Stereo Matching Project

Description of the project:

The aim of this project is to compute the disparity maps associated to pairs of already rectified stereo images. In order to do so, we firstly have to find the corresponding points between the two images along the epipolar lines and then assign a disparity value to the matches.

As a standard stereo geometry is assumed, the searching is performed through horizontal colinear epipolar lines. Since disparity is inversely proportional to depth, we expect to get nearer objects that are nearer to the camera, lighter than the farther ones.

The dataset provided comprehend four images named *Map*, *Sawtooth*, *Tsukuba* and *Venus*, with their respective parameters, which are: the disparity range, the scale factor needed to obtain a coherent map and the width of the image border to be ignored during evaluation.

Map

Sawtooth

Tsukuba

Venus

Base algorithm implementation:

To pursue the stereo matching problem, a local, area-based algorithm employing the SAD (Sum of Absolute Difference) has been used. The disparity is calculated by determining a measure of dissimilarity between the pixels within a window in the two images. Thus, stereo matching can be performed as a one-dimensional template matching.

Following the convention, we chose to use as reference image the left one (*imageL*). Then, for each point in the reference image, the supporting window (*K*x*K* sized) fixed and centred at the point is shifted horizontally and compared to all those centred at the pixels of the test image (*imageR*), laying on the same row and within the disparity range ([*disp_min*, *disp_max*]).

```
def AD_left (imgL, imgR, i, j, disp, m, n):
    S=np.abs(int(imgL[i+m, j+n])-int(imgR[i+m, j+n+disp]))
    return S
```

In the right image we add the disparity in order to recover the displacement between corresponding image points in a stereo system, following the relationship: $u_l - u_r = d$

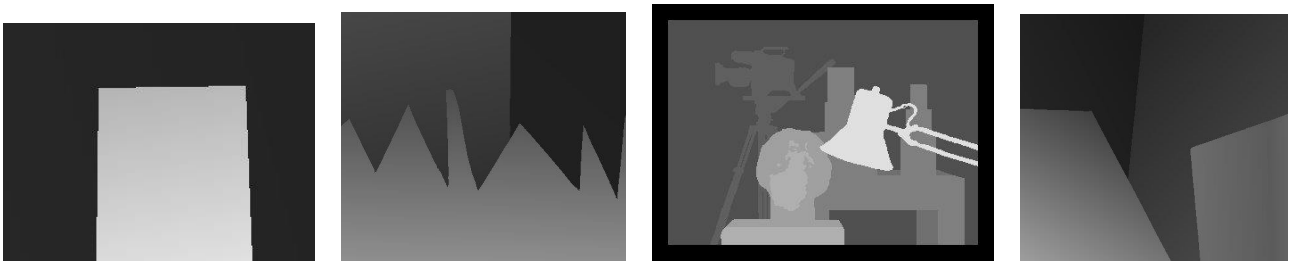
In the following piece of code, the ranges for the iterations have been chosen in order to not overflow.

The algorithm provides us with the lowest SAD within the disparity range according to the winner-takes-all approach. In fact, being the SAD a dissimilarity function, the lower its score, the higher the similarity is. Taking into account these considerations, we obtain our disparity map.

```
#We select the pixel and move along its row
for i in range (K, img_raw-K-ignore_border):
    for j in range (K+disp_max, img_column-K-ignore_border-disp_max):
                min_SAD=500000
        #moving along the epipolar line
        for d in range (disp_min, disp_max+1):
            SAD=0
            for m in range (-K, K+1):
                for n in range (-K, K+1):
                    SAD=SAD+AD_left(imageL, imageR, i, j, d, m, n)
            if (SAD<=min_SAD):
                min_SAD=SAD
                disp=d
        #saving the disparity value associated with the lowest SAD found
        MapL[i,j]=np.asarray([disp*disp_scale])
```

Evaluation of the results:

In local algorithms, the selection of window size plays a major role in determining the quality of the resulting disparity map. This is because a fixed window size does not yield reliable disparity estimates for all the pixels in a stereo pair of images. As a consequence, we tried different sizes seeking for the one that better fits each image. The dataset provides us with the ground-truth disparity map (*target*) (i.e. error-free) which we exploited in order to evaluate quantitatively the precision our resulting map (*img*).



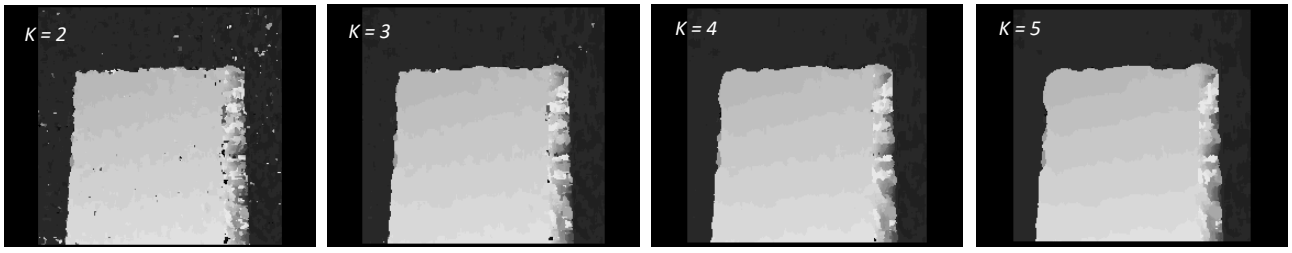
The error has been calculated as follows:

```
def Error (target, img, w, h, m, scale_factor, ig_border, d_max):
    error=0
    window=(w-2*m-ig_border-2*d_max)*(h-2*m-ig_border)
    for j in range (m+d_max, w-m-d_max-ig_border):
        for i in range (m, h-m-ig_border):
            if (abs(target[i,j]/scale_factor - img[i,j]/scale_factor)>1):
                error=error+1
    return (error/window)
```

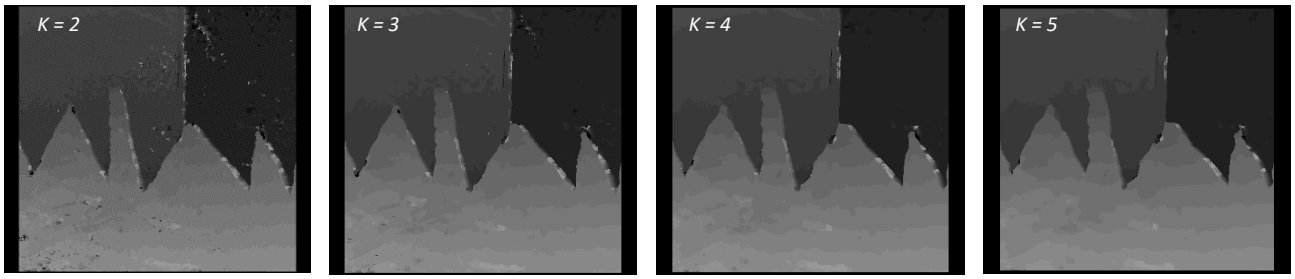
This function increases the error each time the absolute difference between the intensities of corresponding pixels is greater than one and normalizes it over the area on which we have executed the matching. The ground-truth disparity map must be divided by the scale factor to yield to unit gain. Our resulting map, instead, is built already normalized but the division by the scale factor results still necessary since it has been saved in a format coherent with the ground-truth map.

