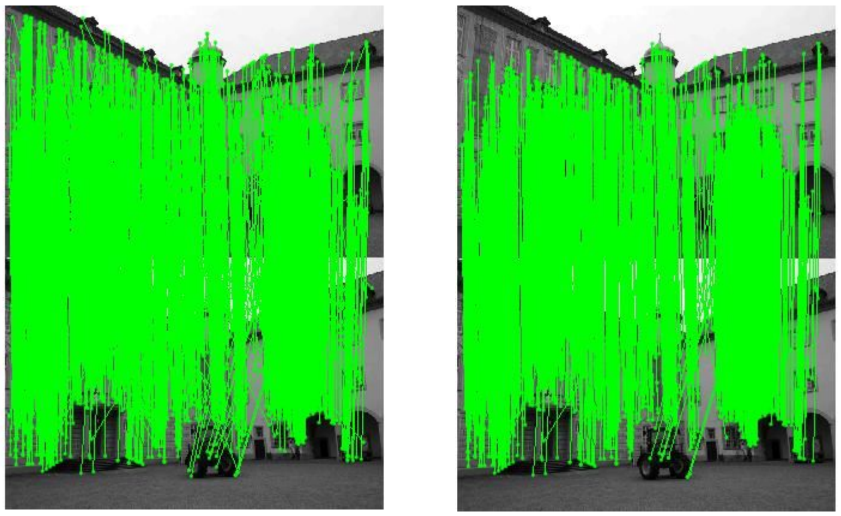


Image 2: Left: Obtained SIFT keypoints and matches between image 1 and 2. Right: Inliner matches obtained estimating F using RANSAC

From the given camera calibration matrix K and the previously estimated fundamental matrix F, we will go towards obtaining the candidate camera matrices. The essential matrix we require to continue towards our goal can be estimated from the following relationship between the Fundamental matrix F, the calibration matrices K, K' and the essential matrix E.

$$\mathbf{E} = \mathbf{K}'^{\top}\ \mathbf{F}\ \mathbf{K}$$

By decomposing the essential matrix E with SVD, and taking the last column of U we can obtain the translation vector T'. We can get that as we the have the following relationships expressing the essential matrix E in two different ways; as a SVD decomposition and as the cross product of the translation T and rotation R.

$$E = UDV^T \qquad E = [T'_\times] R = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

We will have 4 different scenarios as we cannot obtain the sign from T' and we have rotation choices. These are reversed translations depending on the sign of *u3* and a possible 180º rotation. In the equations below, we are naming *u3* as the last column of U.

$$P'_1 = \left[UWV^T \mid +\mathbf{u}_3 \right] \quad P'_2 = \left[UWV^T \mid -\mathbf{u}_3 \right]$$
$$P'_3 = \left[UW^TV^T \mid +\mathbf{u}_3 \right] \quad P'_4 = \left[UW^TV^T \mid -\mathbf{u}_3 \right]$$

From the different possible results we can obtain, we'll use the one that makes realistic geometrical sense, as it makes sense that the reconstructed point has to be in front of both cameras A and B, as in the diagram below.

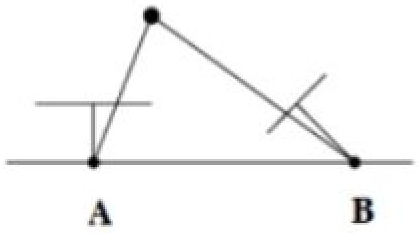


Image 3: Diagram of geometric interpretation

Here we can see the actual plots of the possible results we have using different points of view to better visualize the result. We can visually see in our case that the camera matrix B to use that makes realistic sense from the possible solutions is the one at the top right, by looking at the image on the right (Notice that camera A is the one centered at (0,0)).

