Depth estimation by stereovision

Advanced Topics in Computer Vision - Master in Automation and Robotics (UPC)

Havard Pedersen Brandal & Olmo Vigil Antuña

MAY 2018

I. Introduction

In this practice, we are tasked to calculate the distance (depth) of the objects that appear on a scene to the camera. To perform this calculation, pairs of images which have some lateral shift in between them will be used (Figure 1). Following the pseudo-code provided, the shirai algorithm has been implemented along with a cross correlation similarity function in order to compute the disparity map of the images, where the gray level indicates the proximity of the image region to the camera.

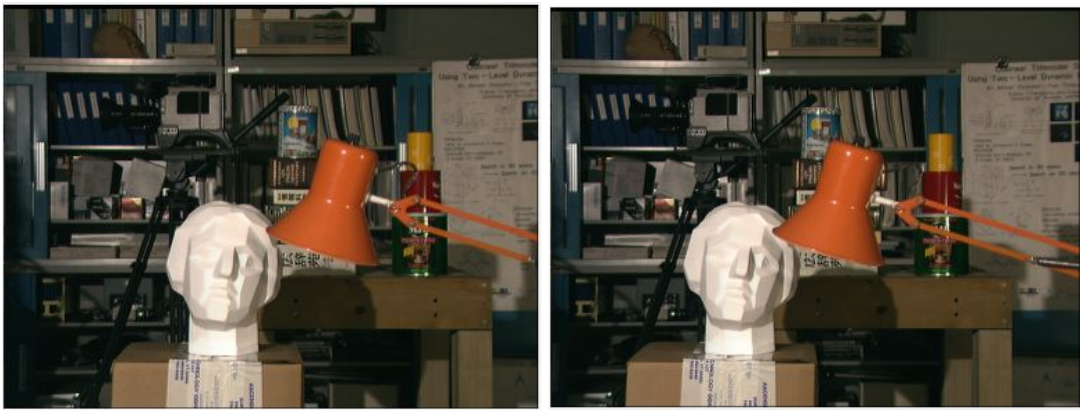

Figure 1: Example of the original images with horizontal shift.

Different parameters and operator sizes have been tested to see which yields the best result. Finally, the function is compared with the MATLAB built-in function *disparity()*.

II. Implementation

This section aims to explain how the main parts of the code implemented work: the shirai algorithm and the implemented cross correlation similarity function.

A. Shirai algorithm

One of the main parts of the practice consists on the implementation of the shirai algorithm using the pseudo-code given in the statement. The main purpose of the algorithm is to match a point from the left image with the corresponding point in the right image. In order to do this, a similarity function is used whose global minimum will indicate possible candidates for the matching. The search interval size **[I_i, I_f]** and the range for the windows size ***k*** are parameters that need to be

introduced by the user. Furthermore, three different thresholds (***d1***, ***d2***, ***d3***) are used to help with the search, which have to be manually tuned depending on the images used:

- ***d1*****:** the first threshold, used to check if there is a unique minimum in the similarity vector smaller than it.

- ***d2*** **(*d2>d1*)**: the second threshold is used to check if the search at that point similarity(i,j) should be carried on or stopped. If all the values of the similarity vector are above d2, the search will stop and the next pixel in the left image will be chosen to continue with the algorithm. Alternatively, the search will also stop if the window size ***k*** reaches its maximum value.

- ***d3 (d3>=d2)***: if there are similarity values between the previous two thresholds, the interval **[I_i, I_f]** is examined for a possible reduction. If a boundary value (the first or the last element of the similarity vector) is greater or equal to ***d3***, it is eliminated and then the search continues as before.

B. Cross correlation

To measure the similarity between regions in the two images, the similarity function chosen has been the cross correlation. Given a point in the left image, the similarity function will extract a block around this point depending on the window size parameter **k.** This block will act as a template to be searched for in the right image like a template matching algorithm.

In our application we assume that the two images are correctly rectified such that the template matching is done only along the epipolar line, and is further restricted to be done in the interval **[I_i, I_f]** given by the shirai algorithm. Blocks that correlate strongly with the template will have a high score so to adapt the result to the shirai algorithm, the resulting correlation is normalized and flipped. This gives highly correlated blocks low scores. An example similarity graph is shown in the Figure 2. The lower the score on the y-axis the more similar are the blocks. The x-axis describes the distance between the blocks in pixels.

