16-720 Computer Vision: Homework 3
Homographies & RANSAC

Instructor: Srinivasa Narasimhan
TAs: Arun Venkatraman, Aayush Bansal, Yiying Li

Due: Tuesday, March 10th, 2015 11:59 p.m.

If you have any questions, please post to the discussion board on blackboard or contact the TAs - Arun (`arunvenk@cs.cmu.edu`), Aayush (`aayushb@cs.cmu.edu`), and Yiying (`yiying@cmu.edu`). **Start early! Start early!**

Note: In order to complete the later implementation questions, **Q1.4** must first be solved.

Introduction

In this homework, we will be exploring the world of homographies, specifically planar homographies. Why is this useful? In many robotic applications, robots often must deal tabletops, walls, and the ground among other flat planar surfaces. When two cameras observe a plane, there exists a relationship between the images captured. This relationship is defined by a 3×3 transformation matrix, called a planar homography. Interestingly, as we shall see, we can also derive a homography for views separated by a pure rotation.

A planar homography allows us to compute how a planar scene would look from second camera location, given only the first image. In fact, we can compute how images of the plane will look from any camera at any location without knowing any internal camera parameters and without actually taking the pictures, all with the planar homography.

1 Planar Homographies: Theory (35 pts)

Suppose we have two cameras $\mathbf{C}_1$ and $\mathbf{C}_2$ looking at a common plane Π in 3D space. Any 3D point $\mathbf{P}$ on Π generates a projected 2D point located at $\mathbf{p} \equiv (u_1, v_1, 1)^T$ on the first camera $\mathbf{C}_1$ and $\mathbf{q} \equiv (u_2, v_2, 1)^T$ on the second camera $\mathbf{C}_2$. Since $\mathbf{P}$ is confined to the plane Π, we expect that there is a relationship between $\mathbf{p}$ and $\mathbf{q}$. In particular, there exists a common 3×3 matrix $\mathbf{H}$, so that for any $\mathbf{p}$ and $\mathbf{q}$, the following conditions holds:

$$\mathbf{p} \equiv \mathbf{Hq} \tag{1}$$

We call this relationship *planar homography*. Recall that both $\mathbf{p}$ and $\mathbf{q}$ are in homogeneous coordinates and the equality $\equiv$ means $\mathbf{p}$ is proportional to $\mathbf{Hq}$ (recall homogeneous coordinates). It turns out this relationship is also true for cameras that are related by pure rotation without the planar constraint.

- **Q1.1 (10 pts):** Prove that there exists an $\mathbf{H}$ that satisfies Equation 1 given two 3×4 camera projection matrices $\mathbf{M_1}$ and $\mathbf{M_2}$ corresponding to cameras $\mathbf{C_1}$, $\mathbf{C_2}$ and a plane Π. Do not produce an actual algebraic expression for $\mathbf{H}$. All we are asking for is a proof of the existence of $\mathbf{H}$.

Note: A degenerate case may happen when the plane Π contains both cameras' centers, in which case there are infinite choices of $\mathbf{H}$ satisfying Equation 1. You can ignore this case in your answer.

- **Q1.2 (10 pts):** Prove that there exists an $\mathbf{H}$ that satisfies Equation 1 given two cameras separated by a pure rotation. That is, for $\mathbf{C_1}$, $\mathbf{p} = \mathbf{K_1}\begin{bmatrix}\mathbf{I} & 0\end{bmatrix}\mathbf{P}$ and for $\mathbf{C_2}$, $\mathbf{p'} = \mathbf{K_2}\begin{bmatrix}\mathbf{R} & 0\end{bmatrix}\mathbf{P}$. Note that $\mathbf{K}$ is not the same for the two cameras.

- **Q1.3 (5 pts):** Why is the planar homography not completely sufficient to map any arbitrary scene image to another viewpoint? State your answer concisely in one or two sentences.

- **Q1.4 (10 pts):** We have a set of points $\mathbf{p} = \{p^1, p^2, \ldots, p^N\}$ in an image taken by camera $\mathbf{C}_1$ and corresponding points $\mathbf{q} = \{q^1, q^2, \ldots, q^N\}$ in an image taken by $\mathbf{C}_2$. Suppose we know there exists an unknown homography $\mathbf{H}$ between corresponding points for all $i \in \{1, 2, \ldots, N\}$. This formally means that $\exists\ \mathbf{H}$ such that:

$$p^i \equiv \mathbf{H}q^i \tag{2}$$

  where $p^i = (x_i, y_i, 1)$ and $q^i = (u_i, v_i, 1)$ are homogeneous coordinates of image points each from an image taken with $\mathbf{C_1}$ and $\mathbf{C_2}$ respectively.