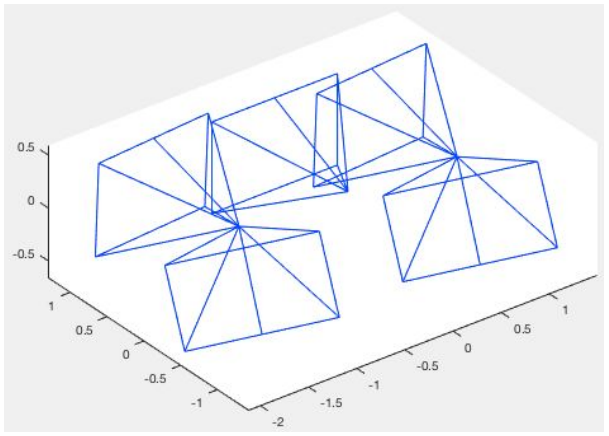
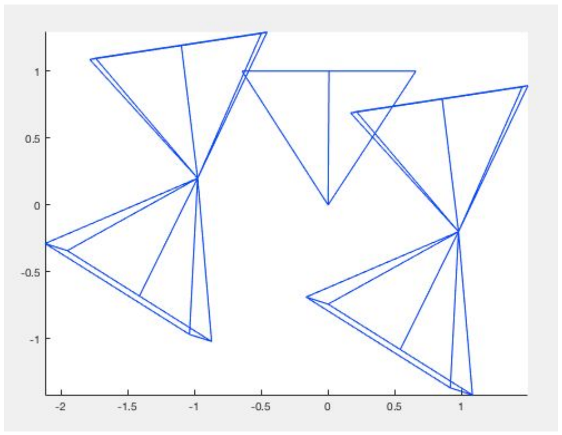
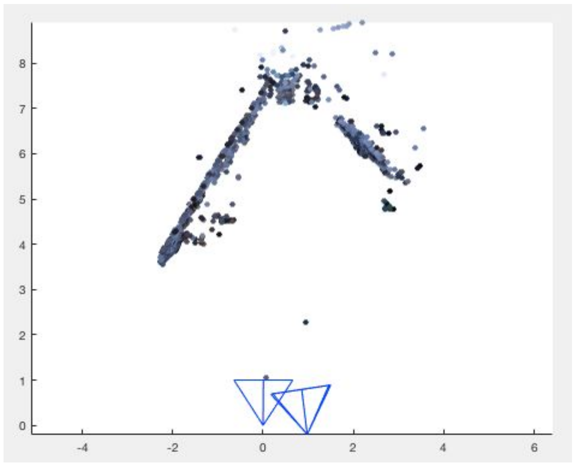
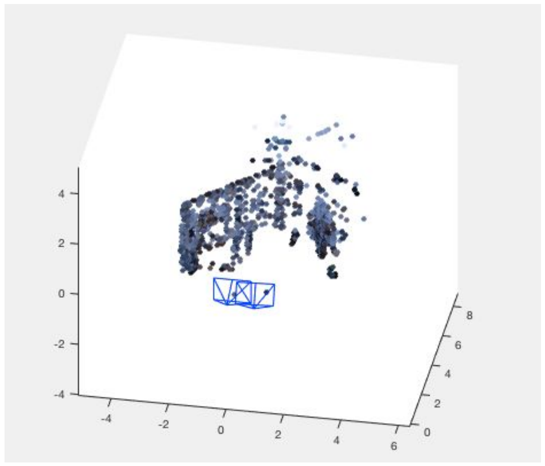



Image 4: First camera A and the four possible solutions for the second. Both images are the same one rotated to better observe the results.

Instead of doing it visually, we can obtain the second camera candidate by using triangulation of matching points by projecting them into every camera and checking when the projected point is in front of both.

In the next two figures, we can appreciate the results obtained after the reconstruction from different points of view where we can observe the 3D reconstruction of the building using two images:

Images 5,6: 3D Reconstruction using two images

After the 3D reconstruction, reprojection error must be considered caused by the triangulation. Therefore, we have used the next equation to compute the reprojection error:

$$\sum_i d(\mathbf{x_i}, \hat{\mathbf{x}}_\mathbf{i})^2 + d(\mathbf{x_i}', \hat{\mathbf{x}}_\mathbf{i}')^2$$

In the next figure we can see the histogram of the reprojection error. Also apart from the histogram we compute the mean of the reprojection error which was 0.5908 and we represented using the following normal distribution:

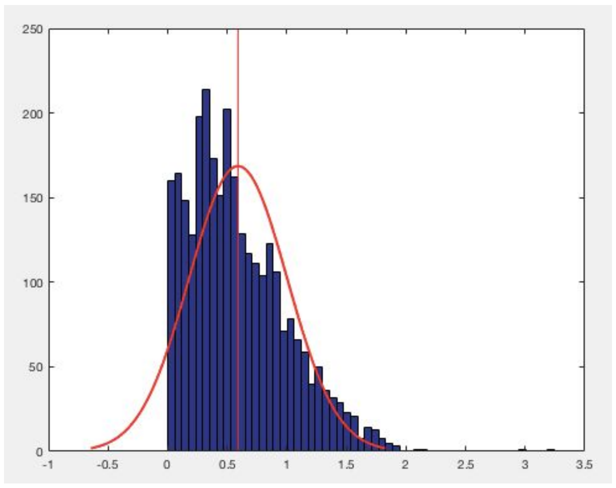


Image 7: Reprojection error histogram and mean

3. Depth map computation with local methods (SSD)

The aim of this exercise is to obtain a depth map using the Sum of Squared Differences (SSD) local method implemented in *stereo_computation.m* function. For this part, we will use the two provided stereo rectified images shown below.

Image 8: Left and right stereo rectified images used to obtain the depth map

As the images are already stereo rectified, this simplifies the pixel correspondence to horizontal scan lines. Here, we will be using a minimum disparity of 0 and a maximum of 16. The method is described as a local method as we will use each point and the neighboring ones that fall inside the window instead of the whole image at once to compute a Cost C depending on the position.

We will use windows on the right and left images *wr, wl* of a specified size (we will change them later to see the effect on the results) centered at a point on the image. These windows will slide horizontally to and compare right content to left content using a Cost. This SSD cost will take into account the intensity differences between right and left windows at position *p* as a function of the disparity *d*. Best matching disparity will be the one with the lower SSD cost.

$$C(\mathbf{p}, \mathbf{d}) = \sum_{\mathbf{q} \in N_\mathbf{p}} w(\mathbf{p}, \mathbf{q}) \left| I_1(\mathbf{q}) - I_2(\mathbf{q} + \mathbf{d}) \right|^m, \quad \sum_{\mathbf{q} \in N_\mathbf{p}} w(\mathbf{p}, \mathbf{q}) = 1 \qquad \textit{where, } m = 2$$

And the *nxn* window at position *p, q* is defined as:

$$N_\mathbf{p} = \{\mathbf{q} = (q_1, q_2)^T \mid p_1 - \tfrac{n}{2} \leq q_1 \leq p_1 + \tfrac{n}{2},\ p_2 - \tfrac{n}{2} \leq q_2 \leq p_2 + \tfrac{n}{2}\}$$

We evaluated the SSD method windows of sizes to see if the results improve or not, and the effects it has on the depth map computation. Sizes evaluated are 3x3, 9x9, 20x20, 30x30.