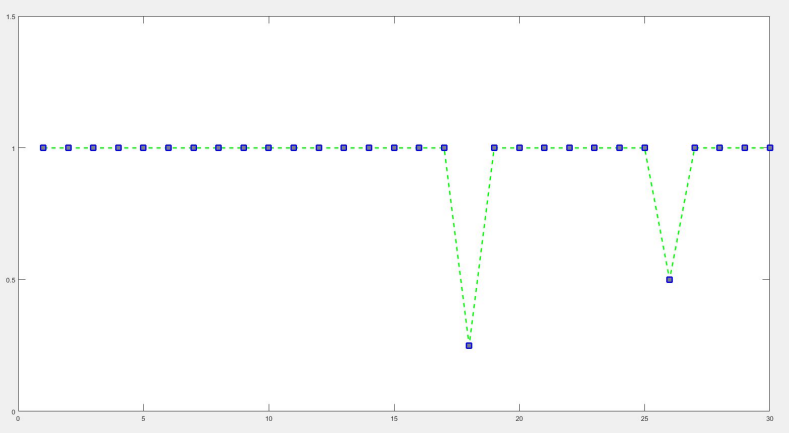

Figure 2: Similarity scores graph from the cross correlation. The two dips in the graph indicates that there are elements similar to the template at around 18 and 26 pixels distance.

III. Results

In this section, all the different phases that an image goes through will be shown, and then some results with different images will be displayed, comparing the results with the MATLAB *disparity()* function.

First, the left and right images are transformed into grayscale and a Canny edge detection is performed in both (Figure 3).

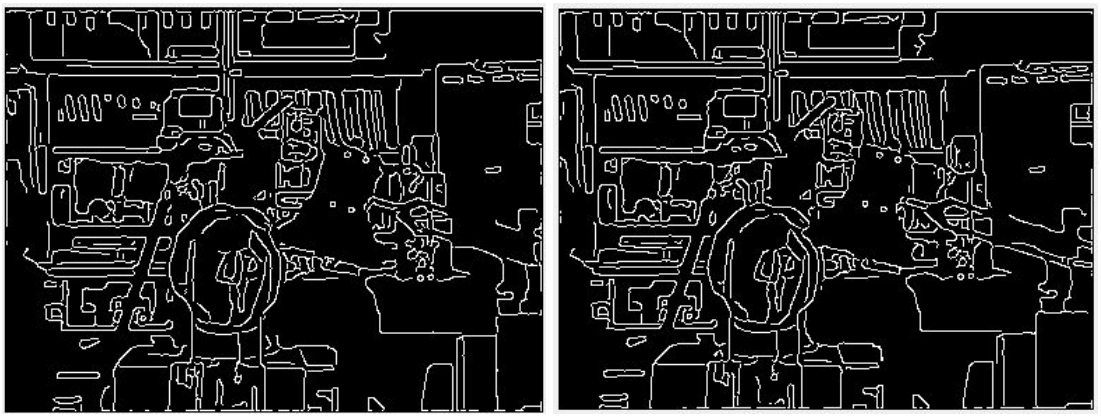

Figure 3: Example of the input edge images.

Then, the image goes through the shirai algorithm implemented using cross correlation, whose output can be seen in Figure 4. The resulting disparity map holds the displacement of each pixel, which in our case ranges from 0 to 30. As we can see, the head and the lamp are the brightest parts of the disparity map, which relates nicely to the original image as the two objects are placed in the front.

