Submission details

- This is a **LONG** assignment. Please don't wait with it for the last minute.
- This assignment should be done in singles or in pairs.
- The topics of this assignment are Segmentation and Image Features.
- The submission date is **30/06/2024**. Please pay attention to the late submission policy.
- Coding should be done in Python.
- You are not required to use any specific function or library, unless stated otherwise. If in doubt, please contact me via email or Moodle forum.
- For submission, package up your code as a zip file. Include your written answers as a pdf file named writeup.pdf. Include graphs and images in your writeup, and please label them so we know which figures go with which sub-problem. Alternatively, write everything inside your python notebook.
- Name the zip file with ID number(s). Submit your zip file in Moodle.
- If you have any questions about this assignment, please contact me.

Task 1: Segmentation with Superpixels

In this exercise you will get to know a simple method for creating Superpixels, called "Simple Linear Iterative Clustering" (SLIC). Creating Superpixels is an important segmentation technique, used for many Computer Vision applications such as Object Detection, Depth Estimation and Human Pose Estimation. It allows us to perform many processes on perceptually meaningful regions rather than on the pixels themselves, which is both more meaningful and more efficient.

The algorithm:

a. Create K region centers on an equally sampled regular grid on the input $M \times N$ image.
b. For each center, create a $2S \times 2S$ window centered at the center, where $S = \sqrt{\frac{MN}{K}}$.
c. For each pixel inside the window, extract a feature vector and check the distance to the center's feature vector using some distance metric.
d. Decide whether to allocate the pixel to the center or not using some threshold value.
e. After going over all the region centers, recalculate the center's position for each cluster of its associated pixels by averaging over the cluster's members.
f. Repeat stages b-f until convergence or reaching a predefined number of iterations. The resulting clusters are the Superpixels.

For more details, see the paper:

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S., "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods", *IEEE Trans. on PAMI*, Vol. 34, No. 11, pp. 2274 - 2282, 2012.

1. Implement the SLIC method for a general feature vector and a general distance metric. You may implement on your own or use the existing library function. This choice affects your ability to complete certain item ahead, so pay attention.
2. At first we will use a simple feature vector and distance metric. Our vector will be similar to the Mean-Shift one: $\begin{bmatrix} r & g & b & x & y \end{bmatrix}^T$. What will be a good metric for measuring similarity between a pixel and the center in this 5D space? Explain your choice.

3. Take the castle images and try three different weights for the color values and the spatial coordinates in your distance metric. Show the results and explain them.
4. For the same images, show a heat map of the distance between the pixel and its corresponding center in the 5D space, both at initialization and after convergence (with the same scale). Explain the result.
5. Take the castle images and try four different K values: 16, 64, 256, 1024. Measure the time it takes to create each of the results and show them. Explain the results and times you get.
6. Implement the Boundary Recall measure for Superpixel evaluation. This is exactly the Recall measure from HW1, measured on edge maps produced from the Superpixel segmentation. Consider the detection as true even if it is three pixels far from a ground-truth edge.
7. Implement the Under Segmentation measure for Superpixel evaluation:

$$U = \frac{1}{N} \sum_{S \in GT} \sum_{P \in SP} \min\left(|P \cap S|, |P| - |P \cap S|\right)$$

   Where S is a segment in the ground-truth segmentation GT, P is a Superpixel in your segmentation SP and N is the number of pixels in the image.
8. Select four K values and calculate the above two measures and the running time for running your SLIC implementation on the given BSDS images, using their ground-truth segmentations. Use the given ground-truth edge maps for the Boundary Recall measure instead of calculating them from the ground-truth segmentations. Average both measures results over the 5 ground-truth segmentations and edge maps for each image, and then over all the BSDS images to achieve a single working point. Plot 3 graphs of K vs the relevant measure and explain them.
9. BONUS: Try to improve your results in item 8 using the following suggestions:
   a. A more complex distance metric.
   b. A different color space.
   c. A richer feature vector (e.g. using gradients, etc.)
   d. A modification to the algorithm stages.

