Assignment 2

CS283, Computer Vision
Harvard University

Due Monday, Sep. 24, at 5:00pm

This assignment deals with projective transformations. It is *substantially* longer than Assignment One, and I strongly recommend that you complete Questions 1 & 2 before Friday. As usual, there is a helpful "Hints and Information" section at the end of this document. Be sure to format your submission according to the guidelines and submit using the iSites "drop-box".

In Question 4, you will be computing and applying planar projective transformations to images. In recent versions of Matlab there are functions that can do some of this for you. But since we want you to write your own code, you are not allowed to use them. Specifically, the functions `cp2tform`, `imtransform`, `tformarray`, `tformfwd`, `tforminv`, `maketform`, and `findbounds` are to be avoided.

In what follows, the notation is such that $\mathbf{x}$ and $\tilde{\mathbf{x}}$ indicate homogeneous and inhomogeneous vectors, respectively.

1. *(20 points)* A similarity transformation is a composition of a rotation, translation, and scaling. In the special case of no rotation, a similarity can be written

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{1}$$

   where, as usual, '=' indicates equality up to scale. In matrix notation, this is written $\mathbf{x}' = \mathbf{T}\mathbf{x}$.

   (a) Suppose you are given a set of N inhomogeneous points $\tilde{\mathbf{x}}_i = (x_i,\ y_i)^\top$, $i = 1 \ldots N$. Write expressions for s, t_x and t_y in Eq. 1 such that the set of points $\{\tilde{\mathbf{x}}_i\}$ is mapped to the set of points $\{\tilde{\mathbf{x}}'_i\}$ with the following properties.
      i. The centroid of the points $\{\tilde{\mathbf{x}}'_i\}$ is the origin, $(0,\ 0)^\top$.
      ii. The average distance from the origin to a point $\tilde{\mathbf{x}}'_i$ is $\sqrt{2}$.

   (b) Write a Matlab function `T=getT(X1)` that takes an $N \times 2$ arrays of points (where each row is (x_i, y_i)) and returns the 3×3 similarity matrix $\mathbf{T}$ defined above.

   (c) Test your function by plugging your transformation $\mathbf{T}$ into the following script.

```
X=rand(50,2)*100;  % 50 random points in square [0,100]x[0,100]
Xn=(T*[X';ones(1,50)])';
Xni=Xn(:,1:2)./repmat(Xn(:,3),[1 2]);

% display results
figure;
subplot(1,2,1); plot(X(:,1),X(:,2),'.'); axis equal; axis tight;
subplot(1,2,2); plot(Xni(:,1),Xni(:,2),'.'); axis equal; axis tight;
```

   Submit the resulting plot, and write a brief description of the operations being performed in lines two and three of this script.

2. *(20 points)* A homography relating two images can be estimated from four or more pairs of corresponding points. When these four points correspond to known metric points on a plane, such a homography can be used to remove projective distortion via metric rectification.

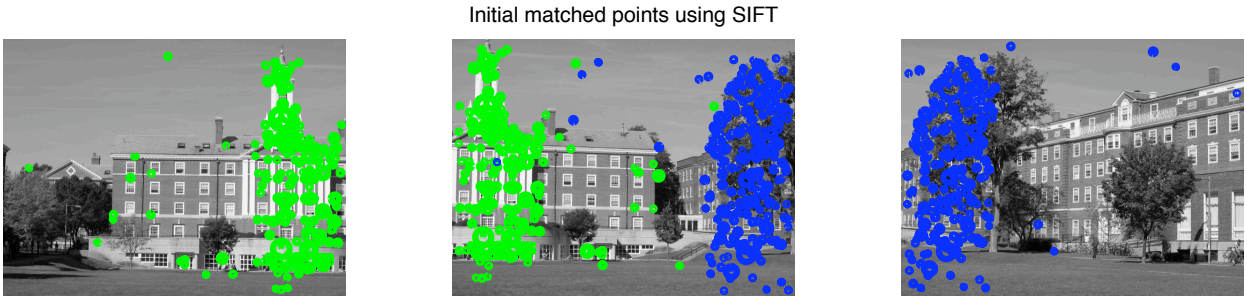
   (a) Write a function `H=getH(X1,X2)` that takes two $N \times 2$ arrays of image coordinates (where each row is (x_i, y_i)) and returns the 3×3 homography matrix that maps points in the first image (`X1`) to points in the second (`X2`). Use your function `getT()` from Problem 1 to implement normalization as discussed in Sect. 4.4.4 (and Algorithm 4.2) of Hartley & Zisserman.

(b) Write a function `Iout=applyH(Iin,H)` that computes a new image by applying the homography `H` to an $h \times w$ image `Iin`. The resolution of the output image should (on average) be comparable to that of `Iin`, and the horizontal and vertical limits of the output image plane should be large enough to include all mapped pixels from `Iin`. Pixels in `Iout` that do not correspond to any point in `Iin` should be set to zero.

