Report: Seventh Assignment (optional assignment)

Disparity Computation

Giovanni Battista Borrè, Mario Ciranni, Tommaso Gruppi, Federico Tomat

June 20, 2019

1 Introduction

In traditional stereo vision, two cameras, displaced horizontally from one another are used to obtain two different views on the same scene, in a manner similar to human binocular vision. By comparing these two images the relative depth information can be obtained in the form of a disparity map, which encodes the difference in horizontal coordinates of corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location.

1.1 The disparity Map

Disparity map refers to the apparent pixel difference or motion between a pair of stereo images. In order to grasp this concept, one can try closing one eye and then rapidly opening it while shutting the other one. Objects that are closer to you will appear to jump a significant distance while objects further away will move very little. That motion is the disparity. In a pair of images derived from stereo cameras, you can measure the apparent motion in pixels for every point and make an intensity image out of the measurements.

1.2 The Dense correspondence algorithms

While sparse matching algorithms are still occasionally used, most stereo matching algorithms today focus on dense correspondence, since this is required for applications such as image-based rendering or modeling. This problem is more challenging than sparse correspondence, since inferring depth values in textureless regions requires a certain amount of guesswork. This dense correspondence have a categorization scheme consists in "building blocks" from which a large set of algorithms can be constructed. It is based on the observation that stereo algorithms generally perform some subset of the following four steps:

1. matching cost computation;
2. cost (support) aggregation;
3. disparity computation and optimization;

4. disparity refinement.

Some examples are the **local (window-based) algorithms** compute the disparity at a given point depends only on intensity values within a finite window, usually make implicit smoothness assumptions by aggregating support and the **traditional sum of squares differences (SSD)** algorithm can be described as:

1. The matching cost is the squared difference of intensity values at a given disparity.
2. Aggregation is done by summing the matching cost over square windows with constant disparity.
3. Disparities are computed by selecting the minimal (winning) aggregated value at each pixel.

Some local algorithms, however, combine steps 1 and 2 and use a matching cost that is based on a supporting region, e.g. **normalized cross-correlation** seen in the previous assignment. Such algorithms typically do not perform an aggregation step, but rather seek a disparity assignment (step 3) that minimizes a global cost function that consists of data (step 1) terms and smoothness terms. The main distinctions among these algorithms is the minimization procedure used. In between these two broad classes are certain iterative algorithms that do not explicitly specify a global function to be minimized, but whose behavior mimics closely that of iterative optimization algorithms.

2 Problem Description

This assignment consists in first of all open the scripts provided and try them in order to better understand the different challenges of a stereo vision pipeline and then to implement an alternative version of disparity computation. The guideline advice to implement three different functions: the first that takes in input two image patches and computes the SSD, the second one that computes the disparity between two images taking all pixels in the first one and each possible disparity in the disparity region using the first function implemented and the third one that computes left-right and right-left disparities and left-right consistency.

2.1 SSD function

To calculate the SSD we implement the function:

$$d = my\_ssd(N1, N2) \tag{1}$$

The sum of squares disparity (SSD) is a statistical technique used in regression analysis to determine the dispersion of data points. The goal of this function is to computes the differences between 2 images of the same size and to understand the similarity between them. The formula used to compute this value is:

$$\Phi_{SSD}(N_1, N_2) = -\sum_{k,l=-\frac{W}{2}}^{\frac{W}{2}} (N_1(k,l) - (N_2(k,l))^2$$

2.2 Disparity function

The disparity function is used to compute the disparity map, it is called:

$$disparityMap = my\_disparity(I1, I2, W, dmin, dmax) \tag{2}$$

This function provides us with the disparity map, a matrix in which for every pixel of the first image, we put the column, with respect to that point, of the most similar point in the second one. The disparity check takes place only on the rows because the two camera are the same height.
To compute the disparity we need other parameters to pass to the function in addition to the two images:

- A window W which says how many "neighborhood" pixel are taken into consideration when we compute the SSD.
- The range of the $dmin$ and $dmax$ that represents where it has to search for the similarity point in the second image.