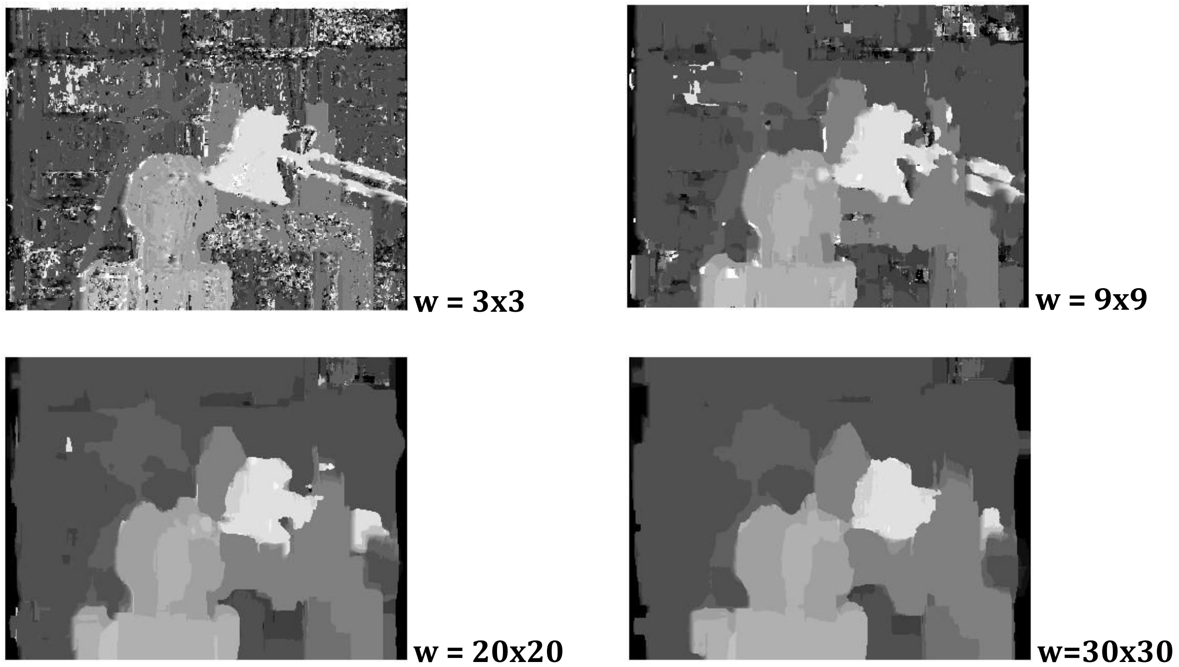


Image 9: Depth map computation with SSD using different window sizes

We can see there is a clear effect on using different window sizes for the depth map computation. Using smaller windows get noisier results, and that noise is reduced as the window used gets larger, probably due to the fact that we have fewer values inside each window to compute the cost every time.

However the images are much sharper when using the smaller windows, as they allow more detail to be kept at the cost of higher noise sensitivity. Larger windows produce a clearer "averaging" effect allowing it to be more robust to the noise but losing detail, we can see how the regions merge together, which can make smaller objects even disappear. So basically, depending on the application one could be interested in one or the other (need of detail vs importance of noise), or even using middle sized windows to have less noise but still retain some of the details. This parameter can be highly optimizable.

4. Depth map computation with local methods (NCC)

This time, we want to obtain a depth map using the Normalized Cross Correlation (NCC) local method which is implemented in *stereo_computation.m* function. We'll use the same stereo rectified images as in the previous task.

This method is also a local method which uses the sliding window as in the SSD previously seen, but this time, we'll use a different matching cost instead (using Normalized cross-correlation (see below)). The use of the windows will be the same as before.

$$NCC(\mathbf{p},\mathbf{d}) = \frac{\sum_{\mathbf{q}\in N_\mathbf{p}} w(\mathbf{p},\mathbf{q})(I_1(\mathbf{q}) - \bar{I}_1)(I_2(\mathbf{q}+\mathbf{d}) - \bar{I}_2)}{\sigma_{I_1}\sigma_{I_2}}$$

where;

$$\bar{I}_1 = \sum_{\mathbf{q}\in N_\mathbf{p}} w(\mathbf{p},\mathbf{q}) I_1(\mathbf{q}); \quad \bar{I}_2 = \sum_{\mathbf{q}\in N_\mathbf{p}} w(\mathbf{p},\mathbf{q}) I_2(\mathbf{q}+\mathbf{d});$$
$$\sigma_{I_1} = \sqrt{\sum_{\mathbf{q}\in N_\mathbf{p}} w(\mathbf{p},\mathbf{q})(I_1(\mathbf{q}) - \bar{I}_1)^2};$$
$$\sigma_{I_2} = \sqrt{\sum_{\mathbf{q}\in N_\mathbf{p}} w(\mathbf{p},\mathbf{q})(I_2(\mathbf{q}+\mathbf{d}) - \bar{I}_2)^2}$$

Here, we also evaluate the NCC method using windows of different sizes to see the effects it has on the depth map computation. Sizes evaluated are 3x3, 9x9, 20x20, 30x30.