   Implement your suggestion and demonstrate its advantage on three different images of your choice, comparing to item 8. Explain what you changes and why, and also why do you think it worked (or not). Successful suggestions will be rewarded by a higher bonus.

Task 2: Creating a Panorama Image using SIFT

In this task we will see how to create the ever-so-popular Panorama image from a set of images using the SIFT image features. You may use any library function, apart from functions that create Panorama images. Do **NOT** implement SIFT on your own! Possible sources for SIFT implementation:

1. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
2. http://www.janeriksolem.net/2009/02/sift-python-implementation.html

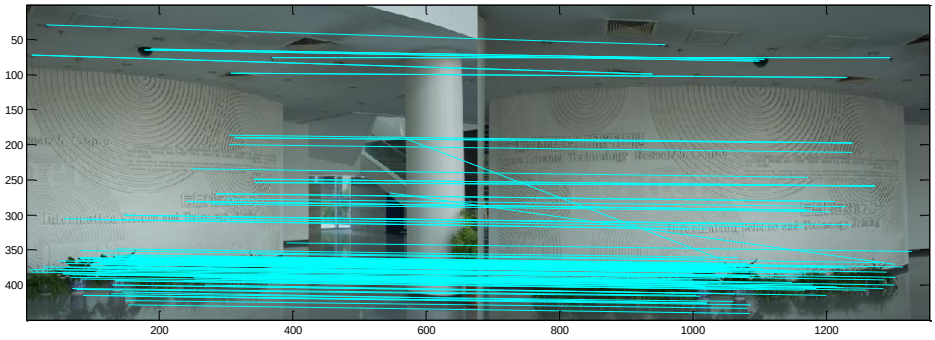
The original SIFT paper:
Lowe, D. G., "Object recognition from local scale-invariant features", *Proc. of ICCV,* pp. 1150–1157, 1999.

You are given a series of 3 images with a part of a single scene in each with some overlapping, while the camera moved from one part to the next. You will attempt to merge the parts into one big Panorama image using SIFT. For example, these three images

will give us this Panorama:

We will be working on the “Working set” until the very late stages of this task. To save on text, I will be referring to items from task 1 in certain places.

1. **Viewing.** Display the images and set them in the correct order of appearance.
2. **Detection.** Detect features on the first two images of the set using SIFT, similarly to item 2 in task 1. Note that here we will use whatever SIFT gives us. SIFT returns a 128 element vector as the feature vector for each interest point. Show the features’ locations and discuss them.
3. **Matching.** Find matching feature points on both images by comparing feature vectors. Calculate the Euclidean distance between the vector of one point from the first image and the vectors of points in the second image. Set the match for the smallest distance, but pay attention that you do not allocate the same point twice. Show 50 of your matches by drawing lines between them like this (It is ok to get a few errors here):



4. **Homography.** Find the Homography matrix between the two images by using RANSAC (see task 1 item 7). In every RANSAC iteration the Homography matrix is calculated from scratch using a minimal number of interest point pairs. What is that minimal number?
   When RANSAC is done, calculate the final Homography from the biggest inliers group of interest point pairs. Use the Direct Linear Transformation method to do that (See below). Show and explain the Homography you got.

5. **Mapping.** Map the pixels from the projection plane of the first image to the projection plane of the second image using the Homography matrix you calculated. You need to create a new image that will hold this projection, and this image may be bigger and may contain areas of black pixels (zero values). Show the projected image.

6. **Merging I.** Now we are going to place both images on the same canvas. Note that we chose to project Image A on the projection plane of Image B, therefore you have to place image B and the projection of image A on that canvas. you need to create a canvas (a big black matrix) that can hold the two merged images. To calculate the size, you can project the coordinates of the corners of image A to the plane of image B. For example, the upper left corner:

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \\ \tilde{w}_i \end{bmatrix} = H \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

   Important notes:
   a. Remember you have to normalize the resulting coordinates.
   b. It is possible to get negative coordinates (since they are relative). Take this into account when calculating the locations on the canvas. The mapping of the corners of image A should be reasonable and not too far away from image B. Check the quality here.
   c. Note that in the overlapping regions in the Panorama, one image may "run over" the other image, but that should not create a visual problem, apart from noticeable "stitching"edges. That is, if you placed the images correctly.