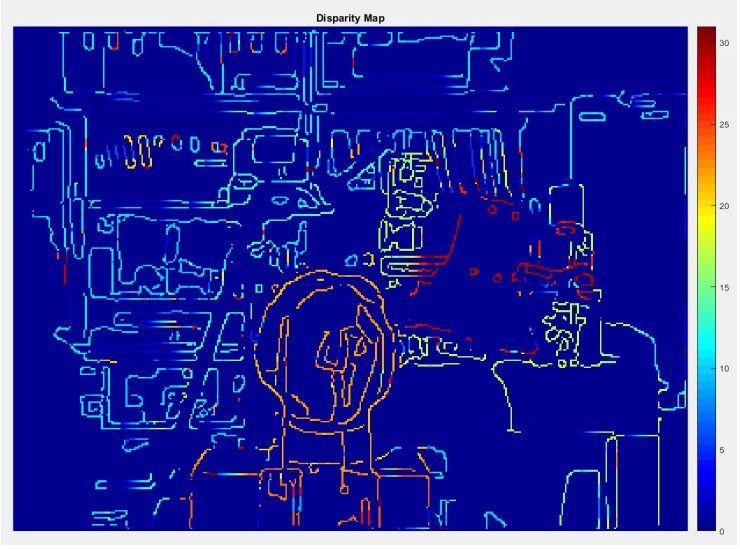

Figure 4: Output of the shirai algorithm using cross correlation.

Finally, the disparity map is filled. To do so, the image is gone through from left to right and from the top to the bottom. Each time that a value different from zero is encountered, this value is stored and put in all the following pixels with value zero. When a different non-zero value is reached, this new value is stored and the process carries on. The disparity map obtained is shown in Figure 5:

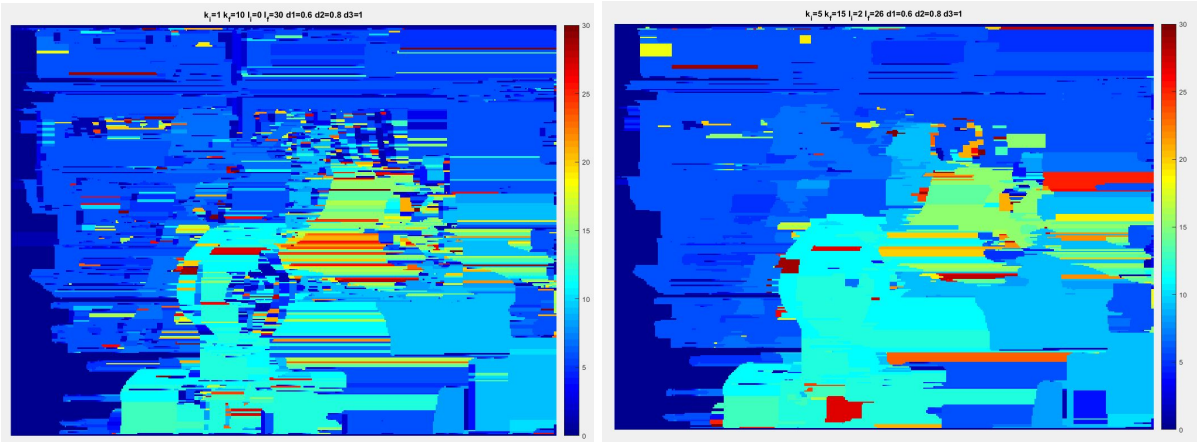

Figure 5: Disparity maps obtained (one direction) with different sizes of k.

It seems that increasing the window size of the similarity search is more robust to noise, but comes at the expense of detail. To try to improve the result, this same process has been done 4 times, turning the original image 90 degrees each time, so we change the orientation in which the map is drawn. Then the four maps obtained are summed and divided by 4, obtaining the disparity map shown in Figure 6.

