Homework 2: Image Stitching

October 9, 2025

Due Date: October. 30 by 23:59:59

Introduction

Late delivery will not be accepted, you will get 1 final score off per day after the deadline(at most 5 final points off). e.g. if you submit four days after the deadline and you get 90 + 10 bonus, your score will be 9 + 1 - 4 = 6. You may get 12 final points out of 10 points for this homework.

We have learned about some basic descriptors of computer vision, let's do something cool! The purpose of this project is to stitch images with image descriptors.

In this project, you are tasked to handle multiple sets of individually shot images, detect and extract their feature descriptors, stitch, and finally blend them.

Most of the algorithms are presented in the course materials. If you have any further questions, here are some supplemental materials:

- Image Stitching: https://zhuanlan.zhihu.com/p/148300210
- Harris Corner: https://www.cnblogs.com/klitech/p/5779600.html
- Harris Corner: https://www.cs.cmu.edu/ 16385/s17/Slides/6.2_Harris_Corner_Detector.pdf
- RANSAC: https://blog.csdn.net/gongdiwudu/article/details/112789674
- SIFT: https://zhuanlan.zhihu.com/p/536619540
- HoG: https://blog.csdn.net/Ning_yuan/article/details/127175708

1 Harris Corner Detector (35 pts.)

1.1 Calculate spatial derivative (10 pts.)

Please choose an appropriate filter to produce the gradient of the image along the X and Y axis. You can reuse the convolution function implemented in Problem Set 1. For simplification, the input image needs to be converted to grayscale. The code should be formatted as follows, where we input a grayscale image and receive its gradient `grad_x`. The function that calculates `grad_y` is similar in format.

```
def gradient_x(img):
    '''
    Input:
        img (np.array (W,H))
    Output:
        grad_x (np.array (W,H))
    '''
    #Your implementation
    return grad_x
```

1.2 Calculate Harris response (15 pts.)

To implement the Harris corner detector, we need to calculate the corner response value R, as shown in Equation 1:

$$R = \det(M) - k \cdot \mathrm{tr}(M) = \lambda_1\lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2, \tag{1}$$

where k is a constant usually chosen in the range of 0.04 to 0.06. You can refer to page 29 for details.

In practice, the derivatives in M are convolved with a window w, usually a Gaussian window to reduce noise and enhance the detection of corners:

$$A = I_x \circ I_x \otimes w; \tag{2}$$

$$B = I_x \circ I_y \otimes w; \tag{3}$$

$$C = I_y \circ I_y \otimes w. \tag{4}$$

Thus you can derive $\det(M)$ and $\mathrm{tr}(M)$ with these three matrices.

Please implement the function that calculates R for each window (small image patches) when moved in both X and Y directions. You should call the `gradient_x` and `gradient_y` in step 1.

```
def harris_response(img, k, win_s):
    '''
    Input:
        img  (np.array (W,H))
        k    (float): Harris corner constant
        win_s(int): Size of the sliding window
    Output:
        R (np.array (W,H)): Harris response of each pixel
```

```
    '''
    #Your implementation
    return R
```

1.3 Select candidate corners and non-maximal suppression (10 pts.)

Now it's time to select the pixels that are likely to be a corner. A non-maximal suppression can help us avoid too much duplicated information carried by nearby corner pixels. Only the candidate corner with maximal response is kept within a given area.

```
def corner_selection(R, th, min_dist):
    '''
    Input:
        R (np.array (W,H)) Pixel-wise Harris response
        th (float) Threshold of R for selecting corners
        min_dist (int) Minimum distance of two nearby corners
    Output:
        pixels (list:N) A list of tuples containing pixels
        selected as corners. e.g., pixels = [(5,8), (3,9)]
        means two pixels at (5,8) and (3,9)
    '''
    # implementation
    return pixels
```

2 Implement Gradient Histogram (15 pts.)

In this part, you will implement a Gradient Histogram for SIFT as a feature descriptor. Refer to the slides or supplemental materials for the details of implementation. You are not required to list the variables (e.g., `window_size`) as input of the function. Instead, hard-code them in the program. There's no fixed template for coding, but you should take into consideration at least the modules below:

1. Horizontal and vertical gradients
2. Gradient direction calculation
3. Prominent gradient selection
4. Histogram for a given cell

5. Feature vector construction

Overall, the main function should be like this:

```
def histogram_of_gradients(img, pix):
    '''
    Input:
        img (np.array (W,H))
        pix (list: N) A list of tuples that contain N indices
            of pixels selected as corners in Q1.
    Output:
        features (np.array (N,L)) A list of L-dimensional
            feature vectors corresponding to the input pix.
    '''
    # implementation
    return features
```

Note that the order of `features` should be consistent with `pix`.