2.3 Disparity final function

This function carries out the task of computing the $left-right$, $right-left$ disparities and in addition the $left-right$ consistency.

$$disparityMap = my\_disparity\_final(I1, I2, W, dmin, dmax) \tag{3}$$

When computing the right-left disparity we can use the previous function, exchanging the position of $dmin$ and $dmax$ with opposite sign, when is reserved the order of images, so that is consistent with the previous disparity map. The choice of the range value is crucial: if the values are too low, we don't find the right correspondence point in the second image, if the values are too big we end up putting too much noise (correspondence point could be a wrong distant point). To find nice values for this it may be useful rectified images, which plots the 2 image one above the other with different colors. This could help to understand how much the two images are shifted. The problem in computing only the disparity between one image and another one is that we can have no consistency, so we need also to combine the disparity from left to right and the disparity from right to left to find the right consistency.
Having both the maps, we check if:

$$D_{1r}(i,j) = D_{r1}(i, j + Drl(i,j))$$

If this is true then the consistency is confirmed and we put $D_{final}(i,j) = D_{r1}(i,j)$ and if it is false we put in $D_{final}(i,j)$ the minimum floating point that we are provided with by the MATLAB® function *realmax* [3].

3 Result

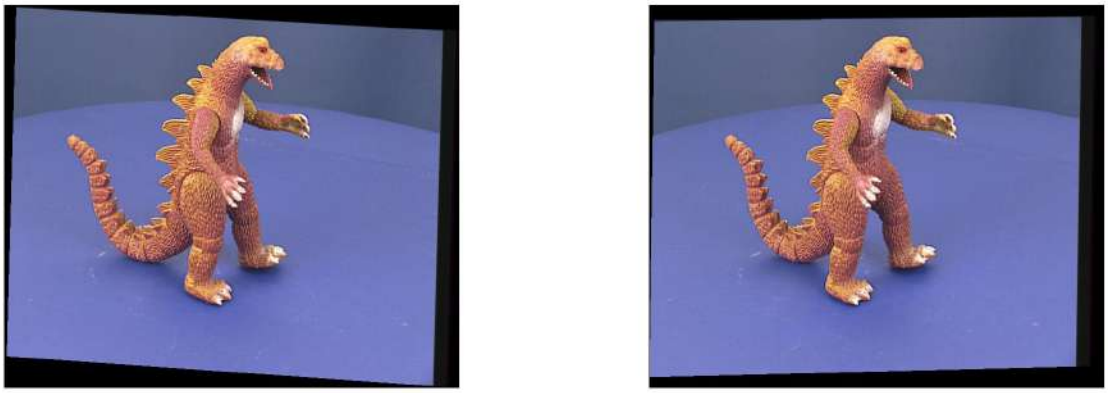

Figure 1: Input Images

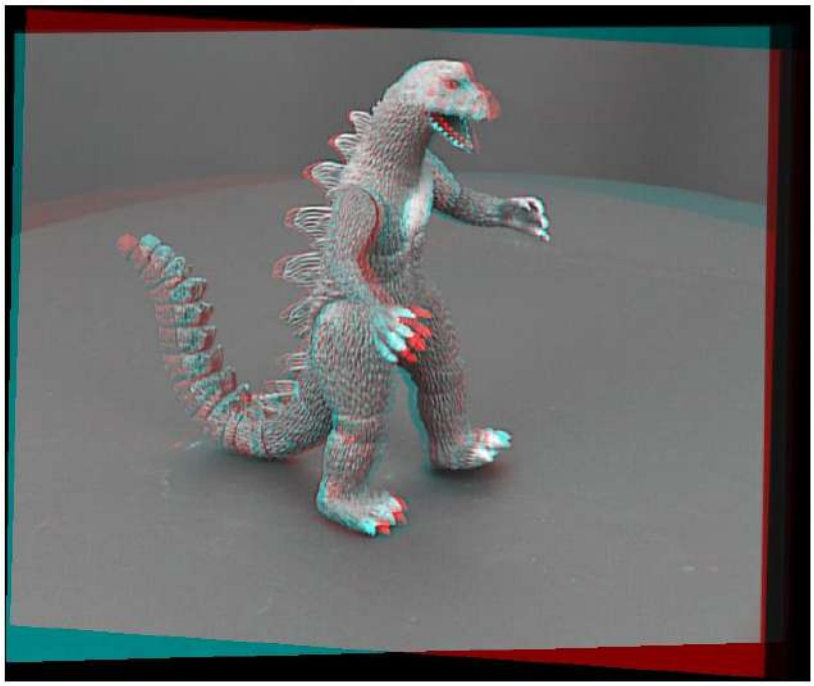

Figure 2: The imshowpair comparison

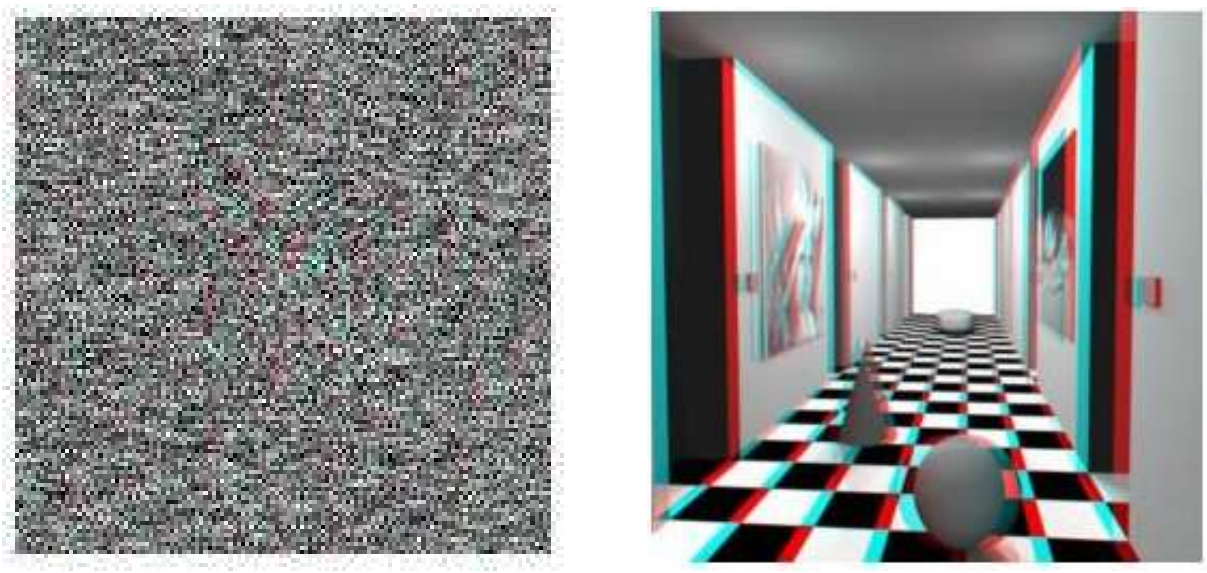

Figure 3: Input Images

The previous images show the overlapping of some couples of different input images on which the disparity map will be computed. This is done in order to underlie tha major differences between them before computing the disparity map and for doing that the MATLAB® function *imshowpair* [4] is used in the code.

