Introduction to Computer Vision (ECSE 415)
Assignment 1: Image Filtering

DEADLINE: February 8, 11:59 PM

Please submit your assignment solutions electronically via the **myCourses** assignment dropbox. The submission should include a single Jupyter notebook. More details on the format of the submission can be found below. Submissions that do not follow the format will be penalized 10%. Attempt all parts of this assignment. The assignment will be graded out of a total of **100 points**. There are *50 points* for accurate analysis and description, *40 points* for bug-free and clean code, and *10 points* concerning the appropriate structure in writing your report with citations and references. Each assignment will be graded according to defined rubrics that will be visible to students. Check out MyCourses, the "Rubrics" option on the navigation bar. You can use **OpenCV**, **Scikit-Image**, and **Numpy** library functions for all parts of the assignment except those stated otherwise. Students are expected to write their own code. (Academic integrity guidelines can be found **here**). Assignments received up to 24 hours late will be penalized by 30%. Assignments received more than 24 hours late will not be graded.

Submission Instructions

1. Submit a single Jupyter notebook consisting of the solution of the entire assignment.
2. Comment your code appropriately.
3. Give references for all codes which are not written by you. (Ex. the code is taken from an online source or from tutorials)
4. Do not forget to run **Markdown** ('Text') cells.
5. Do not submit input/output images. Output images should be displayed in the Jupyter notebook itself.
6. Make sure that the submitted code is running without error. Add a **README** file if required.
7. If external libraries were used in your code please specify their name and version in the **README** file.
8. We are expecting you to make a path variable at the beginning of your codebase. This should point to your working local (or google drive) folder.
   Ex. If you are reading an image in the following format:

   ```
   img = cv2.imread ( '/content/drive/MyDrive/Assignment1/images/shapes.png' )
   ```

   Then you should convert it into the following:

   ```
   path = '/content/drive/MyDrive/Assignment1/images/'
   img = cv2.imread(path + 'shapes.png')
   ```

   Your path variable should be defined at the top of your Jupyter notebook. While grading, we are expecting that we just have to change the path variable once and it will allow us to run your solution smoothly. Specify, your path variable in the **README** file.
9. Answers to reasoning questions should be comprehensive but concise.

1 Denoising (20 points)

You are **not allowed** to use **OpenCV/Scikit-image** functions for this section. For this question, you are expected to **write your own code in NumPy for convolution.**

Note: For this question, you should work with **grayscale** images. **DO NOT** forget to convert your images to GrayScale from RGB images.
Note 2: This question is worth **20 points**:

- 10 points for correct analysis and displaying desired outputs.
- 10 points for correct, complete, and clean code. Do not forget to implement Convolution with NumPy, it has the biggest impact on your grade.

1.1 Gaussian Noise

You are given a clean image named 'messi.png' (Figure 1) and an image corrupted by additive white Gaussian noise (Figure 2).
Apply the following filtering operations:

1. Filter the noisy image using a 3×3 Gaussian filter with variance equal to 3. Display the filtered image along with the original image (you can use **openCV/scikit-learn** function only for computing the **gaussian filter**).
2. Filter the noisy image using a box filter of the same size. Display the filtered image along with the original image.
3. Compare the Peak-Signal-to-Noise-Ratio (PSNR) of both of the denoised images to that of the clean image and state which method gives the superior result. Use the PSNR function provided by **opencv/scikit-image**.

1.2 Salt and Pepper Noise

You are also given an image corrupted by salt and pepper noise (Figure 3). Apply the following filtering operations:

1. Filter the noisy image using the same Gaussian filter as used in the previous question. Display the filtered image along with the original image.
2. Filter the noisy image using a median filter of the same size. Display the filtered image along with the original image.
3. Compare the PSNR of both of the denoised images to that of the clean image and state which method gives a better result. You are **allowed** to use the PSNR function provided by **opencv/scikit-image**.
4. Describe the discrepancies between the two reference corrupted images: How are Gaussian noise and salt and pepper noise different? What is the best denoising method for these two types of noisy images? (You can use your findings and materials discussed in the classroom)

Figure 1: Clean reference image (source)

Figure 2: Image corrupted with Gaussian noise.

Figure 3: Image corrupted with salt and pepper noise.

2 Canny Edge Detection (20 points)

For this question, you will experiment with the Canny edge detection algorithm that was described in class. Experiments will be performed on the 'cntower.jpg', Figure 4 image. Your goal is to develop an output like Figure 5.

Note: Do not forget to convert the reference image into **grayscale** format. Also note that Figure 5 is probably not the best answer to the question.

Note 2: This question is worth **20 points**:

- 10 points for correct analysis and displaying desired outputs.
- 10 points for correct, complete, and clean code.

2.1. Briefly describe the 4 main steps of Canny edge detection.

2.2. Canny edge detection has three main parameters, as described in the tutorials. The Gaussian smoothing Kernel size (K), the Lower (L), and the Higher (H) thresholds are used for Hysteresis. You will study the effects of changing these parameters in this part. You should run your experiments with this setup: (K = 3, 5, 13; L = 10, 30, 50; H = 80, 100, 150). Note that you should vary the values of each parameter while keeping the other parameters constant. Repeat this procedure for all combinations of parameters mentioned above. This should result in a total of 27 triplets of parameters. Run the Canny edge detection algorithm (cv2.GaussianBlur and cv2.Canny) on the input image for each of these triplets and display the results.

