CMSC 733, Computer Processing of Pictorial Information
Project 1: Myautopano!
Due on: 11:59:59PM on Wednesday, Feb 15 2017

Prof. Yiannis Aloimonos,
Nitin J. Sanket

February 2, 2017

The aim of this project is to implement an end-to-end pipeline to do image panorama stitching of unordered images. We all use the panorama mode on our smart-phones. You'll be implementing a simple pipeline which does the same. Aren't you excited?

In the next few sections, we'll detail how this has to be done along with the specifications of the functions for each part.

1 Capture Images

For this project, you need to capture multiple images of a scene, which you will use to stitch a panorama from. In general, you should limit your camera motion to purely translational, or purely rotational (around the camera center). Make sure you have about 30-50% overlap between consecutive images to have enough common features to be able to stitch panoramas.

You'll need creative ways to make it work for harder images. A set of sample images are shown in Fig. 1.

Figure 1: Sample input images used for panorama stitching.

2 Adaptive Non-maximal Suppression (ANMS) to find corners distributed equally over the image - 25Pts

The aim of this step is to detect corners spread all across the image to avoid weird artifacts in warping. Implement the following steps for ANMS:

Detect corner features in the image using `cornermetric` with the appropriate parameters. The output is a matrix of corner scores. The higher the value, the higher the score/probability of that pixel being a corner. Visualize the output using the MATLAB function `imagesc` or `surf`.

Now, find the N_{strong} strongest corners using the MATLAB function `imregionalmax`. Because, when you take a real image, the corner is never perfectly sharp, each corner might get a lot of hits out of the N_{strong} corners - we want to choose only the N_{best} best corners after ANMS. In essence, you'll get a lot more corners than you should. ANMS will try to find corners which are local maxima. The ANMS algorithm is described next:

Input : Corner score Image (C_{img} obtained using `cornermetric`), N_{best} (Number of best corners needed)
Output: (x_i, y_i) for $i = 1 : N_{best}$

Find all local maxima using `imregionalmax` on C_{img};
Find (x, y) co-ordinates of all local maxima;
((x, y) for a local maxima are inverted row and column indices i.e., If we have local maxima at $[i, j]$ then $x = j$ and $y = i$ for that local maxima);

Initialize $r_i = \infty$ for $i = [1 : N_{strong}]$

for $i = [1 : N_{strong}]$ **do**
  for $j = [1 : N_{strong}]$ **do**
    if $(C_{img}(y_j, x_j) > C_{img}(y_i, x_i))$ **then**
      $\text{ED} = (x_j - x_i)^2 + (y_j - y_i)^2)$
    end
    if $ED < r_i$ **then**
      $r_i = \text{ED}$
    end
  end
end

Sort r_i in descending order and pick top N_{best} points

Algorithm 1: ANMS algorithm.

Consider the case where we are stitching first 2 images in Fig. 1. The output of ANMS are shown in Fig. 2. To plot dots on the images you can do `imshow(..); hold on; plot(..); hold off;`. Note that plot uses x and y wherein x is the column number and y is the row number.

Figure 2: Output of ANMS on first 2 images shown in Fig. 1.

3 Feature Descriptor - 25Pts

In the previous step, you found the feature points (locations of the N_{best} best corners after ANMS are called the feature points). You need to describe each feature point by a feature vector, this is like encoding the information at each feature points by a vector. One of the easiest feature descriptor is described next.

Take a patch of size 40×40 centered (this is very important) around the keypoint. Now apply gaussian blur (feel free to play around with the parameters, for a start you can use MATLAB's default parameters in `fspecial` command, Yes! you are allowed to use `fspecial` command in this project). Now, sub-sample the blurred output (this reduces the dimension) to 8×8. Then reshape to obtain a 64×1 vector. Standardize the vector to have zero mean and variance of 1 (Done by subtracting all values by mean and then dividing by the standard deviation). Standardization is used to remove bias and some illumination effect.

4 Feature Matching - 10Pts

In the previous step, you encoded each keypoint by a 64×1 feature vector. Now, you want to match the feature points among the two images you want to stitch together. In computer vision terms, this step is called as finding feature correspondences within the 2 images. Pick a point in image 1, compute sum of square difference between all points in image 2. Take the ratio of best match (lowest distance) to the second best match (second lowest distance) and if this is below some ratio keep the matched pair or reject it. Repeat this for all points in image 1. You will be left with only the confident feature correspondences and these points will be used to estimate the transformation between the 2 images also called as *Homography*. Use the function `dispMatchedFeatures` given to you to visualize feature correspondences (Sample output is shown in Fig. 3).

Figure 3: Output of `dispMatchedFeatures` on first 2 images of Fig. 1.

5 RANSAC to estimate robust Homography - 15Pts

Note that not all the matches we found in the previous step will be right. We will use a robust method called Random Sampling Concensus or RANSAC to compute homography.