Figure 4: Disparity Map

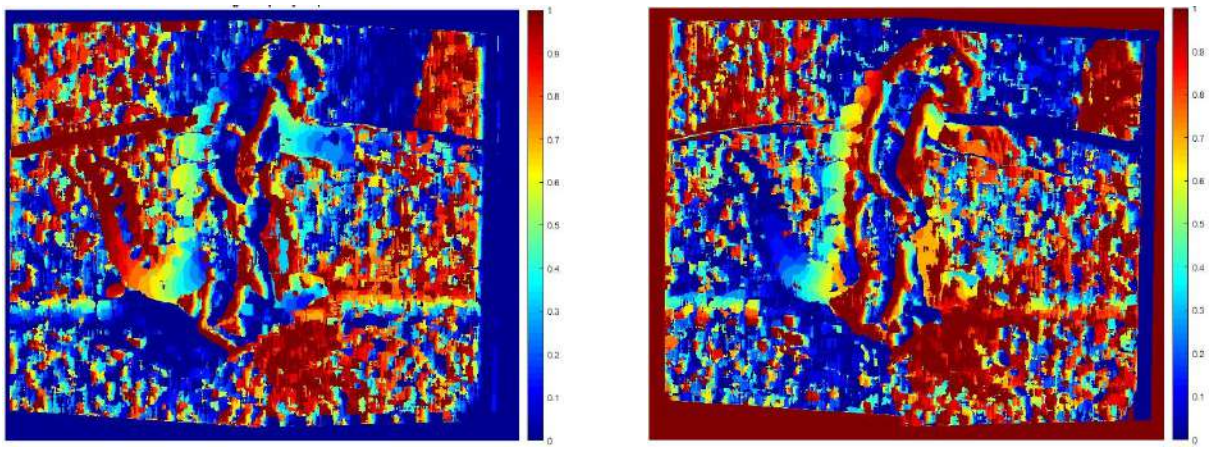

Figure 5: Left-Right and Right-Left disparity Map

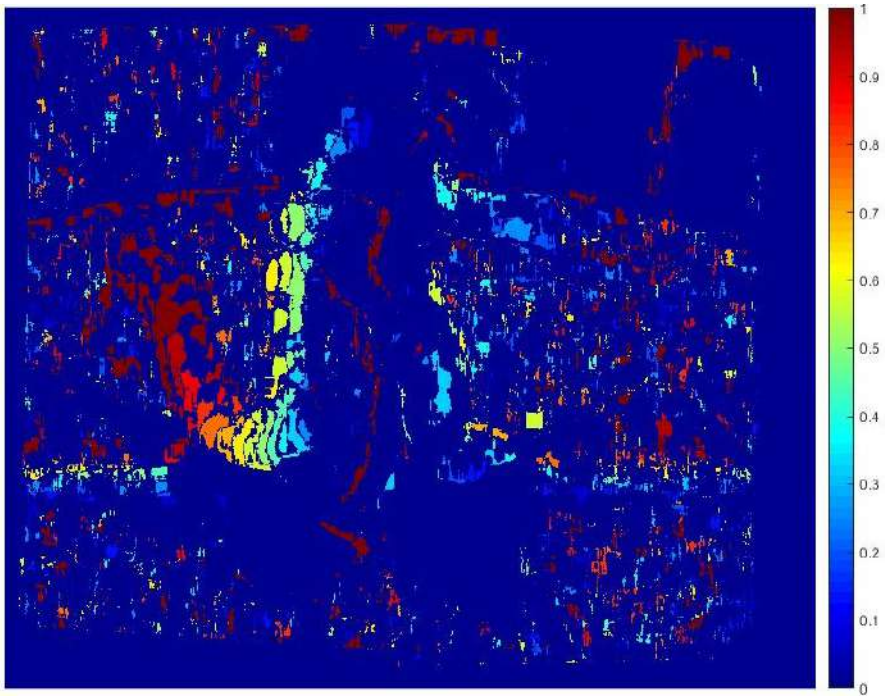

Figure 6: Final disparity Map

The Figure 4 shows the disparity map between the first two input images but the results are not appreciable enough because the output image in grayscale does not underlie the disparities between the two input images. This result comes from the function *my_disparity* and so the computation of the left-right and right-left disparities are requested. This step is done through the *my_disparity_final* function and the results are shown in Figure 5. It can be note that the different disparity maps (Left-Right first and Right-Left then) highlights respectively different variations of disparities. In the first case the most accentuated disparities are the ones on the "left side" of the object while in the second one they come from the "right-side". The combination of these output images results in a final map that shows

with brighter pixels the objects which are closer to the camera while with the darker ones the farther objects from it: the results are really more appreciable with respect to the first disparity map that we obtained with the *my_disparity* function. In the following images we propose the results of the algorithm applied on a different couple of input images: these results better underlie the differences between the Left-Right and the Right-Left disparity maps.

