Assignment 2

CS283, Computer Vision
Harvard University

Due Friday, Sep. 22, at 5:00pm

This assignment deals with projective transformations. It is *substantially* longer than Assignment One, and we strongly recommend that you complete Questions 1 & 2 before Tuesday. As usual, there is a helpful "Hints and Information" section at the end of this document.

As always, your submission must follow the Submission Guidelines and take the form of a single live script (`.mlx`) with accompanying `data` and `src` folders. (The `src` folder will be empty this week.) A skeleton script is included in the same archive as this PDF.

In Questions 3 and 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 points $\{\tilde{\mathbf{x}}'_i\}$ 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 using the following snippet of code (also available in the accompanying skeleton script). In addition, write a brief description of the operations being performed in lines three and four.

```
X=rand(50,2)*100;  % 50 random points in square [0,100]x[0,100]
T = getT(X);
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;
```

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 corresponding 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 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 (black).

(c) The image `MaxwellDworkin_01.jpg` is available in the `data` folder. Using your new Matlab functions `getT`, `getH`, and `applyH`, along with some manually-identified points in that image, compute and display a metric rectification of the image. Use the fact that the aspect ratio of the grey bricks, when measured on the building with a tape measure, is 2.5 : 1. *Hints*: In the rectified image, the corners of one of the grey 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` and `impixelinfo` commands can be used to manually identify pixel coordinates in the image. To begin, it is easier to start with a grayscale version of the image; once that is working you may choose to adapt it to color. See the skeleton script.

3. *(15 points)* Homographies can also be used to create a panoramic image by blending together multiple photographs captured with the same center of projection.

(a) Write a function `Iout = applyH2(I1,I2,I3,H12,H32)` that computes a panoramic image by: (1) applying the homography `H12` to image `I1`, (2) applying homography `H32` to image `I3`, and (3) blending these together with image `I2`. The resolution of the output image should (on average) be comparable to that of `I2`, and the horizontal and vertical limits of the output image plane should be large enough to include all mapped pixels from `I1` and `I3`. Pixels in `Iout` that do not correspond to any point in the pre-image of `Iout` should be set to zero (black). You can blend the overlapping regions of the mapped images using any simple method you prefer (e.g., choosing for each pixel in the overlap region the color from one of the overlapping images, or averaging the colors from both of the overlapping images).

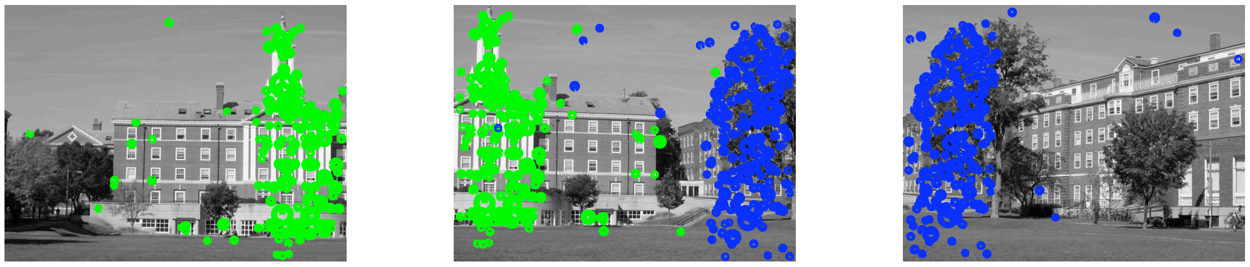
(b) Test your new `applyH2` function by creating a panorama from the three `quad_x.jpg` images in the `data` folder. For this, you should use the center image `quad_middle.jpg` as image `I2`, manually identify a sufficient number of corresponding points between overlapping images, and use calls to your function `getH` to estimate the homographies `H12` and `H32` from these correspondences. Your code for these steps must be executable in your live script. (See the skeleton script.)

4. *(15 points)* Your next task is to build on your functions above to handle noisy correspondences between pairs of images, such as those produced by completely automated systems that detect "interest points" in an image and then match these points between images. The correspondences that are produced by these systems contain outliers, which you must identify and eliminate using RANSAC.

