CS4495 Computer Vision – Fall 2014

Project 2 – Window-based Stereo Matching

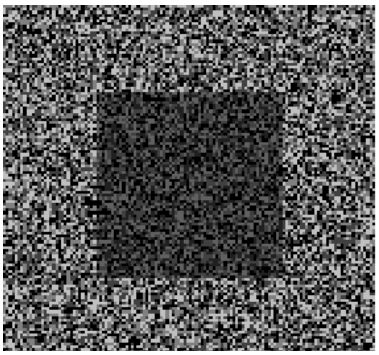
Due **Wed** September **24th**, 2014 5 minutes before midnight

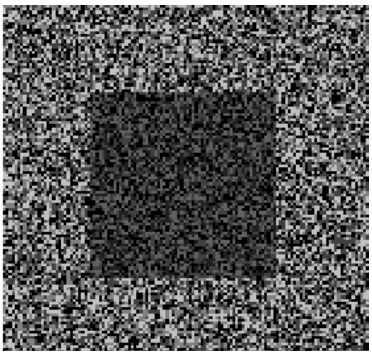
In class and in Forsyth and Ponce, chapter 7 we discussed window-based approaches to estimating dense stereo correspondence. In this project you will implement such approaches and evaluate it on some standard stereo pairs.

As a reminder as to what you hand in: A Zip file that has

(1) Images (either as JPG or PNG or some other easy to recognize format) clearly labeled using the convention PS<number>-<question number>-<question sub>-counter.jpg
(2) Code you used for each question. It should be clear how to run your code to generate the results. All code has to be grouped into separate folders. Each folder is named like PS1-1-code and includes self-contained code for one problem. For some parts – especially if using Matlab – the entire code might be just a few lines.
(3) Finally a PDF file that shows all the results you need for the problem set. This will include the images appropriately labeled so it is clear which section they are for and the small number of written responses necessary to answer some of the questions. Also, for each main section, if it is not obvious how to run your code please provide brief but clear instructions.

1 There are a few images located in the project directory at: http://www.cc.gatech.edu/~afb/classes/CS4495-Fall2014/ProblemSets/PS2/Data. For this first problem we provided a test pair leftTest.png and rightTest.png that are just a central square moved 2 pixels to the right.





Implement the basic stereo algorithm of taking a window around every pixel in one image and search for the best match along the same scan line in the other image. **You will do this both left to**

right and right to left. Remember: because of depth changes (discontinuities) some pixels visible in the left image are not in the right and vice a versa. So you will match in both directions.

For this part implement the simplest thing imaginable: look for the smallest difference between the template window (source) and the proposed location window. Use the *sum of squared differences measure (SSD).* I am going to take the definitions from: http://software.intel.com/sites/products/documentation/hpc/ipp/ippi/ippi_ch11/ch11_image_proximity_measures.html

SSD is defined by:

$$S_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[t(j,\ i) - x\left(r + j - \frac{tplRows}{2},\ c + i - \frac{tplCols}{2}\right)\right]^2$$

Basically you just sum up the squares. A "good" match, then, is when this value is at a minimum. That is, you are looking for the same image patch in both images. **YOU MAY NOT USE BUILT-IN SSD FUNCTIONS THAT BASICALLY DO THIS ENTIRE STEP.** But it is easy in Matlab to sum the squares of a matrix; check out the operator: .^2 (Yes you need the 'dot'.)

a) Implement the SSD match algorithm, and create a disparity image D(x,y) such that L(x,y) = R(x+D(x,y),y) when matching from left to right. Also match from right to left.

Output: First , show $D_L(x,y)$ [matching from left to right] and $D_R(x,y)$[matching from right to left] as **images**. They should show a central square moved 2 pixels to the left or right. These disparity images may need to be **scaled** and shifted (in intensity) to display correctly. An ideal solution would be zero outside and plus or minus 2 in the middle. Of course the world is not ideal.

Output: Now *plot* your disparities as a 3D plot. The easiest way to do this is to use the surf function. For example:

```
surf((double(mydisp)),'FaceColor','interp','EdgeColor',…
    'none','FaceLighting','phong')
```

Gives a nice plot that allows you to manipulate the view. This will also make it easier for you (and us) to see what you've recovered. You might have to scale or clip you disparities to make it look good.

2 Now we're going to try this on a real image pair: proj2-pair1-L .png and proj2-pair1-R .png. Since these are color images create gray scale versions. You can use **rgb2gray** or your own function.

a) Again apply your SSD match algorithm, and again create a disparity image D(x,y) such that L(x,y) = $R(x+D_L(x,y),y)$ when matching from left to right. Also match from right to left.

Output: Show $D_L(x,y)$ [matching from left to right] and $D_R(x,y)$[matching from right to left] as images. Again, these disparity images may need to be scaled and shifted to display correctly.

b) Also in this directory are the ground truth disparity images proj2-pair1-Disp-L .png and proj2-pair1-Disp-R .png. Compare your results.

Output: description of the differences between your results and the ground truth

3 SSD is not very robust to certain perturbations. We're going to try to see the effect of some perturbations:

a. Add some Gaussian noise to the image; either one or both. Make the noise sigma big enough that you can tell some noise has been added. Run SSD again.
Output: Disparity images and analysis of result compared to part 2.

b. ***Instead*** of the Gaussian noise, increase the contrast (multiplication) of one of the images by just 10%. Run SSD again.
Output: Disparity images and analysis of result compared to part 2.

4 Now you're going to **use** (not implement yourself unless you want) an improved method, called ***normalized correlation*** – this is discussed in the book. The basic idea is that we think of two image patches as ***vectors*** and to compute the angle between them – much like normalized dot products. This can be written as:

$$R_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j,\ i) \cdot\ x\left(r+j-\frac{tplRows}{2},\ c+i-\frac{tplCols}{2}\right)$$

where the above is just the explicit dot product which then has to be normalized by the magnitude of each:

$$\rho_{tx}(r,\ c) = \frac{R_{tx}(r,\ c)}{\sqrt{R_{xx}(r,\ c)R_{tt}\left(\frac{tplRows}{2},\ \frac{tplCols}{2}\right)}}$$

a) Implement a window matching stereo algorithm using some form of normalized correlation. **Matlab has its own function `normxcorr2(template, A)`** which implements:

$$\gamma(u,v) = \frac{\sum_{x,y}\left[f(x,y)-\overline{f}_{u,v}\right]\left[t(x-u,y-v)-\overline{t}\right]}{\left\{\sum_{x,y}\left[f(x,y)-\overline{f}_{u,v}\right]^2\sum_{x,y}\left[t(x-u,y-v)-\overline{t}\right]^2\right\}^{0.5}}$$

OpenCV has a variety of relevant functions as well such as CV_TM_CCOEFF_NORMED**. You MAY use these built in normalized correlation functions.**

Test it on the original images both left to right and right to left.
Output: disparity images and description of how it compares to the SSD version and to the ground truth

b) Now test it on both the Gaussian noise and contrast boosted versions from 2a and 2b.
Output: Again disparity images and analysis of results.

EXTRA CREDIT

5X There is a second pair of images: proj2-pair2-L .png and proj2-pair2-R .png

a. Try your algorithms on this pair. Play with the images – smooth, sharpen, etc. Keep comparing to the ground truth.
Output: disparity images and analysis of what it takes to make stereo work using a window based approach.

6X For either part 2a or 4a or 5, convert your disparity images into something that looks like depth – or is at least proportional to depth – and plot that as a 3D plot.

a. **Output**: your best shot at this.