CS 378 Computer Vision
Problem set 3
Out: Tuesday, October 13
Due: Tuesday, October 27, 11:59 PM

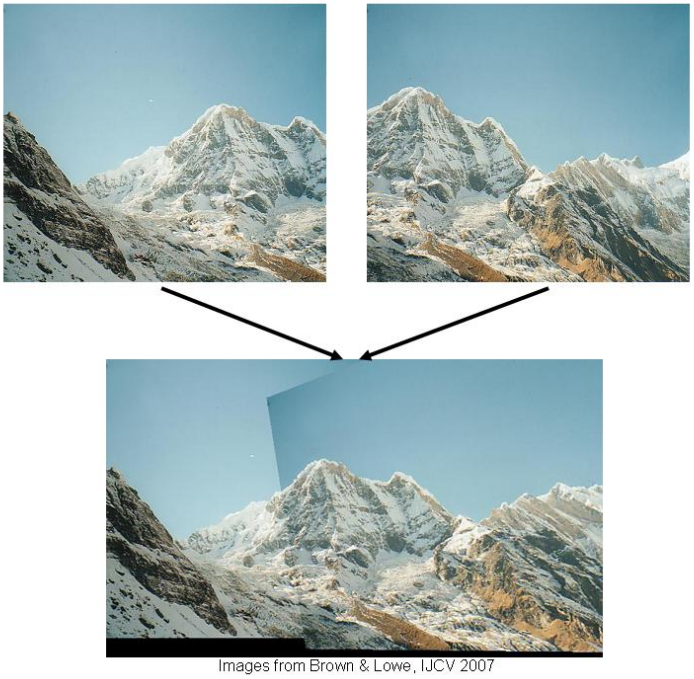
See submission instructions at the end of this document.

I. Short answer problems [20 points]

1. As a camera's focal length gets smaller, what is the effect on the image's field of view? Explain briefly.

2. Define a method to determine if a building's (planar) surface is parallel to the image plane. State any assumptions.

II. Programming problem: image mosaics [80 points]



Images from Brown & Lowe, IJCV 2007

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.

****** 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`. ******

Here are the main components you will need to implement:

- **Getting correspondences**: write code 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.

- **Computing the homography parameters [20 points]**: write a function that takes a set of corresponding image points and computes 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 $\lambda p_i = Hp_i'$. Note that p_i and p_i' are both 3-vectors. 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.

  We can set up a solution using a system of linear equations $Ax = b$, where the 8 unknowns of H are stacked into an 8-vector x, the $2n$-vector b contains image points from one view, and the $2n \times 8$ matrix A is filled appropriately so that the full system gives us $\lambda p_i = Hp_i'$. Let $H_{3,3} = 1$. Solve for the unknown homography matrix parameters. [Useful Matlab functions: '\' operator (help `mldivide`), `reshape`]

  Verify that the homography matrix your function computes 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.

- **Warping between image planes [20 points]:** write a function that can take the recovered homography matrix and an image, and return a new image that is the warp of the input image using H. Since the transformed coordinates will typically be sub-pixel values, you will need to sample the pixel values from nearby pixels. For color images, warp each RGB channel separately and then stack together to form the output.

  To avoid holes in the output, 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. [Useful Matlab functions: `round`, `interp2`, `meshgrid`, `isnan`.]

- **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.

After writing and debugging your system:

1. **[10 pts]** Apply your system to the provided pair of images, and display the output mosaic.
2. **[15 pts]** Show two additional examples of mosaics you create 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. Or, make a mosaic using two images from the same room where the same person appears in both.
3. **[15 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.

III. [OPTIONAL] Extra credit [up to 10 pts each, max 30 pts]

1. 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.
2. Refine the initial correspondences automatically by searching small patches near the clicked points for a good alignment, as measured by SSD between the pixels. Show an example where the refined points improve the mosaic alignment.
3. Rectify an image with some known planar surface (say, a square floor tile, or the rectangular face of a building façade) 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 commercials. 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.

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 specifed 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, if you use the `interp` function, be sure to pass it 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. 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.

- There are two weeks allocated for this assignment. Please look through it early, and come see the instructors if/when you have questions.

Submission instructions: what to hand in

Electronically:

- Your well-documented Matlab code .m files.
- A first pdf file file1.pdf containing:
    - Your name and CS login ID at the top.
    - Your answers to Section I, numbered.
- A second pdf file file2.pdf containing
    - Your name and CS login ID at the top.
    - Your image results and accompanying brief explanations for Section II. This file will be posted online; be sure to include credit for images if necessary.
    - (optional): any results and descriptions for extra credit portions in Section III.

Submit all the above with one call to turnin:

```
>> turnin --submit jaechul pset3 file1.pdf file2.pdf
codeFileXYZ.m codeFileABC.m etc.
```

Hardcopy: Print out the pdf files, and either bring it to class or drop off at Jaechul's office in CSA on Tues 10/26. Do not print out code. As usual, the hardcopy must be the same as what is submitted electronically.

Remember that partial submissions are allowed; we just ask that you write especially good comments on your code in this case.