ECE 4984/5554: Computer Vision, Fall 2015

PS3

Christopher Mobley

Due: Monday, 19 OCT 15

1 Programming: Image mosaics [100 points]

In this exercise, 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 or inbuilt matlab functions. For extra credit, you can optionally implement an automated correspondence process with local feature matching. Implement the following components as required:

1. Getting correspondences: write code to get manually identified corresponding points from two views. For Matlab, look at ginput function for an easy way to collect mouse click positions. Or checkout the function cpselect in Matlab's Image Processing Toolbox for help selecting corresponding points. For Python, look at matplotlib.widgets.Cursor function or simply collect the mouse in the plot. 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.

2. Computing the homography parameters: [20 points]
   Write a function H = computeH(t1, t2) that takes a set of corresponding image points t1, t2 (both t1 and t2 should be 2xN matrices) and computes the associated 3 x 3 homography matrix H. The function should take a list of $n \geq 4$ pairs of corresponding points from the two views, where each point is specified with its 2d image coordinates. Verify that the homography matrix your function computed is correct by mapping the clicked image points from one view to the other, and displaying them on top of each respective image. (imshow, followed by hold on and plot). Be sure to handle homogenous and non-homogenous coordinates correctly. Save this function in a file called computeH.m(py) and submit it. Note: Your estimation procedure may perform better if image coordinates range from 0 to 2. Consider scaling your measurements to avoid numerical issues. Look at notes on how to estimate a homography here.

