Computer Vision, Fall 2025, IIIT Sri City

Programming Assignment

0.1 Instructions:

- It is strongly recommended that you work on your assignment on an individual basis. If you have any questions or concerns, feel free to talk to the instructor.
- Please turn in Python Notebooks with the following notation for the file name: your-roll-numberQ_no.ipynb.
- Please use the same names for images in your code as mentioned in the problem statements. While evaluating the same names will be used.
- During evaluation, you may be asked to run your code and show the demo.
- Copying or any kind of misconduct will be taken seriously.

Q1:Implementation Guidelines (10 points)

The aim of this project is to help you become comfortable with **Python**, **NumPy**, and basic image filtering techniques. Once you have a working image filtering routine, creating hybrid images will follow naturally. You are expected to implement five functions, each depending on the previous one:

1. `cross_correlation_2d`
2. `convolve_2d`
3. `gaussian_blur_kernel_2d`
4. `low_pass`
5. `high_pass`

- **Image Filtering:** Filtering (or convolution) is one of the most fundamental operations in image processing. For background, refer to Szeliski and the lecture slides on linear filtering. While NumPy provides fast built-in operations for filtering, in this assignment you will **develop your own implementation from scratch**. Start by coding `cross_correlation_2d`, then extend it to build `convolve_2d`, which will reuse the correlation function.
- **Gaussian Blur:** As covered in lectures, blurring can be performed by averaging neighboring pixel values. A Gaussian blur is a more advanced technique, where neighbors are weighted according to a Gaussian distribution. To implement this, you will write `gaussian_blur_kernel_2d`, which generates a Gaussian kernel of a specified size. This kernel can then be passed into your `convolve_2d` function, along with an image, to create a blurred version.
- **Low-Pass and High-Pass Filters:** A low-pass filter removes finer details and preserves smooth image content. A high-pass filter keeps the edges and sharp transitions, while suppressing coarse structures. Using Gaussian blur as the basis, you will implement the `low_pass` and `high_pass` functions.
- **Hybrid Images:** A hybrid image is formed by combining a low-pass filtered version of one image with a high-pass filtered version of another. This process requires choosing a parameter known as the **cutoff frequency**, which determines how much detail is removed or retained. The cutoff is adjusted by setting the standard deviation (σ) of the Gaussian kernel. You are encouraged to experiment with different values, and even use different cutoffs for each image.

Restrictions

For this assignment, you **must not** use ready-made filtering functions from NumPy, SciPy, OpenCV, or any other library. These will be permitted in future tasks, but here the focus is on understanding the mechanics. You may use helper functions like `np.shape`, `np.zeros`, `np.transpose`, `np.pad`, and `np.flip`, but the actual filtering logic must be written by you using loops or NumPy vectorization.

Q2: Edge Detection (10 points)

Write a function that finds edge intensity and orientation in an image. Display the output of your function for one of the given images *uttower_left.JPG* and *uttower_right.JPG*

```
img1 = myEdgeFilter(img0, sigma)
```

The function will input a grayscale image (`img0`) and a scalar (`sigma`). Here, `sigma` is the standard deviation of the Gaussian smoothing kernel used before edge detection. The function should output `img1`, the *edge magnitude* image. First, use your convolution function to smooth the image with a Gaussian kernel defined by `sigma`. This step helps reduce noise and remove spurious edges. Use `scipy.signal.gaussian` to generate the Gaussian kernel. The filter size should depend on `sigma` (for example: $hsize = 2 * ceil(3 * sigma) + 1$)

Compute the edge magnitude image `img1` using gradients in both x and y directions. For the x-direction gradient (`imgx`), convolve the smoothed image with the x-oriented Sobel filter. For the y-direction gradient (`imgy`), convolve with the y-oriented Sobel filter. Combine these to obtain `img1`. Optionally, output `imgx` and `imgy` as well.

Non-Maximum Suppression: Gradient magnitudes often produce thick edges. To thin them:

- For each pixel, examine its two neighbors along the gradient direction.
- If either neighbor has a larger gradient magnitude, suppress the current pixel (set its value to zero).
- Quantize gradient orientation into four cases: $0°$, $45°$, $90°$, and $135°$. For example, an angle of $30°$ maps to $45°$.

