Problem Set 3: Window-Based Stereo Matching

Description

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

What to submit

Download and unzip: ps3.zip

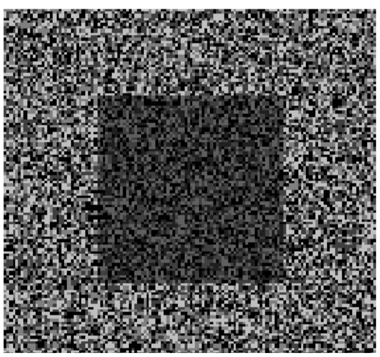
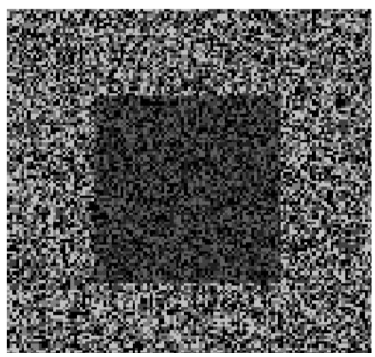
ps3/

- input/ - input images, videos or other data supplied with the problem set
- output/ - directory containing output images and other files your code generates
- ps3.py - main code for completing each part, esp. function calls
- *.py - Python modules, any utility code
- ps3_report.pdf - a PDF file with all output images and text responses

Zip it as `ps3.zip`, and submit on T-Square.

Questions

1. Use the pair `pair0-L.png` and `pair0-R.png` - a central square moved 2 pixels horizontally:

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 image 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).* We are going to take the definitions from: https://software.intel.com/en-us/node/504333

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.

a. Implement the SSD match algorithm as function **`disparity_ssd`**`(L, R)` that returns a disparity image `D(y,x)` such that `L(y,x) = R(y,x+D(y,x))` when matching from left (L) to right (R).

   The images supplied to this function (L and R) should be of the same size (height, width), single-channel (grayscale), and of `double` type with values in range `[0.0, 1.0]`. The disparity map returned should be of the same size as L and R, of `double` type, and each element should contain the disparity at that point, in pixels.

   Apply it to the two test images, matching from left to right:
   ```
   L = cv2.imread(... 'pair0-L.png' ...) * (1 / 255.0)  # scale to [0, 1]
   R = cv2.imread(... 'pair0-R.png' ...) * (1 / 255.0)
   D_L = disparity_ssd(L, R)
   ```

   Also match from right to left:
   ```
   D_R = disparity_ssd(R, L)
   ```

   They should indicate a central square moved 2 pixels to the left or right, e.g., D_L should have value -2 in the approximate region of the central square, 0 elsewhere.

   Function: `disparity_ssd`
   Output: Save disparity images:
   - D_L`(y,x)` [matching from left to right] as `ps3-1-a-1.png`
   - D_R`(y,x)` [matching from right to left] as `ps3-1-a-2.png`
   These disparity images may need to be scaled and shifted to display/write correctly.

2. Now we’re going to try this on a real image pair: `pair1-L.png` and `pair1-R.png`. Note that these are color images - so make sure you read in grayscale or convert.

   a. Again, apply your SSD match function, and create a disparity image `D(y,x)` such that `L(y,x) = R(y,x+D(y,x))` when matching from left to right. Also, match from right to left.
   Output: Save disparity images, scaling/shifting as necessary:
   - D_L`(y,x)` [matching from left to right] as `ps3-2-a-1.png`
   - D_R`(y,x)` [matching from right to left] as `ps3-2-a-2.png`

   b. In the input directory are ground truth disparity images pair1-D_L .png and pair1-D_R .png. Compare your results.
   Output: Text response - description of the differences between your results and ground truth.

3. SSD is not very robust to certain perturbations. We're going to try to see the effect of perturbations:
    a. Using pair1, add some Gaussian noise, either to one image or both among `pair1-L.png` and `pair1-R.png`. Make the noise sigma big enough that you can tell some noise has been added. Run SSD match again.
    Output: Disparity images (D_L as `ps3-3-a-1.png` and D_R as `ps3-3-a-2.png`), text response - analysis of result compared to question 2.
    b. Instead of the Gaussian noise, increase the contrast (multiplication) of one of the images by just 10%. Run SSD match again.
    Output: Disparity images (D_L as `ps3-3-b-1.png` and D_R as `ps3-3-b-2.png`), text response - analysis of result compared to question 2.

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

    The explicit dot product of two image patches (treated as flat vectors) is:

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

    This result is then normalized:

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

    a. Using some form of normalized correlation, implement a window matching stereo algorithm. Again, write this as a function **`disparity_ncorr`**`(L, R)` that returns a disparity image `D(y,x)` such that `L(y,x) = R(y,x+D(y,x))` when matching from left (L) to right (R).
    OpenCV has a variety of relevant functions and supported methods such as CV_TM_CCOEFF_NORMED, which implement correlation similar to:

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

    You MAY use these built-in normalized correlation functions.

    Test it on the original images both left to right and right to left (pair1-L.png and pair1-R.png).
    Output: Disparity images (D_L as `ps3-4-a-1.png` and D_R as `ps3-4-a-2.png`),
    Text response - description of how it compares to the SSD version and to the ground truth.

b. Now test it on both the noisy and contrast-boosted versions of pair1 from 3-a and 3-b.
Output: Disparity images (Gaussian noise: D_L as `ps3-4-b-1.png` and D_R as `ps3-4-b-2.png`; contrast-boosted: D_L as `ps3-4-b-3.png` and D_R as `ps3-4-b-4.png`),
Text response - analysis of results comparing original to noise and contrast-boosted images.

5. Finally, there is a second pair of images: pair2-L.png and pair2-R.png
   a. Try your algorithms on pair2. Play with the images – smooth, sharpen, etc. Keep comparing your results to the ground truth (pair2-D_L.png and pair2-D_R.png).
   Output: Disparity images (D_L as `ps3-5-a-1.png` and D_R as `ps3-5-a-2.png`), Text response - analysis of what it takes to make stereo work using a window based approach.