CS 6476: Computer Vision, Fall 2019
PS3

Instructor: Devi Parikh

Due **before**: Wednesday, October 16th, 11:58:59 pm

Instructions

1. Answer sheets, code and input/output images must be submitted on Canvas. Hard copies will not be accepted.
2. Please provide a pdf version of your answer sheet named: LastName_FirstName_PS3.pdf.
3. If your scripts are in Python, please use '.py' as file extension. If your scripts are in Matlab, please use '.mat' as file extension.
4. Please put all your code, input/output images and answer sheets in a folder (no subdirectories). Make sure your code is bug-free and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.
5. If plots are required, you must include them in your answer sheet (pdf) and your code must display them when run. Points will be deducted for not following this protocol.
6. Your code and plots should use the same filenames mentioned in the question (if present). Variables in your code should use the same names that are mentioned in the question (if present).
7. Please make sure that the folder is named LastName_FirstName_PS3_mat if using Matlab and LastName_FirstName_PS3_py if using Python.
8. Zip the above folder and name the zipped file LastName_FirstName_PS3_mat.zip if using Matlab and LastName_FirstName_PS3_py.zip if using Python. Submit *only* the zip file.

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.

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]

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]

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]

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

1. Replace the manual correspondence stage with automatic interest point detection and local feature matching. Check out available code here to compute the local interest points and features:

http://www.vlfeat.org/overview/sift.html
http://www.robots.ox.ac.uk/~vgg/research/affine/detectors.html

2. Implement RANSAC for robustly estimating the homography matrix from noisy correspondences. Show with an example where it successfully gives good results even when there are outlier (bad) correspondences given as input. Compare the robust output to the original (non-RANSAC) implementation where all correspondences are used.

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.

4. Make a short video in the style of the HP commercial's video which you saw in class. Building on #3 above, let the frame in the output video move to different positions over time, and warp the framed image into the correct position for every video frame in the sequence.

Matlab hints:

1. Useful functions: `round, interp2, meshgrid, isnan, eig, inv`

2. 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`.

Python hints:

1. Useful Modules: `numpy, scipy, skimage, matplotlib`

2. Useful Functions: `numpy.linalg.eig, numpy.linalg.inv, numpy.tile, scipy.misc.imread numpy.meshgrid`

3. There are some Python library 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.

This assignment is adapted from the following two sources:

1. PS3 from Kristen Grauman's CS 376: Computer Vision at UT Austin.
2. HW1 of Martial Herbert's course 16-720 Computer Vision taught in Fall 2005.