M4 - Lab 4: Reconstruction from two views

Sergio Montoya, Laia Albors, Ibrar Malik, Adam Szummer

Pompeu Fabra University
{ sergio.montoyadepaco, laia.albors, ibrar.malik, adam.szummer
}@e-campus.uab.cat

1 Introduction

This week's main goal is to study stereo vision for 3D reconstruction, using two images and cameras with known intrinsics. In section 2 and section 3 we will focus on reconstructing by estimating the fundamental and essential matrices using pairs of correspondances and applying triangulation. On section 4 we will compute disparity maps from pairs of rectified images using local methods.

2 Triangulation

This week first task is triangulation. That is, the process of determining a 3D point given its projections onto two images (knowing the projection matrices of these two images). In particular, we will use the homogeneous algebraic linear method (DLT).

To solve this problem we have the correspondence between the 2D points from the two images $\boldsymbol{x_1} \longleftrightarrow \boldsymbol{x_2}$, and the camera matrices P_1 and P_2. With this information we want to find the corresponding 3D point $\boldsymbol{X}$ such that:

$$\boldsymbol{x_1} \equiv P_1\boldsymbol{X} \qquad\qquad \boldsymbol{x_2} \equiv P_2\boldsymbol{X}$$

First the homogeneous scale factor can be eliminated by a cross product as follows:

$$\boldsymbol{x_1} \equiv P_1\boldsymbol{X} \;\longrightarrow\; \lambda\boldsymbol{x_1} = P_1\boldsymbol{X} \;\longrightarrow\; \lambda\boldsymbol{x_1} \times \boldsymbol{x_1} = \boldsymbol{x_1} \times P_1\boldsymbol{X} \;\longrightarrow\; \boldsymbol{0} = \boldsymbol{x_1} \times P_1\boldsymbol{X}$$

Considering $\boldsymbol{x_1} = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$, $\boldsymbol{x_2} = \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix}$, $P_1 = \begin{pmatrix} \boldsymbol{p_1^{1T}} \\ \boldsymbol{p_1^{2T}} \\ \boldsymbol{p_1^{3T}} \end{pmatrix}$ and $P_2 = \begin{pmatrix} \boldsymbol{p_2^{1T}} \\ \boldsymbol{p_2^{2T}} \\ \boldsymbol{p_2^{3T}} \end{pmatrix}$, the previous expression gives us three equations for each image point:

$$\begin{aligned} eq_1 &= x_1(\boldsymbol{p_1^{3T}}\boldsymbol{X}) - (\boldsymbol{p_1^{1T}}\boldsymbol{X}) = 0 \\ eq_2 &= y_1(\boldsymbol{p_1^{3T}}\boldsymbol{X}) - (\boldsymbol{p_1^{2T}}\boldsymbol{X}) = 0 \\ eq_3 &= x_1(\boldsymbol{p_1^{2T}}\boldsymbol{X}) - y_1(\boldsymbol{p_1^{1T}}\boldsymbol{X}) = 0 \end{aligned}$$

but only two are linearly independent, since $eq_3 = y_1 \cdot eq_2 - x_1 \cdot eq_1$, so we have just two equations per point. Reproducing the same procedure for the point in the second image we can construct the matrix A as follows:

$$A = \begin{pmatrix} x_1\boldsymbol{p_1^{3T}} - \boldsymbol{p_1^{1T}} \\ y_1\boldsymbol{p_1^{3T}} - \boldsymbol{p_1^{2T}} \\ x_2\boldsymbol{p_2^{3T}} - \boldsymbol{p_2^{1T}} \\ y_2\boldsymbol{p_2^{3T}} - \boldsymbol{p_2^{2T}} \end{pmatrix}$$

such that $A\boldsymbol{X} = \boldsymbol{0}$. But this would only be the case in the ideal case. Instead, we can find the $\boldsymbol{X}$ that minimizes $\|A\boldsymbol{X}\|_2$. To avoid the trivial solution, we also have to impose that $\|\boldsymbol{X}\|_2 = 1$. This is equivalent to computing the singular value decomposition of $A = USV^T$ and taking as $\boldsymbol{X}$ the last column of V (the singular vector associated to the minimum singular value of A).

2.1 Normalization

Unfortunately, the DLT algorithm is not invariant to similarity transformations, so a normalization transformation to the data needs to be applied before applying DLT. This normalizing transformation will nullify the effect of the arbitrary selection of origin and scale in the coordinate frame of the image. In particular, the scaling and translation so that both pixel coordinates are in the interval $[-1, 1]$ is applied with matrix H:

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

where $nx \times ny$ is the size of the images.