Recall the RANSAC steps are:

- Select four feature pairs (at random), p_i from image 1 , p_i^1 from image 2.
- Compute homography H (exact). Use the function `est_homography` given to you.
- Compute inliers where $SSD(p_i^1, Hp_i) < thresh$. Here, Hp_i computed using the `apply_homography` function given to you.
- Repeat the last three steps until you have exhausted N_{max} number of iterations (specified by user) or you found more than a percentage of inliers (say 90% for example).
- Keep largest set of inliers.
- Re-compute least-squares $\hat{H}$ estimate on all of the inliers. Use the function `est_homography` given to you.

6 Cylindrical Projection - 10Pts

When you try to stitch a lot of images (stitching images is discussed in the next section) with translation, a simple projective transformation (applying homography) will give bad results and the images will be stretched/shrunken to a large extent on the edges. A sample case is shown in Fig. 5.

Figure 4: Final Panorama output for Fig. 1 image.

Now, to overcome the distortion problems at edges, we will be using cylindrical projection on the images first before performing other operations. Essentially this is a pre-processing step.

The following equations transform from normal image co-ordinates to cylindrical co-ordinates.

$$x' = f \tan\left(\frac{x - x_c}{f}\right) + x_c$$

$$y' = \left(\frac{y - y_c}{\cos\left(\frac{x - x_c}{f}\right)}\right) + y_c$$

In the above equations, f is the focal length of the lens in pixels (feel free to experiment with values, generally values range from 100 to 500, however this totally depends on the camera and can lie outside this range). The original image co-ordinates are (x, y) and the transformed image co-ordinates (in cylindrical co-ordinates) are (x', y'). x_c and y_c are the image center co-ordinates. Note that, x is the column number and y is the row number in MATLAB.

You can use `meshgrid, ind2sub, sub2ind` to speed up this part. **Using loops will TAKE FOREVER!**

A sample input image and it's cylindrical projection output is shown in Fig. 6.

Note that, the above equations talk about pixel co-ordinates are not pixel values. The idea is you compute the co-ordinate transformation and copy paste pixel values to these new pixel co-ordinates (in all 3 channels, i.e., RGB). However, when you compute the values of (x', y') they might not be integers. A simple way to get around this is to use round or actually interpolate the values. If you decide to round the co-ordinates off you might be left with black pixels, fill them using some weighted combination of it's neighbours (gaussian works best). A trivial way to do this is to blur the image and copy paste pixel values on-to

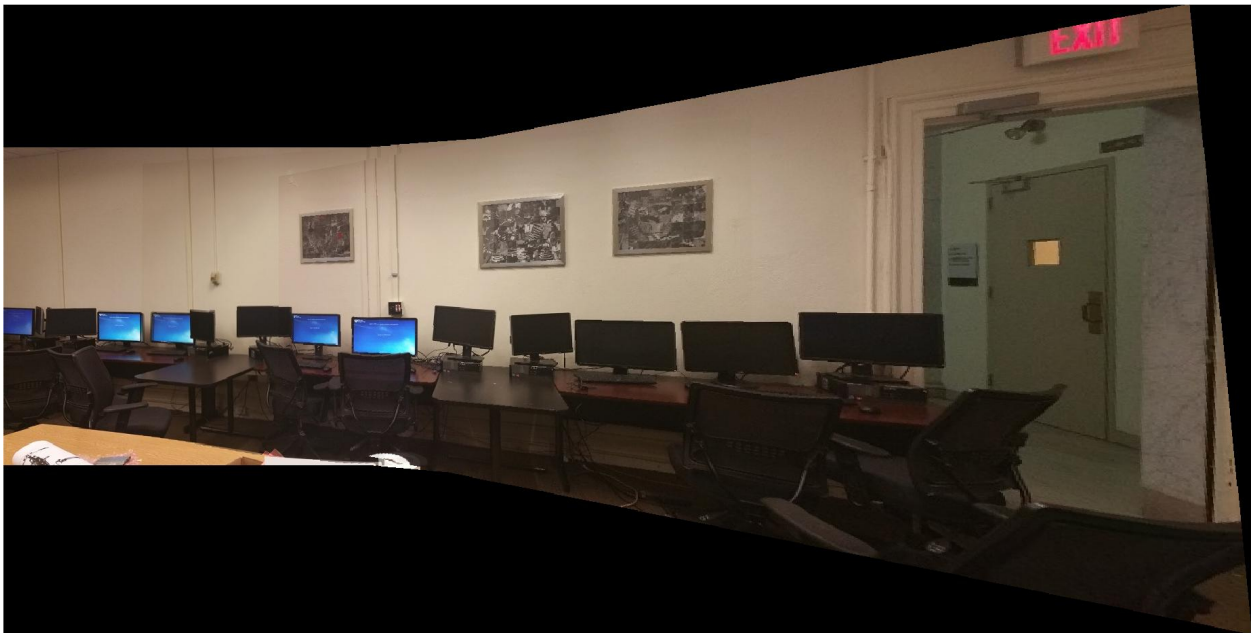

Figure 5: Panorama stitched using projective transform showing bad distortion at edges.

Figure 6: Left: Original input image, Right: Cylindrical output image.

original image where there were pure black pixels. (You can also initialize pixels to NaN's instead of zeros to avoid removing actual zero pixels).

