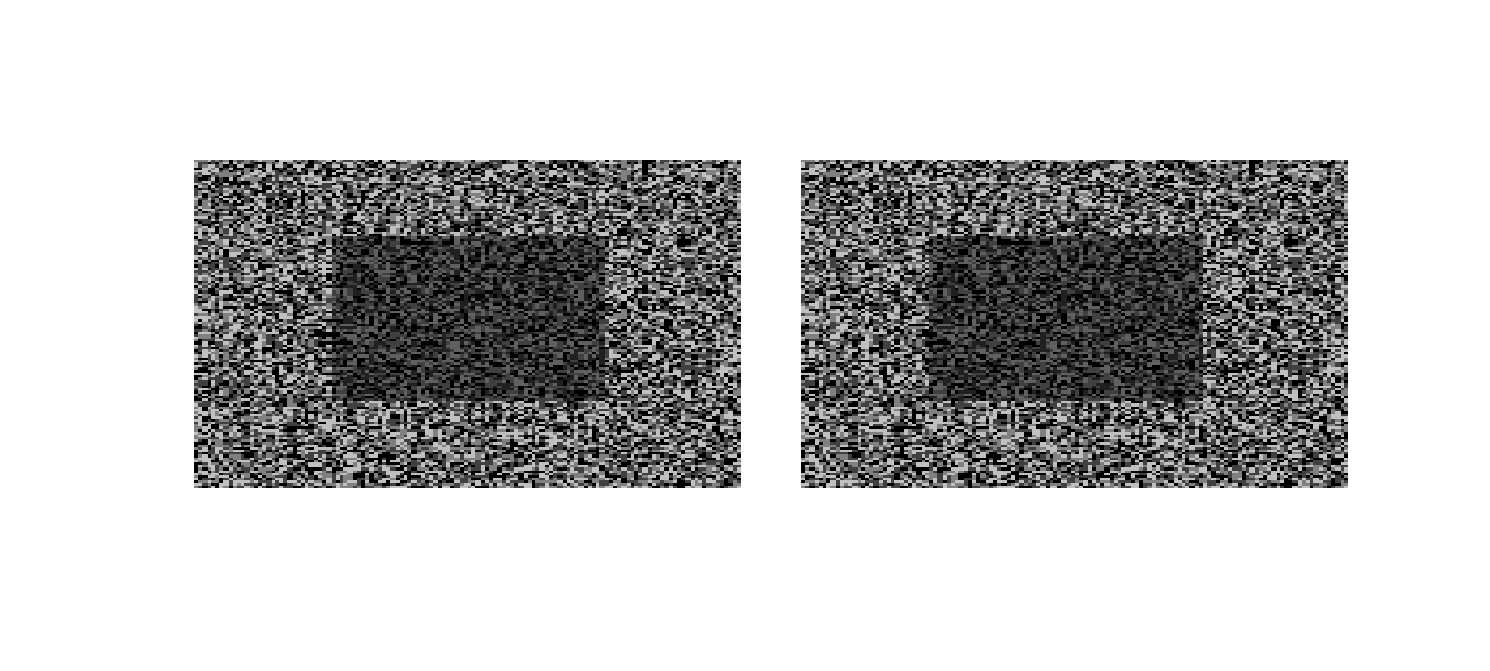
Instruction:

* Submit your code after it has been run and place the output images in the output folder.
* You may use any external lib. It is recommended to work with: numpy, opencv (cv2).

Question 1:

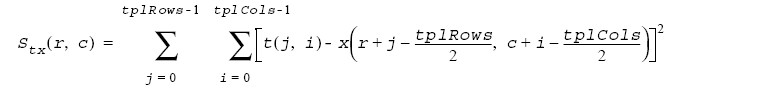
Use the pair pair0-L.png and pair0-R.png - a central square moved 2 pixels horizontally:



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 and vice a versa. So you will match in both directions.

cv2.matchTemplate(………, method=cv2.TM\_SQDIFF\_NORMED) will help

SSD is defined by:



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.

1. 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).  
     
   Apply it to the two test images, matching from left to right:  
           L = im2double(imread(fullfile('input', pair0-L.png')));  
           R = im2double(imread(fullfile('input', pair0-R.png')));  
           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 file**: disparity\_ssd.m containing function disparity\_ssd (identical name)  
   **Output**: Save disparity images:  
   - DL(y,x) [matching from left to right] as ps2-1-a-1.png  
   - DR(y,x) [matching from right to left] as ps2-1-a-2.png  
   These disparity images may need to be scaled and shifted to display/write correctly.

Question 2:

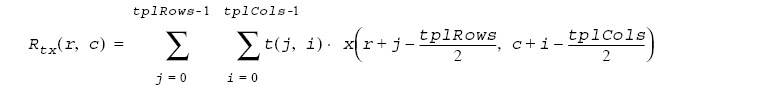
Now we’re going to try this on a real image pair: pair1-L .png and pair1-R .png.  Since these are color images, create grayscale versions. You can use **rgb2gray** or your own function.

1. 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:  
   - DL(y,x) [matching from left to right] as ps2-2-a-1.png  
   - DR(y,x) [matching from right to left] as ps2-2-a-2.png
2. Also in the input directory are ground truth disparity images pair1-D\_dropL .png and pair1-D\_R .png.  Compare your results.  
   **Output**: Text response - description of the differences between your results and ground truth.

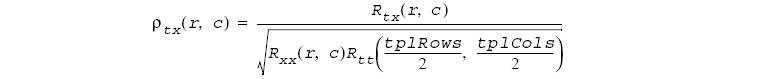
Question 3:

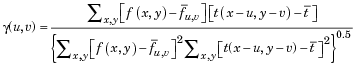
Now you’re going to *use* (not implement yourself unless you want) an improved method, called ***normalized correlation*** – this is discussed in the lecture. The basic idea is that we think of two image patches as ***vectors*** and compute the angle between them – much like normalized dot products.

You will find cv2.matchTemplate(……, method=cv2.TM\_CCORR\_NORMED) a useful method  
  
The explicit dot product of two image patches (treated as flat vectors) is:



This result is then normalized:



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

OpenCV has a variety of relevant functions and supported methods 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 (pair1-L.png and pair1-R.png).  
**Output**: Disparity images (DL as ps2-4-a-1.png and DR as ps2-4-a-2.png), text response - description of how it compares to the SSD version and to the ground truth.

1. Now test it on both the noisy and contrast-boosted versions from 2-a and 2-b.  
   **Output**: Disparity images (Gaussian noise: DL as ps2-4-b-1.png and DR as ps2-4-b-2.png; contrast-boosted: DL as ps2-4-b-3.png and DR as ps2-4-b-4.png), text response - analysis of results comparing original to noise and contrast-boosted images.

Question 4: Edge detection

Implement the Sobel edge detection algorithm (you may use the numpy convolutions but not the implemented method for Sobel edge detection in cv2)

Use OpenCV and implement Canny edge detection algorithm (learned in lecture only). You may use any OpenCV method except the one that implements Canny edge detection (you can use it for debug)

You need to choose 2 images (any image from google is fine, funny once will get extra credit and make me happy) place them in the input folder run both Sobel and Canny on them and place the output in the output folder.