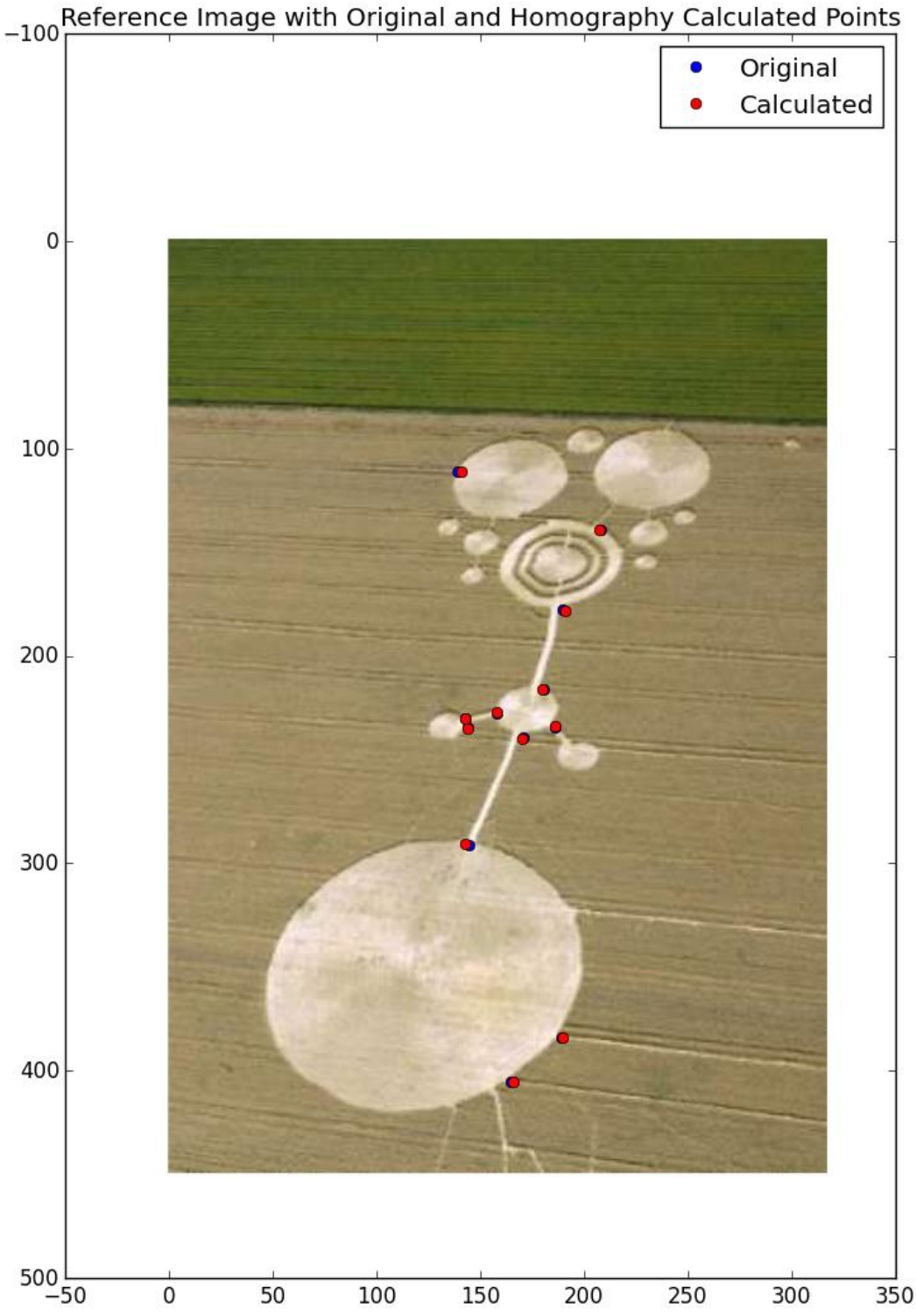

Figure 1: Homography accuracy on crop2.jpg

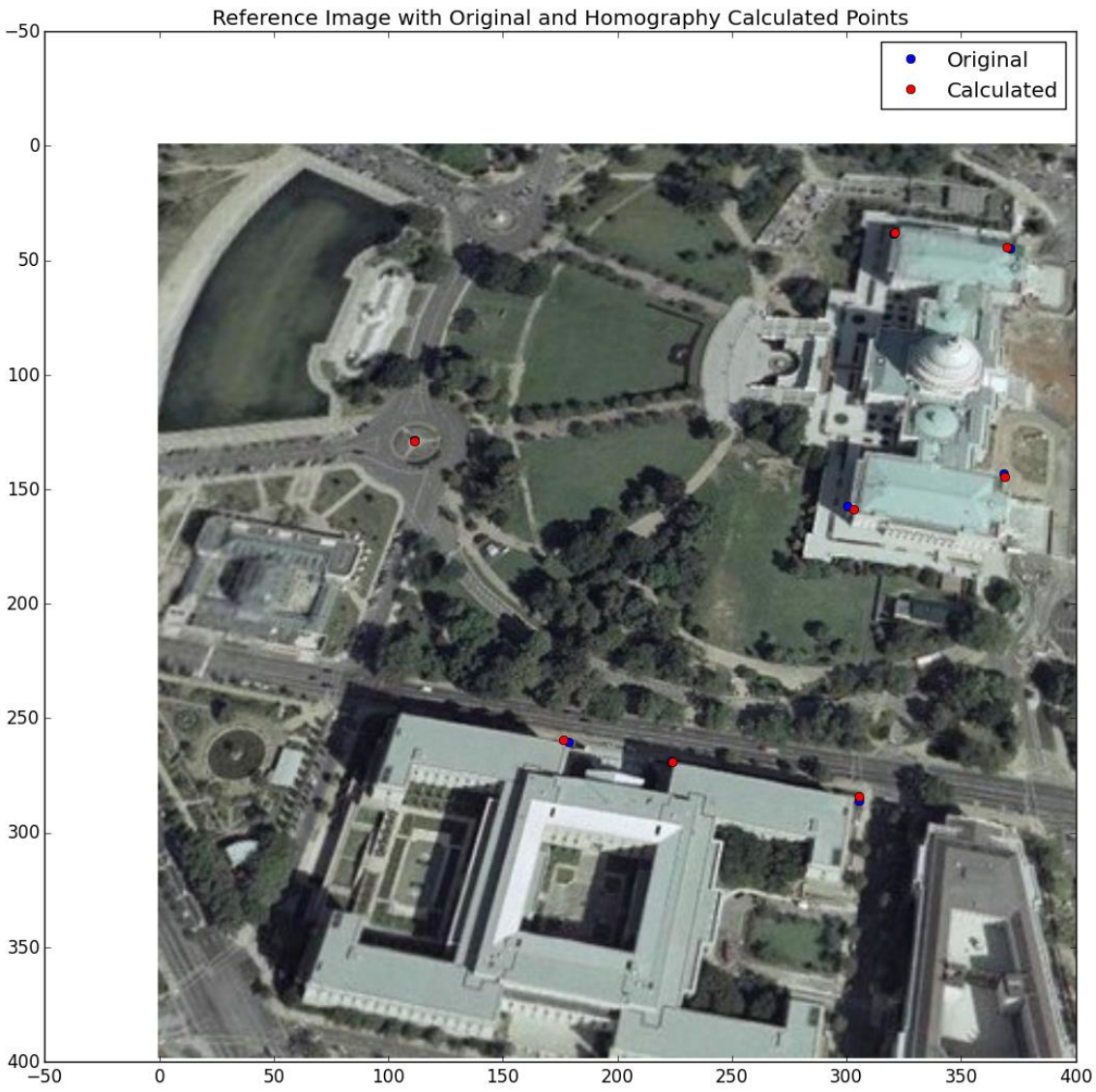

Figure 2: Homography accuracy on wdc2.jpg

3. Warping between image planes: [30 points]
   Write a function [warpIm,mergeIm] = warpImage(inputIm, refIm, H) which takes as input an image inputIm, a reference image refIm, and a 3x3 homography matrix H, and returns 2 images as outputs. The first image is warpIm, which is the input image inputIm warped according to H to be in the frame of the reference image refIm. The second output image is mergeIm, a single mosaic image with a larger field of view containing both the input images. All the inputs and outputs should be MxNx3 matrices. To avoid holes, use an inverse warp. Warp the points from the source image into the reference frame of the destination, and compute the bounding box in that new reference frame. 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. Also note that the input and output images will be of different dimensions. Once you have the input image warped into the reference image's frame of reference, create a merged image showing the mosaic. Create a new image large enough to hold both the 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. Save this function in a file called warpImage.m(py) and submit it.

4. Apply your system to the following pairs of images, and display the output warped image and mosaic in your answer sheet.
   Pair 1: crop1.jpg, crop2.jpg. For this pair use these corresponding points: cc1.mat, cc2.mat (Matlab) or cc1.npy, cc2.npy (Python) .
   Pair 2: wdc1.jpg, wdc2.jpg. For this pair use appropriate corresponding points of your choice. Name the variables containing these points as points1 and points2 and submit them in a file called points.mat(npy). points1 and points2 should be matrices of size 2xN. [15 points]

