Assignment 1

Digital Image Processing - M25

Deadline: 29th Aug, 2025 11:59PM

General Instructions:

1. Follow the specified repository structure.
2. Make sure you run your Jupyter notebook before submitting to save all outputs.
3. Answer any questions asked in a markdown cell in your notebook.
4. Label your plots.

---

1. Spatial Filtering [25 marks]

Implement the following spatial filters. You may use a library to read and write images, but the filtering itself should be your implementation. You may use any image of your choice for the following tasks:

a) **Smoothing Filters**

- **Mean/Average Filter** Create a function that applies a mean filter to an image with a specified kernel size. Apply the mean filter using three different kernel sizes (e.g., 3, 7 and 11) on the same image. Display the original image alongside the three filtered results in a single row of plots. Discuss how increasing kernel size affects noise reduction and image blurring. [4]
- **Gaussian Filter** Create a function that applies a Gaussian filter to an image, given kernel size and standard deviation parameters. Apply the Gaussian filter with three different kernel sizes and three different standard deviations, resulting in nine filtered images. Arrange and display these results in a 3×3 grid of plots. Discuss the effect of kernel size and standard deviation on smoothing strength and edge preservation. [4]
- **Comparison** Select two different images:
  - One where the mean filter is more effective than the Gaussian filter.

– One where the Gaussian filter outperforms the mean filter.

For each image, apply both filters with appropriate parameters, display the results side-by-side, and explain which filter is better and why. [2]

b) **Edge Detection Filters [5]**

- Create separate functions to implement the Prewitt, Sobel, and Laplacian edge detection filters.
- Apply all three filters to a chosen grayscale image (you may use a smoothed version from step 1 to improve results).
- Display the original image and the three edge-detected images side-by-side.
- Discuss the characteristics of each edge detector and explain the differences in the edge maps produced.

c) **Image Sharpening**

- Create functions for:
  - Unsharp Masking, where the sharpened image is obtained by adding the Laplacian of the image to the original image, i.e.,
    $I'(u, v) = I(u, v) + \nabla^2 I(u, v)$
  - Highboost Filtering, where the sharpened image is calculated by adding a scaled difference between the original and a Gaussian blurred version of the image to the original, i.e.,
    $I'(u, v) = I(u, v) + a \times \big(I(u, v) - \text{gaussBlur}(I(u, v))\big)$

  Display the original, unsharp masked, and highboost filtered images side-by-side. [8]
- Discuss how increasing the boost factor 'a' (in high boost filtering) affects the output. Plot the output for at least 3 different a-values. [2]

2. Histogram Matching [25 marks]

You are given two images: source image and target image. Your task is to:

(a) Source Image (b) Target Image

Figure 1: Given images

a) **Implement the following functions:**

- `compute_histogram_2d(image_channel)`: Compute the histogram for a given image channel. Do not use the inbuilt functions for computing histogram. [3]
- `compute_cdf(histogram)`: Compute the cdf of the histogram. [3]
- `match_histograms_2d(source, target)`: Match the histogram of the source image channel to that of the target image channel. Do not use the inbuilt match histograms function. [8]

Once again, the three functions that are to be implemented are to be done without the use of any library (other than read/write/display).

b) **Visualization:** Create a 2×3 subplot showing the source image, target image and the histogram matched image with each channel matched separately in the first row. In the second row, plot the respective histograms (with the three channels represented with different colors). [4]

c) **Observations:**

- What happens when you match an image to its own histogram (self-matching)? Verify with a plot. [1]
- When matching a bright image to a dark image, what happens to the noise visibility in the darker regions and why? Verify with a plot. You may choose any one bright image and one dark image and match them. [4]
- If the target histogram is uniform, how does the result differ from standard histogram equalization? [2]

Figure 2: Stego Image

3. Steganography With Bit-plane Slicing [25 marks]

A stego image is an image that contains hidden information embedded within it using steganography techniques. You are given a stego image that contains a hidden secret image embedded into one of its bit-planes.

a) Find which bit-plane contains the secret by extracting and plotting all 8 bit-planes of the image. [8]

b) Explain the difference between embedding the secret in the Most Significant Bit (MSB) versus the Least Significant Bit (LSB) — discuss implications for visibility and robustness. [3]

c) Create your own stego image. Implement a function `encrypt(cover_img, secret_img)` to embed a (binary) secret image of your choice into chosen bit-plane(s) of a cover image. Ensure the secret is not visible to the naked eye after embedding. Also implement a function `decrypt(img)` to get the secret image based on your encryption. [12]

d) Till now we assumed a binary secret image and hence can be encrypted in one bit plane. Can you think of any ways to encrypt a grayscale secret image? [2]

4. HDR Images [25 marks]

In digital photography, High Dynamic Range (HDR) imaging is used to overcome the limitations of standard 8-bit images, which cannot represent the full range of real-world brightness levels. Explain why HDR imaging is required. [1]

Often, images with different exposures are combined to create a single HDR image. You are provided with three images of the same scene: low exposure, medium exposure, and high exposure. Your task is to merge these into a single HDR image.

(a) Low exposure (b) Medium exposure (c) High exposure

Figure 3: Input images with different exposures

a) **HDR Merge Function [10]:** Implement the function:

`hdr_merge(images, exposures)`

using a linear combination of the given images with inverse exposure weighting:

$$HDR(x,y) = \frac{\sum_{i=1}^{N} \frac{I_i(x,y)}{t_i}}{\sum_{i=1}^{N} \frac{1}{t_i}}$$

where:

- $I_i(x,y)$ is the pixel value of image i at (x,y)
- t_i is its exposure time

Use the following exposure times:

Low exposure:	0.25 sec
Medium exposure:	2.5 sec
High exposure:	15 sec

b) **Tone Mapping Function [10]:** Implement the function:

`tone_map(hdr_merged_img)`

Since the HDR merge may produce pixel values beyond the 0–255 range that displays can show, apply: Logarithmic compression to strongly compress high values followed by Gamma correction with $\gamma = 2.2$ to adjust brightness to match human visual perception.

c) **Visualization [1]:** Create a 1×4 subplot showing the given images and the final HDR result after tone mapping

d) **Observations [3]:** Compare and discuss the details preserved or lost in the HDR output compared to the original exposures. Specifically, comment on the shadows, highlights and the overall contrast.