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

DEADLINE: February 10, 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 submission format 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, clean code and correct implementation, and *10 points* concerning the appropriate structure in writing your report (within markdown cells) 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. You may use any tools for your assignments, but you are responsible for any misrepresentation, inaccuracy, or plagiarism. Citations of non-existent material or obvious factual inaccuracies may result in no credit. (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. The output images should be displayed in the Jupyter Notebook itself.
6. Make sure that the submitted code runs without errors. 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 expect 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:

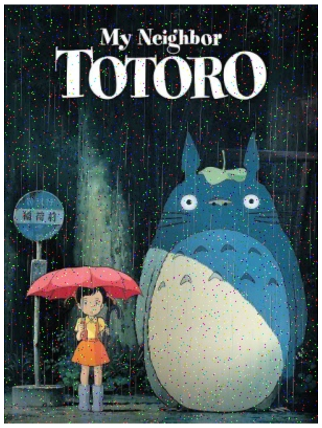
   ```
   path = '/content/drive/MyDrive/Assignment1/images/'
   img = cv2.imread(path + 'shapes.png')
   ```

   Your path variable should be defined at the top of your Jupyter Notebook. While grading, we are expecting that we only have to change the path variable once and that 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.

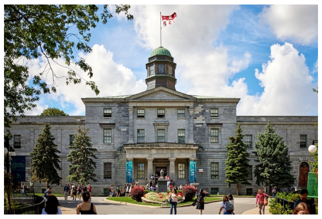
1 Image Filtering [20 points]

In this section, you will implement Gaussian filtering and Median filtering from scratch. You will apply both filters to an image provided to you and analyze their effects.

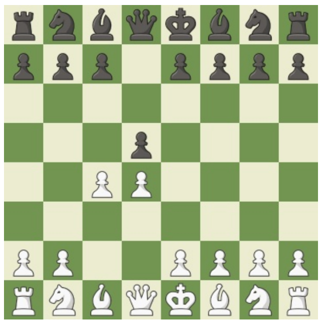
1. Read the image `./images/task1.jpg` and convert it into grayscale. [1 point]
2. Implement Gaussian Filtering [8 points]
   - Create a function `gaussian_filter(image, kernel_size, sigma)` where:
     - `image` is the input grayscale image.
     - `kernel_size` is the size of the filter (e.g., 3x3, 5x5).
     - `sigma` is the standard deviation of the Gaussian function.
   - Calculate the Gaussian kernel using the given `kernel_size` and `sigma` (do not use `cv2.getGaussianKernel()` or any similar built-in function).
   - Normalize the kernel so that the sum of its elements equals 1.
   - Perform convolution between the image and the kernel (do not use `cv2.filter2D()` or any similar built-in function).
3. Implement Median Filtering. [4 points]
   - Create a function `median_filter(image, kernel_size)` where:
     - `image` is the input grayscale image.
     - `kernel_size` is the size of the window (e.g., 3x3, 5x5).
   - Extract a sliding window of size `kernel_size x kernel_size` around each pixel.
   - Replace the center pixel with the median value of the window.
4. Display the grayscale image, the Gaussian filtered image, and the Median filtered image side by side. [3 points]
5. Answer the following questions in a markdown cell: [4 points]
   - How does the Gaussian filter affect the edges and details in the image compared to the Median filter?
   - Which filter is more effective at reducing salt-and-pepper noise? Why?
   - How do the kernel size and sigma parameters impact the results of Gaussian filtering?
   - How does the kernel size impact the results of Median filtering?



(a) task1.jpg



(b) task2.jpg



(b) task3.jpg

Figure 1: Dataset for Q1-Q3.

2 Canny Edge Detection [20 points]

In this section, you will implement the Canny edge detection algorithm and experiment with its adjustable parameters. You will also explore techniques to improve the edge detection results through pre-processing and post-processing steps.

1. Apply the Canny edge detection algorithm to the image `./images/task2.jpg` (convert to grayscale first). Experiment with different threshold values for the parameters `threshold1` and `threshold2` in the `cv2.Canny()` function. Observe how these parameters affect the detected edges. [6 points]
2. Pre-process the image to improve edge detection results. Consider the following techniques: [6 points]
   - Apply denoising filters, such as Gaussian blur or median filtering, to reduce noise in the image.
   - Perform histogram equalization to enhance contrast.

   Re-run the Canny algorithm on the pre-processed image and compare the results with the original.
3. Post-process the edges detected by the Canny algorithm to refine the results. Experiment with the following methods: [6 points]
   - Use morphological operations such as dilation to connect broken edges.
   - Apply erosion to thin out thick edges.