So, instead of using the system of equations $A\boldsymbol{X} = \mathbf{0}$, we will use the normalized version $\hat{A}\boldsymbol{X} = \mathbf{0}$:

$$\left.\begin{array}{ll} \underbrace{H\boldsymbol{x_1}}_{\hat{\boldsymbol{x}}_1} = \underbrace{HP_1}_{\hat{P}_1}\boldsymbol{X} \longrightarrow & \hat{\boldsymbol{x}}_1 = \hat{P}_1\boldsymbol{X} \\ \underbrace{H\boldsymbol{x_2}}_{\hat{\boldsymbol{x}}_2} = \underbrace{HP_2}_{\hat{P}_2}\boldsymbol{X} \longrightarrow & \hat{\boldsymbol{x}}_2 = \hat{P}_2\boldsymbol{X} \end{array}\right\} \longrightarrow \hat{A}\boldsymbol{X} = \mathbf{0}$$

Since $\boldsymbol{X}$ does not change, there is no need to de-normalized the results afterwards.

2.2 Testing the triangulation

We have used the provided code to validate that the function `triangulate` that we have implemented works properly. In particular, we have tested the `triangulation` function with two Cameras and some random points. With it we get an average triangulation error of $1.62 \cdot 10^{-15}$, so the estimated 3D points are practically in the same position as the real 3D points, as we can see in Figure 1.

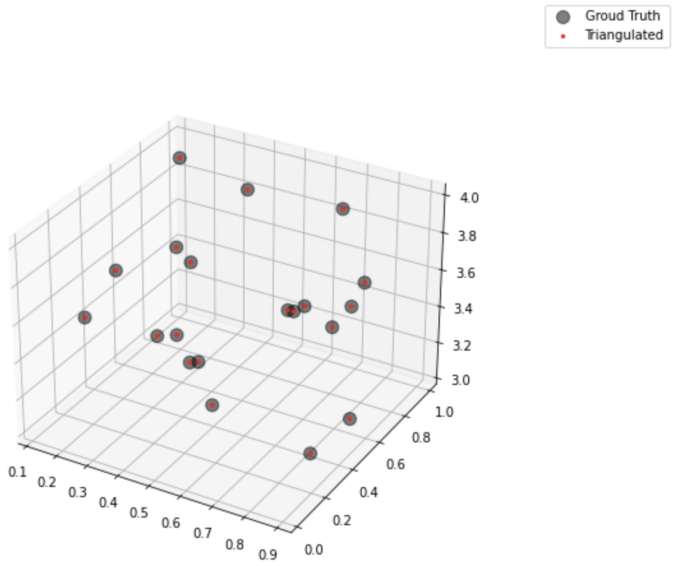


Fig. 1: Visualization of the 3D points: ground truth and estimated with our `triangulation` function.

3 Reconstruction from two views

In this section our goal is to estimate the 3D reconstruction from two views where the image correspondences contain outliers (which is the usual case). To do this we first have to find point correspondences between the two images. In our case, this is done using the ORB local feature detector.

3.1 Estimation of the Fundamental and Essential Matrix

With the found matches we can estimate the Fundamental Matrix that relates the two images. For this we can use the 8-point algorithm. The problem is that this matches are very noise. Therefore, to robustly estimate the Fundamental Matrix, F, the 8-point algorithm can be combined with the RANSAC algorithm, which estimating F by finding a set of *inliers*.

Once we have the Fundamental Matrix we can estimate the Essential Matrix. To do this we can use the mathematical expression that relates these two matricies, which is:

$$E = (K')^T F K$$

where K and K' are the calibration matrices of the two cameras. In our case we assume that both images have been taken with the same camera so we only have one calibration matrix. Therefore, the Essential Matrix can be estimated as $E = K^T F K$.

3.2 Estimation of the Camera Matrices

Next step is to estimate the two camera matrices from the Essential Matrix. For the first one as we have freedom to choose any coordinate system we will just use a $[I|\mathbf{0}]$ canonical camera matrix. For the second camera we will make use of the E matrix. On Equation 1 we can see the equation we have minimized to get the fundamental, and thus essential matrix.

$$x' F x = x'(K')^{-T} E K^{-1} x = x'(K')^{-T}[t'_\times] R K^{-1} x = 0 \tag{1}$$

The decomposition $E = [t_\times]R$ that we have seen in class can be obtained by factorizing E into a skew-symmetric matrix $S = [t_\times]$ which determines t up to scale, and an orthogonal matrix R.

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad Z = \text{diag}(1, 1, 0) W \tag{2}$$

Using two matrices W and Z in equation Equation 2, and the fact that skew-symmetric matrices can be decomposed (up to scale) as $S = UZU^T$ where U is orthogonal, we will factorize E as follows:

$$E = [t_\times]R = UZU^T R = U\text{diag}(1, 1, 0) W U^T R \tag{3}$$

We can see how this is basically the SVD decomposition of E, as W, U^T and R are all orthogonal and they form V^T. To get the rotation we compute the SVD and get two rotations as in Equation 4, because $W^T W = W W = I$.

$$R_1 = U W^T V^T \qquad R_2 = U W V^T \tag{4}$$

Note that we can only assure R being orthogonal, because in Equation 3 the skew-symmetric decomposition is up to scale. Rotation matrices are actually in the special orthogonal $SO(3)$ set. The extra constraint these matrices add is that they cannot produce reflections: a camera will never flip its image between shots. For $R \in SO(3)$, we have both $RR^T = I$ (orthogonality) and also $\det(R) = 1$ (no reflections). If we get a *bad* rotation, we will just flip it with $-R$.

