Homework 3

Feature Extraction
COMP 590, Spring 2020
Due: Feb 20, 2020

Summary
The goals of this assignment are to 1) expand your understanding of image transformations and filtering, and 2) have you gain practical experience with image feature extraction. Here, you will implement a Harris corner feature detector, which uses a measure of 2D "cornerness" to identify salient local regions in an input image. You will also implement a simple patch-based matching procedure to identify corresponding 2D feature points for a pair of images. Skeleton Python code (described below) has been provided for this assignment, along with images for you to experiment with.

Reading

- Szeliski 3.1 (intro), 3.1.1-2, 3.2 (intro), 3.2.1, 3.3.1-2
- (optional) Szeliski 3.1.3-5, 3.2.2+, 3.3.3+, 3.5
- Harris, Chris, and Mike Stephens. "A combined corner and edge detector." *Alvey Vision Conference*. Vol. 15. No. 50. 1988. [link]
- Lowe, David G. "Distinctive image features from scale-invariant keypoints." *International Journal of Computer Vision* 60.2 (2004): 91-110. [link]
  (This is the SIFT feature paper. You will not use SIFT in this assignment, but you should read the paper and understand the basics of the method.)

Assignment
In this assignment, you will implement the following pipeline for feature exaction and matching (specific steps are described in detail below):

1) Harris "cornerness" filter for a gray-value image
2) Non-maximum suppression of cornerness scores
3) Corner keypoint extraction
4) Patch-based feature description
5) One-to-one matching using SSD and NCC

Steps for Feature Detection (Harris Corner Detection)
The general steps of Harris corner detection are as follows:

1) Convert input image $I_{RGB}(x, y)$ to grayscale image $I(x, y)$.
2) Apply a "*cornerness*" filter $h(I)$. This results in an image $R(x, y)$ with larger values corresponding to image points that are more likely to be corners. The filter h is formed from a series of independent filter steps:
   a. Compute the image gradients $X = I_x(x, y)$ and $Y = I_y(x, y)$. It is suggested you use the Sobel filter for increased robustness to noise.

   b. Compute the matrix-valued image $M(x, y)$, as defined by Harris and Stephens. For the Gaussian weighting, consider using SciPy's gaussian_filter function.
   c. Compute $R(x, y) = \det(M(x, y)) - k\ \mathrm{tr}(\mathrm{M}(\mathrm{x}, \mathrm{y}))^2$, for some constant k.
3) Apply non-maximum suppression to $R(x, y)$, keeping only pixel locations that have the strongest response within their $w \times w$ pixel neighborhood. Consider using a maximum filter for this operation.
4) Return the (x, y) coordinates of the strongest corner response maxima, according to some thresholding operation. Here, we will simply select as features the up-to-K strongest maxima with response $R(x, y) > 0$. Note that the keypoints should have integer pixel positions.

Feature Description and Keypoint Matching

We will take a very simple approach to feature description: For each keypoint, take the $n \times n$ image patch of $I(x, y)$ centered at that keypoint. For keypoint matching between two images, you should implement two methods for comparing descriptors:

1) Sum-of-squares difference (SSD)
2) One minus normalized cross-correlation (1 – NCC; this assigns a distance of zero for identical descriptors)

Given two images, compute the distance of every feature in the first image to every feature in the second image and store the result in a *match matrix*, with rows corresponding to first-image features and columns corresponding to second-image features.[1] Then, compute keypoint correspondences using *one-to-one matching*:

- **One-to-one Matching**: For each keypoint in the first image, find the most similar keypoint in the second image. Repeat this for the keypoints in the second image against the first. Keep the feature correspondences that are mutual best matches.

Note: While you could use SciPy's cdist function to compute the pairwise scores (*i.e.*, SSD and 1 – NCC), please implement the metrics on your own for this assignment.

Summary: Algorithm Parameters

- σ : standard deviation of the Gaussian filter when computing $R(x, y)$
- k : typical values range from 0.05 to 0.15
- w : for non-maximum suppression, use a fixed window size of $w = 11$px
- K : maximum number of keypoints to return (note: only return points where $R(x, y) > 0$)
- n : for feature description, use a fixed window size of $n = 7$px
- Matching method: either SSD or (one minus) NCC

[1] For example, the first column will contain the distance from the first keypoint in the first image to every keypoint in the second image. (Note that the keypoint ordering for each image is arbitrary, but fixed.)

Code
There are four files provided to help you get started:

- *main.py*: This file is already set up to extract and match features for two images, and to display the result. For usage, execute "*python main.py --help*".

- *harris_corner.py*: This file contains skeleton code for feature extraction. You will need to fill in two functions (respecting the algorithm parameters already defined in the class):
    - *compute_corner_response()* : Given grayscale image $I(x, y)$, compute $R(x, y)$.
    - *get_keypoints()* : Given $R(x, y)$, apply non-maxima suppression and return the strongest up-to-K keypoints having $R(x, y) > 0$. The N selected keypoints are returned as an $N \times 2$ array, with each row giving the (x, y) coordinate of a detected keypoint.

- *feature_matcher.py*: This file contains skeleton code for feature matching. You will need to fill in four functions (respecting the algorithm parameters already defined in the constructor of the FeatureMatcher class):
    - *get_descriptors()* : Given an image and a set of keypoint locations, extract a feature descriptor for each keypoint. You should decide how to obtain descriptors for keypoints near the edge of the image. For an $N \times 2$ array of keypoints and a descriptor window of size n, this function should return an $N \times n^2$ array of keypoint descriptors, with each row consisting of a flattened image patch.
    - *match_ssd()* : Compute a match matrix for two sets of descriptors using SSD.
    - *match_ncc()* : Compute a match matrix for two sets of descriptors using 1 – NCC.
    - *compute_matches()* : Given a match matrix, compute one-to-one matches. Given M selected matches, this function should return an $M \times 2$ array, where each row contains a pair of corresponding keypoint indices. For example, if the first row of the matches array contains indices (i, j), the associated keypoints are the i^{th} keypoint of the first image and the j^{th} keypoint of the second image.

- *plot_util.py*: This file is already set up to display keypoints and matches.

Submission
Submit a single PDF containing your results for the tasks listed below and separate files for your source code.

In the following, assume $k = 0.05$, $\sigma = 1$, and $K = 1000$ unless otherwise specified. Again, note that you should only return keypoints where $R(x, y) > 0$.

- For the included image "test.png", show results for $k = 0.05$, $k = 0.10$, and $k = 0.15$ with $\sigma = 1$ and $\sigma = 9$ (six total images). Explain the separate effects of increasing k and increasing σ.
- For the image "airport1.png", show results with and without non-maximum suppression. Explain why using non-maximum suppression is desirable.
- Show matches between the images "airport1.png" and "airport2.png" using $K = 50$ and SSD. Clearly circle a feature in the first image that was incorrectly matched, or state that all matches are correct.
- Show matches between the images "oxford1.png" and "oxford2.png" using $K = 50$ for both SSD and 1 – NCC. Compare the performance of the two methods and explain why one might be preferred over the other. Also explain where and why matching errors occur for the image pair.[2]
- Show matches between the images "graffiti1.png" and "graffiti2.png" using $K = 20$ and $K = 100$ with SSD. Explain the observed matching performance for this image pair.[3]

[2] Oxford images credit: James Philbin et al.
[3] Graffiti images credit: Krystian Mikolajczyk and Cordelia Schmid