CMSC 426, Computer Vision
Project 2: Panorama!

Prof. Yiannis Aloimonos,
Jack Rasiel and Kaan Elgin

February 28, 2018

The aim of this project is to implement an end-to-end pipeline for panorama stitching. We all use the panorama mode on our smart-phones– you'll implement a pipeline which does the same basic thing. 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 Due Dates

As with Project 1, this project is divided into two milestones:

- **MILESTONE 1 DUE DATE:** 11:59:59PM on Thursday, **March 01** 2018
  - Corner detection, ANMS, feature descriptors, and feature matching.
- **MILESTONE 2 DUE DATE:** 11:59:59PM on Thursday, **March 08** 2018
  - The rest of the pipeline: RANSAC, cylinder projection, warping and blending. **To be clear, cylinder projection is part of Milestone 2.**
  - Note that much of the extra credit opportunities are in these parts of the project.

Reaching both milestones on time earns 10 points of extra credit. There is no late penalty for missing the milestones. They are intended to encourage timely completion of the projects: each milestone should be roughly the same amount of work, based on our estimates.

Each milestone will be submitted separately on ELMS; for the full submission guidelines **see section 13**.

2 System Overview

A system diagram is given on the following page, with short descriptions of each step on the page after that:

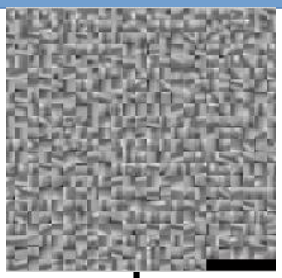
Input Images

Cylinder Projection
(optional, not shown here)

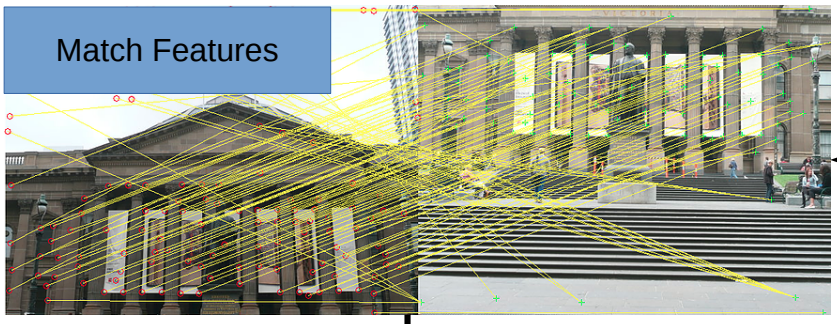
Detect Corners

ANMS

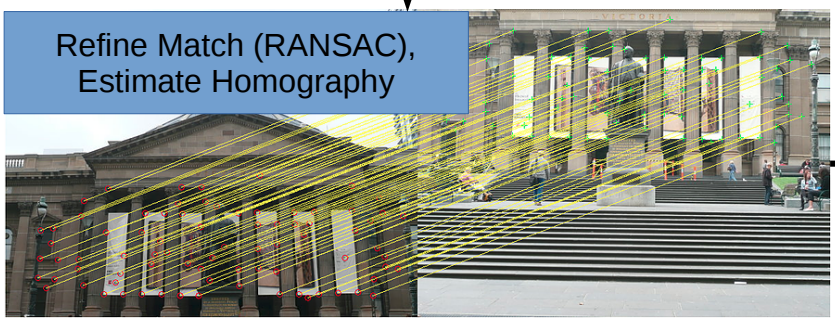
Feature Descriptors



Match Features



Refine Match (RANSAC),
Estimate Homography



Warping and Blending

(result)

2.1 System Overview

A brief description of the panorama stitching pipeline:

- **Cylinder Projection:** Project images onto a cylinder, to reduce distortion at the panorama's edges.
- ***Detect Corners:*** *identify corner points in your images. For this project, it's just a function call.*
- **ANMS:** pick out the stronger corner points.
- **Feature Descriptors:** create descriptors for the corner points, so they can be matched between images (in the next step).
- **Feature Matching:** Match feature descriptors from different images, to find possible point correspondences.
- **RANSAC and Homography Estimation:** refine the feature point matches, and use the correspondences to estimate homographies between images.
- **Image Warping and Blending** Use the estimated homographies to warp the images onto one another, and apply blending to reduce the appearance of seams where they fit together.

3 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 pure translation, or pure rotation (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.

We might give bonus points for interesting image selections!