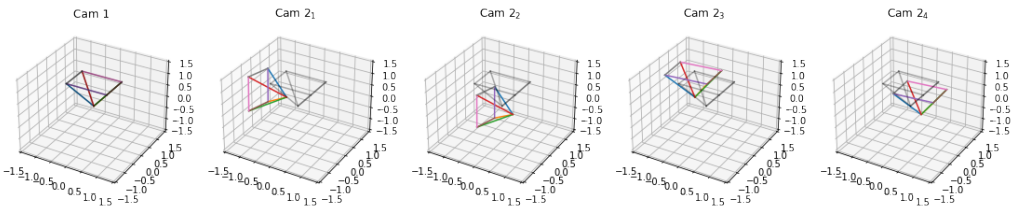


Fig. 2: Candidate cameras for a stereo scene.

To get the translation, we can use the fact that the self cross-product is $St = 0$. Setting $|t| = 1$, we can find a least squares solution minimizing $|St|$ by simply doing the SVD decomposition of S and using the last column of U, u_3 as the solution. As we can see in Equation 3, $S = U\text{diag}(1, 1, 0)WU^T$ and thus it shares the U matrix from the essential matrix. Note that as $|t| = 1$, we have two possible solutions t and $-t$. Combined with the two rotation matrices, we have up to four candidate camera matrices. We can see an example on Figure 2

To find the correct matrix, we simply change the coordinates of a 3D correspondence into the camera coordinates for both views, and if this point has a positive z for both cameras we have a valid candidate. An explanation of why this works is given at Hartley & Zisserman 9.6.3, where they derive the relative rotation of both cameras.

3.3 Computation of the reprojection error

The reprojection error can be computed by *projecting* triangulated points back into the image plane, with the use of the retrieved camera/projection matrix P. If the matrices are not normalized (we apply K to the points), then the axis units should correspond to *pixel* lengths. The reprojection error will correspond to the euclidean difference of the projected triangulated points, at every image.

We have to keep in mind that we are working with noisy correspondences, and we might never get a zero reprojection error, e.g. maybe an Essential matrix fullfilling the epipolar contraint for all points does not exist. On Figure 3 we can see the reprojection error for the estimated cameras: although the error concentrates on the lower values it's still non-zero.

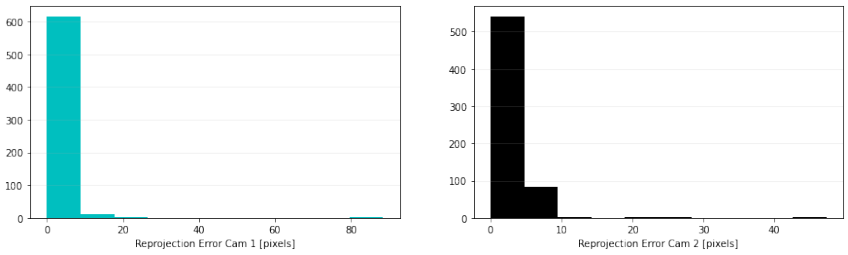


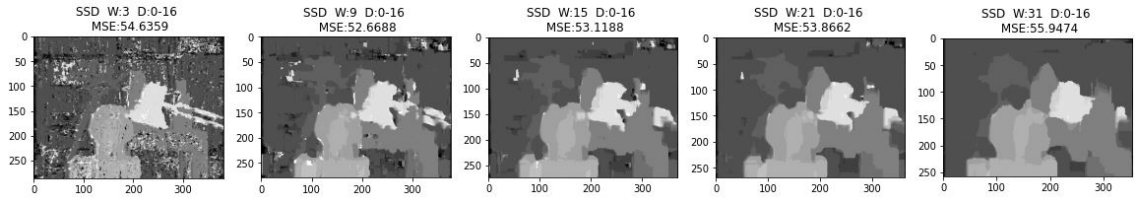
Fig. 3: Reprojection error for two views of the same scene.

4 Depth map computation - Stereo Matching

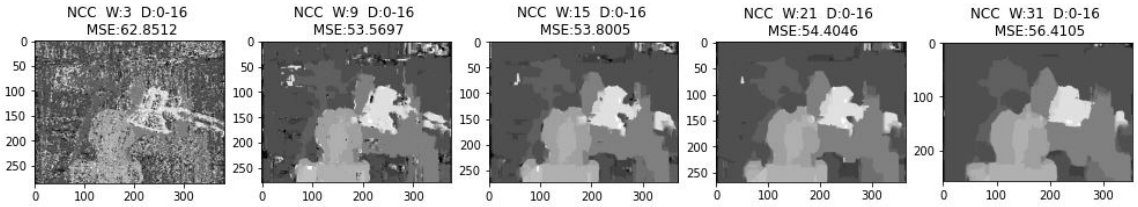
In this section, we compute disparity maps from pairs of rectified images with local methods. For rectified images, given a point x in the first image I_1, the epipolar line l' in I' with respect to a point x is parallel to the x axis, therefore the correspondence must be at the same height in the second image. This eases the correspondence search, as we must search the correspondence in the same row in the second image. For each pixel from the first image, we must search its corresponding point in the second image by applying a sliding window technique with a local metric. The search in the second image will be done at the same row by comparing the content to that of the reference window. Then, after comparing the reference patch to all possible patches delimited by the minimum and maximum disparities, we pick the patch with the minimum matching cost (WTA, Winner-Takes-All). The local matching cost between a reference patch with central pixel p in the first image and a target patch p' in the second image can be defined as:

