University at Buffalo
Department of Computer Science and and Engineering
CSE 473/573 - Computer Vision and Image Processing

Fall 2021
Project #2
Due Date: 12/1/21, 11:59PM

1 Image Stitching (5 points)

The goal of this task is to stitch two images (named "left.jpg" and "right.jpg") together to construct a panorama image. Please follow the steps below to complete the task:

(1) Find keypoints (points of interest) in the given images, e.g., Harris detector or SIFT point detector.
(2) Use SIFT or other feature descriptors to extract features for these keypoints.
(3) Match the keypoints between two images by comparing their feature distance using KNN, k=2. Here, "crosss-checking" and "ratio testing" are commonly used to filter good matches. You can adjust the acceptable distance ratio for better matching (0.75 is a good starting value for ratio testing).
(4) Compute the homography matrix using RANSAC algorithm. (Optional: since RANSAC contains randomness, if you would like to make your best homography matrix and matching result reproducible, you are recommended to set a fixed random seed (which should be the seed for your best result) in your submission. To set random seed for Numpy, you can use `np.random.seed(<int>)` for `numpy.random` or use `random.seed(<int>)` for python basic `random` package.)
(5) Use the homography matrix to stitch the two given images into a single panorama. Note: your final panorama image should NOT be cropped, i.e., it should contain black background and all the regions/contents of both left and right images (see Fig. 3). If your result is cropped and missing some regions of left/right image, you will lose point.

Following is an example of image stitching. Fig. 3 is the expected result. Note that Fig. 1 and Fig. 2 are just examples, and the two images you need to stitch are provided in the .zip file. **Please include a pdf report for this task in your submission showing the methods and steps you implemented for completing the task.**

- You will be provided with a "task1.py" file and two original images. Please strictly follow the format in the "task1.py" and do not modify the code format provided to you.
- Any Python Standard Library or API provided by Numpy could be used.
- Any API provided by OpenCV could be used, **EXCEPT FOR cv2.findHomography() and APIs that have "stitch", "Stitch", "match" or "Match" in their names or functionality**, e.g., `cv2.BFMatcher()` and `cv2.Stitcher.create()`.
- If you decide to use SIFT, please note it has been patented and it has been removed from the latest OpenCV, but it is included in the older versions. Therefore, in order to use SIFT, please install the OpenCV(3.4.2.17) as follows:

- Remove installed opencv (Skip this step if you don't have it installed)

```
$ pip uninstall opencv-python
```

- Install opencv 3.4.2.17

```
$ pip install opencv-python==3.4.2.17
```

- Install opencv contribution library 3.4.2.17

```
$ pip install opencv-contrib-python==3.4.2.17
```

OpenCV(3.4.2.17) has been tested and it does provide fully functional API like `cv2.xfeatures2d.SIFT_create()`.
You might find the following link helpful if you plan to use SIFT feature from OpenCV API. `https://docs.opencv.org/3.4.2/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html`.

Figure 1: Sample Left Image

Figure 2: Sample Right Image

Figure 3: Sample Successful Stitching

2 Grayscale Image Processing (4 points)

The goal of this task is to experiment with two commonly used image processing techniques: image denoising and edge detection. Specifically, you are given a grayscale image with salt-and-pepper noise, which is named "task2.png" (Fig. 4). You are required to write programs to denoise the image first, and then detect edges in the denoised image along both x and y directions. You should follow the steps below to implement your code in "task2.py":

- You are NOT allowed to use OpenCV library (the only two exceptions are `cv2.imread()` and `cv2.imwrite()`, which are already imported in "task2.py"). In addition, you are allowed to use Numpy for basic matrix calculations **EXCEPT FOR any function/operation related to convolution or correlation**. You need to write the convolution code ON YOUR OWN (You should NOT use any other libraries, which provide APIs for convolution/correlation or median filtering).

(i) In order to denoise the image with salt-and-pepper noise, you need to implement a 3×3 **Median Filter** and complete the function `filter(img)` in "task2.py". This function will apply median filtering to the input image and return a denoised image, which will be saved as "task2_denoise.jpg". HINT: remember to do the **zero-padding** to the input image, so that the returned image after filtering can have the same size as the original input. (1 points)

(ii) After denoising, you need to first implement the convolution function `convolve2d(img, kernel)`. Then, based on `convolve2d()`, you should complete the edge detection function `edge_detect(img)` using **Sobel** operators along both x and y directions. This function will take the denoised image "task2_denoise.jpg" as input and should return three images, the first one showing edges along x direction, the second one showing edges along y direction, and the last one showing the magnitude of edges by combining edges along x and y directions. The three edge images will be saved as "task2_edge_x.jpg", "task2_edge_y.jpg" and "task2_edge_mag.jpg" separately. HINT: **zero-padding** is still needed before convolution to keep the original input size. Moreover, in order to save the edge images with correct range and contrast, you need to **normalize** these images before returning them from `edge_detect()`. You should normalize a given image using the following equation: $\text{normalized\_img} = 255 \times \frac{\text{img} - min(\text{img})}{max(\text{img}) - min(\text{img})}$, so that the maximum pixel value is 255 and the minimum pixel value is 0. (2 points)

(iii) Design two 3×3 kernels to detect the diagonal edges of the denoised image "task2_denoise.jpg" along both 45° and 135° directions (0° direction is defined as the horizontal (x) direction of the image). Similar to (ii), you need to complete the function `edge_diag(img)` in "task2.py" and return two normalized edge images, one showing edges along 45° direction, and the other one showing edges along 135° directions. These two edge images will be saved as "task2_edge_diag1.jpg" and "task2_edge_diag2.jpg" separately. Please also print out the two diagonal kernels you designed for this task. (1 point)

