COMP.SGN.320

3D and Virtual Reality Project Assignment

Dense disparity estimation via local stereo matching

Project overview

Implement the local color-weighted disparity estimation algorithm and evaluate its performance on a set of stereo image pairs. **The project is completed alone,** i.e. it is not a group assignment. Sharing of code is not allowed.

The available Moodle folder is divided into three folders. First, demo examples in which you can check and visualize the result of each task separately by running *demo_mandatory.m* and *demo_4_5_6_7.m*. Second folder is function templates in which contains several functions as a skeleton for your implementations. You can execute the functions by running *run_mandatory.m* and *run_4_5_6_7.m.* The third folder, teddy, includes the dataset required for tasks.

The algorithm includes the following steps:

1. Cost function calculation *(mandatory)*
2. Cost aggregation based on three methods (*mandatory*)
   a. Box filtering
   b. Gaussian filtering
   c. Local color-weighted filtering (guided filter)
3. 'Winner-takes-all' disparity estimation and quality evaluation *(mandatory)*
4. Detection of occlusions *(1p)*
5. Computation of confidence values for disparity estimates *(1p)*
6. Post-filtering to tackle occlusions and bad pixels *(1p)*
7. Implementation of cross-bilateral filter-based aggregation *(1p)*

Steps marked as mandatory should be completed in order to pass the exercise (grade 1). Each additional point will be directly added to the grade. The tasks should be implemented in numerical order since 6 depends on 4 and 5, and 7 is the most challenging one. Grade 5 requires all steps to be completed.

Your implementation should produce equivalent (not necessarily identical) output to the demo files.

Deliverables

- A report: a pdf with all of the figures generated by the demo scripts using your implementation of the functions. No textual content is required, but if there are problems with some tasks, please make a note of it in the report and describe the issue. **List clearly, which steps you are delivering** with a table such as**:**

Step	**Completed**
Steps 1, 2, 3	X
Step 4	X
Step 5	X
Step 6	
Step 7	

- Your own implementation of the requested functions for those tasks which you have marked as completed.

The evaluation of the submission will be done on a version 2021a Matlab installation with all toolboxes available by executing *run run_mandatory.m* and *run run_4_5_6_7.m.* Do not make any modifications in the demo scripts, and do not change the names of the function templates. Those are not necessary and will prevent the semi-automatic evaluation from running.

The script **has to run without errors**. Tasks marked as completed without any additional remarks in the report are expected to work. If your implementation is generating obviously wrong results and you don't describe the problem in the report in detail (empty images, extreme noise, heavy artifacts etc.), you won't be getting a second chance after the evaluation. Well documented problems may get feedback and a more time to fix them.