$$E(p, q) = \frac{\sum_{u \in N_p, t \in N_q} comb(w(p, u), w(q, t))e(u, t)}{\sum_{u \in N_p, t \in N_q} comb(w(p, u), w(q, t))} \tag{5}$$

where $comb$ defines how the weight matrix from the target patch and reference patch are combined, and N_p and N_q defines the neighbors of p and q, respectively. e defines a difference metric like the Sum of Squared Differences (SSD) or the Normalized Cross Correlation (NCC).



(a) Sum of Squared Differences in the Tsukuba pair of images.



(b) Normalized Cross Correlation in the Tsukuba pair of images.

Fig. 4: Comparison of SSD and NCC as a local matching method. W denotes the window size, D denotes the minimum and maximum disparity, and MSE denotes the Mean Squared Error between the estimation and groundtruth disparity map.

The difference in the x coordinate between corresponding points in a pair of rectified images is defined as the disparity. As we have rectified images, we have a parallel camera configuration in which the depth Z of a point X that projects at points p and p' can be defined as:

$$Z = f\frac{T}{p_x - p'_x} \tag{6}$$

where f is the focal length, T is the baseline, and p_x and p'_x is the x coordinate of points p and p', respectively.

In subsection 4.1, we explore the sum of squared differences as a local matching cost, and in subsection 4.2, we explore the use of the normalized cross correlation. In subsection 4.3, we explore the use of bilateral and exponential support weights to improve the disparity map estimation.

4.1 SSD: Sum of squared differences

The first local matching cost that we implement is the SSD. The squared differences are multiplied by a weight matrix that for now will give the same importance to every pixel in the patch. For the first two local matching costs (SSD and NCC), we use the average weight matrix. The SSD matching cost is defined as:

$$e_{SSD}(u,t) = (I(u) - I'(t))^2 \tag{7}$$

where I is the first image and I' is the second image. We perform the experiments with the Tsukuba pair of images from the 2001 Middlebury stereo dataset (3). As denoted by the authors, the disparity groundtruth map contains the disparity values multiplied by 8. We set the minimum and maximum disparity to 0 and 16, respectively. We decide to evaluate the MSE with respect to the groundtruth disparity map only in the image part where the disparity is computed (i.e. the disparity values are not computed where the window size does not fit the image size, therefore a greater window size will result in less disparity values in the borders of the image). The results are provided in Figure 4a. As we can see in the figure, the window size of 3x3 computes better disparities at small parts or detailed parts (e.g., the body of the lamp), while it incorporates a big amount of noise, resulting in a noisy solution. A bigger window size produces smoother results with more spatial coherence, but does not compute the disparity of small parts correctly.

We also define a vectorized version for computing the SSD (in the function *stereo_computation_vectorized* and *compute_disparity_at_i_vectorized*). This vectorized version takes advantage of vectorized operations with numpy. Specifically, for each row, we create a reference tensor of shape (WxDxB), where W is the number of columns at which we want to compute the disparity at that row, D is the number of blocks in the target image to which we want to compare each column in the reference image, and B is the block size (window size squared). We also create a target tensor of shape (Wx1xB) containing the blocks to which we want to compare the reference block. As a result of computing the SSD in a vectorized way, we obtain a result tensor of shape (WxD) which contains the SSD measure of each column to each possible position delimited by the minimum and maximum disparities. Then, all that is left is to compute the argmin for each column. We observe that in a computer with an AMD Ryzen 7 3700X 8-Core Processor this vectorized approach takes 30 seconds in average, while the previous approach takes 1 minute. On a computer with more limited resources, with an AMD EPYC 7571 processor, we see the advantage of using this vectorized approach, which takes around 50 seconds in comparison to the previous approach that takes around 5 minutes. We only implemented SSD in a vectorized way, as it was harder to implement and we did have enough computational resources.

4.2 NCC: Normalized cross correlation

The SSD matching method is not robust against different illumination conditions, which could yield to a high cost due to different illumination of the scene. This could lead to many false matches and bad disparity map estimation. The Normalized Cross Correlation helps alleviating this effect because it normalizes each vector before computing the correlation of values. It is defined as:

$$E(p,q) = \frac{\sum_{u \in N_p, t \in N_q} comb(w(p,u), w(q,t))(I(u) - \bar{I}(N_p))(I'(t) - \bar{I}(N_q)))}{\sum_{u \in N_p, t \in N_q} comb(w(p,u), w(q,t))\sigma(I(N_p))\sigma(I(N_q))} \tag{8}$$