Error

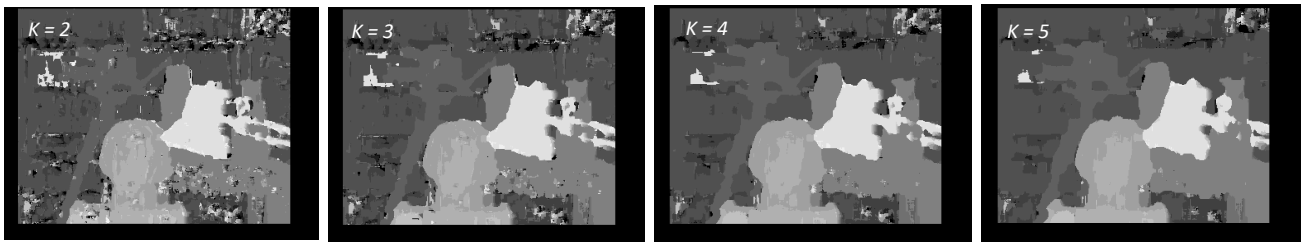
	K = 2	**K = 3**	**K = 4**	**K = 5**
Map	0.09812	0.08958	0.09317	0.09852
Sawtooth	0.08156	0.06856	0.06791	0.07050
Tsukuba	0.19961	0.16302	0.14536	0.13655
Venus	0.15687	0.10340	0.07863	0.06799



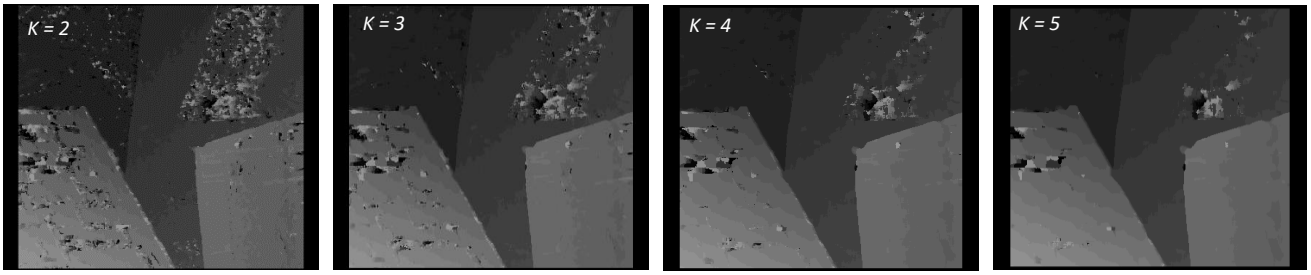
Resulting disparity map of Map for K increasing



Resulting disparity map of Sawtooth for K increasing



Resulting disparity map of Tsukuba for K increasing



Resulting disparity map of Venus for K increasing

It's easy to notice how for *Tsukuba* and *Venus* images, increasing the supporting window on which we rely during the SAD, leads us having lower values of the error. On the contrary, this advantage is lost passing from K=3 to K=4 in the first two images. In fact, reliable disparity estimation requires the window to cover a region of sufficient intensity variations as well as constant disparity. *Map* and *Sawtooth* are images that have large disparity variation between the regions of the image (i.e. objects at very different distances from the camera), so the window should be kept small to avoid smoothed disparity mapping. In the former case instead, since there are regions of uniform intensity, the larger the window is kept, the more noiseless the estimation turns out. However, it's remarkable to underline that increasing the size brings to higher computational costs.

Main issues:

Area-based algorithms, such as the one we used, suffer from different drawbacks.

1. **Variation of depth within the window's area**

   In general, local algorithms are plagued by the problem of blurring of edge boundaries, known in the literature as boundary overreach or border localization. This problem occurs when correlating windows overlap depth discontinuities. In this case, the global minimum returned by the SAD is significantly less sharp and also less distinctive. When the window is located across a depth discontinuity, the points within the window do not maintain their spatial relationship when projected into the right image. Therefore, due to the geometry of the stereo system, when searching along the epipolar line, I may not find a complete match.

2. **Occlusion**

   Images that have large disparity range can feature many points that are occluded in the other image, especially near the edges (i.e. near a depth discontinuity). Finding a match for such points is difficult, so they are often left unmatched.

3. **Fixed size window**

   A central issue in local stereo matching algorithms is the empirical selection of an appropriate window size. In fact, it results in two major problems: a noisy disparity map if the selected window is small, covering a region of insufficient intensity variations (as the signal to noise ratio would be low), and a smoothed disparity map or boundaries if the selected window is too large, covering a region of varying disparities (trade-off). Hence, as the right size depends on the specific point that I'm considering, algorithms that modify the window shape and dimension adaptively, depending on the local intensity and disparity variations, produce better results than the standard ones.

   Another problem related to the use of a fixed-size window is the vanishing of details in the disparity map when they are small compared to the size of the correlation window. In this case the signal strength embodied in the texture of the detail is balanced by the contributions of the other points within

the window. Shrinking the window can mitigate this problem, even if it causes the reduction of the signal-to-noise ratio as a side-effect, leading to more matching errors.

The last aspect to point out is that these algorithms rely on the assumption of frontal-parallel surfaces, so it's presumed that the surfaces within the window are parallel, which isn't always the case.

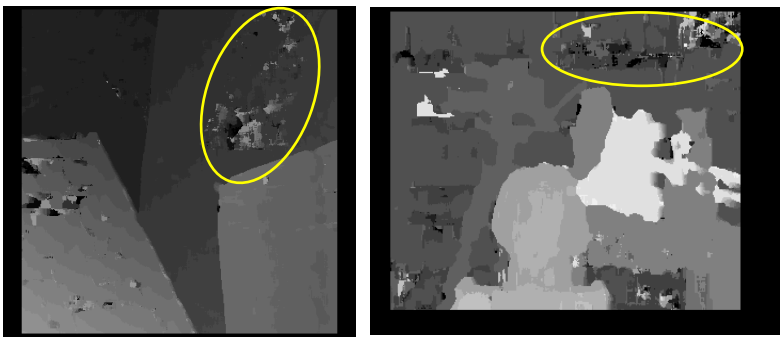
4. **Low textured regions**

   Matching turns out to be highly unreliable when dealing with textureless areas, since the horizontal intensity variation in such regions is very low. On the contrary, in greatly textured regions, the minima provided by the SAD are definitely sharp. This occurs because textured regions are associated to a high signal' strength, that yields higher error scores for non-homologous points. However, this does not guarantee match reliability in presence of repetitive patterns, because the global minimum of the error might be sharp but close to some other local minimum.

