CS 663 : Digital Image Processing : Assignment 3

Instructor : Suyash P. Awate

Due Date : 20 Sep 2015, Sunday, 11:55 pm

Instructions for submission are at `www.cse.iitb.ac.in/~suyash/cs663/submissionStyle.pdf`

The folder structure is at
`www.cse.iitb.ac.in/~suyash/cs663/assignment3_FourierSegmentation.zip`

Note: The input data / image(s) for a question is / are present in the corresponding data/ subfolder.

5 points are reserved for submission in the described format.

1. (25 points) Fourier Analysis.

   Input image: `1/data/boat.mat`.

   Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes.

   Corrupt the image with independent and identically-distributed additive zero-mean Gaussian noise with standard deviation set to 10% of the intensity range. Note: in Matlab, randn() gives random numbers drawn independently from a Gaussian with mean 0 and standard deviation 1.

   Write code for Butterworth filtering in the frequency domain. Use a Butterworth filter with $n = 2$ (see slides for the equation).

   Define the root-mean-squared difference (RMSD) as the square root of the average, over all pixels, of the squared difference between a pixel intensity in the original image and the intensity of the corresponding pixel in the filtered image, i.e., given 2 images A and B with N pixels each, RMSD$(A, B) := \sqrt{(1/N)\sum_p (A(p) - B(p))^2}$, where $A(p)$ is the intensity of pixel p in image A.

   Apply the Butterworth filter to smooth the image. Tune the frequency parameter D_0 to minimize the RMSD between the filtered and the original image.

   • Write a function `myButterworthFiltering.m` to implement this.

   • Show the Butterworth filter, in the frequency domain, as an image.

   • Show the original, corrupted, and filtered versions side by side, using the same (gray) colormap.

   • Report the optimal parameter value found, say D_0^* along with the optimal RMSD.

   • Report RMSD values for filtered images obtained with (i) $0.95D_0^*$ and (ii) $1.05D_0^*$.

   • Consider a circular / disk mask around the pixel in the Fourier-transform image that corresponds to the zero-frequency in the centered image (i.e., after applying fftshift() in Matlab). Report the

indices of this center pixel (containing the so-called "DC" offset). Let the radius of the circle be R pixels. Report radii R such that the sum, over pixels inside the mask, of the squared magnitudes of the Fourier transform is closest to (i) 88%, (ii) 91%, (iii) 94%, (iv) 97%, and (v) 99% of the total energy of the Fourier spectrum. Apply each of these masks to the Fourier transform, compute the inverse Fourier transform, take the magnitude of the resulting image values, and report the RMSDs between the resulting images and the original image.

2. (30 points) Harris Corner Detection.

   Input image: `2/data/boat.mat`.

   Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes.

   Shift and rescale the intensities in the image to lie within the range $[0, 1]$.

   Implement the Harris corner detector algorithm. The parameters underlying this algorithm are: two Gaussian smoothing levels involved in computing the structure tensor, the constant k in the corner-ness measure. Tune these three parameters to get the best results.

   • Write a function `myHarrisCornerDetector.m` to implement this.

   • Display the derivative images, corresponding to the derivatives along the X and Y axes.

   • Display the images (along with a colormap) of the two eigenvalues of the structure tensor, evaluated at each pixel.

   • Display the image (along with a colormap) of the Harris corner-ness measure. Positive values in this image must correspond to a corner structure in the image.

   • Report all three parameter values used.

3. (40 points) Image Segmentation using mean shift.

   Input image: `3/data/baboonColor.png`.

   Take this 512×512 pixel image, smooth it using Gaussian convolution with standard deviation 1 pixel width, and subsample the smoothed image by a factor of 2 in each spatial dimension to produce a 256×256 image. Use this smaller-sized image for the following experiment. If this image is still too large for your computer's memory, then you may resize further.

   Implement the algorithm for mean-shift image segmentation using both color (RGB) and spatial-coordinate (XY) features. Tune parameters suitably to get a segmented image with at least 5 segments and no more than 50 segments. To improve code efficiency, you may use Matlab functions like knnsearch(), bsxfun(), etc. For this image, about 20 iterations should be sufficient for reaching close to convergence. You may select a random subset of nearest neighbors, in feature space, for the mean-shift updates to reduce running time. Each iteration can run in about 10-20 seconds on a typical personal computer.

   • Write a function `myMeanShiftSegmentation.m` to implement this.

   • Display the (i) original image along with (ii) the segmented image that shows color-coded pixels (and, thus, segments) using the color component of the converged feature vectors.

   • Report the following parameter values: Gaussian kernel bandwidth for the color feature, Gaussian kernel bandwidth for the spatial feature, number of iterations.