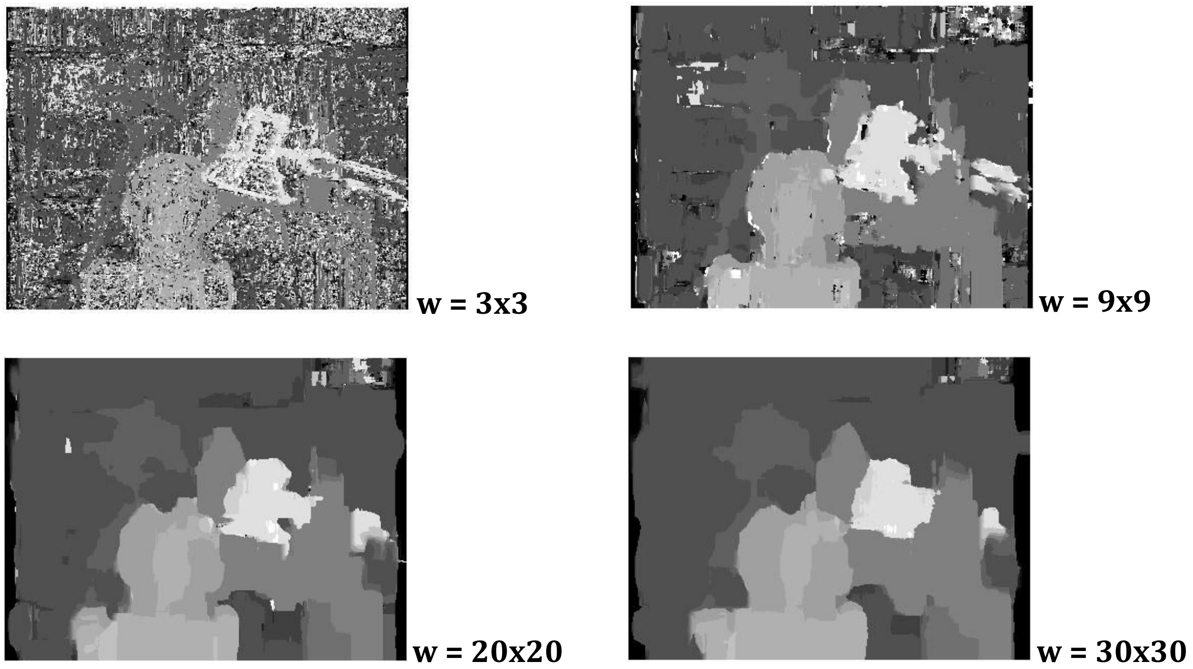


Image 10: Depth map computation with NCC using different window sizes

Similarly to what we've seen with the SSD case, small windows have high sensitivity to noise and keep finer detail while larger windows are more robust to noise while losing the detail. As in the previous case, this is an effect of using more samples to for the computation on the larger windows vs using smaller amount of neighboring values for small windows.

We may try to qualitatively see some differences between the local methods SSD and NCC evaluated on the same images with the same window sizes. The with the smallest window used (3x3) NCC seems to be much more sensitive to noise compared to the SSD. So if we want to keep detail we might favor the SSD in this case. For large windows the results are quite similar in both cases.

5. Depth map computation with local methods using facade images

Here we will evaluate the same local methods (SSD, NCC) previously tested on Middlebury images on the left-right stereo rectified castle facade images shown below.

Image 11: Left and right stereo rectified facade images used to obtain the depth map

First, let's evaluate the SSD method using windows of different sizes of 3x3, 9x9, 20x20, 30x30 on the facade images.