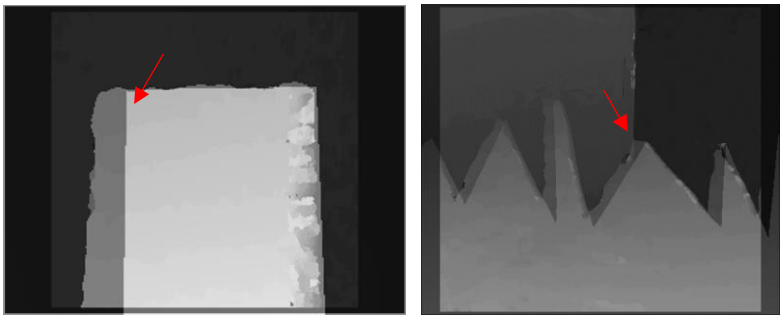
Global algorithms have been developed to deal with the problems associated with local algorithms. They calculate disparity relying on the minimisation of a global cost function. These techniques remove the dependency of the disparity map on the window size and, due to their global support nature, they provide reliable disparity estimates even for regions containing low textured and occluded points. However, their high computational cost limits their application, making it infeasible to exploit them in real-time tasks.

Let's now analyse these issues in our dataset.

The *Tsukuba* stereo pair contains complex objects at different depths, as well as repetitive and poorly textured regions in the background, such as, for example, the wall at the top-right corner. Moreover, this stereo pair contains some specular and non-planar regions, like the face of the statue, and small details, such as the lamp's wire and switch. Also, there are uniform areas in the picture's background, as well as in *Venus*, that lead to noisy disparity map.



Map and *Sawtooth* feature instead simpler planar objects at a larger disparity range, such that larger occlusions are present. However, compared to *Tsukuba*, these stereo pairs contain very few small details. Moreover, *Sawtooth* contains several low textured regions.



Improvement of the baseline algorithm:

In order to try to increase the performance of our baseline algorithm, we introduced several additional tools:

a. Left-right consistency check
b. Rejection of ambiguous minimum
c. Robust similarity function
d. Prefiltering
e. Postfiltering

Anyway, it should be said that the selection of a reliable disparity estimate is a challenging task if no prior knowledge about the nature of the image is provided.

In the following discussions, we have kept *K*=3 for *Map* and *Sawtooth* and *K*=4 for *Tsukuba* and *Venus*, according to the results found before.

a) Left-right consistency check

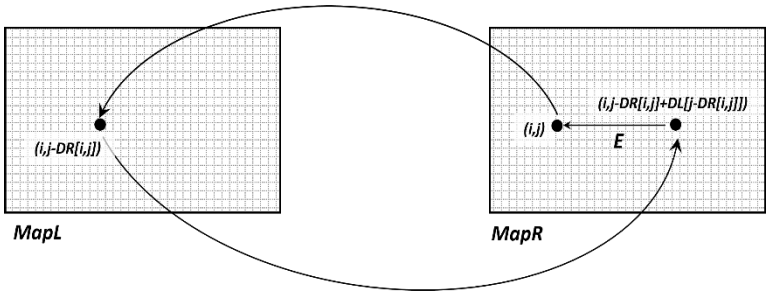
In this step we attempted to detect wrong matches by performing the so-called Bidirectional matching. It creates and compares the two disparity maps relative to each image, applying the standard procedure twice. Initially, for each point in the left image the best match in the right image is sough (direct matching phase, left image taken as reference), then their role is reversed, and for each point in the right image found the correspondence in the left one (reverse matching phase, right image taken as reference). To do so, we have implemented a specular function (*AD_right*) where the coordinates are transformed coherently to the right image. The disadvantage is that the computational time required significantly increases.

```
def AD_right(imgL, imgR, i, j, disp, m, n):
    S=np.abs(int(imgL[i+m, j+n-disp])-int(imgR[i+m, j+n]))
    return S
```

To compare the two disparity maps (*MapL* and *MapR*), we proceeded as follows:

- Select pixel (i,j) from the right image, its disparity value is *MapR[i,j]*
- Its corresponding point in the left image is at position (i,j-*MapR[i,j])* and its disparity value is *MapL[i,j-MapR[i,j]]*
- The matching point in the second image is at position (i, j-*MapR[i,j]*+*MapL[i,j-MapR[i,j]]*)
- Calculate the projection error as $E = MapR[i,j] - MapL\left[i, j - MapR[i,j]\right]$

If $E = 0$ the match is coherent, as if we start on a point on the right image, move to its respective on the left one and jump back on the right image we reach the initial point.




```
for i in range (K, img_raw-K-ignore_border):
    for j in range (K+disp_max, img_column-K-ignore_border-disp_max):
        #We select the pixel and move along its row
        min_SAD_L=500000
        min_SAD_R=500000
        #moving along the epipolar line
        for d in range (disp_min, disp_max+1):
            SAD_L=0
            SAD_R=0
            for m in range (-K, K+1):
                for n in range (-K, K+1):
                    #matching images both from left to right and from right to left
                    SAD_L =SAD_L+AD_left(imageL, imageR, i, j, d, m, n)
                    SAD_R =SAD_R+AD_right(imageL, imageR, i, j, d, m, n)
            if (SAD_L<=min_SAD_L):
                min_SAD_L=SAD_L
                disp_L=d
            if (SAD_R<=min_SAD_R):
                min_SAD_R=SAD_R
                disp_R=d
        MapL[i,j]=np.asarray([disp_L*disp_scale])
        MapR[i,j]=np.asarray([disp_R*disp_scale])

valid_pixel=0
MapL2=MapL
for j in range (K+disp_max, img_column-K-disp_max-ignore_border):
    for i in range (K, img_raw-K-ignore_border):
         #select a pixel in the right side
        DR=MapR[i,j]/disp_scale
         #pick its counterpart in the left map
        DL=MapL[i,j-int(DR)]/disp_scale
        #check the consistency
        if (DL!=DR):
            MapL2[i,j]=np.asarray([255])
        else:
            valid_pixel= valid_pixel+1
```

This operation allows us to reject incoherent matches that can't be found with just the base computation. We decided to assign white colour to such pixels in the disparity map.