- The gradient angle can be computed as:

$$\theta = \arctan\left(\frac{\texttt{imgy}}{\texttt{imgx}}\right)$$

Restrictions: Your implementation must **not** call OpenCV's `Canny` function or similar edge detectors. You may, however, use `Canny` for debugging or comparison purposes.

Q3: Stitching pairs of images (10 points)

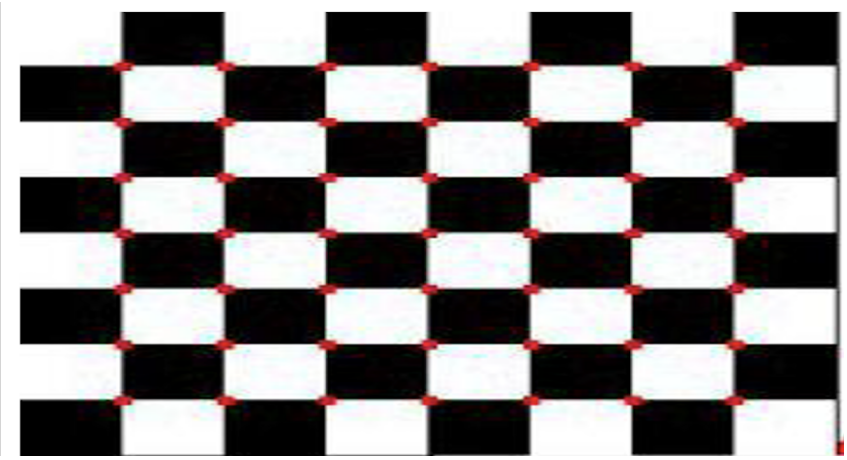
The first step is to write code to stitch together a single pair of images. For this part, you will be working with *uttower_left.JPG* and *uttower_right.JPG* pair:

Steps:

- Compute SIFT key-points and descriptors for both images.
- Calculate distances between descriptors in the first image and those in the second to build a similarity matrix.
- From this matrix, select candidate matches. You may either keep all pairs below a distance threshold or choose the top few hundred pairs with the smallest distances.
- Run RANSAC to estimate a homography mapping one image onto the other. Report the number of inliers and the average residual for the inliers (squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images. For RANSAC, a very simple implementation is sufficient. Use four matches to initialize the homography in each iteration. You should output a single transformation that gets the most inliers in the course of all the iterations. For the various RANSAC parameters (number of iterations, inlier threshold), play around with a few "reasonable" values and pick the ones that work best.
- Warp one image onto the other using the estimated transformation.
- Create a canvas large enough to hold both warped images. Blend them into a panorama by averaging pixel values in the overlapping region.

Q4: Corner Detection (10 points):

In this question, you will implement Harris corner detection algorithms for given two input images (Image1.jpg, Image2.jpg).



- Implement Harris Corner Detection algorithm for images given. In this problem, consider the cornerness measure (C) as [Det(H) - k.Tr(H)], where H is the hessian matrix, Det() is the determinant, Tr() is the trace, and k is a constant $\in$ [0.02, 0.04]. Please use non-negative k = 1/25 as a starting value and try to optimize it by trying different values and comment about it.
- In the previous question, replace cornerness measure (C) with $\lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)$ and determine the time efficiency of this system and the system in the previous question. Report the time efficiencies, you are supposed to get the same accuracy results but different time complexities.

Submission Instructions

Submit all required files for each problem. For example for Q1:

- Your completed `hybrid.py` file. Ensure that all five required functions are implemented.
- `left.png`, `right.png`: the input images used for generating the hybrid image (any supported format is acceptable). `hybrid.png`: the final hybrid image created using your implementation.
- `README`: must include Parameters used for the filters (kernel size, kernel sigma, and mix-in ratio), and Clarification on which image contributed high-frequency details and which contributed low-frequency details.

Similarly for all problems as required.