   Visualize and compare the refined edge maps with the original Canny output.
4. Summarize your findings in a markdown cell. Discuss how each pre-processing and post-processing step influenced the edge detection results. Highlight the parameter settings that produced the best edges. [2 points]

3 Harris Corner Detection [20 points]

In this section, you will implement the Harris Corner Detection algorithm from scratch. You will calculate image gradients, smooth the results using Gaussian filtering, and compute the corner response to detect corner points in an image.

1. Read the image `./images/task3.jpg` and convert the image to grayscale. [1 point]
2. Compute image derivatives. [4 points]
   - Calculate the image derivatives in both the x and y directions using Sobel filters.
   - Compute the following products of the derivatives:

     $$I_x^2, \quad I_y^2, \quad I_x I_y$$

     where I_x and I_y are the gradients in the x and y directions, respectively.
3. Smooth the derivative products I_x^2, I_y^2, and $I_x I_y$ using the Gaussian filter in 1.2. You can use any reasonable `kernel_size` and `sigma` values. [2 points]
4. Compute the Corner Response Function. [5 points]
   - For each pixel, calculate the corner response function using the formula:

     $$R = \det(M) - k \cdot (\text{trace}(M))^2$$

     where M is the second moment matrix formed from the smoothed derivative products:

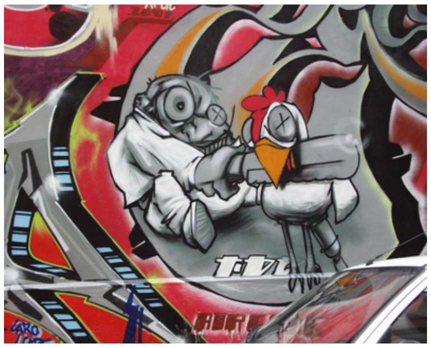
     $$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

     The parameter k is a sensitivity factor, typically set to 0.05.
5. Perform Non-Maximum Suppression. [2 points]
6. Select a reasonable threshold and mark the local maxima in R that exceeds the threshold as detected corners. Overlay the detected corners on the grayscale image for visualization. [3 points]
7. Experiment with different threshold values to control the sensitivity of corner detection and summarize your findings on the effect of varying thresholds in a markdown cell. [3 points]

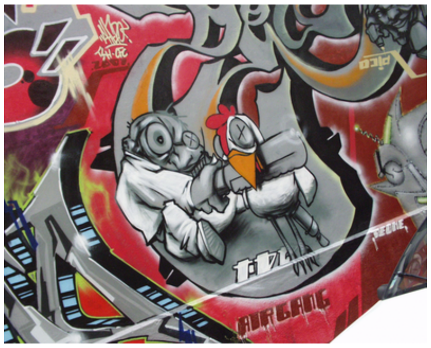
4 SIFT Features [30 points]

In this section, you will implement the SIFT algorithm to detect and match keypoints between images. You will explore the algorithm's ability to handle scale and rotation changes through a series of matching tasks and visualizations.

1. Compute SIFT keypoints and descriptors for the image pair `./images/graf1.png` and `./images/graf2.png`. Use a brute-force matching method to find correspondences between the images. Sort the matches by distance in ascending order. Visualize the top ten matches by overlaying the matched keypoints (with their orientation and size shown as circles) on both images. [10 points]
2. Scale Invariance. [10 points]
   - Scale `./images/graf1.png` using scaling factors of 0.5, 2, and 4 and display them.
   - Compute SIFT keypoints and descriptors for each scaled image and match them with the original image using brute-force matching.
   - For each pair of images (original vs. scaled), sort the matches by distance and visualize the top ten matches.
   - Create a plot showing the matching distances for the top 100 keypoints, with the keypoint index on the x-axis and the matching distance on the y-axis.
   - Discuss how scaling affects the matching distances in a markdown cell.
3. Rotation Invariance. [10 points]
   - Rotate `./images/graf2.png` by 60°, 120°, 180°, and display them.
   - Compute SIFT keypoints and descriptors for each rotated image and match them with the original image using brute-force matching.
   - For each pair of images (original vs. rotated), sort the matches by distance and visualize the top ten matches.
   - Create a plot showing the matching distances for the top 100 keypoints, with the keypoint index on the x-axis and the matching distance on the y-axis.
   - Discuss how rotation affects the matching distances in a markdown cell.



(a) graf1.png



(b) graf2.png

Figure 2: Image pair for Q4.