Since this improvement implies the rejection of invalid matches, the resulting disparity maps turn out not as dense as the previous ones. Therefore, only those pixels featuring a valid disparity value should be taken into account in the evaluation of the error.

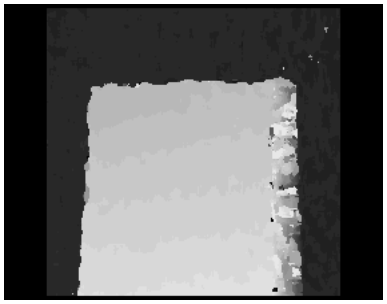
```
def Error (target, img, w, h, m, scale_factor, v_pixel, d_max, ig_border):
    error=0
    for j in range (m+d_max, w-m-d_max-ig_border):
        for i in range (m, h-m-ig_border):
            if (img[i,j]!=255):
                if (abs(target[i,j]/scale_factor - img[i,j]/scale_factor)>1):
                    error=error+1

    return (error/v_pixel)
```

The error is eventually incremented when the pixels are not white (valid ones).

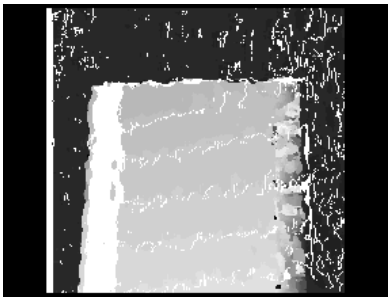
	Base error	**LR error**	**% of invalid matches**	**% of improvement**
Map	0.08958	0.06199	15.5%	30.7%
Sawtooth	0.06856	0.06527	9.5%	4.7%
Tsukuba	0.14536	0.13297	13.0%	8.5%
Venus	0.07863	0.06199	11.7%	21.1%

We can see that the major improvement occurs on the first image that, in fact, has the greatest number of occlusions. Indeed, these points are left unmatched during the left-right consistency check.

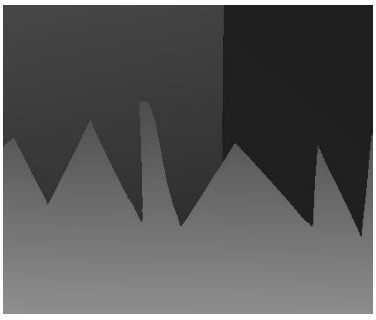
Ideal



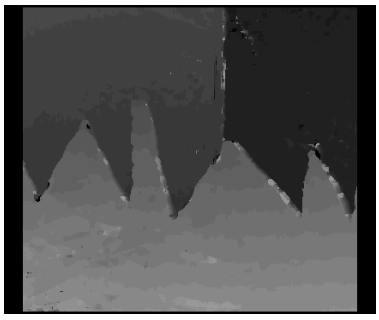
Baseline algorithm



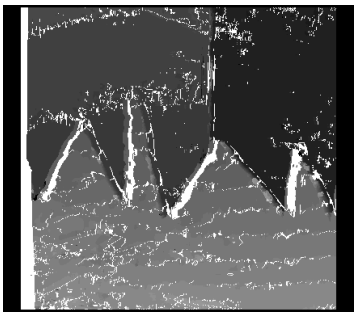
Left-Right consistency check



Ideal

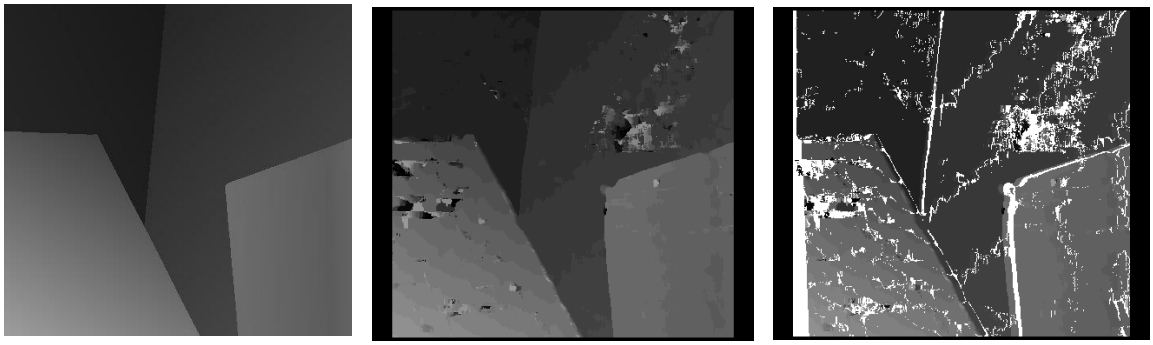


Baseline algorithm



Left-Right consistency check

Ideal *Baseline algorithm* *Left-Right consistency check*



Ideal *Baseline algorithm* *Left-Right consistency check*

In these images we can also notice the presence of a vertical white line generated by the displacement between the left and right perspective.

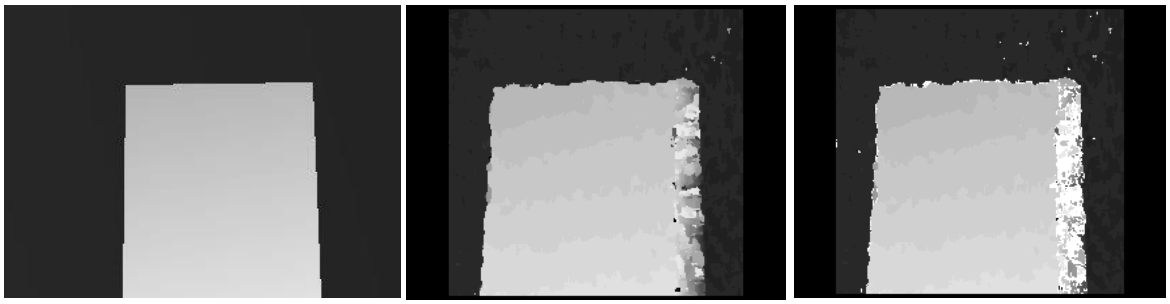
b) Rejection of ambiguous minimum

We have analysed the dissimilarity function in order to accept a match only in case of a sharp and unambiguous minimum. To achieve this goal, for each pixel, we save the values of the SAD within the disparity range in a vector sorted by increasing values. Then we compare the first element of the vector (minimum), with the third one. The match is considered valid if their difference is greater than a certain threshold, set at 10%. We have considered the third smallest value and not the second one because there can be pixels whose SAD is nearly equal for two neighbouring disparities. These "double minima" are valid and occur frequently between regions that differ in their disparity by one. Again, the unmatched points are coloured in white and the error is computed only between valid matches.