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

The aim of this step is to detect strong corners, spread widely across the image. Initially, we'll detect corner features using Matlab's `cornermetric`. The output is a matrix of corner scores: the higher the score, the higher the probability of that pixel being a corner. Visualize the output using the MATLAB function `imagesc`.

We want to find particularly strong corners, spread across many different parts of the image. Find the N_{strong} strongest corners using the MATLAB function `imregionalmax`. The ANMS algorithm is described next:

Consider the case where we are stitching first 2 images shown in the system diagram. The output of ANMS is shown in Fig. 1. To plot dots on the images: `imshow(..); hold`

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**
      ED $= (x_j - x_i)^2 + (y_j - y_i)^2$
    end
    if $ED < r_i$ **then**
      $r_i =$ ED
    end
  end
end

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

Algorithm 1: ANMS algorithm.

`on; plot(..); hold off;`. Note that plot uses x and y where x is the column number and y is the row number.

Figure 1: Output of ANMS on the sample images.

5 Feature Descriptor - 25Pts

In the previous step, you found the feature points (i.e., the locations of the N_{best} best corners after ANMS). You need to generate a *descriptor* for each feature point. This is much like how we generated Textons in project 1: each of the filters encoded different information about the texture surrounding a pixel. The information we'll encode here is a bit simpler:

Take a patch of size 40×40 centered (this is very important) around the keypoint. 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`*** *in this project!*). Now, sub-sample the blurred output to 8×8 (this reduces the dimensionality of the descriptor; in practice the gain in computational efficiency outweighs the information loss). Then reshape to obtain a 64×1 vector. Standardize the vector to have zero mean and variance of 1 (i.e., subtract all values by mean and then dividing by the standard deviation). Standardization makes the descriptor more robust to variations in appearance between images.

6 Feature Matching - 10Pts

In the previous step, you represented each feature point as a 64×1 feature vector. Now, we'd like to identify matching points in different images. In Vision lingo, we call this finding "feature correspondences" between the 2 images.

We'll quantify the similarity between two feature descriptors with a simple sum of squared differences on their pixels. For each point in image 1, compute the sum of square differences between with all points in image 2. Take the ratio of the best match (lowest distance) to the second best match (second lowest distance)– if this is above some ratio, keep the match. Repeat for all points in image 1. This process yields a set of points correspondences, with which we will now estimate the transformation between the two images (AKA the *homography*).

Use the function `showMatchedFeatures` provided to visualize feature correspondences (Sample output is shown in Fig. 2).

Figure 2: Output of `showMatchedFeatures` on first 2 images.

7 RANSAC to estimate robust Homography - 15Pts

Not all the matches we found in the previous step will be useful. To help separate good matches from bad, we use the RANSAC algorithm.

Recall the RANSAC steps are:

- Select four feature pairs (at random), p_i , p_i^1.
- 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.

8 Warping Images Together - 5 Points

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. The panorama output obtained for Fig. **??** is shown in Fig. 3.

Figure 3: Final Panorama output.

9 Cylindrical Projection

The pipeline, as implemented so far, doesn't work well for very wide panoramas. When you try to stitch a lot of images with translation, a simple projective transformation (applying homography) the images will be stretched or shrunken too much at the edges. A sample case is shown in Fig. 4.

To overcome these distortion problems, we will add a **cylindrical projection** step to the start of the pipeline. For an intuition of what we are doing, and the problem we're solving, see the relevant link in the **Resources** section.

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. 5.

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 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).