Crop1.jpg

Crop2.jpg

Figure 3: Input Crop images

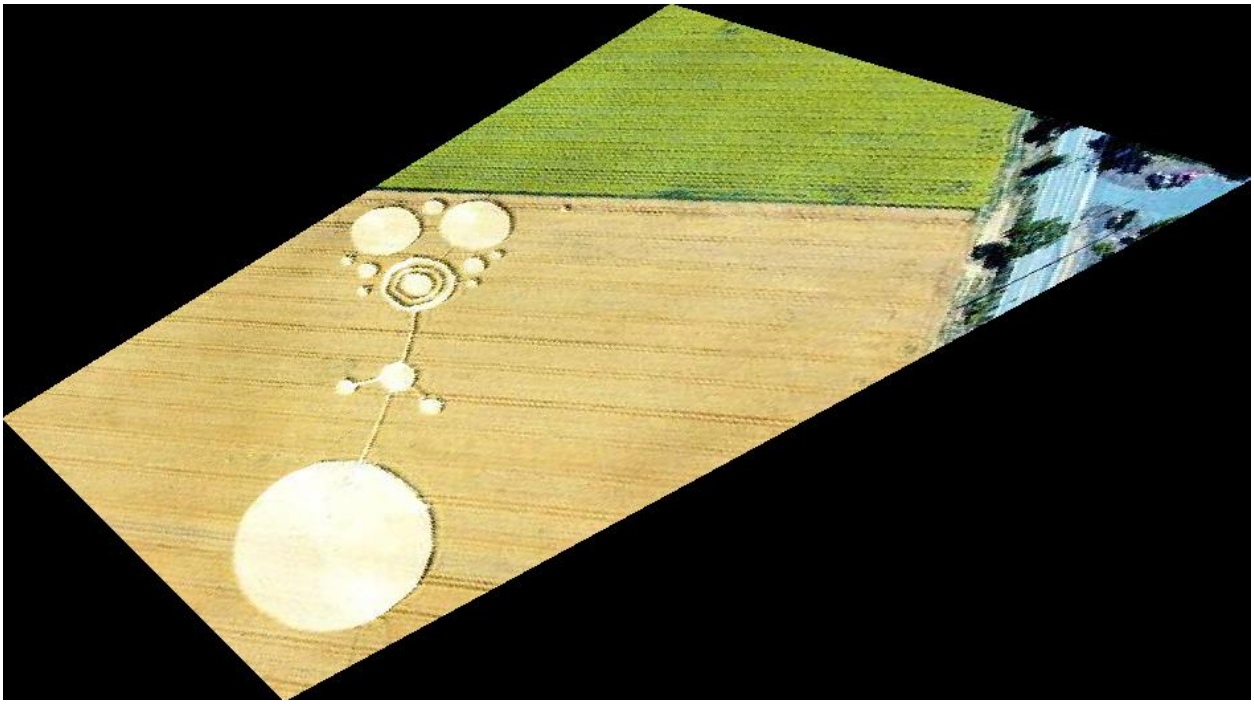

Figure 4: Crop1.jpg warped image

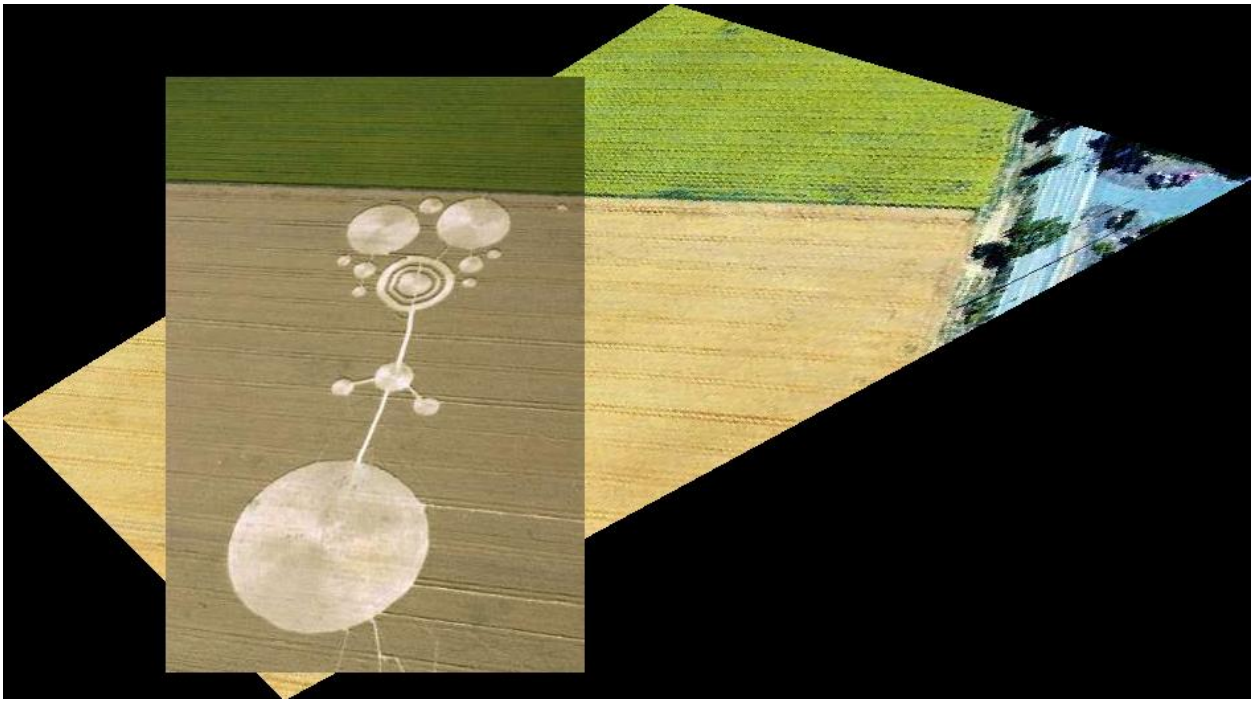

Figure 5: Crop image mosaic

wdc1.jpg

wdc2.jpg

Figure 6: Input Washington DC images

