CSCI 4830/5722 Computer Vision, Spring 2018
Instructor: Fleming
Homework 2, Due Sunday, February 18th, by 11:55pm

Image Mosaics

For this assignment, you will implement an image stitcher that uses image warping and homographies to automatically create an image mosaic. We will focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps. For simplicity, we'll specify corresponding pairs of points manually using mouse clicks.

Note: There are some built-in Matlab functions that could do much of the work for this project. However, to get practice with the workings of the algorithms, we want you to write your own code. Specifically, you may not use any of these functions in your implementation: cp2tform, imtransform, tformarray, tformfwd, tforminv, maketform.

Provided files:
Two image files that can be used for the mosaic: *uttower1.jpg* and *uttower2.jpg*

What You Have to Do

Task 1 (10 points) Getting correspondences:
Write a function named `getPoints()` to get manually identified corresponding points from two views. Look at Matlab's `ginput` function for an easy way to collect mouse click positions. The results will be sensitive to the accuracy of the corresponding points; when providing clicks, choose distinctive points in the image that appear in both views.
Your function will take in two provided images: *uttower1.jpg* and *uttower2.jpg.* The function will return a 10 x 4 matrix, where each row is a pair of corresponding points, and the 4 columns represent the (row,column) position from the 1st image, followed by the (row,column) position from the 2nd image.

Note: in the next task, you will have to compute the homography that relates these point pairs. Remember to choose points (features) that are as much as possible:

a) coplanar, and
b) visible in both images.

Task 2 (30 points) Computing the homography parameters:
Write a function `computeH()` that takes the set of corresponding image points returned by the function `getPoints()` from Task 1, and computes the associated 3 x 3 homography matrix H, with the help of a RANSAC-like algorithm:

1. Pick 4 random point correspondences from the 10 supplied.
2. Compute the associated 3 x 3 homography matrix H

This matrix transforms any point p_i in one view to its corresponding homogeneous coordinates in the second view, p_i', such that

$$\lambda * p_i = H * p_i'$$

Note that p_i and p_i' are both vectors with 3 elements. Useful Matlab functions include the '\' operator (help `mldivide`), `cat` and `reshape`.

3. Calculate the Euclidian distance between the selected points in one image, and the projection of their corresponding points using the computed homography. Note that p_i' is in homogeneous coordinates, but after multiplying with H, the resulting 3 x 1 vector will not have 1 as the third value. You will need to divide by the 3rd value the get the new (x,y) coordinates.

4. Save the distance value and repeat from step 1 for 20 times.

Your function should return the homography matrix with the smallest distance.

Task 3 (20 points) Warping between image planes:
Write a function `warp1()` that can take the recovered homography matrix and one image, and return a new image that is the warp of the input image using H (or its inverse) . Since the transformed coordinates will typically be sub-pixel values, you will need to sample the pixel values from nearby pixels. Feel free to use the `sampleBilinear.m` you developed for the first assignment. For color images, warp each RGB channel separately and then stack together to form the output. To avoid holes in the output, use inverse warp rather than direct mapping.

To compute the bounding box of the destination image, you will need to warp the points from the source image into the reference frame of the destination. Then sample all points in that destination bounding box from the proper coordinates in the source image. Note that transforming all the points will generate an image of a different shape / dimensions than the original input. It is ok to have some areas of the new image be black (0).

Useful Matlab functions: `round, interp2, meshgrid, isnan, inv`.

Task 4 (20 points) Create the output mosaic:
Once we have the source image warped into the destination image's frame of reference, we can create a merged image showing the mosaic. Create a new image large enough to hold both (registered) views; overlay one view onto the other, simply leaving it black wherever no data is available. Don't worry about artifacts that result at the boundaries.

You are free to use a method/convention of your own choosing, for the overlap areas.

Task 5 (20 points) After writing and debugging your functions:

1. [5 pts] Write a script in which you apply your functions to the provided pair of images and display the output mosaic. Use subplots to display the original images and the resulting mosaic.
2. [5 pts] Show two additional examples of mosaics you created using images that you have taken. You can make a mosaic from two or more images of a broad scene that requires a wide-angle view to see well. Include these examples in the same script.
3. [10 pts] Warp one image into a "frame" region in the second image. To do this, let the points from the one view be the corners of the image you want to insert in the frame, and the let the corresponding points in the second view be the clicked points of the frame (rectangle) into which the first image should be warped. Use this idea to replace one surface in an image with an image of something else. For example -- overwrite a billboard with a picture of your dog, or project a drawing from one image onto the street in another image, or replace a portrait on the wall with someone else's face, or paste a Powerpoint slide onto a movie screen, …
For all examples, play around a bit with the choice of points for the correspondence pairs until you get a reasonable alignment. Include this exercise in the same script.

Submitting the assignment:
Make sure each script or function file is well commented and it includes a block comment with your name, course number, assignment number and instructor name. Zip all the .m and image files together and submit the resulting *.zip* file through Moodle as Hmwk 2 by Sunday, February 18th, by 11:55pm.

Tips:
- It can be useful when debugging to plot the corners and clicked points from one view on top of the second view after transforming them via H. Use axis([minx, maxx, miny, maxy]); to adjust the viewing window so that you can see all points.
- You will need the inverse of the homography matrix to transform "backwards".

- Be aware that Matlab's image (matrix) indices are specified in (row,col) order, i.e., (y,x), whereas the `plot` and `ginput` functions use (col,row) order, i.e., (x,y).
- Check the order of the clicked corresponding points, to make sure your code uses the intended corresponding point pairs.
- As usual, be careful with how images are cast for computations and display (double vs. uint8). In particular, for your `sampleBilinear.m` function, be sure to pass a matrix of doubles for the image input.
- When collecting your own images, be sure to either maintain the same center of projection (hold the camera at one location, but rotate between views), or else take shots of a scene with a large planar component (a building, maybe). In either case, use a static scene. Textured images that have distinctive points you can click on are good. Also ensure that there is an adequate overlap between the two views.