(a) Write a function `H=ransacH(X1,X2)` that takes N correspondences, in the form of two $N \times 2$ arrays of corresponding image coordinates, and returns a 3×3 homography matrix. Each pair of corresponding rows in the input (`X1(i,:)`, `X2(i,:)`) represents a correspondence, and some of these correspondences are outliers. Your function must use RANSAC to identify the subset of inlying correspondences, and return a 3×3 homography matrix that maps inlying points of `X1` to the corresponding points in `X2`. You should follow Algorithm 4.4 of Hartley & Zisserman, using a call to your function `getH` in step (i).

(b) Test your `ransacH` function by creating another panorama from the three `quad_x.jpg` images in the `data` folder, this time using the noisy correspondences stored in data file `quad_points.mat` and shown in the figure below. Similar to Question 3, you should use the center image `quad_middle.jpg` as image `I2`, but this time use calls to `ransacH` for estimating the homographies `H12` and `H32` from the noisy correspondences in `quad_points.mat`. Your code for these steps must be executable in your live script. (See the skeleton script.)

Initial matched points using SIFT



Hints and Information

- In Matlab, functions begin 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)
  ```

  Function definitions can be stored in the their own m-files and then called from the command line. Alternatively, as you will do in his assignment, they can be included inside of a live script, so they are only available locally within the script.

- To apply homography **H** to image I, you actually need to operate in reverse. Typically, you: 1) apply **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-spaced 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;
  ```

- The noisy correspondences for Question 4 were computed using the open-source **VLFeat** Matlab toolbox from http://www.vlfeat.org. This toolbox includes an implementation of the "scale-invariant feature transform" (SIFT) for the robust detection and matching of interest points. In case you are curious, here is the Matlab code that produced the correspondences in `quad_points.mat` and created the image above. (*You do not need this code for the assignment, it is only here for your information.*)

```
% check that VLfeat is installed and on the Matlab path
if ~exist('vl_sift')
  error('VLfeat package not found. See: http://www.vlfeat.org/install-matlab.html.')
end

% read images, convert to grayscale with single precision
im1=single(rgb2gray(imread(fullfile('..','data','quad_left.jpg'))));
im2=single(rgb2gray(imread(fullfile('..','data','quad_middle.jpg'))));
im3=single(rgb2gray(imread(fullfile('..','data','quad_right.jpg'))));

% detect interest regions and compute descriptors
[X1,D1]=vl_sift(im1);
[X2,D2]=vl_sift(im2);
[X3,D3]=vl_sift(im3);

% match descriptors (in both directions) between each pair of overlapping images
[M12,S12]=vl_ubcmatch(D1,D2);
[M21,S21]=vl_ubcmatch(D2,D1);

[M23,S23]=vl_ubcmatch(D2,D3);
[M32,S32]=vl_ubcmatch(D3,D2);

% Eliminate matches that are not symmetric.
% Store results in 2xK arrays: M12_sym, M23_sym
M12_sym=[];
for i=1:size(M12,2)
  M21_i=find(M21(2,:)==M12(1,i));
  if ~isempty(M21_i)
    if M21(1,M21_i)==M12(2,i)
      M12_sym=[M12_sym, M12(:,i)];
    end
  end
end

M23_sym=[];
for i=1:size(M23,2)
  M32_i=find(M32(2,:)==M23(1,i));
  if ~isempty(M32_i)
    if M32(1,M32_i)==M23(2,i)
      M23_sym=[M23_sym, M23(:,i)];
    end
  end
end

% strip away the unmatched points
X12=X1(:,M12_sym(1,:))';  % points in im1 that match im2
X21=X2(:,M12_sym(2,:))';  % points in im2 that match im1
X23=X2(:,M23_sym(1,:))';  % points in im2 that match im3
```

```
X32=X3(:,M23_sym(2,:))';  % points in im3 that match im2

% for assgn 2, we only need the point locations
X12=X12(:,1:2);
X21=X21(:,1:2);
X23=X23(:,1:2);
X32=X32(:,1:2);
save quad_points X12 X21 X23 X32

% display correspondences
figure;

subplot(1,3,1); imshow(im1,[]); hold on;
h1=vl_plotframe(X12');

subplot(1,3,2); imshow(im2,[]); hold on;
h2a=vl_plotframe(X21');
h2b=vl_plotframe(X23'); set(h2b,'color','b');

subplot(1,3,3); imshow(im3,[]); hold on;
h3=vl_plotframe(X32'); set(h3,'color','b');
```