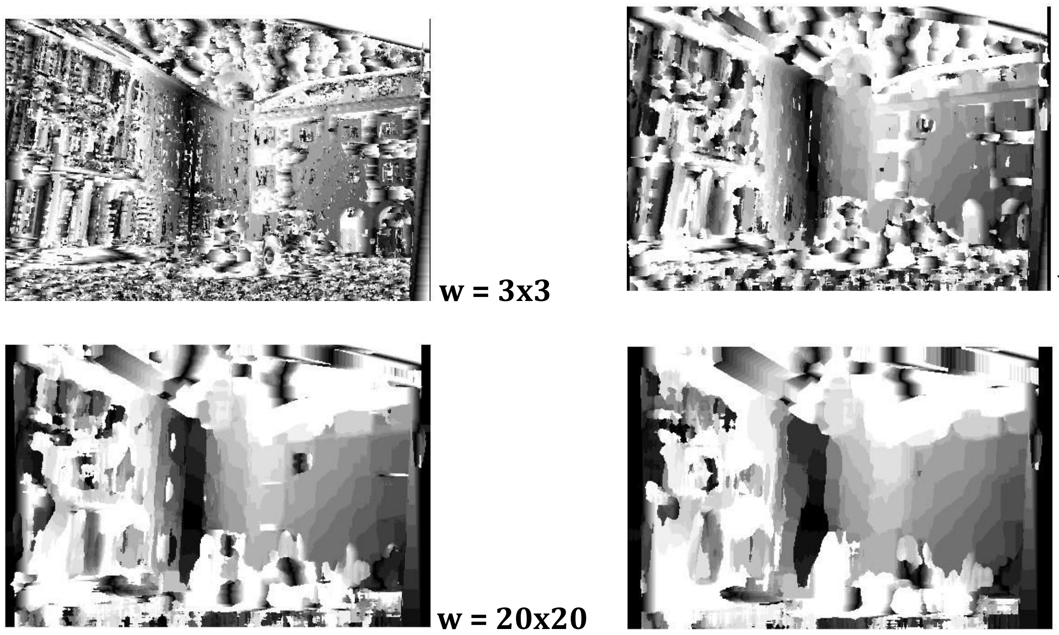


Image 12: Depth map computation with SSD using different window sizes

Now, let's evaluate the NCC method using windows of different sizes of 3x3, 9x9, 20x20, 30x30 on the facade images.

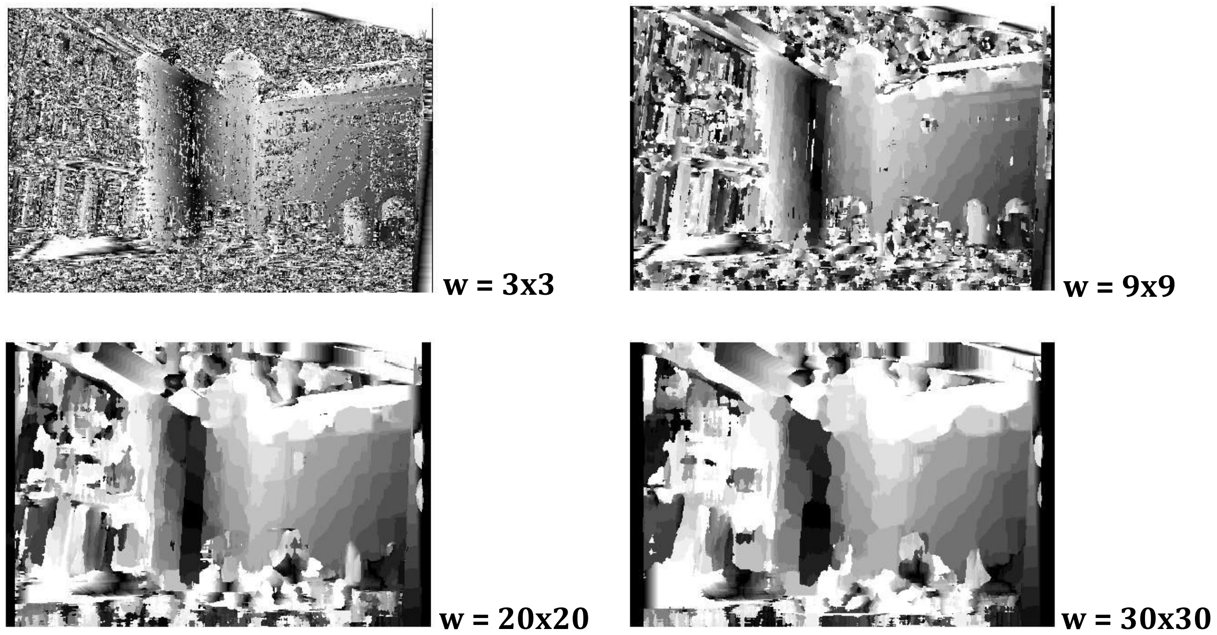


Image 13: Depth map computation with NCC using different window sizes

As expected, the same effects we appreciated when evaluating the Middlebury images are present here. Window size show the same effects we saw. Also NCC shows a higher amount of noise than SSD for smaller windows, which we also saw when using the previous image pair.

Comparing the depth maps obtained in Middlebury images vs Facade images, one can see how better results seems to be obtained with the Middlebury pair. One reasoning could be the fact that most of facade image has most objects “far” away from the camera, resulting in very similar disparities between images, which makes the noise affect more the results. In Middlebury images, however, we have object much closer than others meaning that they’ll have higher disparities. Another issue with the facade images is that occlusions appear on the truck which may also hinder the performance of the local methods.