3 Local feature matching (5 pts.)

Given a pair of images, extract paired interest points. First, detect corners using `corner_selection` written in Q1. Next, generate corresponding features using `histogram_of_gradients` written in Q2. Finally, you need to match the two sets of features according to the Euclidean distance and a certain threshold. The outputs should be two sets of pixel indices `pixels_1` and `pixels_2`. For example, `pixels_1 = [(1, 3), (2, 4)]`, `pixels_2 = [(2, 5), (3, 7)]` means (1, 3) in `img_1` matches (2, 5) in `img_2`, and (2, 4) in `img_1` matches (3, 7) in `img_2`.

```
def feature_matching(img_1, img_2):
    '''
    Input:
        img_1, img_2 = (np.array (W,H,3))
    Output:
        pixels_1, pixels_2 = (list:N) A list of tuples (x,y) that
        contains indices of pixels selected as corners in Q1.
    '''
    # implementation
    return pixels_1, pixels_2
```

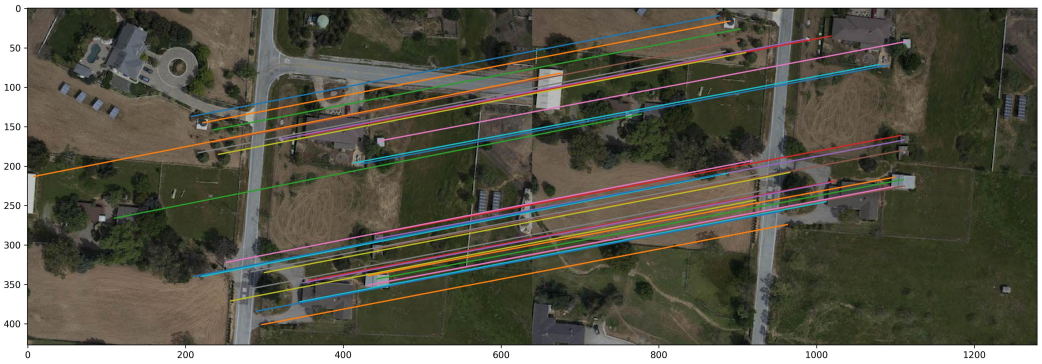

Figure 1: An example for feature matching

We provide the implementation of this function, together with some reasonable hyper-parameters and visualization function *test_matching*. Your task is to understand how it works and use it for the following tasks. You can try different hyperparameters for better performance. Everyone can get points for this part when you finish the tasks above(task 1, task 2). You can test if the tasks above are correct using visualization function *test_matching*. If your codes are correct, matching points should be correct as following figure shows:

4 Image stitching and Blending (30 pts + 10 bonus pts

Warp a pair of (which means two) images so that corresponding points align. Make full use of the functions you have written. We provide 5 pairs of images with parallel image planes. These images can also help you to validate the functions in Q.1 Q.3.

4.1 Compute the alignment of image pairs (10 pts.)

`compute_homography` takes two feature sets from `image_1` and `image_2`, and a list (lens¿4) of feature matches and estimates a homography from image 1 to image 2.

```
def compute_homography(pixels_1, pixels_2):
    '''
    Input:
        pixels_1, pixels_2 (list:N') A list of tuples (x,y) that
        contains N' indices of pixels.
    Output:
        homo_mat (np.array (3,3)) The estimated homography
        matrix
    '''
```

```
    # implementation
    return homo_mat
```

Note: In `compute_homography`, you will compute the best-fit homography using the Singular Value Decomposition (SVD) utilizing `np.linalg.svd(M)`. Also, converting the coordinates to homogeneous coordinates would be a good idea.

4.2 Align the image with RANSAC (15 pts.)

`align_pair` takes two-pixel sets, the list of pixel matches obtained from the feature detecting and matching component, a *motion model*, m (described below) as parameters. Then it estimates and returns the inter-image transform matrix `homo_matrix` using the RANSAC to compute the optimal alignment. You will need RANSAC to find the homography with the most inliners.

```
def align_pair(pixels_1, pixels_2):
    '''
    Input:
        pixels_1, pixels_2 (list:N'): A list of tuples (x,y) that
        contains N' indices of pixels.
    Output:
        est_homo (np.array(3,3)): Optimal homography matrix
    '''
    # implementation
    return est_homo
```

4.3 Stitch and blend the image (Bonus: 10 pts.)

Given an image and a homography, figure out the box bounding the image after applying the homography, warp the image into a target bounding box with inverse warping, and blend the pixels with their neighbors. We provide the implementation of alpha-blending, you can either follow the pipeline directly, or rewrite a better one. You can get bonus points accroding to the qulity of your blending results.

```
def stitch_blend(img_1, img_2, est_homo):
    '''
    Input:
        img_1, img_2: (np.array (W,H,3)) The image to process
        est_homo: (np.array (3,3)) Optimal homography matrix
```

```
    Output:
        est_img: (np.array (W',H',3)) The blended image
    '''
    # implementation
    return est_img
```

