Problem Set 5
CMSC 426
Assigned October 25 2012
Due November 1, 2012

Blob Detection

This problem set deals with some of the key components needed to perform blob detection. This was discussed in class and is also described in: David G. Lowe, **"Distinctive image features from scale-invariant keypoints,"** *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110, which is available at: http://www.cs.ubc.ca/~lowe/keypoints/.

1. **20 Points: Gaussian Smoothing.** To find Blobs, you must first generate a multiscale representation of images, by smoothing them repeatedly. We choose the magic numbers sigma = 1.6 and k = $2^{1/3}$. Given these numbers, we want to smooth an input image using a Gaussian with a standard deviation of sigma, sigma*k, sigma*k*k, sigma*k*k*k, …. You may want to use fspecial to help twith this. Also, for this problem set, it is important to be sure that at the boundary of the image, you replicate the nearest pixel when smoothing. Look at the options for imfilter to see how to do this.

   Note that there are two ways to do this. Suppose you have smoothed an image with a Gaussian that has a standard deviation of A, and you want to obtain the image smoothed with a Gaussian that has a standard deviation of B, with B>A. You can just smooth the image with a Gaussian with std dev. of B. Or, you can take the smoothed image, and smooth it again with a Gaussian whose std dev. is sqrt(B*B-A*A). This second approach will use a smaller filter and be faster. Either is acceptable.

   As an example, I include the results you get from smoothing a simple image with a white square on a black background. This is produced with the Matlab commands:

   ```
   K = zeros(100,100);
   K(45:55,45:55) = 255;
   ```

   On the web page, I show examples of smoothing this image six times.

   Smooth the dog image provided on the class web page. Smooth the dog six times, with Gaussians that have standard deviations ranging from sigma to sigma*k*k*k*k*k. Include these six images when you turn in your assignment.

2. **40 points: Blob Detection.** To detect blobs, you must repeatedly smooth your images. Then you subtract the results at each scale from the next scale. This

gives you a set of difference images, which are equivalent to convolving the image with Difference of Gaussian (DoG) filters at multiple scales. Then identify local extrema by comparing a pixel in a DoG image to all neighboring pixels at that scale and at the scales that are the next larger and smaller. So to be a local maximum, a pixel must be bigger than 26 neighbors (8 at the same scale, 9 each at two adjacent scales). Similarly, a local minima must be smaller than all neighbors. In addition, an extrema will only be salient if its absolute value is bigger than .03 (this assumes that the image values range between 0 and 1, not 0 and 255.).

In order to help you test your code, the class web site shows the results I obtain on a simple black image with a white square in the middle. Only one blob is detected, in the middle of the image (the red circle).

I have to say, while these basic steps aren't too hard, it took me quite a while to get this to work. So watch out for subtle issues in implementing this.

You should run your code on the dog image. You should detect all extrema after smoothing 13 times (that means you can compute 12 DoG images, and check 10 of them for extrema. The first and last DoG images needn't be checked, because you can't compare them to scales above and below them). I am including my program's output. The extrema I compute are shown as red circles. (I generate these red circles using plot. I use the option 'MarkerSize' to make the size of the circle equal to twice the standard deviation of the Gaussian at the scale at which they were detected. This looks something like this:

```
plot(x,y,'ro', 'MarkerSize', 2*sigma);
```

Hand in three images that visualize the result of your blob detector on the dog image, the swan image, and on the simple square image.

3. **20 points: Challenge Problem.** As described in David Lowe's paper, this process can be made much more efficient by subsampling the image, every time it has been smoothed with a standard deviation of 1.6. This would correspond to reducing the size of the image by half after every three smoothings. This is a little subtle, of course, because you need to compare each scale to neighboring scales, and because you need to recover the positions of the extrema in the original image. But this is a challenge problem.

   In addition to including your code, show the results that you obtained in an image, and include a brief description of how much of a speed-up you found. Use the profiler or tic and toc commands to measure the run time.