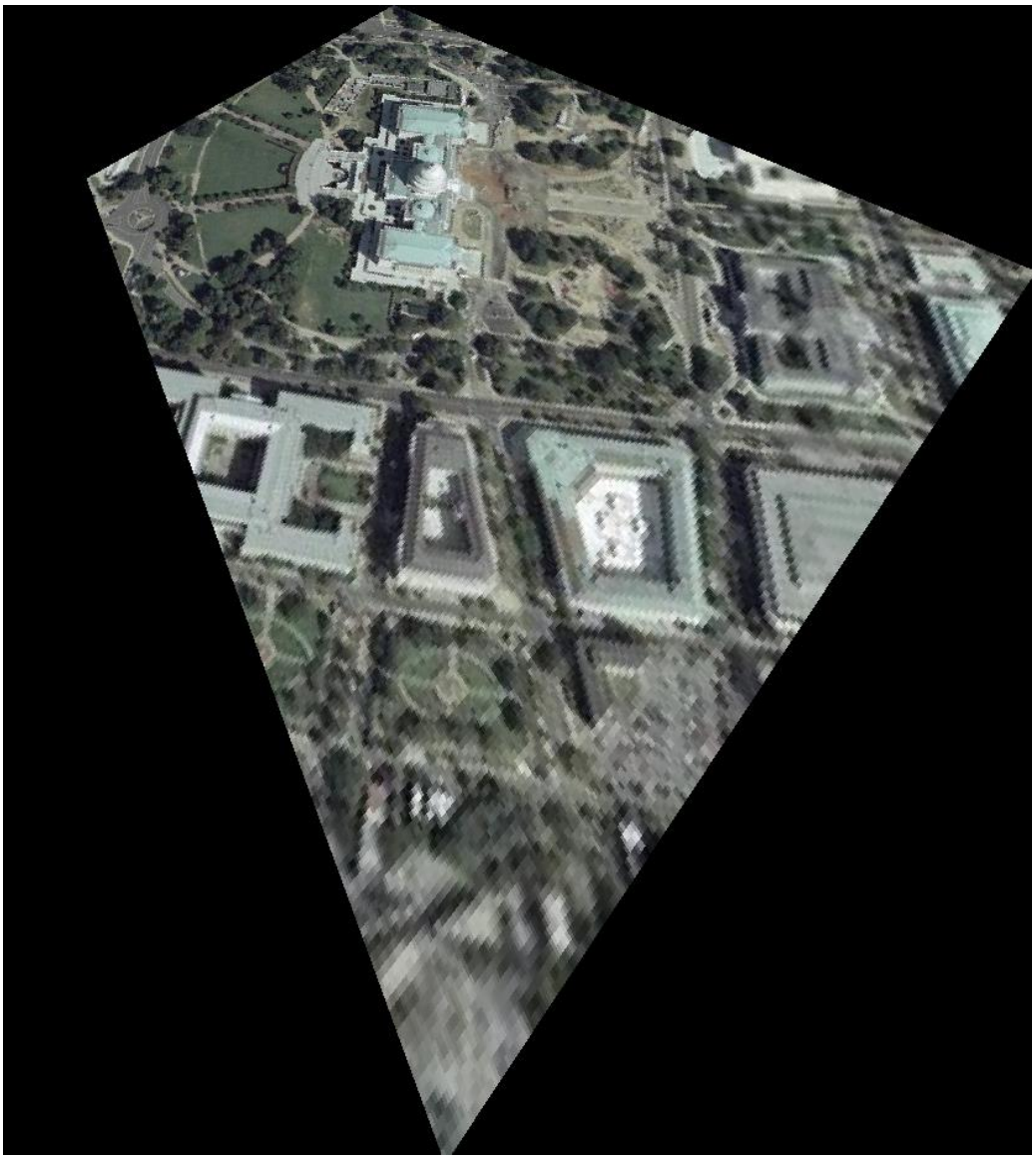

Figure 7: wdc1.jpg warped image

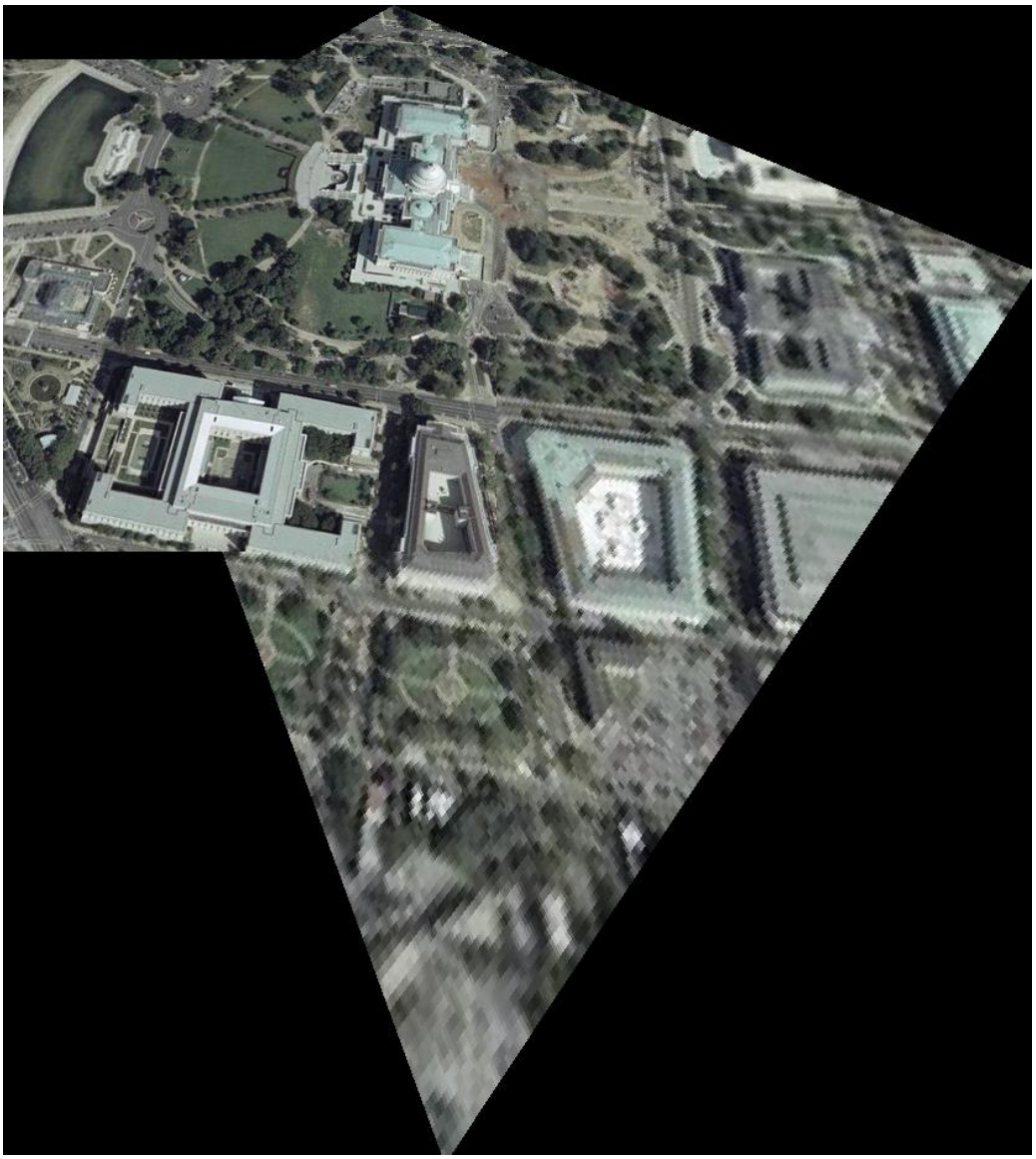

Figure 8: Washington DC image mosaic

5. Show one additional example of a mosaic you create using images that you have taken. You might make a mosaic from two or more images of a broad scene that requires a wide angle view to see well. Or, make a mosaic using two images from the same room where the same person appears in both. [20 points]

