CIS581, Computer Vision
Project 3, Automatic 2D Image Mosaic
Due November 8, 3:00pm

Instructions. This is an individual assignment. All matlab functions should follow the names and arguments stated in the problems in order for them to run properly with the grading script. A test script will be provided shortly that will call your functions to ensure they will run inside the grading script. To submit the assignment, upload a zip file containing all your code via Canvas.

Turn-in

Zip your files into a folder named `PennKey_Project3.zip` and submit it via Canvas.
This should include:

- your .m files for the six required Matlab functions (`corner_detector.m`, `anms.m`, `feat_desc.m`, `feat_match.m`, `ransac_est_homography.m`, `mymosaic.m`)
- any demo .m script(s) for reproducing/showing your results
- any additional .m files with helper functions for your code - this includes any third party code that you used, eg for harris corner detection
- the input images you used
- the resulting stitched resized image(s) and image mosaic(s) respectively.
- a PDF showing your final and intermediate results. It should contains:
  - the final stitched mosaic (which should already be included as a standalone image).
  - corner detection results of an image in gray scale image.

- adaptive non-max suppression of the image (used to show corner detection results) in red dots.
- final matching results (after RANSAC) of all pairs of images used in mosaic in red cross with outliers in blue dot
- describing any additional features of your implementation.
- if you used third party code, make it clear. Make the PDF as simple as possible.

All the files should be in the top level of the ZIP file (no subfolders, please).

Overview

This project focuses on image feature detection, feature matching and image mosaic techniques. The goal of this project is to create an image mosaic or stitching, which is a collection of small images which are aligned properly to create one larger image. We will follow technique outlined in the following papers, which are available on the course website:

"*Multi-image Matching using Multi-scale image patches*", Brown, M.; Szeliski, R.; Winder, S. CVPR 2005

"*Shape Matching and Object Recognition Using Shape Contexts*", Belongie, S., Malik, J. and Puzicha, J. PAMI 2002: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/shape/

Task 0: CAPTURE IMAGES

For this project, you need to capture multiple images of a scene, which you will use to create image mosaic. In general, you should limit your camera motion to purely translational, or purely rotational (around the camera center).

Bonus points will be given for interesting selection of images.

Task 1: Automatic Correspondences

1. Detecting corner features in an image. Recommand to use Matlab `cornermetric` with proper arguments. But, you could follow method outlined in lecture note on Interest Point Detector to build your own image corner detector (for example derived

from edge detector). You can probably find free "harris" corner detector on-line, and you are allowed to use them. Software links are provided on the course website.

Implement the funtion:

`[cimg] = corner_detector(img)`

(INPUT) `img`: $H \times W$ matrix representing the gray scale input image.
(OUTPUT) `cimg`: $H \times W$ matrix representing the corner metric matrix.

2. Implement Adaptive Non-Maximal Suppression. See section 3 of the reference paper, as well as lecture notes. The goal is to create an uniformly distributed points given the number of feature desired:

`[x, y, rmax] = anms(cimg, max_pts)`

(INPUT) `cimg`: $H \times W$ matrix representing the corner metric matrix.
(INPUT) `max_pts`: the number of corners desired.
(OUTPUT) `x`: $N \times 1$ vector representing the column coordinates of corners.
(OUTPUT) `y`: $N \times 1$ vector representing the row coordinates of corners.
(OUTPUT) `rmax`: suppression radius used to get `max_pts` corners.

3. Extracting Feature Descriptor for each feature point. You should use the subsampled image around each point feature. Don't worry about rotation-invariance, just extract axis-aligned 8x8 patches. Note that it's extremely important to sample these patches from the larger 40x40 window to have a nice big blurred descriptor. Don't forget to bias/gain-normalize the descriptors.

`[descs] = feat_desc(img, x, y)`

(INPUT) `img`: $H \times W$ matrix representing the gray scale input image.
(INPUT) `x`: $N \times 1$ vector representing the column coordinates of corners.
(INPUT) `y`: $N \times 1$ vector representing the row coordinates of corners.
(OUTPUT) `descs`: $64 \times N$ matrix, with column i being the 64 dimensional descriptor (8×8 grid linearized) computed at location (x_i, y_i) in `img`.

As a bonus, you can implement the Geometric Blur features.

4. Matching these feature descriptors between two images. You can use k-d tree to find the k nearest neighbour. Remember to filter the correspondences using the ratio of the best and second-best match SSD. You can set the threshold to 0.6.

`[match] = feat_match(descs1, descs2)`

(INPUT) `descs1`: $64 \times N_1$ matrix representing the corner descriptors of first image.