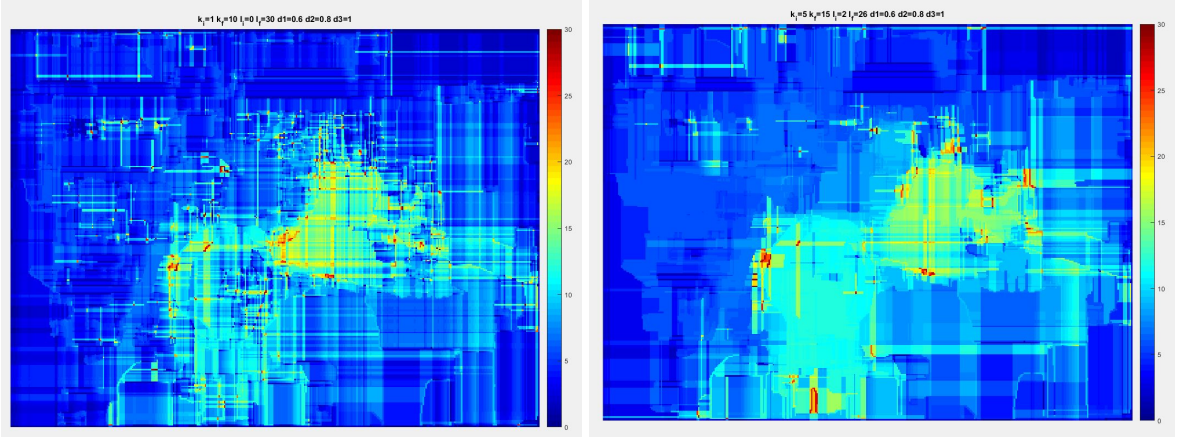

Figure 6: Disparity map obtained (four directions).

The parameters used for the left and the right images of the last two figures are presented in Table 1:

	Left image	Right image
k_i	1	5
k_f	10	15
l_i	0	0
l_f	30	30
d1	0.6	0.6
d2	0.8	0.8
d3	1.0	1.0

Table 1: Parameters used for the left and right images of Figure 5 and Figure 6.

Finally, in Figures 7 and 8, two more input images have been tried. After playing around with the shirai parameters, the chosen ones have been **[4, 10, 0, 30, 0.5, 0.8, 1.0]** following the scheme as in the table above. This parameter list gave detailed classification of disparity while not being to noisy.