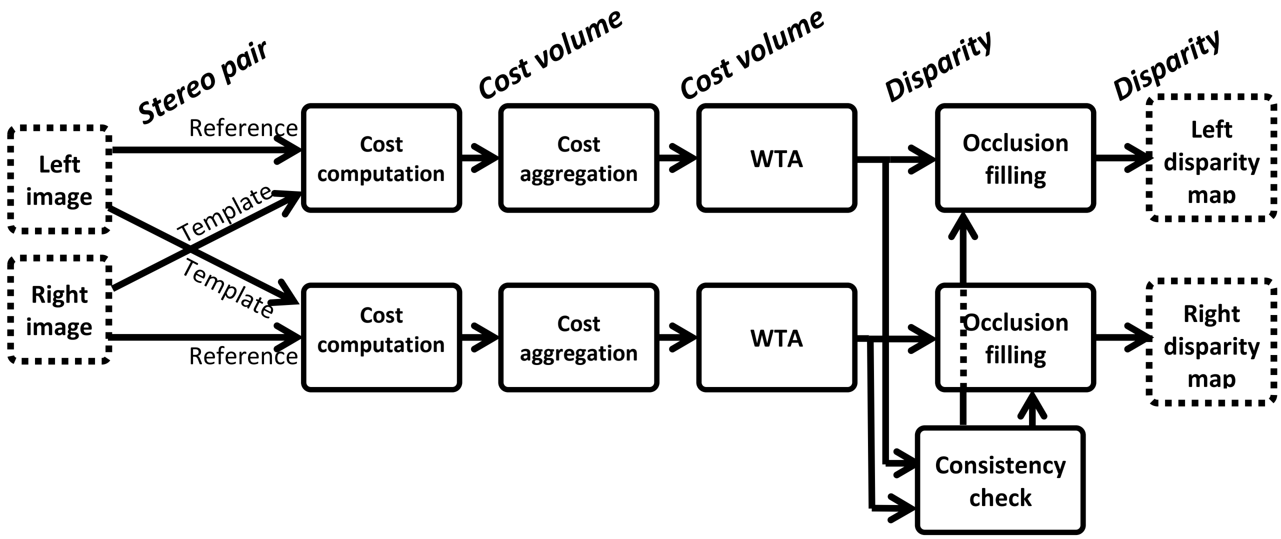


Figure 1. Illustration of the stereo matching pipeline

Test data

Start with a rectified image stereo pair taken from the Middlebury stereo vision site http://vision.middlebury.edu/stereo/ (included in the Moodle files). Use views 2 and 6 as they have the corresponding ground truth disparity maps provided. view 1 and 5

Quality metric

Use the percentage of bad pixels as a quality metric to compare your estimation result with the ground true disparity. Bad pixels are defined here as pixels whose disparity value differ from the ground truth by more than 1 in either direction.

Cost Calculation

Calculate a 3D volume of per-pixel differences of the input images

$$C(x, y, d) = |L(x, y) - R(x - d, y)|$$

Where x, y,and d corresponds to the horizontal and vertical pixel coordinates and shift amount from right image to the left image, respectively. When the images are in color, compute the difference per channel, and sum the differences over the three color channels to get a scalar value for all elements of C.

In order to reduce the effect of outliers on matching quality, the individual cost values in the 3D cost matrices should be capped, so they do not reach values over e.g., 150 (in case of L1 cost metric).

Cost Aggregation

Cost volume aggregation is per-slice filtering of the Cost Volume to make it more consistent within the disparity dimension. Performing this operation on the cost volume is equivalent to block matching (like done e.g. in motion estimation).

$$C_{agg}(x,y,d) = \sum_{u,v\in\Omega(x,y)} C(x,y,d)W(u,v),$$

where $\Omega(x,y)$ is the windowed neighborhood of (x,y) and $W(u,v)$ the weighting function inside the window.

- In block matching, the weights are constant over the whole window (window size 2x radius+1)
- In the Gaussian case they follow the normal distribution (window size 2x radius+1, sigma according to the input parameter)
- In the case of color weighted filtering, the weights are computed from the associated color image

Hint: *help imfilter, fspecial, imguidedfilter*

Bilateral filtering for aggregation (task 7)

This approach is also called cross-bilateral filtering. In addition to considering the spatial distance when computing the contribution of a pixel to the filtering result (e.g. Gaussian filter), the difference in intensity is also taken into account.

Spatial proximity (Euclidian distance) between pixels *p* and *q* in coordinate space *(u,v)*:

$$\Delta g_{pq} = \sqrt{\left(u_p - u_q\right)^2 + \left(v_p - v_q\right)^2}$$

Color similarity can be computed in various color spaces, but here we use the difference in RGB directly, so between pixels *p* and *q*:

$$\Delta c_{pq} = \sqrt{\left(R_p - R_q\right)^2 + \left(G_p - G_q\right)^2 + \left(B_p - B_q\right)^2}$$

The filter weights for the window then become:

$$\mathrm{W(p,q)} = \exp\left(\frac{-\Delta c_{pq}^2}{\gamma_c}\right)\exp\left(\frac{-\Delta g_{pq}^2}{\gamma_g}\right)$$

A suitable starting point for the values of γ_c and γ_g is for instance 10 (assuming the image intensities are [0,255]). It is a good idea to test your filtering function on a regular 2D image to see if it works properly and how the selection of those parameters affects the outcome. E.g. applying it on the image

`I=imnoise(imread('cameraman.tif'))` should suppress some of the added noise, but not blur the edges of the image too much.

Cross-bilateral filtering

The process of applying a cross-bilateral filter is similar to the previously described case, but it involves using two sets of input data: the image to be filtered (or in this case, slice of cost volume) and the reference image. The weights $\mathrm{W}(\mathrm{p},\mathrm{q})$ are computed solely based on the reference image. They are then applied to the target of the filtering. Hint: this filter involves complex processing. Make sure your filter works correctly before you start testing the effect of the filter size.

Disparity by WTA

The disparity *D* can be estimated from the cost volume *C* using the "Winner-takes-all" approach,

$$D(x,y) = \underset{d}{\operatorname{argmin}}\, C(x,y,d).$$

Take a careful look at the previous equation to understand exactly which variable is supposed to be the disparity value!

It would be possible to further improve the depth estimate by fitting a quadratic function to the cost vector along the d-dimension, which can then be evaluated to find a sub-pixel precision disparity estimate (not required).