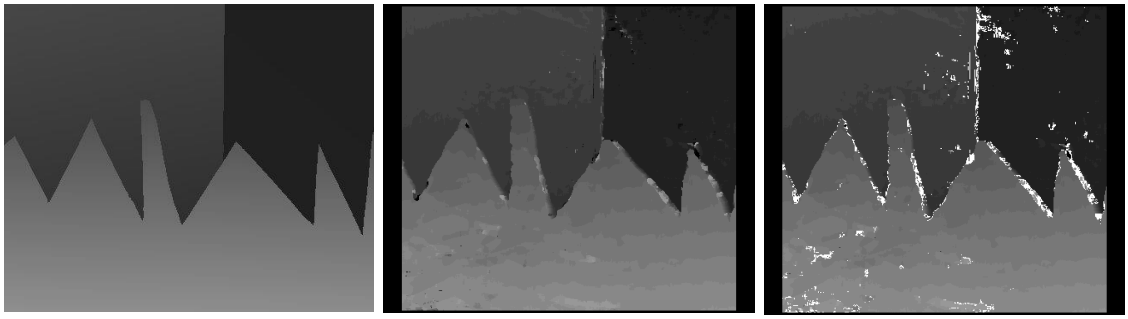
```
#sort the array and mark the match as invalid (white) if the minimum isn't sharp enough
        sad_array=np.sort(sad_array)
        if ((sad_array[2]-sad_array[0])<0.1*sad_array[0]):
            disp=255/disp_scale
        else:
            valid_pixel=valid_pixel+1

        MapL[i,j]=np.asarray([disp*disp_scale])
```

The majority of the discarded minimum are given by large disparity gap within the points of the supporting window as well as noisy/textureless regions in the disparity map.

	Base error	Refinement error	% of improvement	% of invalid matches
Map	0.08958	0.05041	43.7%	4.37%
Sawtooth	0.06856	0.05497	19.8%	2.20%
Tsukuba	0.14536	0.12148	16.4%	5.52%
Venus	0.07863	0.05222	33.6%	5.82%

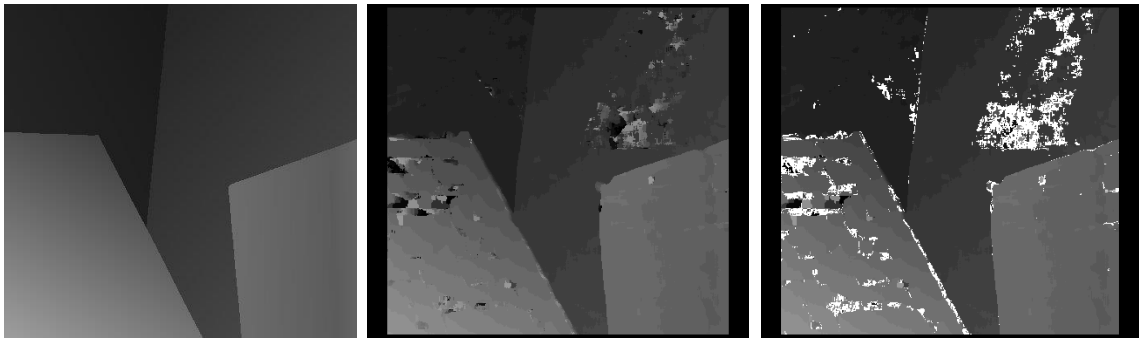


Ideal *Baseline algorithm* *Incorrect measurements*



Ideal *Baseline algorithm* *Incorrect measurements*

Ideal *Baseline algorithm* *Incorrect measurements*



Ideal *Baseline algorithm* *Incorrect measurements*

c) Robust similarity function

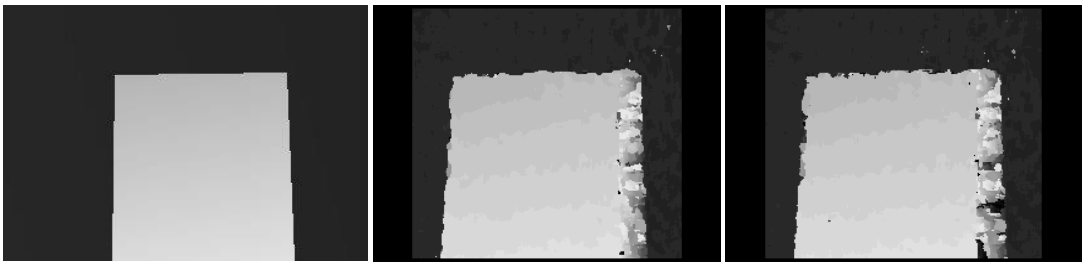
In this step we tried to improve the quality of the disparity map by evaluating it with a more robust similarity function such as the Zero-Mean Normalized Cross-Correlation (ZNCC). It provides a higher degree of invariance, extended to all the affined intensity changes. For each pixel, the algorithm calculates the ratio between the dot product of the zero mean intensities of the two images and the product of their L2 norm, and selects the higher result since the ZNCC is a similarity function.

```
def mean (img, i, j, d, window_size):
    mean=0
    for m in range (-window_size, window_size+1):
        for n in range (-window_size, window_size+1):
            mean= mean+ int(img[i+m, j+n+d])
    return mean/(window_size*window_size)

def ZNCC_num (imgL, imgR, i, j, d, m, n, meanL, meanR):
    num=(int(imgL[i+m, j+n])-meanL)*(int(imgR[i+m, j+n+d])-meanR)
    return num

def ZNCC_den_l (img, i, j, m, n, mean):
    den=(int(img[i+m, j+n])-mean)**2
    return den

def ZNCC_den_r (imgR, i, j, d, m, n, meanR):
    den=(int(imgR[i+m, j+n+d])-meanR)**2
    return den

for i in range (K, img_raw-K-ignore_border): #Select a pixel and move along its row
    for j in range (K+disp_max, img_column-K-ignore_border-disp_max):
        max_ZNCC_L=0
        #moving along the epipolar line
        for d in range (disp_min, disp_max+1):
            num_ZNCC=0
            den1_ZNCC_L=0
            den2_ZNCC_L=0
            mean_L=mean(imageL, i, j, 0, K)
            mean_R=mean(imageR, i, j, d, K)
            for m in range (-K, K+1):
                for n in range (-K, K+1):
                    num_ZNCC=num_ZNCC+ZNCC_num(imageL,imageR,i,j,d,m,n,mean_L,mean_R)
                    den1_ZNCC_L=den1_ZNCC_L+ZNCC_den_l(imageL, i, j, m, n, mean_L)
                    den2_ZNCC_L=den2_ZNCC_L+ZNCC_den_r(imageR,i,j,d,m,n,mean_R)
            ZNCC_L=num_ZNCC/np.sqrt(den1_ZNCC_L*den2_ZNCC_L)
            if (ZNCC_L>=max_ZNCC_L):
                max_ZNCC_L=ZNCC_L
                disp_L=d
        MapL[i,j]=np.asarray([disp_L*disp_scale])
```