(c) The image `MaxwellDworkin_01.jpg` is available in the `assgn2_files.zip` on the course website. Using your new Matlab functions, compute a metric rectification of the image of the wall. (You may want to convert the image to grayscale first.) Submit your results and your code. Hint: In the rectified image, the corners of one of the bricks will have inhomogeneous coordinates $(0,\ 0)$, $(0,\ h)$, $(w,\ 0)$ and $(w,\ h)$, where $h = 1$ and $w = 2.5$ are the relative height and width of the brick. The `zoom`, `pixval` and `impixel` commands can be used to manually identify pixel coordinates in the image.

3. *(15 points)* Use your `getH()` function and (a modified version of) your `applyH()` function from Prob. 2 to create a planar mosaic from a sequence of three or more overlapping images captured with the same projection center. You may use the images in the file `assgn2_files.zip` (with the center image `quad_middle.jpg` as the reference plane), or you may be creative and capture your own sequence of images. Manually identify a sufficient number of corresponding points between the overlapping images, estimate and apply the homographies, and then blend the overlapping regions using any method you prefer. Submit your results and your code.

4. *(15 points)* You will now create a simple "Auto-stitch" script that accomplishes your matching and stitching automatically! Download the open-source **VLFeat** library from http://www.vlfeat.org and unpack it in a convenient location (see hints below for more details). This library includes an implementation of SIFT for robust matching. Begin with the skeleton code `autostitch.m` on the course website, which will find (noisy, but pretty darn good) correspondences and produce the following figure.



As directed at the bottom of the skeleton code, and following Algorithm 4.4 of Hartley & Zisserman, implement RANSAC to robustly estimate the homographies from these correspondences. (You should be making a call to `getH()` in step (i) of the algorithm.) Then use these homographies to create a mosaic similar to that of the previous question. Submit your results and your code.

Hints and Information

- To create a Matlab function, open a new text file named `<function name>.m` and begin the file with a line of the form

  ```
  function [output1,output2,...]=<function name>(input1,input2,...)
  ```

  For example, your `getH.m` function will begin with a line such as

  ```
  function H = getH(X1,X2)
  ```

  An M-file that does not begin with a `function` line can also be executed from the command line, but it will be interpreted as a script instead of a function, and it will not have local variables or accept input arguments.

- To get the **VLFeat** library to work, you will need to run the `vl_setup` script in the package's 'toolbox' sub-directory. For example, if you unpacked the library in `/home/jdoe/vlfeat`, type the following commands in your Matlab command window before running `autostitch.m`.

  ```
  >> addpath /home/jdoe/vlfeat/toolbox
  >> vl_setup
  ```

  For your submission, you should assume that your code is being executed on a machine that has **VLFeat** similary installed somewhere in Matlab's path. Do *n*ot include the library in your submission directory, and do not include the above setup commands in your script.

- If you choose to capture your own panoramic images, here are a few suggestions. First, since your rotation is only roughly about the center of projection, you will obtain better results for distant scenes. Second, use a relatively long focal length (i.e., zoom in) to reduce the radial distortion (straight lines in the world should be straight in your images.) Finally, include enough overlap of significant landmarks to enable manual identification of corresponding points.

- To apply homography $\mathbf{H}$ to image I, you actually need to operate in reverse. Typically, you: 1) apply $\mathbf{H}$ to the corners of I to determine the horizontal and vertical limits of the output image, 2) generate a regular grid of (x, y) coordinates (with appropriate resolution) containing this output range (see `linspace` and `meshgrid`), 3) allocate an array of zeros with the same dimensions as your grid in which you will store the output intensities, 4) apply $\mathbf{H}^{-1}$ to your grid of coordinates to find the corresponding locations in I, and 5) sample I at these (generally non-integer) points, using `interp2`.

  As always, loops should be avoided. All of this can be done in parallel. In general, the Matlab commands `reshape`, `permute`, `repmat`, `ind2sub` and `sub2ind` are very useful.

  Here is some skeleton code that demonstrates some of these steps:

  ```
  % create regularly-space grid of (x,y)-pixel coordinates
  [x,y]=meshgrid(linspace(xmin,xmax,num_x_pts), linspace(ymin,ymax,num_y_pts));

  % reshape them so that a homography can be applied to all points in parallel
  X=[x(:) y(:)];

  % [Apply a homography to homogeneous coordinates corresponding to 'X'. ] %
  % [Compute inhomogeneous coordinates of mapped points.                 ] %
  % [Save result in Nx2 matrix named 'Xh'.                               ] %

  % interpolate I to get intensity values at image points 'Xh'
  Ih=interp2(I,Xh(:,1),Xh(:,2),'linear');
  ```

```
% reshape intensity vector into image with correct height and width
Ih=reshape(Ih,[num_y_pts,num_x_pts]);

% Points in 'Xh' that are outside the boundaries of the image are assigned
%  value 'NaN', which means 'not a number'.  The final step is to
%  set the intensities at these points to zero.
Ih(find(isnan(Ih)))=0;
```