CS 663 : Digital Image Processing : Assignment 2

Instructor : Suyash P. Awate

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. (15 points) Image Sharpening.

   Input images: (1) `1/data/superMoonCrop.mat`, and (2) `1/data/lionCrop.mat`.

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

   Write code for image sharpening using unsharp masking and apply it to both the input images.

   To compare the original and filtered images, linearly contrast-stretch them to the same intensity range, say, $[0, 1]$.

   Tune the parameters (Gaussian standard-deviation parameter and the scaling parameter) to your best judgment, but such that the sharpening in the image is clearly visible.

   You may use the following Matlab functions: fspecial() and imfilter().

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

   • For each image, show the original and sharpened versions side by side, using the same (gray) colormap.

   • Report the tuned parameter values for each image.

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. (50 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.