7 Blending images to get a seamless panorama - 15Pts

Produce a panorama by overlaying the pairwise aligned images to create the final output image. The following MATLAB functions should help you in this part, 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. You can also use any third party code for warping and transforming images. The panorama output obtained for Fig. 1 is shown in Fig. 4.

Come up with a logic to blend the common region between images while not affecting the regions which are not common. Here common means shared region, i.e., a part of first image and part of second image should overlap in the output panorama. Describe what you did in your report. You CAN NOT use any built-in MATLAB function or third party code to do this.

Note that the pipeline talks about how to stitch a pair of images, you need to extend this to work for multiple images - you can re-run your images pairwise or do something smarter. Your end goal is to be able to stitch any number of given images - maybe 2 or 3 or 4 or 100, your algorithm should work. If a random image with no matches are given, your algorithm needs to report an error.

8 Extra Credit

- Interesting selection of images. **(5Pts)**.
- Achieve rotation invariance (for feature matching) in the feature descriptor. Hint: Use dominant direction of image patch around the keypoints. **(5Pts)**.
- Implement SIFT-like multi-scale features and keypoint detectors. **(20Pts)**.
- Panorama recognition: Given a set of images that might form panoramas, automatically discover and stitch them together. **(5Pts)**.
- Automatically create "globe" panoramas (Refer to Fig. 7). **(10Pts)**.
- Extrapolate photo boundaries to fill in missing areas of panoramas. Use any image inpainting code (You can use third party code for this part). **(5Pts)**.
- Remove ghosting/moving objects in panoramas (Refer to Fig. 8). Feel free to use third party code for this part. **(10Pts)**.

9 Starter Code

There is no starter code given for this project.

Figure 7: Sample Globe Panorama.

Figure 8: Left: Ghost image, Right: De-ghosted image.

10 Submission Guidelines

We have given three sets of images (training set) in addition to which you need to beg/borrow/steal (or capture) two sets of images (each with $\geq$4 images). You need to submit the output panorama images for all these 5 sets in the report. We'll also release 2 more sets of images (test set) on the day the project is due (If we forget, remind us - if you don't all of you will lose 20 points). Run your code on all the training set images, your two custom sets and the test set images. Your report should contain all these outputs (good or bad - or you lose 20 points).

Describe how you are blending images (while doing stitching) and any other observations in your report. Your report should be in the IEEE double column format and should be typeset in LaTeXand **NOT** contain any code, feel free to include algorithms and math. Use the format provided in the `Draft` folder. Include outputs after each stage and talk about failure cases - we are interested in this. You should also include a detailed `README` file explaining how to run your code.

Submit your own test images (in `Images/CustomSet1, Images/CustomSet2`) and codes (`.m` files) with the naming convention `YourDirectoryID_P1.zip` onto ELMS/Canvas (**Please**

compress it to .zip and no other format). If your e-mail ID is `ABCD@terpmail.umd.edu` or `ABCD@umd.edu` your Directory ID will be `ABCD`.
To summarize, you need to submit these things and in the following strcture: A zip file with the name `YourDirectoryID_proj1.zip` onto ELMS/Canvas. A main folder with the name `YourDirectoryID_P1` and the following sub-folders:

- `Code` with all your code. Make a function called `Wrapper` which takes input arguments as a list of image file names and returns a stitched panorama. This assignment might be auto-graded so please follow the guidelines very carefully. All the file and folder names are case-sensitive.
- `Report` with your report in pdf format (typeset in LaTeXdouble-column IEEE format given to you in `Draft` folder).
- `Images` with two sub-folders `CustomSet1` and `CustomSet2` which has all your custom set images.

If your submission does not comply with the above guidelines, you'll be given **ZERO** credit.

Also, please mention about all the extra credit you have done in your report.

Also make a presentation (need not be slides, you can use your report or the whiteboard or just talk) if you want to present it in the class **Note: Every student should present AT LEAST once in the class (you can volunteer to present for more than one project) and can choose to do for any of the Project except the last one. Good presentations will receive upto 20 bonus points.**

11 Allowed Matlab functions

`imfilter, conv2, imrotate, im2double, rgb2gray, fspecial, imtransform, imwarp, meshgrid, sub2ind, ind2sub` and all other plotting and matrix operation/manipulation functions are allowed. Do not use any MATLAB function that implements feature descriptor, ANMS, RANSAC, feature matching, cylindrical projection and so on.

12 Collaboration Policy

You are restricted to discuss the ideas with at most two other people. But the code you turn-in should be your own and if you **DO USE** (try not it and it is not permitted) other external codes/codes from other students - do cite them. For other honor code refer to the CMSC733 Fall 2016 website.

DON'T FORGET TO HAVE FUN AND PLAY AROUND WITH IMAGES!.

Acknowledgements

This fun project was inspired from 'Computer Vision & Computational Photography' (CIS 581) course of University of Pennsylvania (https://alliance.seas.upenn.edu/~cis581/wiki/index.php?title=CIS_581:_Computer_Vision_%26_Computational_Photography) and from 'Computational Photography' (CSCI 129) course from Brown University (http://cs.brown.edu/courses/cs129/).