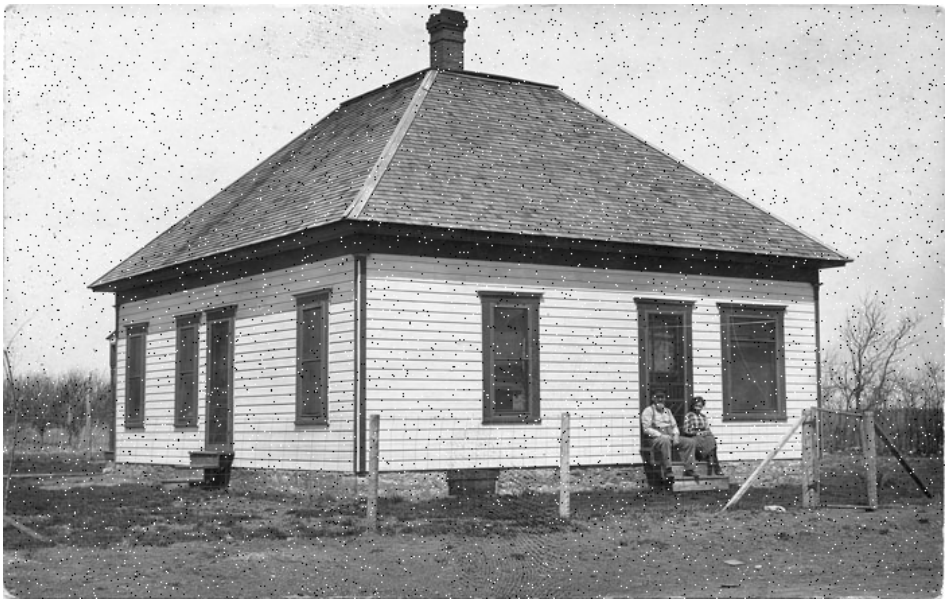

Figure 4: Grayscale image with salt-and-pepper noise.

3 Morphology Image Processing (3 points)

In this task, you are given a binary image with noises as Fig. 5, which is named "task3.png".

- You are NOT allowed to use OpenCV library (the only two exceptions are `cv2.imread()` and `cv2.imwrite()`, which are already imported in "task3.py"). In addition, you are allowed to use Numpy **EXCEPT any functions or APIs directly related to morphology operations, especially erosion, dilation, open and close.** Please implement these morphology operations ON YOUR OWN.

(i) Using the combination of the four commonly used morphology operations, i.e. erosion, dilation, open and close, to remove noises of the image "task3.png". You need to first implement the four operation functions `morph_erode(img)`, `morph_dilate(img)`, `morph_open(img)` and `morph_close(img)`, where 3×3 squared structuring element of all 1's (see Fig. 6) should be used for all the morphology operations. Then you should complete denoising function `denoise(img)` and return the denoised binary image, which will be saved as "task3_denoise.jpg". HINT: **zero-padding** is needed before any morphology operation in order to keep the original input size. (2 points)

(ii) Using the combination of the above four morphology operations to extract the boundary from the denoised image "task3_denoise.jpg". You should implement the boundary extraction function `boundary(img)` and return the binary boundary image, which will be saved as "task3_boundary.jpg". (1 point)

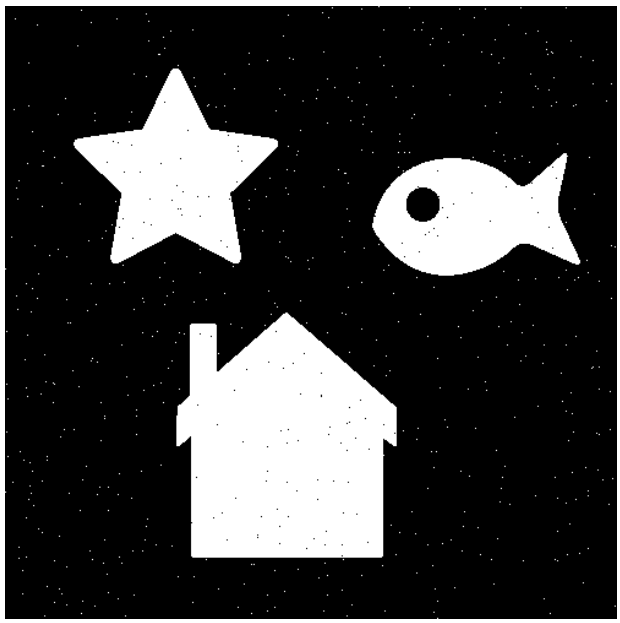

Figure 5: Binary image with noises

1	1	1
1	1	1
1	1	1

Figure 6: Structuring element

Instructions:

- Compress the three python files ("task1.py", "task2.py" and "task3.py"), the pdf report for task1 and all the four given images ("left.jpg", "right.jpg", "task2.png" and "task3.png") into a zip file, name it as "**UBID.zip**" (replace "UBID" with your eight-digit UBID, e.g., 50505050) and upload it to UBLearns before the due.

- Strictly follow the format in the scripts, i.e., "task1.py", "task2.py", "task3.py". **Do Not** modify the code provided to you.

- The input images for all the tasks are provided to you for your own testing only. When grading your codes, different images will be used as inputs and your grade will be based on the corresponding output images. So please do not "hard-code" any solutions in your submissions.
- Anyone whose code is evaluated as plagiarism, your grade will be 0 for this project.
- For all students whose code raises "RuntimeError", your grade will be 0 for this task.
- Please read the guidelines of each task carefully and check the APIs/libraries that are allowed to use. **Do Not** import any library or APIs besides what has been listed.
- Late submissions within 24 hours are allowed and will result in a 50% penalty. After one day, submissions will not be accepted.