where the mean of patches are denoted by $\bar{I}(N_p)$ and $\bar{I}(N_q)$, and are computed on the neighborhood of p and q, respectively. $\sigma(I(N_p))$ and $\sigma(I(N_q))$ is the standard deviation of the patches. The results are provided in Figure 4b. We can see how we achieve similar results to SSD. NCC at small window sizes seems very sensitive to noise and at other different sizes the results are comparable, achieving SSD lower MSE. In Figure 5, we compute the disparity maps for another set of images. In this case, we can see that the object with biggest disparity (closest to the camera) is the truck, which moves around 100 pixels. Therefore, we decide to set the maximum disparity to 100. As with the previous set of images, we can see that when using NCC and small window size, the result is very noisy. Small window sizes produce more detailed results, while bigger sizes produce spatially smooth results. When using NCC, we have noticed that the sky disparity is not very smooth, contrary to SSD where at big window sizes, the sky disparity is considerably smooth. This can be due to the fact that we are working with grayscale images and the mean of the patch is substracted in the NCC, it can be easily confused with other smooth areas with different gray intensity level. Also, searching correspondences in the sky is hard as it has no distinctive points to match. Also, in the windows of facades can have different reflections depending on the camera position, hence being hard to compute the disparity in these places.

4.3 Adaptive support weights

In this section, we explore two types of adaptive support weights, exponential weights (5) and bilateral weights (2). They are very similar but have a few differences. Instead of giving the same importance to each pixel like in previous sections, now we give importance to pixels depending on the similarity in position and color to the central pixel of the window. The exponential weights computes the color weighting term by computing the color difference in the CIELab color space. While the bilateral weights compute the color weighting term by comparing pixels in RGB space and measuring the L1 distance. The adaptive support weights can be estimated as:

$$w(p,q) = w_{color}(p,q) \cdot w_{position}(p,q) = exp(-(\frac{\Delta c_{pq}}{\gamma_{col}} + \frac{\Delta g_{pq}}{\gamma_{pos}})) \tag{9}$$

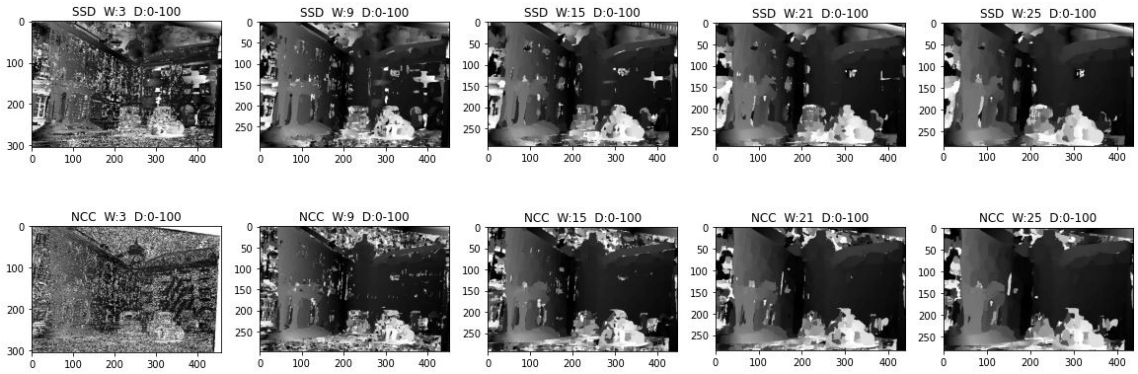


Fig. 5: Comparison of SSD and NCC in the pair of images of facades.