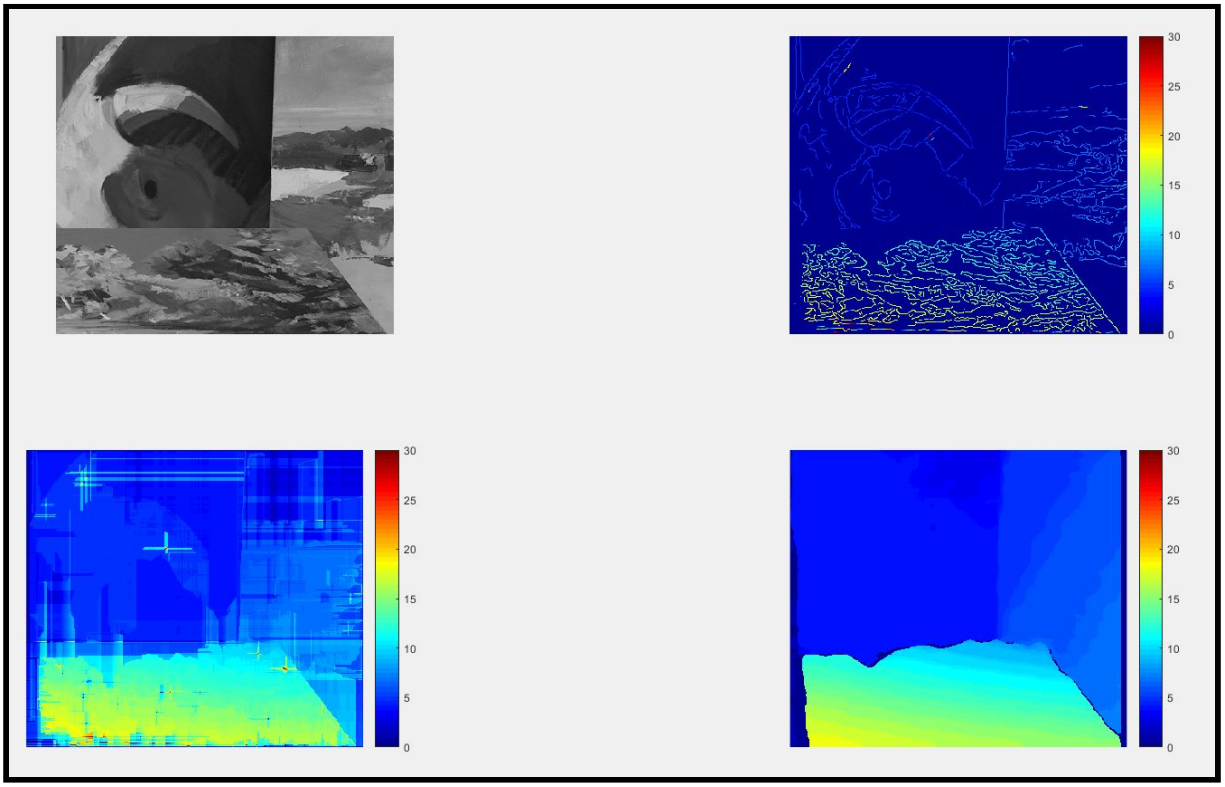

Figure 7: From top left to bottom right, input image, shirai algorithm output, the disparity map filled and the result of the MATLAB function disparity() on the same image.

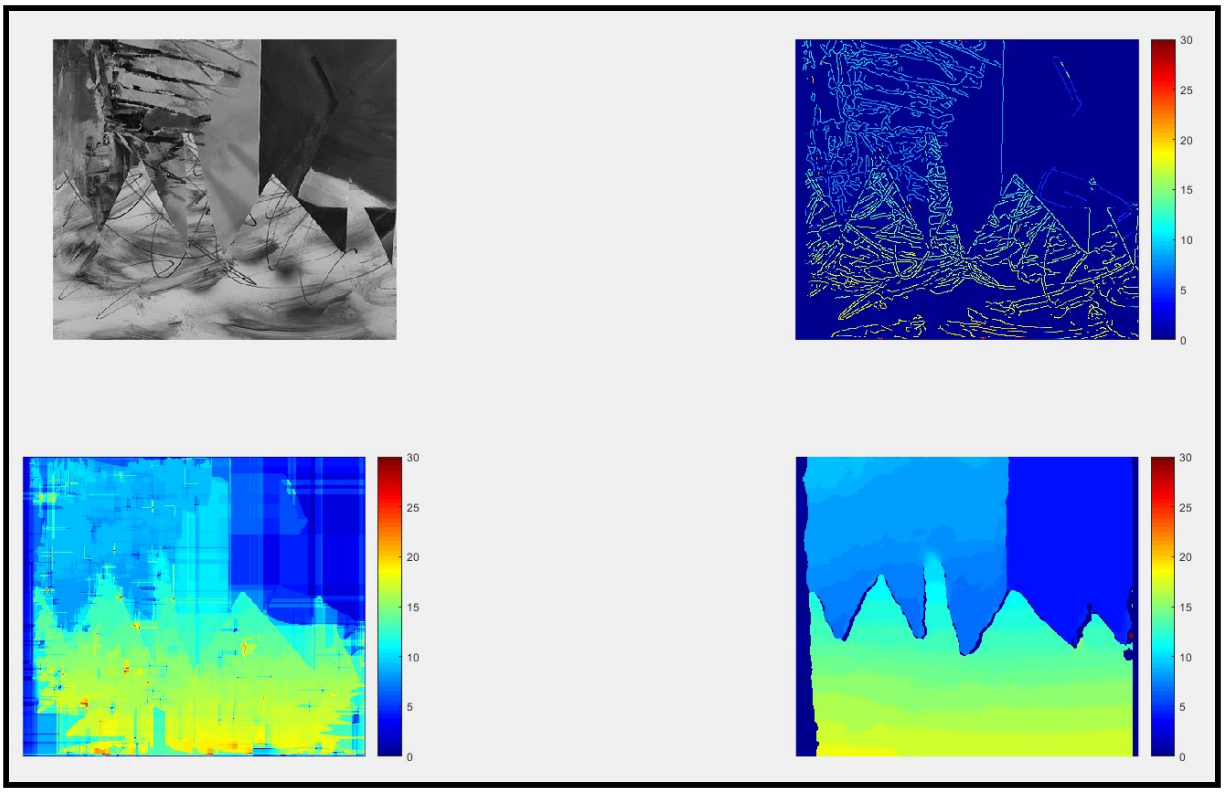

Figure 8: From top left to bottom right, input image, shirai algorithm output, the disparity map filled and the result of the MATLAB function disparity() on the same image.

As it can be seen, the disparity maps obtained are quite similar to the outputs of the MATLAB function disparity(). In order to improve this practice, the filling function could be worked on, and also the parameters could be tuned with more accuracy to get even better results.