goodwin1.jpg

goodwin2.jpg

Figure 9: Input Goodwin images

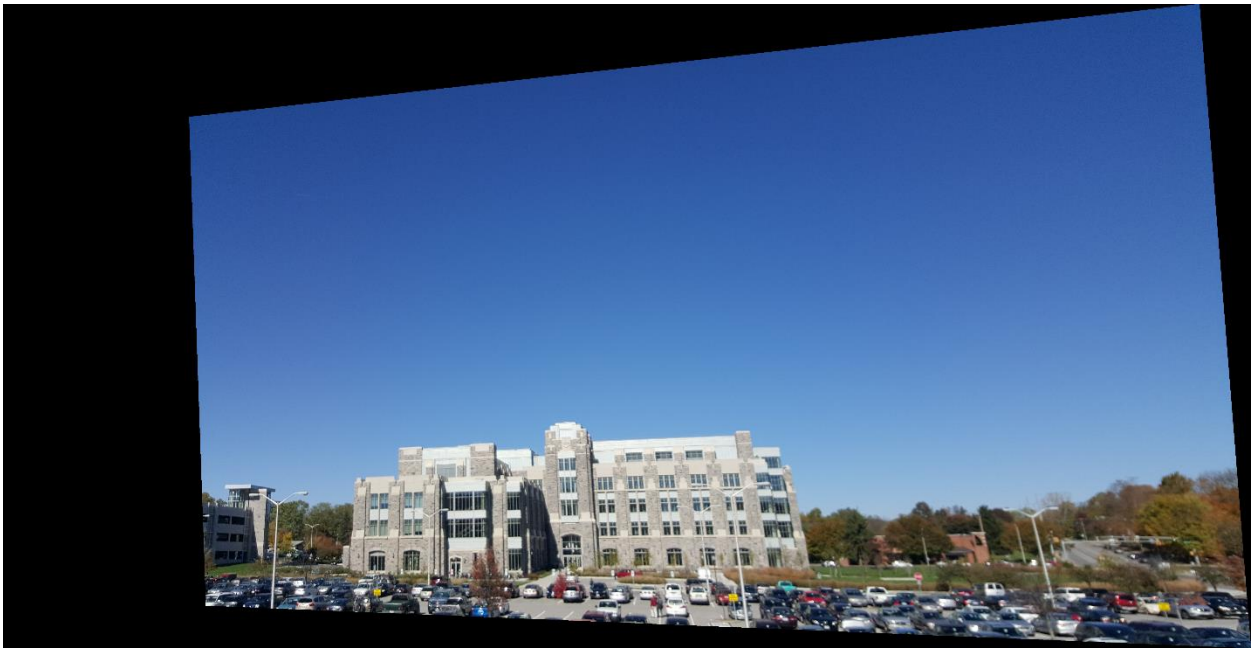

Figure 10: goodwin1.jpg warped image

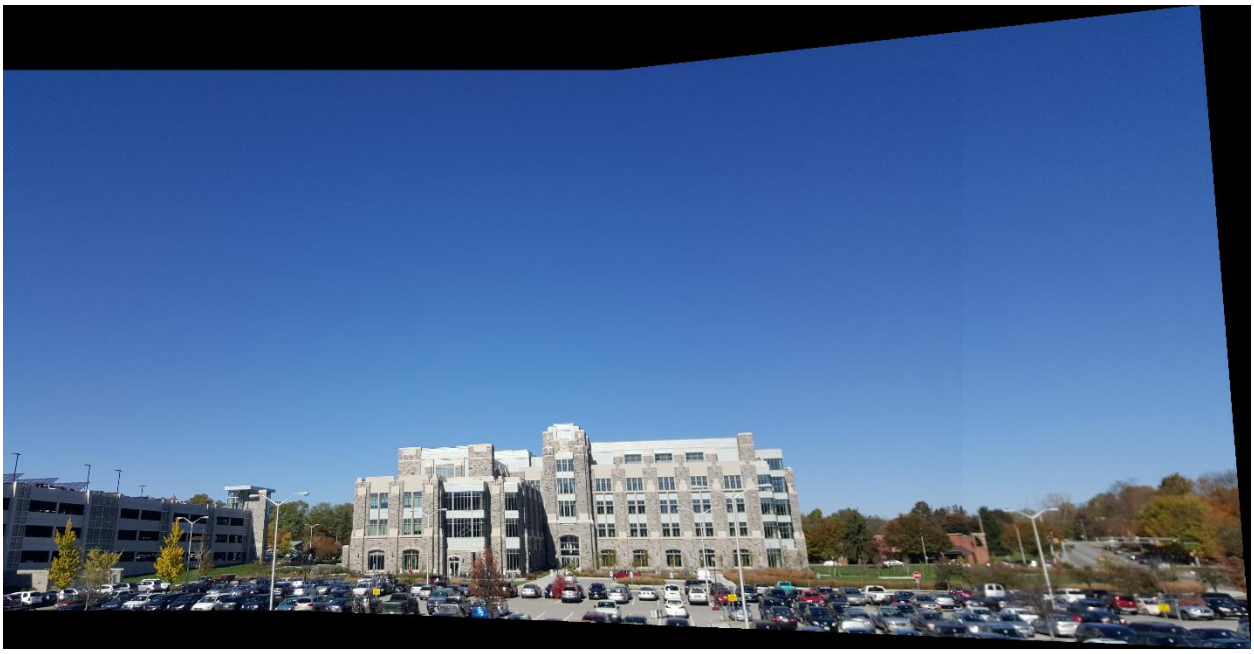

Figure 11: Goodwin image mosaic

6. 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 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, etc. Display the results in your answer sheet. [15 points]

billboard1.jpg

billboard2.jpg

Figure 12: Input billboard images

Figure 13: billboard1.jpg warped image

Figure 14: Billboard image mosaic

2 [OPTIONAL] Extra credit [up to 10 points each, max 30 points]

3 Rectify an image with some known planar surface (say, a square floor tile, or the rectangular face of a building facade) and show the virtual fronto-parallel view. In this case there is only one input image. To solve for H, you define the correspondences by clicking on the four corners of the planar surface in the input image, and then associating them with hand-specified coordinates for the output image. For example, a square tile’s corners from the non-frontal view could get mapped to [0 0; 0 N; N 0; N N] in the output.

Figure 15: Input tile floor image

Figure 16: Output tile image

Note: I discussed this assignment with Murat Ambarkutuk and Orson Lin