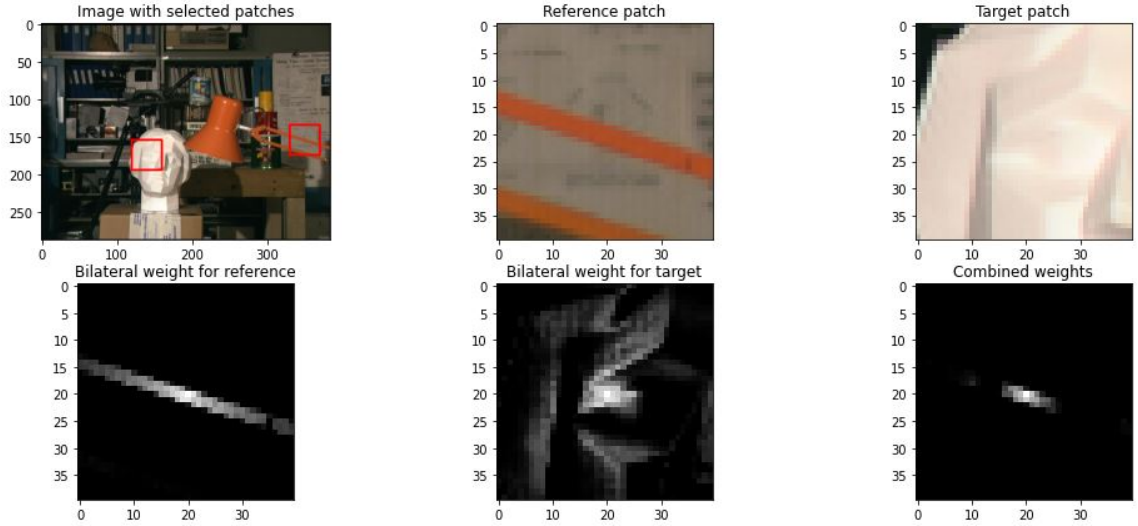


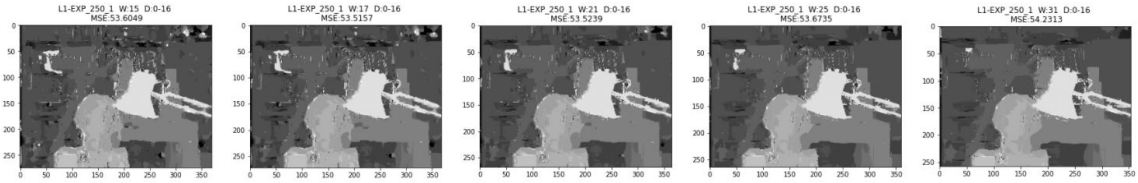
Fig. 6: Example of adaptive weights given a reference and target patch. Reference and target adaptive support weights are combined in a multiplicative way.

where Δc_{pq} denotes the difference in color and Δg_{pq} denotes the distance from the central pixel. γ_{col} and γ_{pos} are hyperparameters that expresses the priority given to color and position similarity. An example of computed adaptive weights are provided in Figure 6. When considering only the reference support window, the computed dissimilarity can be erroneous when the target support window has pixels from different depths. To minimize the effect of such pixels, the support adaptive weights can be combined. The combined support weights favor the points likely to have similar disparities with the centered pixels in both images. We set γ_{col} to 12 and γ_{pos} to $window\_size/2$ in our first experiments.

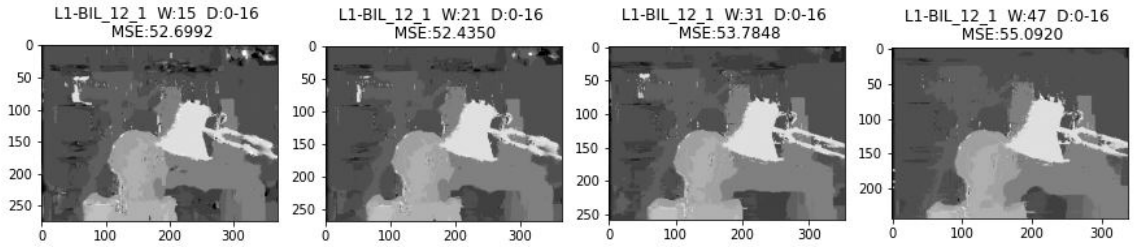
The authors of exponential and bilateral weights use L1 differences for the local matching methods combined with the adaptive support weights. For grayscale images we can also use the L1 and L2 distances for the matching methods. In Figure 7, we have selected the best results with different parameters using the bilateral and adaptive weights. We can see that bilateral weights have achieved lower MSE than exponential weights in our experiments. What is more, as mentioned by the authors of bilateral weights, bilateral weights recover depth discontinuities better (as we can observe in the books on the shelf).

By increasing γ_{col}, we increase the importance of how close the pixel is to the center pixel, and when increasing γ_{pos}, we increase the importance of the similarity in color. In Figure 8, we can see how as we increase the γ_{col} parameter, the spatial smoothness increases as the position weight gets more importance.

As the disparity value of the grountruth image has a maximum value of disparity of 28, we try to set the maximum disparity to 28. We do not obtain better results by changing the maximum disparity.



(a) Exponential weights with L1 matching cost.



(b) Bilateral weights with L1 matching cost.

Fig. 7: Comparison of exponential weights and bilateral weights with L1 as a local matching method. W denotes the window size, D denotes the minimum and maximum disparity, and MSE denotes the Mean Squared Error between the estimation and groundtruth disparity map.

5 Optional: Depth computation with Plane Sweeping

The plane sweeping algorithm consists of projecting the image plane at different depths and backprojecting it into the second image. When the depth of one pixel is matched with the depth of the parallel plane, the pixel will be correctly projected into its corresponding position in the second image. We can see if the pixel depth is correct by applying a local matching method like in the previous sections. We only explore the use of SSD and NCC. The depth of a point X is the 3rd coordinate of the point when expressed in the camera frame:

$$d(x) = (PX)_3 = p_3^T X$$

Then, the fronto-parallel plane at depth d to the first camera P is:

$$\sqcap = p_3^T - (0, 0, 0, d)$$

Let x be a pixel of the first image in homogeneous coordinates. We want to back-project the pixel into the 3D plane $\sqcap$ and reproject it to the second camera P', then:

$$\begin{bmatrix} \sqcap \\ P \end{bmatrix} X = \begin{bmatrix} x \\ 0 \end{bmatrix}$$

Then, we have that the projection x' into the second image is:

$$x' = P' \begin{bmatrix} \sqcap \\ P \end{bmatrix}^{-1} \begin{bmatrix} x \\ 0 \end{bmatrix} \tag{10}$$

As we have a stereo rectified setting, the things we need to know according to Equation 10 are the intrinsic camera parameters and the baseline, as with these parameters we can determine P and P'. Once we obtain the depth map, we can convert it into a disparity map by applying Equation 6. We experiment with the motorbike pair of images from the 2014 Middlebury set of images as the Tsukuba pair of images does not have calibration data. We test different depths from the range of 3000mm to 300000mm, the results can be observed in Figure 9. We observe that for this set of images, we obtain worse MSE than using methods from previous sections.