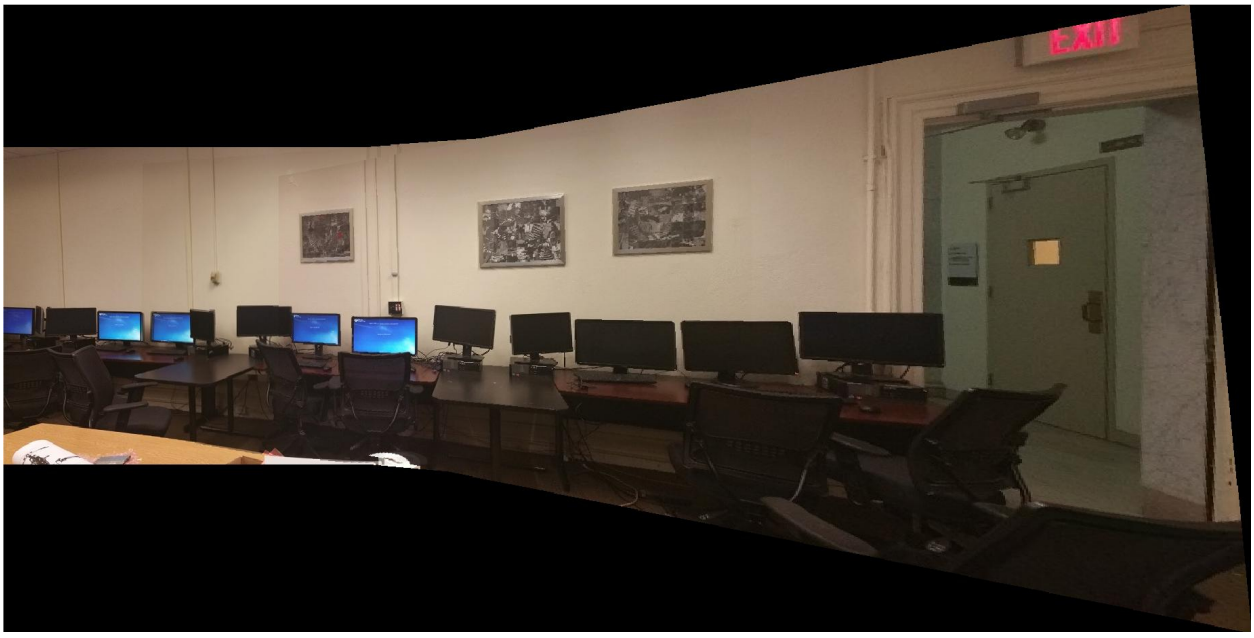


Figure 4: Panorama stitched using projective transform showing bad distortion at edges.

Figure 5: Left: Original input image, Right: Cylindrical output image.

10 Image Blending - 15 Pts

Variations in lighting (and other factors) between images can cause unsightly seams in the resulting panorama. To reduce the appearance of seams, overlapping images should "fade" into one another, rather than having a simple overlap. You CAN NOT use any built-in MATLAB function or third party code to do this.

For some hints on how to do this, see slides 29 through 31 of the Stanford slide deck linked in the Helpful Resources section.**15Pts**

11 Extra Credit

- Interesting selection of images. **(5Pts)**
- Use 8 or more images for stitching. **(5Pts)**
- Achieve rotation invariance (for feature matching) in the feature descriptor. Hint: Use dominant direction of image patch around the keypoints. **(10Pts)**.

12 Starter Code

Put the code which executes your pipeline in `Code/MyPanorama.m`. **DO NOT modify the function definition.** When run, this function must load a set of images from `Images/input/`, and return the resulting panorama. Feel free to put other functions in external files– but make sure you include them in your submission!

13 Submission Guidelines

If your submission does not comply with the following guidelines, you'll be given ZERO credit:

13.1 File tree and naming

Your submission on Canvas must be a zip file, following the naming convention **"YourDirectoryID_proj2_milestone#.zip"**. For example, jrasiel_proj2_milestone1.zip (for milestone 1). The file **must have the following directory structure**, based on the starter files:

```
YourDirectoryID_proj2_milestone#.zip
├── Code/
│   ├── MyPanorama.m
│   └── (Any dependencies of MyPanorama.m)
├── Images/
│   ├── CustomSet1 - Your images here!
│   ├── CustomSet2 - Your images here!
│   ├── Set1 - These are the provided images.
│   ├── Set2
│   ├── Set3
│   └── input
└── report.pdf - Your report.
```

When run, **MyPanorama.m** must load a set of images from `Images/input/`, and return the resulting panorama.

13.2 Report

You will be graded primarily based on your report. We want you to demonstrate an understanding of the concepts involved in the project, and to show the output produced by your code.

Include visualizations of the output of each stage in your pipeline (as shown in the system diagram on page 2), and a description of what you did for each step, including any problems you encountered and/or interesting solutions you implemented. Your report must be full English sentences, **not** commented code.

Be sure to include the output panoramas for **all five image sets**.

A reminder: 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 projects.

14 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.

15 Helpful Resources

- Cylinder Projection - visualization and helpful explanation from the Stanford computer graphics department.
- hrefhttp://pages.cs.wisc.edu/ dyer/cs534/slides/08-mosaics.pdfExcellent slides on panorama stitching, from CS534 at U Wisconsin Madison.
- A good selection on RANSAC from the Hartley and Zisserman textbook was distributed on 02-22; if you did not receive it, please contact the TAs.

16 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 should be the result of you exercising your own understanding of it. If you reference anyone else's code in writing your project, you must properly cite it in your code (in comments) and your writeup. For the full honor code refer to the CMSC426 Spring 2018 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).