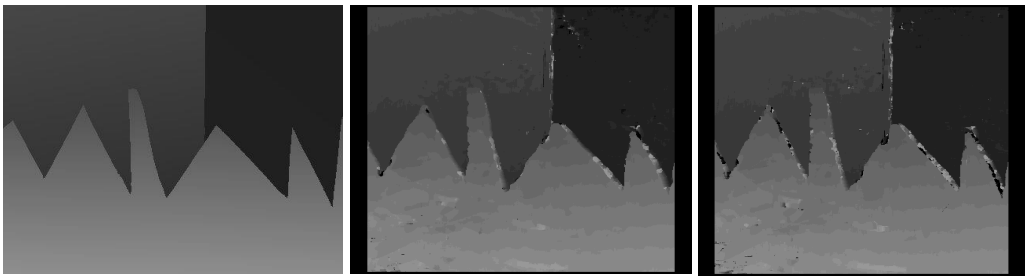
The error instead, is computed as in the base algorithm.

	Base error	**ZNCC error**
Map	0.08958	0.09564
Sawtooth	0.06856	0.07400
Tsukuba	0.14536	0.15746
Venus	0.07863	0.05980

It's quite clear that the new similarity function affects the mapping error in different ways depending on the pair of stereo images considered. In particular, *Map*, *Sawtooth* and *Tsukuba* witness a small increasing in their error, while the quality of *Venus'* disparity map grows. Anyway, the usage of a more robust similarity function is not mandatory in our case as we don't expect strong light changes between the left and right images, having they been captured in the same moment.

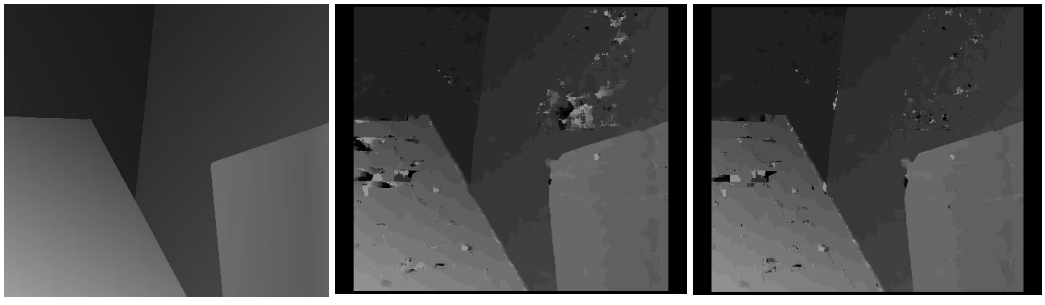


Ideal *Baseline algorithm* *ZNCC algorithm*



Ideal *Baseline algorithm* *ZNCC algorithm*

Ideal *Baseline algorithm* *ZNCC algorithm*



Ideal *Baseline algorithm* *ZNCC algorithm*

d) Prefiltering

We also tried to reduce the mapping error by applying filters to the stereo images during a pre-processing phase, which compensates for the photometric distortion.

In particular, we have applied two different non-linear filters:

- Bilateral Filter

Also called Edge Preserving Smoothing, this type of filter smooths the image without blurring it, keeping the quality of the edges. To achieve this, the filter exploits two gaussian functions and computes the final pixels' weight considering both the relative position and the similarity between the intensities. The drawback of this application is the increasing of the computational cost.

```
#bilateral filtered images
sigmaColor=5
sigmaSpace=5
kernel_size=int(np.ceil((3*sigmaSpace))*2 + 1)
BilateralL=cv2.bilateralFilter(imageL, kernel_size, sigmaColor, sigmaSpace)
```

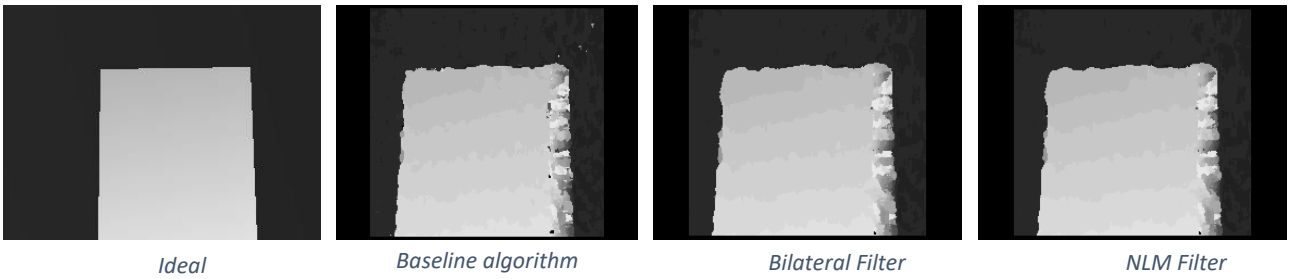
- Non-Local Means Filter

The Non-Local Means Filter is again an Edge Preserving Smoothing Filter, that exploits similarities among patches spread over the image regardless locality. The weighted sum is computed using all pixels in the image, according to how much their intensities are similar to the one given. However, even if the theoretical supporting window should be the whole image, in practice a smaller portion (*SxS*) is used to make the filter faster.

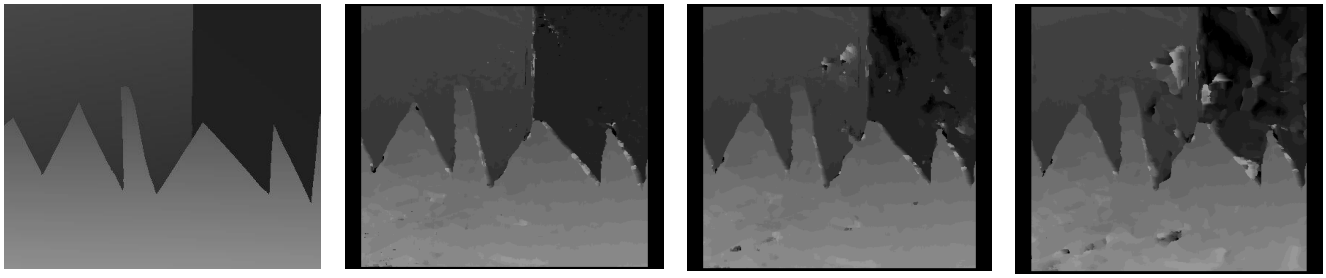
```
#non-local means filter
h=10      #bandwidth
N=7       #size in pixels of the template patch used to compute weights
S=21
NLM_L=cv2.fastNlMeansDenoising(imageL, h, N, S)
```

As we notice from the following table, in both cases the quality of the disparity maps has not improved. This is probably due to the fact that patches became more difficult to be recognised over smoothed images. More specifically, we see that the bilateral filter performs better on *Sawtooth*, *Tsukuba* and *Venus*, while the error of Ma-p is lower when using NLM filter.