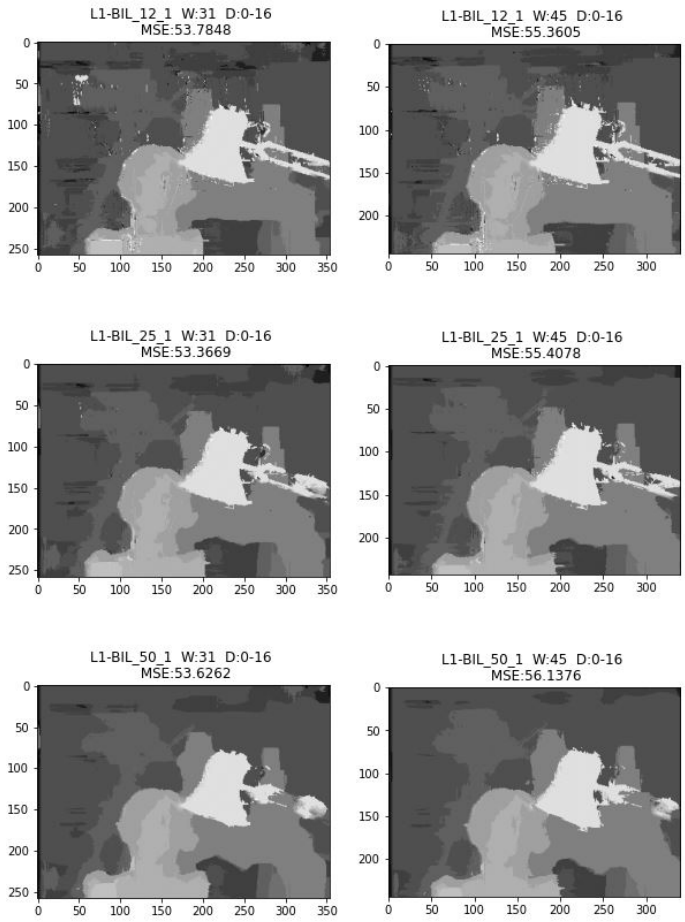


Fig. 8: Comparison of results of using different parameters in the bilateral weights. All experiments have γ_{pos} constant, while setting γ_{col} to 12, 25, and 50, in the first, second, and third rows, respectively.

In the case of having to apply the plane sweeping algorithm to non-rectified images, the only thing that would change is that we also would need to know the rotation of the second camera with respect to the first one and the translation (extrinsic parameters). And depending on the camera setup, it might be difficult to apply this algorithm. For example, if we backproject into the fronto-parallel plane of the first image, if we rotate the second camera by 90º, the plane would project into a line. Also computing the disparity in this case does not make sense, as it only makes sense in the rectified case.

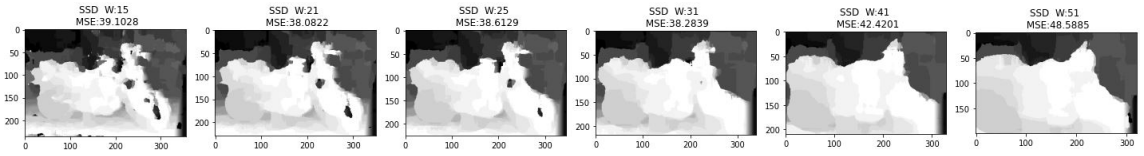
6 Optional: New view synthesis

In this section we will talk about new view synthesis. To go upfront we need to admit that it seems as we did not complete fully this optional. However we still want to share some insights of our founding.

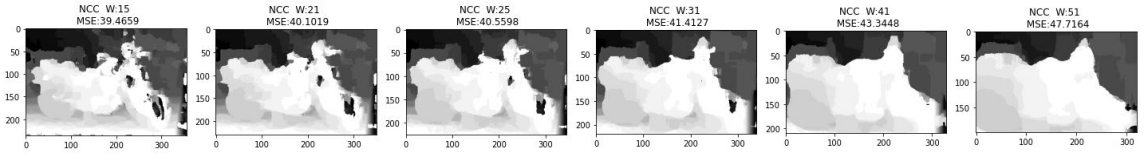
As per understanding of the class and the paper there are several ways to create new view synthesis, including MBR(Model based rendering), or IBR(Image base rendering) in the other words estimation of camera and projective matrices or using correspondences. We have used disparity maps as a foundation for the new view synthesis. Using the equation

$$p = (1-s)x_0 + s * x_1 = (1-s)1/ZP_0X + s1/ZP_1X = 1/ZPsX \tag{11}$$

We have achieved following result for s=0.5 presented in figure 10. As we can see on the disparity map created for the new view taking the half values of positions of the pixels between image 1 and image 2, kind



(a) Sum of Squared Differences in the motorbike pair of images.