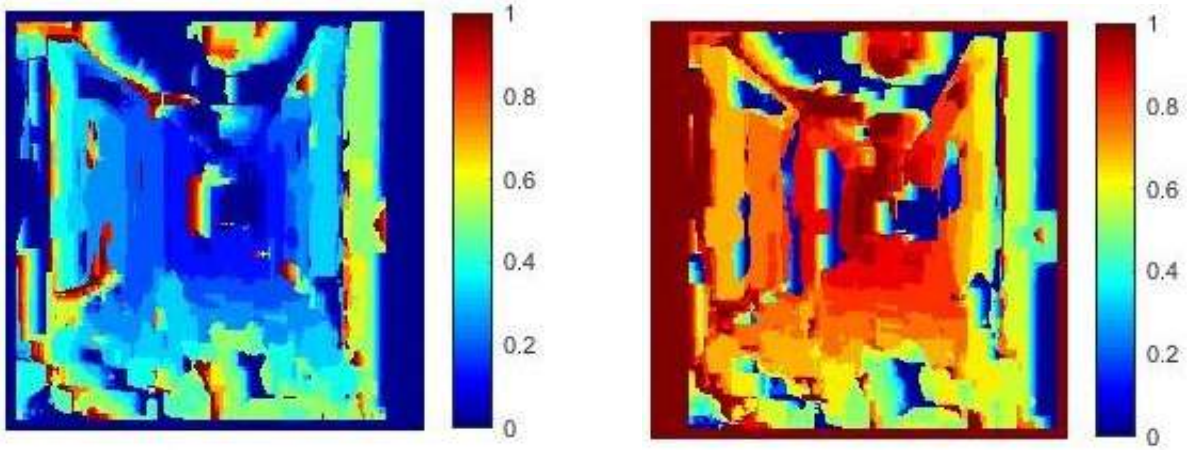

Figure 7: Left-Right and Right-Left disparity Map

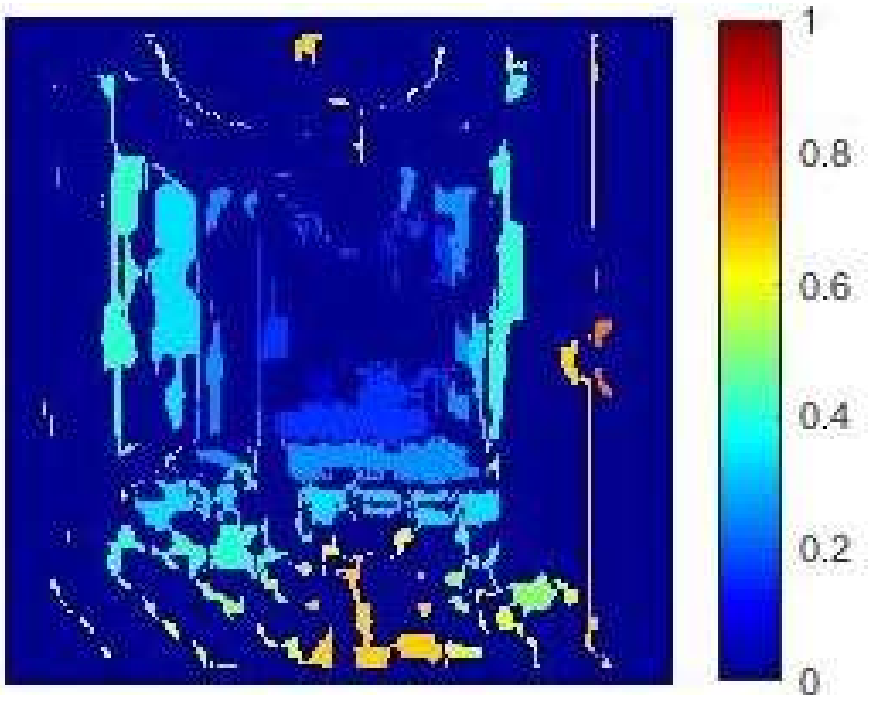

Figure 8: Final disparity Map

4 GitHub Repository

https://github.com/federicotomat/Computer-Vision

References

[1] Richard Szeliski, *Computer Vision: Algorithms and Applications*, (University of Washington, 2010).

[2] Francesca Odone & Fabio Solari, *Computer Vision Course: Image Foundamental*, (University of Study of Genoa, 2019).

[3] Mathworks documentation on *realmax*

[4] Mathworks documentation on *imshowpair*