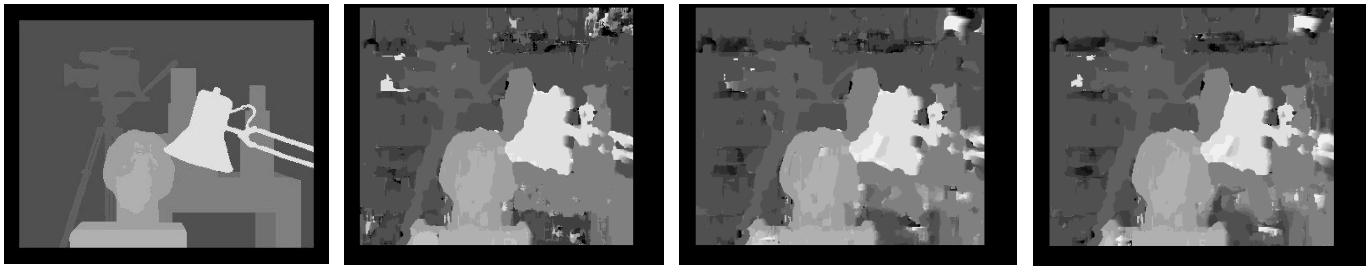
	Base error	**Bilateral error**	**NLM error**
Map	0.08958	0.09390	0.09328
Sawtooth	0.06856	0.1161	0.20454
Tsukuba	0.14536	0.1569	0.17435
Venus	0.07863	0.2428	0.29516



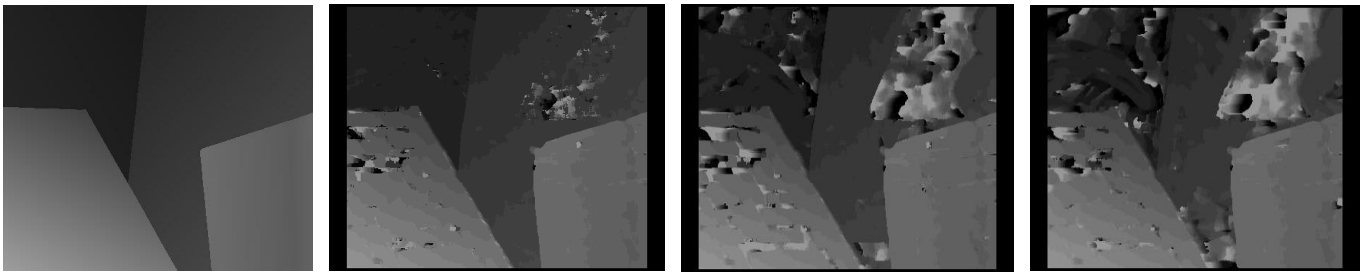
Ideal *Baseline algorithm* *Bilateral Filter* *NLM Filter*



Ideal *Baseline algorithm* *Bilateral Filter* *NLM Filter*



Ideal *Baseline algorithm* *Bilateral Filter* *NLM Filter*



Ideal *Baseline algorithm* *Bilateral Filter* *NLM Filter*

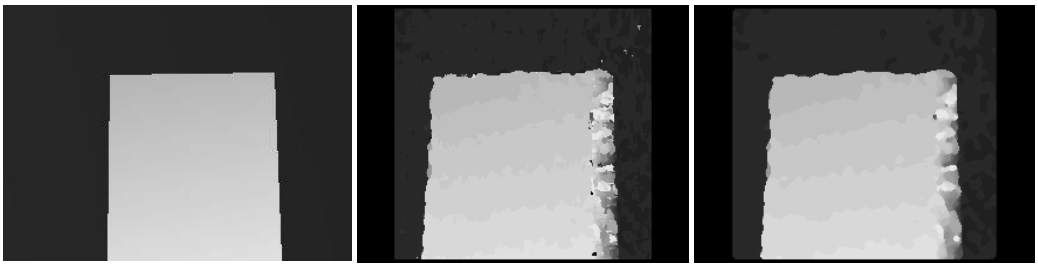
e) **Postfiltering**

As last modification, we tested the performances of a filter applied in the post-processing phase. Our objective was refining the final quality of the map by deleting possible spikes through a Median Filter. It is the simplest non-linear filter, and it replaces every pixels' intensity with the median value over a given neighbourhood. As outliers tend to fall either at the beginning or at the end of their sorted set, they can be easily removed with median filter. Furthermore, this type of filter doesn't blur the image.

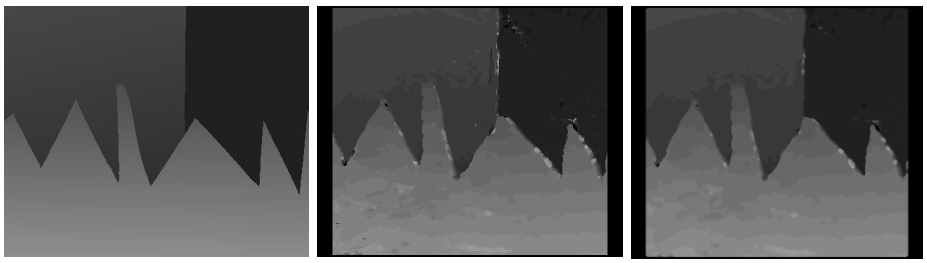
```
#post filter with median filter
k_size=5
MapL_filtered=cv2.medianBlur(MapL,k_size)
```

	Base error	Median error	% of improvement
Map	0.08958	0.08770	2.1%
Sawtooth	0.06856	0.06264	8.6%
Tsukuba	0.14536	0.13440	7.5%
Venus	0.07863	0.06806	13.4%

As we see from the table, the introduction of the Median filter leads to a decreasing of the error function in all the cases, especially in *Venus* whose disparity map indeed looked pretty noisy in the base algorithm case.