(b) Normalized Cross Correlation in the motorbike pair of images.

Fig. 9: Comparison of SSD and NCC as a local matching method when computing depth maps using the plane sweeping algorithm. W denotes the window size, D denotes the minimum and maximum disparity, and MSE denotes the Mean Squared Error between the estimation and groundtruth disparity map.

of shadows are created indicated different view point creation on figure 10a. However when image view is considered we can see discrepancies in the representation which source we were unable to find. In addition to the mentioned equation, solution for occlusion has been developed that considered the disparity value on each pixel. If disparity on particular pixel was higher than the previous data, due to the fact that objects closer to the objects move more (higher disparity) those pixels have been overwritten, nevertheless this did not solve the existing issue.

7 Optional: Depth map fusion

In this section we will analyse and compare the methods introduced by Levoy (1) and Weder (4) for depth map fusion. Both methods take advantage of the interpretation of depth maps as *view-dependant* or projected SDFs, as the depth map gives us the (signed) distance along some direction up to the surface. Levoy et. al. notice that we can simply average the values of these projected SDFs to compute a global SDF function, from which we can extract the surface.

But this averaging cannot be done lightly. Levoy et. al. use *weights* to encode the certainity of the depth maps, e.g. if the geometry normals and the sensor view direction differ by much we might give less importance to those samples. These weights can come from the sensor directly, and we use them to perform a weighted average. Moreover, using the whole range information when averaging these SDFs is not ideal, as we can influence obstructed surfaces, high frequency surfaces, and more. This is why they propose to truncate the SDFs that we are merging near the surface, and with this we also gain some extra speed by skipping many computations.

To work with these continuous SDFs Levoy et. al. use a voxelized grid, on which they keep track of both SDF values and the confidence weights of every view. Iteratively we update this global voxel grid by casting rays from the depth maps or projected SDFs.

Weder et. al. on their Routed Fusion paper propose an improvement over this by use of neural networks. They say that the noise on the depth maps is non-Gaussian and depth-dependant, and thus a linear/weighted method to compute the new SDF values is not ideal. They propose, with the help of neural networks, a non-linear method.

They also use a global voxel grid to store the values, but at each fusion step they align the grid with the image by means of interpolation. This is necessary as they want to incorporate image information into the grid: a neural network is used to compute confidence values and also *clean* the depth maps from outliers. Then, we can concatenate the output features of this network with the grid itself. After computing the updates we will interpolate the grid back to its global alignment.

(a) Disparity representation in a new view

(b) Grayscale-view created for s=0.5.

Fig. 10: New synthesis view generation

How do we compute these updates? Instead of doing weighted averages, Routed Fusion propose using a neural network. This network uses both confidence weights, the cleaned depth map, and the weights and SDF values from the aligned grid to compute updates, and unlike Levoy et. al. they do it for regions instead of just individual rays, incorporating information from the neighbours. Note that with Routed fusion we are not performing truncation explicitly, and instead delegate to the neural network the work of outputting zero valued updates when necessary. Turns out that this works well, as the networks in these method are trained on ground truth data.

The main similarities between both methods is that they use SDFs to merge depth maps, and perform updates *locally* in the sense that a depth map from a specific view will only modify the proper voxels. The main difference is in how they incorporate the global SDF updates, such as using neural networks to compute these updates instead of computing the weighted average, and also doing this update considering the neighbourhood or not.

8 Conclusions

In this week's assignment, we have seen how to extract the camera poses from the essential matrix and how to identify the correct candidate solution. In the disparity map computation methods, we have seen the effects of different window sizes and matching costs in the computation of the disparity map. SSD and NCC achieved similar results and using support adaptive weights improved the results. Smaller window sizes computed better disparities at small parts or detailed parts, while it incorporates a big amount of noise, resulting in a noisy solution. A bigger window size produces smoother results with more spatial coherence, but does not compute the disparity of small parts correctly. The plane sweeping algorithm did not provide better results than the stereo matching method and it can be computationally expensive depending on the

amount of depths to which we project the scene. But a good point of the plane sweeping algorithm is that it can compute depth maps in non-rectified stereo settings. Lastly, we have learned about depth map fusion methods, both more classical and seminal works and recent learning-based methods.

Bibliography

[1] Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. p. 303–312. SIGGRAPH '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237170.237269, https://doi.org/10.1145/237170.237269

[2] Julià, L.F., Monasse, P.: Bilaterally weighted patches for disparity map computation. Image Process. Line **5**, 73–89 (2015)

[3] Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International journal of computer vision **47**(1), 7–42 (2002)

[4] Weder, S., Schönberger, J.L., Pollefeys, M., Oswald, M.R.: Routedfusion: Learning real-time depth map fusion. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)

[5] Yoon, K.J., Kweon, I.S.: Adaptive support-weight approach for correspondence search. IEEE transactions on pattern analysis and machine intelligence **28**(4), 650–656 (2006)