Assignment 1

Image Processing Fundamentals

Academic integrity

This is an individual assignment. Discussing concepts is encouraged; sharing code is not. You may consult documentation or AI tools to remind yourself of syntax, but all code and explanations must be your own. Cite any borrowed snippets. Be ready to explain your solutions. We may conduct brief oral checks.

Why this assignment?

This assignment teaches you to read, transform, filter, and analyze images. You'll experiment with both spatial and frequency domain methods and compare their behavior.

By the end of the assignment, you will be able to:

1. Explain and manipulate binary, grayscale, and RGB images and common file formats (PNG, JPEG, TIFF).
2. Apply and interpret intensity transforms (negative, log, gamma), geometric operations (crop, rotate, interpolate).
3. Implement and use spatial filters (moving average/box, Gaussian, Laplacian, Sobel) and articulate low-pass vs high-pass behavior.
4. Implement frequency-domain filters (ideal/Gaussian/Butterworth LPF/HPF) and a band-reject (notch) filter; compare results to spatial filtering.
5. Detect horizontal and vertical edges and connect them to a simple robotics-style perception task.

Programming and data & tools

- Use MATLAB with the Image Processing Toolbox, or Python with equivalent libraries (e.g., OpenCV, scikit-image).
- Built-in demo images to use: cameraman.tif, peppers.png, coins.png, rice.png.
- Capture one photo yourself (phone is fine): for example, place a strip of black electrical tape on a white sheet of paper at a slanted angle (not perfectly horizontal or vertical) to represent a robot's lane/line. This image will later be used to estimate the line's orientation angle, as if a robot were following it.

Implementation hint

- Normalize images to double in [0,1] for arithmetic; convert back to uint8 for display if needed. For custom filters, pad replicates or handle borders explicitly.
- Sobel kernels:

$$Sx = [-1\ 0\ 1;\ -2\ 0\ 2;\ -1\ 0\ 1\ ];$$
$$Sy = [\ 1\ 2\ 1;\ \ 0\ 0\ 0;\ -1\ -2\ -1\ ];$$

Deliverables checklist

- One MATLAB Live Script (A1_Lastname_Firstname.mlx) with clear section headings and embedded figures; also upload any helper .m functions you wrote.
- Your own photo (.jpg/.png) used in Part E.
- Clear section titles A–F matching the parts below.
- Code cells with comments, figures with titles/captions, and short written answers for each Reflect prompt.
- All .m functions you created (e.g., movingAverage1D.m, boxFilter2D.m, sobelXY.m).

Marking rubric (100 pts)

Section	Points
A. Image types and formats	10
B. Intensity and geometry	12
C. Spatial filtering	35
D. Frequency filtering	25
E. Edges and a robotics mini task	12
F. Reflections	6

Quality expectations: Clear plots (axis labels, colorbars where meaningful), side-by-side comparisons, consistent random seeds if used, and concise, insightful reflections (2–6 sentences each). Code should be readable and modular.

Part A: Image types and file formats (10 pts)

Goal: Understand binary vs grayscale vs RGB; compare PNG/JPEG/TIFF; see lossy vs lossless tradeoffs.

1) Loading and inspecting
   Load peppers.png (RGB) and cameraman.tif (grayscale). Display size, class, and range. Convert peppers.png to grayscale with rgb2gray and save in a matrix as original_pepper_gray. Display the images using imshow()

2) Format
   Save the original_pepper_gray as PNG, TIFF, and JPEG at qualities 95, 75, 50. What are their sizes on disk.

3) Reload the saved images in different formats. Compute the MSE (Mean Squared Error) between the original_pepper_gray image and the compressed images. Then compute PSNR (Peak Signal-to-Noise Ratio) for each one. Display a montage of small crops showing compression of any artifacts.

4) Binary image
   Threshold coins.png to a binary image using imbinarize() with global and adaptive options. Show the binary image and its inverted version; report foreground/background pixel counts.

Reflect

1. Which formats were lossless? How did file size track visual quality?
2. Why is PSNR appropriate for comparing JPEG qualities?
3. What is the difference between the Global and Adaptive binarizing method.
4. What changed visually when you inverted the binary image? Where might binary images be used in robotics?

Part B: Intensity transforms and geometry (12 pts)

Goal: Build intuition for tone mapping and simple geometric operations.

1) Negative, log, gamma
   For cameraman.tif, produce: a negative image, a log-transformed image, and three gamma-corrected images with γ = 0.4, 1.0, 2.5 (with proper scaling). Display the image and show histogram (imhist) for each image.

2) Contrast stretching & equalization
   Write a short function contrastStretch(I, [r1 r2], [s1 s2]) that maps input range [r1,r2] to

[s1,s2]. Apply contrastStretch on cameraman.tif. Compare with built-in functions: imadjust and histeq and comment on differences.