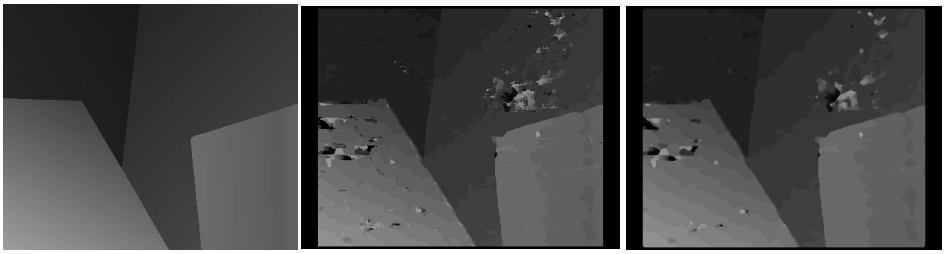


Ideal *Baseline algorithm* *Postfilter*



Ideal *Baseline algorithm* *Postfilter*

Ideal *Baseline algorithm* *Postfilter*



Ideal *Baseline algorithm* *Postfilter*

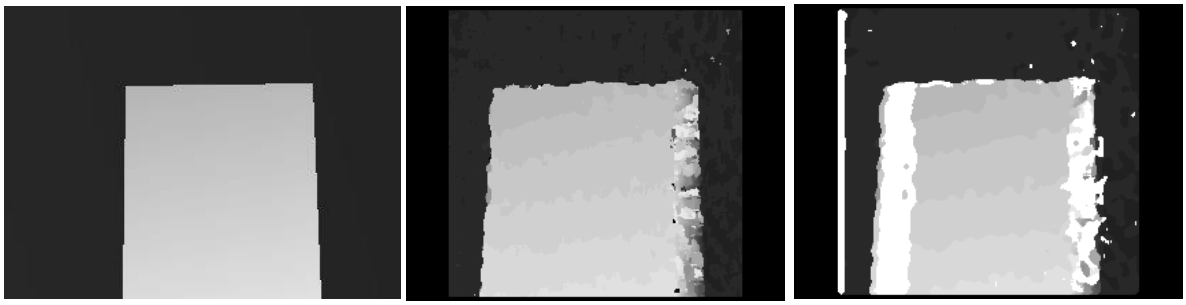
Conclusions

Overall, some of the techniques tested in this second phase of the project enabled us to achieve better results in the stereo matching. In particular, the best algorithm should include Left-Right check, postfiltering and the rejection of ambiguous matches.

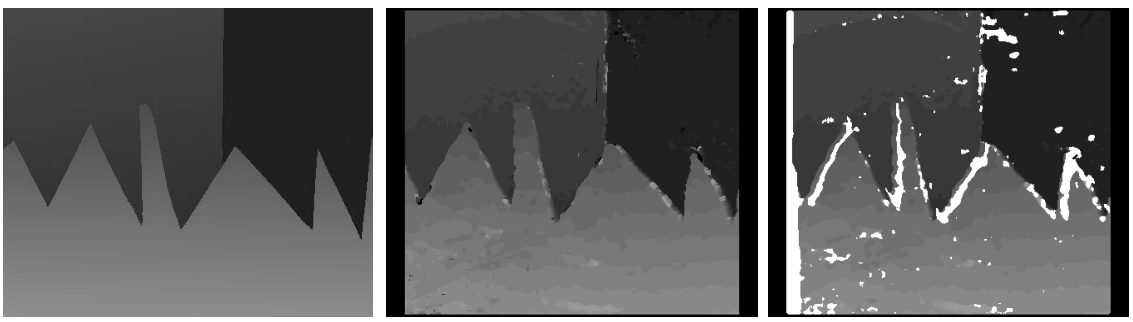
Regarding the choice of the best (dis)similarity function, as the ZNCC reaches better results only for *Venus* and implies a higher computational cost, we decided to implement SAD for all the images. Consistently with the analysis made before, we also chose not to apply any filter in the pre-processing phase.

Obviously, the application of both the Left-Right consistency check and the weak minimum rejection carries to an augmentation of the number of invalid matches in the final disparity maps. As a result, the proposed method could not be very useful, as long as the low final error is computed taken into account a smaller number of pixels.

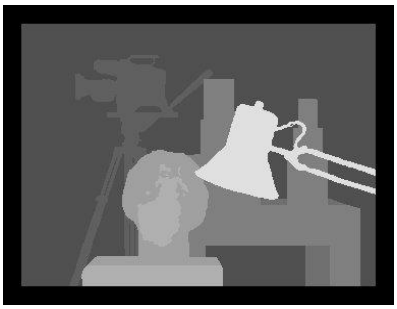
	Base error	**Final error**	**% of invalid matches**	**% of improvement**
Map	0.08958	0.05484	15.7%	38.8%
Sawtooth	0.06856	0.05797	10.3%	15.4%
Tsukuba	0.14536	0.11207	14.8%	22.9%
Venus	0.07863	0.04228	14.0%	46.2%



Ideal *Baseline alaorithm* *Final algorithm*

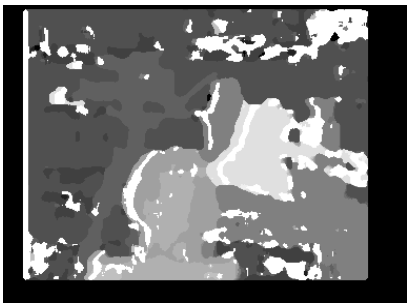


Ideal *Baseline algorithm* *Final algorithm*

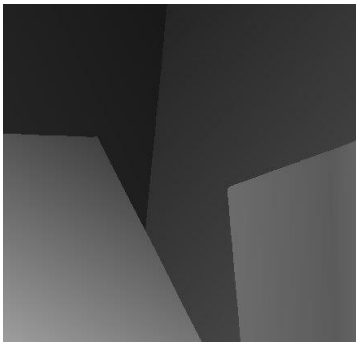


Ideal

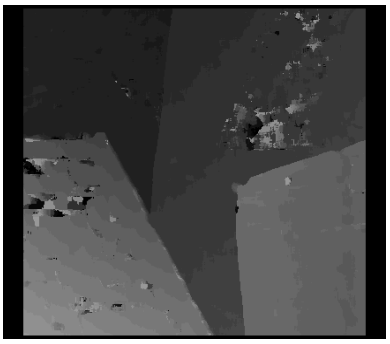
Baseline algorithm



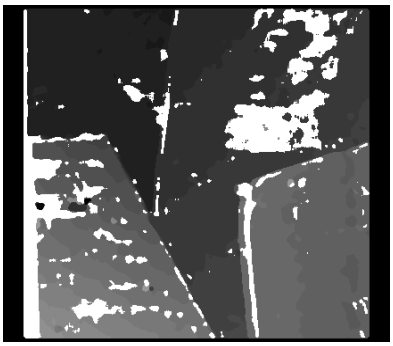
Final algorithm



Ideal



Baseline algorithm



Final algorithm