CS 663 : Digital Image Processing : Assignment 2

Instructor : Suyash P. Awate

Due Date : ****

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) **Spatially varying blurring**

   Input Images: 1/data/flower.jpg, 1/data/bird.jpg

   Each input image has a visually distinct foreground and a background. In case of *petal.jpg*, the flower at the center is the foreground and rest is background. Similarly, in case of *bird.jpg*, the bird is the foreground and rest is background. Your task is to keep the foreground intact and blur the background according to a spatially varying scheme through the following steps:

   (a) (5 Points) Generate a mask M with values 1 for the foreground and 0 for the background. Come up with a partially/fully automatic scheme to generate this mask. Fully manual solutions will not be considered.

   (b) (20 Points) Once the mask is generated, your task is to blur background using a spatially varying kernel. The kernel will be a uniform disc, with weights summing to 1. However, it's radius at every point in the background will vary as a function in the following manner. Let P be a point in the background and d_p be the shortest distance from P to the boundary of the mask *M*. The radius r of the blur kernel will be as follows.

   $$r = \begin{cases} \mathrm{d}_p, & \text{if } \mathrm{d}_p \leq d_{thresh} \\ \mathrm{d}_{thresh}, & \text{if } \mathrm{d}_p \geq d_{thresh} \end{cases}$$

   where d_{thresh} = 20 for *petal.jpg* and d_{thresh} = 40 for *bird.jpg*.

   Write a function *mySpatiallyVaryingKernel.m* implementing this scheme. Note that you **SHOULD NOT** blur the foreground.

   (c) We will run your main.m file and the following items should be displayed at the run time. Not displaying any of the below items will attract a penalty of 5 points per bullet. Since there are many items to be displayed, all the sub-items for a single bullet should be displayed as a collage in a single matlab figure. Not adhering to this and creating an explosion of images on the evaluator's screen will attract another penalty of 5 points.

      i. (a) The mask M (b) the foreground image with background intensities set as 0 and (c) the background image with foreground intensities set a 0. We will visually evaluate

a, b, c to decide whether you have got the mask correctly, especially at the boundaries. Also, in the report, explain the process you adopted to generate the mask. Submit your code for the same.

ii. A **contour plot** or a **jet plot** of r, demonstrating the variation of r w.r.t the distance from the mask boundary.

iii. Display the kernels you used to blur the background at distances $0.2d_{thresh}$, $0.4d_{thresh}$, $0.6d_{thresh}$, $0.8d_{thresh}$, d_{thresh} from the mask boundary.

iv. The output image, with visually distinct sharp foreground and blurred background

You may optionally downsize the images to half their original sizes, if your code takes a while to run.

2. (30 points) Edge-preserving Smoothing using Bilateral Filtering.

Input images:
(1) `2/data/barbara.mat`
(2) `2/data/grass.png`
(3) `2/data/honeyCombReal.png`

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 5% 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 bilateral filtering (standard "slow" algorithm is also fine) and apply it (one pass over all pixels) to all the input images. For efficiency in Matlab, the code should, ideally, have maximum 2 "for" loops to go over the rows and columns of the image. At a specific pixel "p", the data collection with a window, weight computations, and weighted averaging can be performed without using loops.

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.

Tune the parameters (standard-deviations for Gaussians over space and intensity) to minimize the RMSD between the filtered and the original image.

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

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

• Show the mask for the spatial Gaussian, as an image.

• Report the optimal parameter values found, say σ^*_{space} and $\sigma^*_{\text{intensity}}$, along with the optimal RMSD.

• Report RMSD values for filtered images obtained with (i) $0.9\sigma^*_{\text{space}}$ and $\sigma^*_{\text{intensity}}$, (ii) $1.1\sigma^*_{\text{space}}$ and $\sigma^*_{\text{intensity}}$, (iii) σ^*_{space} and $0.9\sigma^*_{\text{intensity}}$, and (iv) σ^*_{space} and $1.1\sigma^*_{\text{intensity}}$, with all other parameter values unchanged.

3. (40 points) Edge-preserving Smoothing using Patch-Based Filtering.

   Input images:
   (1) `2/data/barbara.mat`
   (2) `2/data/grass.png`
   (3) `2/data/honeyCombReal.png`

   Redo the previous problem using patch-based filtering. If you think your code takes too long to run, (i) resize the image by subsampling by a factor of 2 along each dimension, *after* applying a Gaussian blur of standard deviation around 0.66 pixel width and (ii) apply the filter to the resized image.

   Use 9×9 patches. Use a Gaussian, or clipped Gaussian, weight function on the patches to make the patch more isotropic (as compared to a square patch). Note: this will imply neighbor-location-weighted distances between patches. For filtering pixel "p", use patches centered at pixels "q" that lie within a window of size approximately 25×25 around "p".

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

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

   • Show the mask used to make patches isotropic, as an image.

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

   • Report RMSD values for filtered images obtained with (i) $0.9\sigma^*$ and (ii) $1.1\sigma^*$, with all other parameter values unchanged.