3) Crop & rotate (interpolation)
Crop a 256×256 region of peppers.png and rotate it by 20° using nearest, bilinear, and bicubic interpolation. Zoom 300% and comment on aliasing and smoothness.

Reflect

1. When is $\gamma < 1$ helpful? What about $\gamma > 1$?
2. What visual evidence tells you which interpolation was used?
3. In robotics, how might histogram equalization help a mobile robot navigate?

Part C: Spatial filtering (35 pts)

Goal: Smooth and sharpen images via neighborhood operations.

1) Your own moving averages
Implement movingAverage1D(x, w) that returns a 1-D moving average with window length, w and normalized. Simulate a noisy 1-D signal and validate the function. Then implement boxFilter2D(I, k) (k×k, odd k, normalized). Apply your function on grayscale cameraman.tif. Compare your output to imfilter(I, fspecial('average', k), 'replicate').

2) Gaussian smoothing
Apply a 2-D Gaussian blur (imgaussfilt) with $\sigma = 1, 2, 4$ to to cameraman.tif. Comment on fine detail loss vs noise suppression. Compare the result for each σ.

3) unsharp masking and high-boost filtering
Sharpen cameraman.tif by subtracting a blurred version from the original image.

$$I\_sharp = I + \alpha(I - I\_blur). \quad For\ \alpha \in \{0.5, 1.0, 1.5\}.$$

4) Laplacian
Apply a Laplacian filter (fspecial('laplacian') to cameraman.tif and explain the halo effect.

5) Sobel gradients

Using built-in function imfilter(), write your own function sobelXY.m that

a) Takes an input image I (grayscale)
b) Convolves it with the Sobel kernels Sx and Sy to compute G_x and G_y

$$S_x = [-1\ 0\ 1;\ -2\ 0\ 2;\ -1\ 0\ 1\ ];$$
$$S_y = [\ 1\ 2\ 1;\ \ 0\ 0\ 0;\ -1\ -2\ -1\ ];$$

c) Returns Gx, Gy, the full gradient magnitude G = hypot(Gx,Gy), and orientation $\theta = atan2(G_y, G_x)$

6) Apply your sobelXY(I) function to cameraman.tif. Visualize horizontal and vertical edges by showing |Gx| and |Gy| separately.

Reflect

1. Why do larger kernels blur more? What tradeoffs did you see with σ?
2. How is a high-pass filter related to "original minus low-pass"?
3. What kinds of structures appear more strongly in |Gx| vs |Gy|?

Part D: Frequency-domain filtering (25 pts)

Goal: See how filters act in the Fourier domain and design a notch (band-reject) filter to remove periodic noise.

1) Fourier magnitude and phase
For cameraman.tif, compute F = fftshift(fft2(I)). Display log(1+abs(F)) using imshow() and the phase image. Briefly describe where low vs high frequencies appear.

2) Design LPF/HPF in frequency domain
Create and apply ideal, Gaussian, and Butterworth (n=2) low-pass filters at two cutoff radii (e.g., 20 and 60 pixels) and their corresponding high-pass filters on cameraman.tif. Use ifft2 to reconstruct the filtered images. For the Gaussian case, also apply imgaussfilt() in the spatial domain for comparison. Discuss similarities and differences you observe.

3) Notch (band-reject) to remove periodic noise

   a. Simulate a sinusoidal stripe pattern and add it to cameraman (choose frequencies so bright symmetric spikes are visible off-center in the spectrum).
   b. Build a pair of Gaussian notch-reject filters centered at those spike locations (tunable notch radius).
   c. Apply the combined band-reject filter in the frequency domain and show before/after images and spectra.

Reflect

1. What advantages did the frequency-domain approach give you for removing the synthetic interference?
2. How would you pick cutoff radii?

Part E: Edge detection and a robotics mini task (12 pts)

Goal: Use edges to infer simple scene structures for a robot.

1) On your tape-line photo, apply a mild Gaussian blur ($\sigma \approx 1$). Apply sobelXY you wrote in part C on the blurred image. Visualize |Gx| (vertical edges) and |Gy| (horizontal edges). Threshold the gradient magnitude G to obtain a binary edge map

2) Extract the edge pixels of the line. Use polyfit to fit a straight line through those points. Compute its orientation angle in degrees, overlay the fitted line on the image, and interpret the result in one sentence (e.g., "robot should steer left by 20°").

Reflect

1. Which preprocessing made the line more stable?
2. In real robotic systems, what challenges could cause this method to fail?

Part F: Reflection (6 pts)

Answer briefly (2 - 6 sentences each):

1) Spatial vs frequency domain: When would you prefer one over the other in practice?
2) What's one thing that surprised you in this assignment?