2.3. Describe how changing values of each parameter (K, L, U) affects the overall edge detection results. Is there any relationship between parameters?

2.4. Combine parameters in such a way that edges related to the CN tower are detected (remove other edges related to clouds, etc.) The best solution will not necessarily be found in the values listed above, but you can report the best possible option from the 27 combinations that you have already experimented with. Both approaches are acceptable.

Figure 4: Reference image for Question 2 (source)

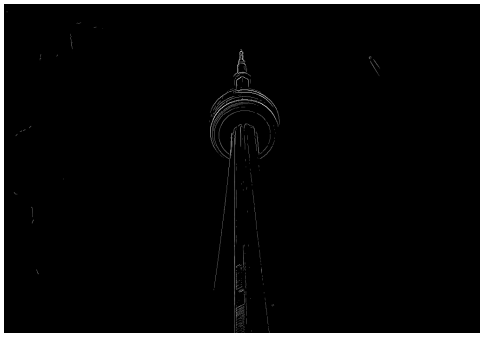

Figure 5: Sample output showing edges

3 Harris Corner Detection (20 points)

For this part of the assignment, you will examine the workings of the Harris corner detector. Implement the Harris corner detector as described in class (Lecture 5, Slide 48, and Tutorial 2) mainly **using NumPy** , going through each of the described steps:

1. Compute the image derivatives (optionally, blur first).
2. Compute the square of the derivatives.
3. Apply Gaussian Filtering on the output of Step 2.
4. Compute the cornerness function response: $Determinant(H) - kTrace(H)^2)$, where k=0.05.
5. Perform non-maximum suppression.

Note 1: note that you can use OpenCV functions for gaussian filtering and computing image derivatives.
Note 2: This question is worth **20 points**:

- 10 points for correct analysis and displaying desired outputs.
- 10 points for correct, complete, and clean code.

Apply the algorithm to three different images:

1. **Maze image** (Left part of Figure 6): Change the value of the threshold to attain detected corners that are similar to those in the right part of Figure 6. Observe and report the effect of changing the threshold values.
2. **Shape image** (Figure 7): Explore the different thresholds and report your observations.
3. **Building image** (Figure 8): Explore the different thresholds and report your observations.

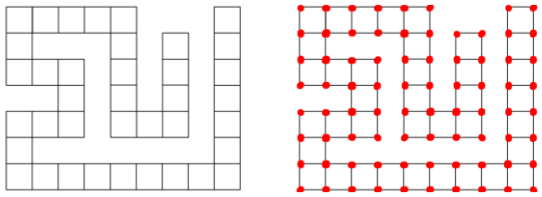

Figure 6: Maze input image and the expected Harris corner detector output. (red dots represent the detected Harris corners) (source)

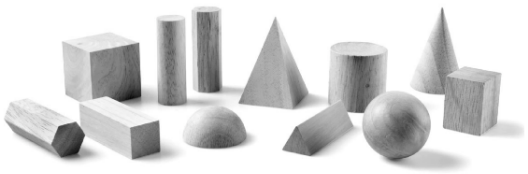

Figure 7: Image depicting different shaped blocks. (source)

Figure 8: Image of a building. (source)

4 Invariance of SIFT Features (30 points)

Note 1: This question is worth **30 points**:

- 10 points for correct analysis and displaying desired outputs for section 4.1 and 4.2.
- 10 points for correct analysis and displaying desired outputs for section 4.3.
- 10 points for correct, complete, and clean code.

You are given a reference image of a Trevi Fountain in Rome as shown in Figure 9. Verify the invariance of SIFT features under changes in image scale and rotation. You are **allowed** to use inbuilt OpenCV/Scikit-Image functions for this question.

4.1 SIFT in a nutshell

Briefly describe the 4 main actions/steps of SIFT method.

4.2 Invariance Under Scale

1. Compute SIFT keypoints for the reference image.
2. Scale reference image using scaling factors of (0.25, 0.6, 3, 5). This should result in a total of 4 different transformed images. Display scaled images.
3. Compute SIFT keypoints for all (total 4) transformed images.
4. Match all keypoints of the reference image to the transformed images using a brute-force method.
5. Sort matching keypoints according to the matching distance.
6. Display the top ten matched keypoints for each pair of the reference image and a transformed image.
7. Plot the matching distance for the top 100 matched keypoints. Plot indices of keypoints on the x-axis and corresponding matching distance on the y- axis.
8. Discuss the trend in the plotted results. What is the effect of increasing the scale on the matching distance? Explain.

4.3 Invariance Under Rotation

1. Rotate reference image at the angle of (30, 75, 90, 180). Display rotated images.
2. Compute SIFT keypoints for all (total 4) transformed images.
3. Match all keypoints of the reference image to the transformed images using a brute-force method.
4. Sort matching keypoints according to the matching distance.
5. Display the top ten matched keypoints for each pair of the reference image and a transformed image.
6. Plot the matching distance for the top 100 matched keypoints. Plot indices of keypoints on the x-axis and corresponding matching distance on the y-axis.
7. Discuss the trend in the plotted results. What is the effect of increasing the angle of rotation on the matching distance? Explain.

Figure 9: Reference image for Question 4 (source)