Note:

1. When working with homogeneous coordinates, don't forget to normalize when converting them back to Cartesian coordinates.
2. Watch out for black pixels in the source image when inverse warping. You don't want to include them in the accumulation.
3. When doing inverse warping, use linear or other interpolation for the source image pixels.
4. First try to work out the code by looping over each pixel. Later you can optimize your code using array instructions and numpy tricks (numpy.meshgrid, cv2.remap).
5. Save the output blended image as *blend_id.png*

4.4 Generate a panorama (5 pts.)

Your end goal is to be able to stitch any number of given images - maybe 2 or 3 or 4; your algorithm should work, at least not report any errors. We provide the implementation of this function, you can either follow the pipeline directly, or rewrite a better one. Everyone can get points for this part when the result are correct.

```
Def generate_panorama (ordered_img_seq)
    '''
    Input:
        ordered_img_seq: (List) The list of images to process
    Output:
        est_pano: (np.array (W',H',3)) The panorama image
    '''
    # implementation
    return est_pano
```

Note:

1. The input is an ordered sequence of images, which means the i th image is supposed to match with the i-1 th and i+1 th image.

2. If random paired images with no matches are given, your algorithm must report an error.

5 Report and Analyze (15 pts + 10 bonus)

5.1 Report (10 pts.)

You have now completed the entire process of image stitching using Harris corner detection. Please use the images we provided to verify the correctness of the algorithm and finish a report. It should include your implementation philosophy and visualizations on Harris Corner Detection, RANSAC for homography, Pair-image stitching(3 pairs), and Panorama generation with multiple images(4 scenes). Contain all of the stuff in one PDF report. This part is worth 10 points: report on implementation(3 pts) and 7 stitched images(7 pts).

5.2 Analyze (5 pts + 10 bonus)

If you have reached this point, you may have noticed that the quality of the stitched image is determined by various factors, including hyperparameters, how to move camera during shooting, what kind of descriptors you use, how good the descriptors are, and how you blend the images. List your experiments, make a brief analysis and you can get 5 points.

Therefore, we encourage you to experiment with the following combinations of problems to earn the bonus 10 points (you can choose either one to get the bonus):

1. Explore how to use more powerful and robust feature detectors and descriptors to improve the panorama, make comparisons to show the improvements, and analyze. (10 pts.)

2. Explore how to combine filtering and advanced blending techniques (e.g., pyramid blending, Poisson blending) to get a better panorama. Make comparisons to show the improvements and analyze. (10 pts.)

For those who want to delve deeper and are unwilling to capture real-world data, you can choose to tackle the last two questions. For the rest, you have to run the experiments on your own dataset. It's important to emphasize that this section only contributes just 1 point to the overall 10-point assignment, so if you're short on time,

you can choose the easier part to get some of the scores. If you opt to tackle any listed questions, kindly document your progress in terms of shooting the sequences and designing the algorithms. Wish you good luck and enjoy the homework!

Requirements You are required to finish the algorithm framework and report with LaTeX or Word with vivid description images. Either Chinese or English is acceptable, but any handwriting homework will get a deduction by 20% of scores this time.

Be sure to zip your code, generated images, and final report and name it *"StudentID_YourName_HW2.zip"*. Save the output panorama image into folder *output* as *panorama_id.png*(id: 1-grail, 2-library, 3-parrington, 4-Xue-Mountain-Entrance). You can add some helper functions but make sure to implement tasks under the given framework *homework2.py*. The helper functions should be placed in another file named *utils.py*. Do not rename the functions and do not modify input and output in *homework2.py*. Any wrong name will reduce your score by 1 pts. You may lose up to 10 pts if you hand in homework with the wrong names.

The teaching assistant is not responsible for addressing any specific inquiries related to code debugging. Students are encouraged to refer to PPT presentations and online learning resources.

For implementation, you may use `NumPy`, `SciPy`, and `OpenCV2` functions to implement mathematical, filtering, and transformation operations. Do not use functions that implement keypoint detection or feature matching. Make sure your code outputs a single panorama in 180s, or you get 20% off of your score.

When using the Sobel operator or Gaussian filter, you should use the 'reflect' mode, which gives a zero gradient at the edges.

Here is a list of potentially useful functions; you are not required to use them:

- `scipy.ndimage.sobel`
- `scipy.ndimage.gaussian_filter`
- `scipy.ndimage.filters.convolve`
- `scipy.ndimage.filters.maximum_filter`
- `scipy.spatial.distance.cdist`
- `np.max, np.min, np.std, np.mean, np.argmin, np.argpartition`
- `np.degrees, np.radians, np.arctan2`