# Problem Set 5 CMSC 426
# Due: Thursday, May 8

1) Brute-Force, SSD matching.

The class web page contains an image and a template that you can download. Your goal is to find the regions in the image that best match the template.

a) Write code to find the sum of square differences between two rectangular portions of images that are of equal size. d = SSD(R1, R2) should take two matrices representing portions of images and return the SSD between them. Test your function on:

R1 = [1, 0, 0, 2; 2, 1, 0, 1]; R2 = [0, 2, 1, 1; 1, 0, 0, 1];

Turn in your result and code. Hint: You should do this without using any loops, just with matrix operations. If you use loops, you may find that your subsequent code is very slow. **40 pts**

b) Write code to compare a small template to an image in all possible translations of the template, computing the SSD. Only consider translations that place the template completely inside the image. A=brute_force_ssd(T,I) should return a matrix, A, that gives the SSD between T and a subset of I, for every possible shift of T. You only need to shift T by integer values, don't worry about interpolation. Test your code using the template and image from the web page. Display and print the result A with the command imagesc(A). Turn in this display and your code. **30 pts.**

c) Now write code to find the best instances of the template in the image. The best instance is the one that positions the template in the image to minimize the SSD. The second best one has minimal SSD of all templates that don't overlap the best one (if you don't watch this condition, your second best template will probably be shifted one pixel from your best template; not a useful result). Display the position of the six best, non-overlapping instances of the template in the image. You can visualize the position of the templates as you like. For example, you might use a matlab command to draw six red rectangles on top of the image, each the size and position of the template. Turn in a plot of your results and your code. **30 pts.**

2) Challenge Problem: Make the code to Problem 1 faster by building a Gaussian pyramid, and starting the matching at a small scale. Then move to a larger scale, restricting your matching to be near the best match at the smaller scale.

Experiment with starting the matching at different scales. How small a scale can you start with and still get good results?

Turn in pictures showing the Gaussian pyramid for the template and the image. Turn in pictures showing the six best templates you found, at the largest scale. Turn in several of these pictures, illustrating how the results vary depending on the scale you begin matching. Use the functions tic and toc to measure the speedup you gained with multiscale matching, and report the results. Also turn in your code. **20 pts.**

3) Challenge Problem: The file on the web page called: 'Problem Set 5 Image.mat' contains the positions of 24 points in 4 images, and the 3D locations of the first four of these points. Execute the command load('Problem Set 5 Image.mat') and the variable I will contain the 8x24 image matrix, and the variable P4 will contain a 3x4 matrix containing the x,y,z world coordinates of the first four points. Use the factorization algorithm discussed in class and in the text to find the 3D coordinates of all 24 points, up to a linear transformation. Then find a linear transformation (with translation) that will align the first four points with the four points described in P4. This will give you the world coordinates of all 24 points. Plot the x and y coordinates of these points with a command such as plot(pts(1,:), pts(2,:), 'ko'); (you will know if you've got the right answer). Turn in the plot and your code. **20 pts.**