   Show the Panorama image that you received and discuss its quality in general.

7. **Merging II.** Now let's deal with those "stitching" lines. Upgrade your merging using a Pyramid Blending algorithm, similar to the one from HW2, task 1. Try two or three levels only in the pyramid. Repeat item 6, but instead of just placing the images on the canvas, do that with Pyramid Blending. Calculate the appropriate mask for each image as if it is placed on the canvas alone, while paying attention that that certain areas from the overlapping regions between the images will be taken from one of the images only (The "stitching" line between a pair of images will define the border of the mask of each image). Show what you got.
   Can you improve this result? Hint: it's in the mask. Show the improvement.
   <u>Bonus:</u> If the brightness still doesn't match, fix that too automatically.

8. **Generalizing I.** Until now we built a Panorama only from two images. Now let's add the third image by repeating the above process (with the appropriate changes). Explain the needed changes and show the Panorama image composed of the three images.

9. **Generalizing II.** Go wild and try the other image sets. Some of them are composed of nine images. Consider changing the order of the image projection and placing on the canvas –

Maybe better results can be achieved by starting from a different image? Show your results and discuss them. Reduce image resolutions only if necessary.

Direct Linear Transformation:

We want to calculate the Homography matrix H from the projection planes of image A to the projection plane of image B, by using pairs of matching points:

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}}_{H} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Where (x_i, y_i) are image A coordinates and $(\tilde{x}_i, \tilde{y}_i)$ are the matching coordinates of interest point i in image B. Given n interest point pairs, it is possible to write the following equation:

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -\tilde{x}_1 x_1 & -\tilde{x}_1 y_1 & -\tilde{x}_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -\tilde{y}_1 x_1 & -\tilde{y}_1 y_1 & -\tilde{y}_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -\tilde{x}_n x_n & -\tilde{x}_n y_n & -\tilde{x}_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -\tilde{y}_n x_n & -\tilde{y}_n y_n & -\tilde{y}_n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\mathbf{h}} \cong \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It is possible to approximate **h** by solving the following Least Squares problem:

$$\hat{\mathbf{h}} = \arg\min_{\mathbf{h}} \|A\mathbf{h}\|_2^2 \ \ subject\ to\ \ \|\mathbf{h}\|_2 = 1$$

Which is solved by: $\hat{\mathbf{h}} = \text{eigenvector of } A^T A \text{ corresponding to the smallest eigenvalue of } A^T A$.

Task 3: Computing Textons

In this task we will get to know a type of features called textons. They provide us a great example of how to use gradients as features. You may use any library function in this task, apart from functions that compute textons.

1. Load the given images and show them. Describe the type of content you see in the images (edges, textures, smooth surfaces, …)
2. Apply Sobel operators (in X and Y directions separately) to the image. Do **NOT** threshold the results. We will mark the outputs M_x and M_y. Show them.
3. Compute the orientation based on M_x and M_y like this: $\theta = \tan^{-1}\left(\frac{M_y}{M_x}\right)$ for each pixel, and also the magnitude $M = \sqrt{M_x^{\,2} + M_y^{\,2}}$ for each pixel. Show the results and explain.

4. Discretize the orientation image uniformly to 8 different angles using round. Pay attention to the fact that the data is cyclic.
5. For each pixel, compute a descriptor that will be the histogram of discretized orientations in a neighborhood of 11×11 pixels. Orientations should only contribute to the histogram if the magnitude M at the corresponding pixel is above some threshold (you have to select this threshold empirically).
6. Using K-Means, cluster the descriptors into 5, 10 and 50 clusters which are the textons. For each setting, assign textons to each pixel (allocate each pixel to a texton using some distance function), and color each texton differently.
7. Plot the results and discuss them. What do you see? Does it make sense? Explain.
8. Try some grayscale images of your own.

Note: Possible distance functions for this task include, apart from the regular known metrics, also the χ^2 **(Chi-squared)** distance between histograms. See scipy.stats.chisquare(f_obs, f_exp).