(INPUT) `descs2`: $64 \times N_2$ matrix representing the corner descriptors of second image.
(OUTPUT) `match`: $N_1 \times 1$ vector where $\texttt{match}_i$ points to the index of the descriptor in `descs2` that matches with the feature i in descriptor `descs1`. If no match is found, you should put $\texttt{match}_i = -1$.

Task 2: RANSAC

Not all the matches computed in Task 2 will be correct. One way to remove incorrect matches is by implement the additional step of RANSAC (see lecture notes).

Use a robust method (RANSAC) to compute a homography. Use 4-point RANSAC as described in class to compute a robust homography estimate:

```
[H, inlier_ind] = ransac_est_homography(x1, y1, x2, y2, thresh)
```

(INPUT) `x1, y1, x2, y2`: $N \times 1$ vectors representing the correspondences feature coordinates in the first and second image. It means the point $(x1_i, y1_i)$ in the first image are matched to $(x2_i, y2_i)$ in the second image.
(INPUT) `thresh`: the threshold on distance used to determine if transformed points agree.
(OUTPUT) `H`: 3×3 matrix representing the homograph matrix computed in final step of RANSAC.
(OUTPUT) `inlier_ind`: $N \times 1$ vector representing if the correspondence is inlier or not. 1 means inlier, 0 means outlier.

Recall the RANSAC steps are:

1. Select four feature pairs (at random), p_i, p_i^1
2. Compute homography H (exact). Code is available for this on the Project page.
3. Compute inliers where $SSD(p_i^1, Hp_i) < thresh$
4. repeat step 1-3
5. Keep largest set of inliers
6. Re-compute least-squares H estimate on all of the inliers

Task 3: IMAGE MOSAIC

Produce a mosaic by overlaying the pairwise aligned images to create the final mosaic image. Also, check Matlab functions, e.g `imtransform` and `imwarp`. If you want to implement `imwarp` (or similar function) by yourself, you should apply bilinear interpolation when you copy pixel values. As a bonus, you can implement smooth image blending of the final mosaic.

`[img_mosaic] = mymosaic(img_input)`

(INPUT) `img_input`: M elements cell array, each element is a input image.
(OUTPUT) `img_mosaic`: $H \times W \times 3$ matrix representing the final mosaic image.

FAQ

Q: Suppose I have the matrix cimg which contains corner strengths for each pixel. Do I need to set a threshold to get corners candidates? And then I only need to calculate radius for each corner candidate. Or I need to calculate radius for all pixels?

A: Ultimately you want corners that are at distinctive locations and well distributed throughout the image. If you don't threshold at all then in areas that are relatively constant (say a plane white wall) you'll get some corners due to noise that won't end up being matches anyway since the location was essentially random. If you threshold too much initially, however, you might only get corners in a few strong regions and cut out fainter but justified corners. Since for the panorama the overlap between images (and hence corners that can correctly match between two images) is a small region near one of the boundaries, you want to be sure that some corners respond in all portions of the image. The anms will then prune the corners that survived the thresholding to a manageable set that is also nicely distributed.

Q: I was having a problem getting the `feat_desc` function to account for the window size at the edges. I tried scaling the size of the window based on the location of the corner, but this would obviously not match with the $P_{(64\times n)}$ matrix output.

A: You could try zero-padding those corners. Doing so makes it more likely that the corner won't find a match or will find a close ssd score to other corners that got zero padded. You could also remove corners within 20 pixels of the edge from consideration so that all of your features can fit a window around them. Your region of overlap will probably be large enough to get plenty of matches past the 20 pixels on the boundary. Additionally assuming pixels near the boundary in one image aren't also near the boundary in the other images (some instances where this assumption isn't true) then they will likely be poor matches and get not make it past your threshold for matching anyway.

Q: In RANSAC, the part of the algorithm that calls for $SSD(p_i^1, Hp_i)$, I am assuming this is talking about the locations of the features and not the feature descriptors since the `ransac_est_homography` doesn't see the descriptors. At any rate, how is this SSD computed and why don't we just use euclidean distance?

A: SSD on the coordinates would be the euclidean distance squared. There is little value in taking a square root because we have to set an arbitrary threshold anyway.

Q: For the `mymosaic` function, are there any assumptions that can be made about the order of the images passed into the function? Meaning, can I assume that the order that the images are passed in is the order I should be stitching? Or do we need to compare every image with each other to find the ones that match best?

A: Assume that the order they come in is the order they stitch together.