6. Bilateral weights

As we did in the previous sections, we have used sliding windows to compute the depth map, but in this case the weights that they are used are the bilateral ones. This kind of windows apply a bilateral filter averaging the weighted pixels from a neighbourhood. The bilateral weights are computed not only taking into account the euclidean distance between the colors of the pixels, but also the spatial difference. Below we can see how the bilateral weights are formulated:

$$w(p,q) = \exp\left(-\left(\frac{\Delta c_{pq}}{\gamma_c} + \frac{\Delta g_{pq}}{\gamma_p}\right)\right).$$

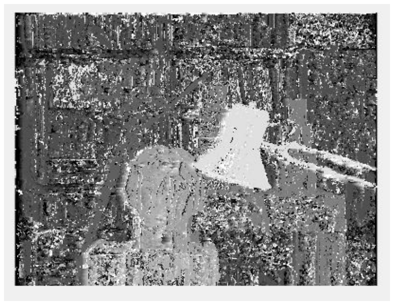
Where ΔCpq is the euclidean distance between the pixels, the color, and ΔGpq is the euclidean distance between the positions, p and q. The gammas are constant parameters, in special gamma c is related to the perception of the color and gamma p is related to the size of the window.The previous weight is computed for the two images, the right and left mentioned above. And the our cost or the dissimilarity is computed as:

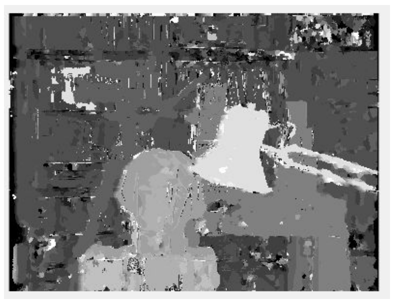
$$E(p,\bar{p}_d) = \frac{\sum_{q\in N_p,\bar{q}_d\in N_{\bar{p}_d}} w(p,q)w(\bar{p}_d,\bar{q}_d)e(q,\bar{q}_d)}{\sum_{q\in N_p,\bar{q}_d\in N_{\bar{p}_d}} w(p,q)w(\bar{p}_d,\bar{q}_d)},$$

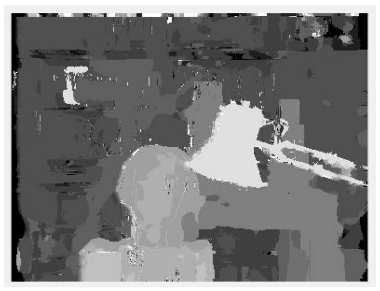
Where e(q,q’) is the truncated AD absolute difference that we can see below:

$$e(q,\bar{q}_d) = \min\left\{\sum_{c\in\{r,g,b\}} |I_c(q) - I_c(\bar{q}_d)|, T\right\}$$

In the next images we can the resulting depth maps:


w = 3x3


w = 9x9


w = 20x20

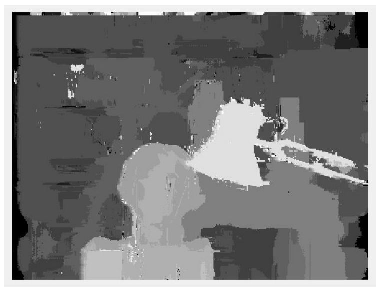

w = 30x30

Image 14: Depth map computation with bilateral weights using different window sizes

After see the results obtained and comparing with the previous costs (SSD and NCC) we can notice that using bilateral weights with small window sizes such as 3x3 or 9x9 we have obtained more noise in our image with respect to SSD and NCC. However, increasing the window size we observe that using bilateral weights we have obtained a better results where the depth map retain more the geometric information and the details of the different objects of the scene.

7. (Optional) New view synthesis

Based on the paper *"Seitz, S. M., & Dyer, C. R. (1996, August). View morphing."* we tried to implement a view morphing algorithm to obtain new views without needing to pre-warp or post-warp, as we're given a pair of stereo rectified images (see below).