Comparison of different aggregation approaches

Compare 3 different aggregation methods for different window sizes in terms of the BAD metric. If everything went fine, the resulting graph should be something similar to Fig 2

- Square window (block matching)
- Gaussian window (only considering spatial proximity)
- Adaptive weights (guided filter)

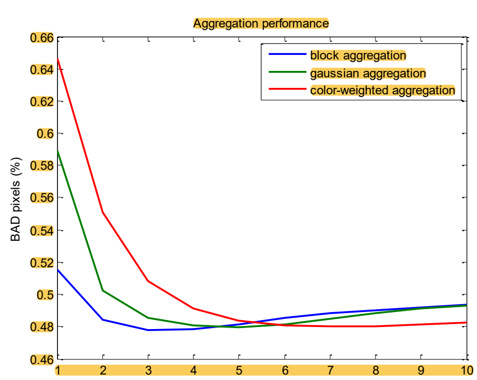


Figure 2. Example of a graph comparing different aggregation methods

Outlier detection 4

Disparity maps suffer from outliers caused by incorrect matches and occluded areas, so post processing is frequently applied to mitigate such errors. One method of detecting outliers in the estimated disparity maps is performing a **left-to-right correspondence check**. The disparity estimation process should provide you two disparity maps, one corresponding to the point of view of each of the views. These maps should be consistent – the same physical point (note the difference, not the same pixel!) should have the same disparity in both images. Often a reasonable allowable difference is +-1.

Fortunately, the disparity value itself tells where the corresponding point should be found in the other image. By taking a disparity value *d* from the left image, subtracting it from its x-coordinate and comparing it to the position in the right image, it is possible to check if both estimates agree on the point's value. If not, the pixel is marked as inconsistent.

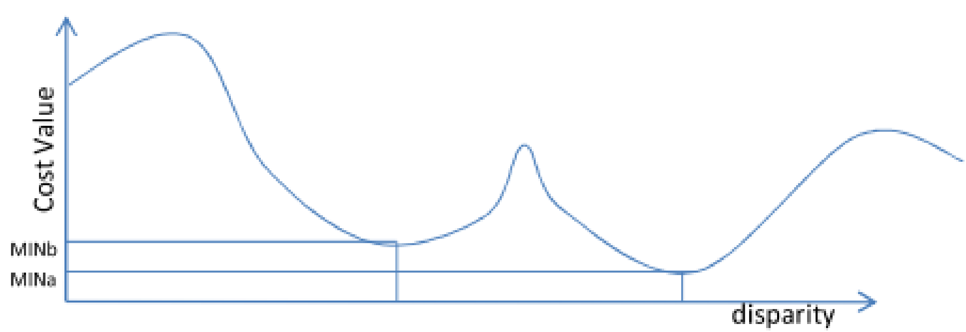
Hint: you can test your implementation by running the consistency check on the ground truth disparity maps. The result should be more or less the same as the occlusion maps given in the source data.

Confidence map

All of the estimates given by disparity estimation are not as reliable as the others. This behavior is examined with a confidence map. Each value of the map gives a numerical estimate on how trustworthy the disparity estimate for the corresponding point is. One way of measuring is to compute the *peak ratio*, which quantifies how "sharp" the minimum of the cost function found by WTA is.

$$PR = \frac{|MinB - MinA|}{MinB}$$

5



The peak ratio can then be combined with the information from the occlusion detection, so that occluded pixels have zero confidence. For determining which pixels should be filled in in the next stage, a reasonable threshold value for the confidence has to be determined. In the reference implementation, anything below confidence 0.1 is considered unreliable, but depending on your implementation, different values may provide better results. Any pixels with confidence less than that will be replaced by the outlier filling step. Hint: *help findpeaks,* though you can expect it to be quite slow to run. If a peak ratio can't be computed (only one minimum or no minima), make sure to assign a confidence value which makes sense.

Outlier filling

For filling in the previously identified faulty or unreliable, pixels, new data has to be generated based on the neighboring areas. A possible filling method is for instance rank order filling, where an occluded pixel is filled in with the e.g., median or minimum value of their neighborhood. Other faulty pixels in the neighborhood shouldn't be considered when computing the median/min. It is a relatively simple method to implement, but the disadvantage is that it doesn't follow real object boundaries and thus might cause objects to "leak" into their surroundings. This approach is implemented in this project. Hint: *help nlfilter, help median.* You will likely have to repeat the filtering process iteratively until the middle sections of the image are filled out and no changes are happening. With the reference implementation, reasonable results are achieved through 5 iterations. You can ignore the blank areas on the edges.