CSCI 4831/5722 Computer Vision, Spring 2021
Instructor: Fleming
Homework 2, Due Monday, February 8th, by 10:00 pm

# Image Mosaics

For Homework 2, 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 you 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 (the list is not final, and more functions might be added later).

**Provided files:**
Two image files that can be used for the mosaic.

# 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. You will need **ten points**. 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: *Image1.jpg* and *Image2.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 1 :** Check the order of the clicked corresponding points, to make sure your code uses the intended corresponding point pairs.

**Note 2 :** 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.

**Note 3 :** After your function returns the corresponding points data, we recommend saving the data to a `.mat` file. In subsequent runs, you don't want to always select new points by clicking inside the image, and you can comment the `getPoints()` function, and load the file instead. Take a look at the built-in Matlab functions `load` and `save`.

**Task 2 (40 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 **four** random point correspondences from the ten 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 `svd, cat` and `reshape.`

3. Project ***all ten points*** from one image onto the other image using the computed homography. Calculate the Euclidean distance between these projected points, and their expected correspondences. Your task is to calculate the average reprojection error.

   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. (See converting *from* homogeneous coordinates to cartesian coordinates in lecture 4 slides).

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

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

**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).

**Task 3 (50 points) Warping images to produce the output mosaic**

Write a function `warp1()` that can take three input parameters (the recovered homography matrix and both images), and return the output mosaic image. Consider the first input parameter image the *reference image*, and then warp the other image on the plane of the reference image.

1. Generate a "blank" final output image. Compute the size of the output image by computing the range of warped image coordinates for each input image.
    a. Map the four corners of each image to find the minimum and the maximum x and y coordinates to determine the size of the output image. This process is named "bounding box" determination.
    b. Compute the x-offset and the y-offset values for the *reference image*.
    c. Initialize all pixel values to "black".
2. Use inverse warping to map each pixel in the output image to one input image, both input images, or none
    a. If the mapping is to just one image, use bilinear interpolation to compute the pixel intensity.
    b. If the mapping is to both images, this means this is a part of the scene that is imaged in both images. Use bilinear interpolation to compute the intensity as mapped from each input image, then use a method of your choosing to decide the final pixel intensity (you could use only the reference image, or only the other image, or a combination/average/maximum/weighted_sum of both intensities)
    c. If the mapping is outside either of the input images, the resulting pixel intensity will be 0 (black), and this is OK.

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

**Tips:**

- As usual, be careful with how images are cast for computations and display (double vs. uint8). In particular, for your `bilinear interpolation` computation, be sure to pass a matrix of doubles for the image input.

**Task 4 (20-30 points)** After writing and debugging your functions:
1. [10 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 each = 10 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] **Only for graduate students.** 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 then 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.

**Tips:**

- 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.

**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 Canvas as Homework 2 by Monday, February 8th, by 10:00pm.