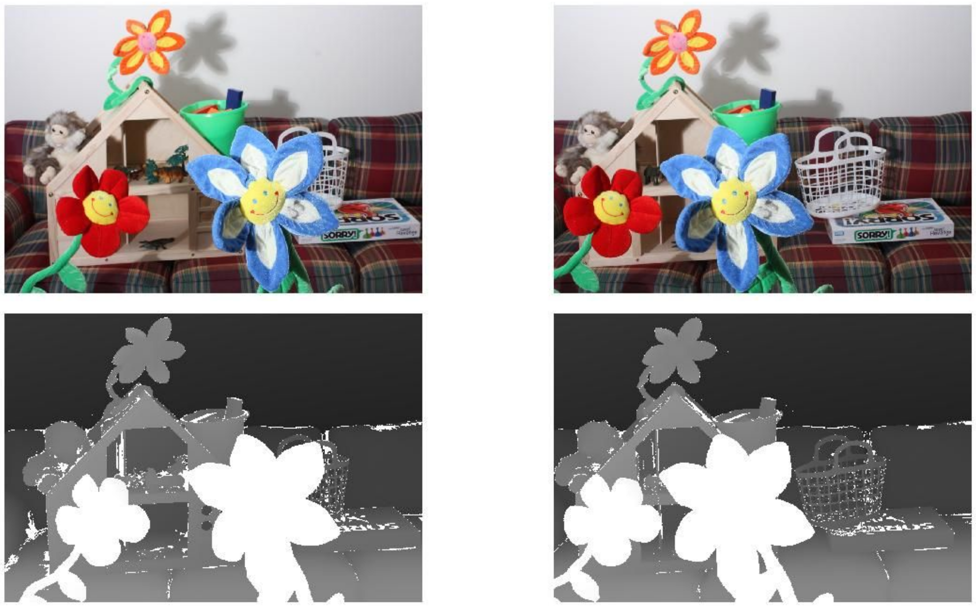


Image 15: Stereo rectified Middlebury images and disparity maps

For every pixel *x, y* of the image, we want to find a new position *p* from the subtraction of the disparity of one of the images to the other which multiplies a factor *s*. As we work with stereo rectified images, we only need to take into account this map into the x dimension. This position *p* will be the correspondent position of a new point which is a morph between the Right ant the Left images. Having values of s = 1 or s = 0 will result in obtaining the other image.

$$\mathbf{p} = (1 - s)\,(x, y) + s\,(x - d_\ell(x, y),\, y)$$

Then, one has to put on that "in-between" position the correspondent intensity values given as a linear interpolation of the image intensities between right and left images such as:

$$I(\mathbf{p}) = (1 - s)\, I_\ell(x, y) + s\, I_r(x - d_\ell(x, y),\, y)$$

Also, occlusions have to be taken care of by checking that:

$$\|d_\ell(x,y) - d_r(x - d_\ell(x,y),\, y)\| \leq \epsilon$$

Here we show the obtained results with an *s* of 0, 0.5 and 1. The results show some evident issues which we didn't have time to properly solve.

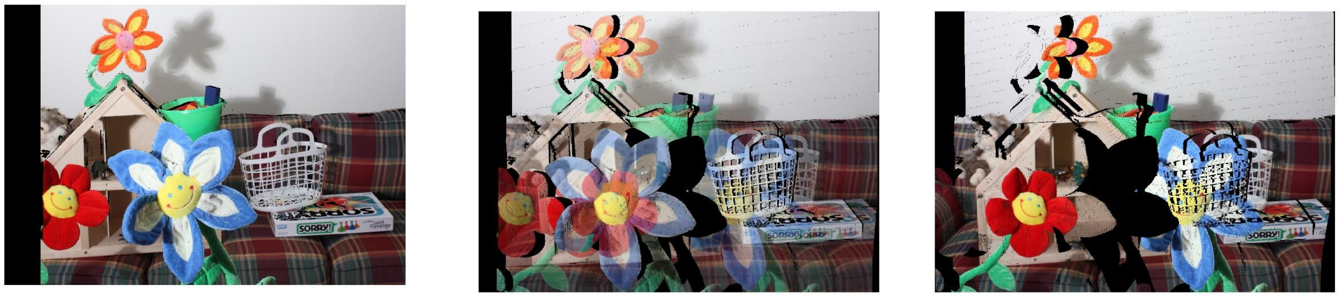


Image 16: View Morphing with *s* = 0, 0.5, 1 respectively

Conclusions:

- The error error obtained by the triangulated points is close to zero which ensures us the implementation seems to properly triangulate.

- We can obtain the second camera candidate visually or mathematically by using triangulation.

- The mean of the reprojection error is also small, so it seems the code works properly.

- Depth map computation gets noisier but more detailed results when using smaller windows and more robust to noise but less detailed for large windows

- NCC seems to be much more sensitive to noise compared to the SSD.

- SSC is better if you want to keep image details.

- Occlusions hinder the performance of the local methods.

- Bilateral weights help keeping the shape of the object allowing to reduce the noise by using larger windows.

- View morphing requires some revising.