  (a) Given N correspondences in $\mathbf{p}$ and $\mathbf{q}$ and using Equation 2, derive a set of $2N$ independent linear equations in the form:

$$\mathbf{Ah} = \mathbf{0} \tag{3}$$

  where $\mathbf{h}$ is a vector of the elements of $\mathbf{H}$ and $\mathbf{A}$ is a matrix composed of elements derived from the point coordinates. Write out an expression for $\mathbf{A}$.
  Hint: Start by writing out Equation 2 in terms of the elements of $\mathbf{H}$ and the homogeneous coordinates for p^i and q^i.

  (b) How many elements are there in $\mathbf{h}$?

  (c) Given N point pairs (correspondences), what are the dimensions of $\mathbf{A}$?

  (d) How many point pairs (correspondences) are required to solve this system?
  Hint: How many degrees of freedom are in $\mathbf{H}$? How much information does each point correspondence give?

  (e) Show how to estimate the elements in $\mathbf{h}$ to find a solution to minimize this homogeneous linear least squares system. Step us through this procedure.
  Hint: Use the Rayleigh quotient theorem (homogeneous least squares).

2 Planar Homographies: Implementation (20 pts)

Now that we have derived how to find $\mathbf{H}$ mathematically in **Q1.4**, we can implement it!

- **Q2.1 (10pts)** Implement the function (stub provided in `matlab/computeH.m`)

```
H2to1=computeH(p1,p2)
```

Inputs: `p1` and `p2` should be $2 \times N$ matrices of corresponding $(x, y)^T$ coordinates between two images.
Outputs: `H2to1` should be a 3×3 matrix encoding the homography that best matches the linear equation derived above for Equation 2 (in the least squares sense). *Hint:* Remember that a homography is only determined up to scale. The Matlab functions `eig()` or `svd()` will be useful. Note that this function can be written without an explicit for-loop over the data points.

- **Q2.2 (10pts)** Normalization improves numerical stability of the solution, as we shall see in the next section. For now, implement the function (stub provided in `matlab/computeH_norm.m`)

```
H2to1=computeH_norm(p1, p2)
```

This version should normalize the coordinates `p1` and `p2` prior to calling `compute_H`. Normalization is a two step process on *each* of the two sets of coordinates.

1. Translate the centroid of the points to the origin.
2. Scale the points so that the average distance from the origin is $\sqrt{2}$.

Note that the resulting `H` should still follow Equation 2.
Hint: Express the normalization as a matrix. Remember that the homography you get is in the normalized coordinate space.

3 Sensitivity to Normalization (15 pts)

In this section we will see why coordinate normalization is something you should ***always*** do when performing operations on point data. Mathematically, coordinate normalization does not change anything. In reality, the computer you use only approximates real numbers with finite precision. Further, the data points you input to your algorithm are never perfect and they contain noise. A well behaved algorithm should not become unstable when small amounts of noise are added to the input.

For the following exercise, we will use these data points:

$$\mathbf{p} = 100 * \begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ 10 & 2 & 1 & 2 & 10 \end{bmatrix}$$

$$\mathbf{p_{test}} = 100 * \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

The homography we want to compute is simply the identity. If you run `computeH(p,p)` on the pristine data (no noise), you will get the identity matrix, which is the result you expect.

- **Q3.1 (15pts)** Write a script `normCompare.m` which performs the following.

  (a) Corrupt one of the point sets that are input to your algorithm with zero-mean Gaussian noise with variance 1 (use Matlab's `randn()`). That is, add independent random noise to *each* of the elements in $\mathbf{p}$ to form

$$\mathbf{p}_{corrupt} = \mathbf{p} + \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

Compute both `computeH(p,p_corrupt)` and `computeH_norm(p,p_corrupt)`. Apply each solution to the test point. Record that resulting point and repeat 1000 times. Don't forget to normalize the result of applying the matrix since it is computed using homogeneous coordinates. You do not need to save these results – we will rerun the script.

(b) Plot the resulting point sets in a single plot with the normalized results (from `computeH_norm()`) as blue dots and the un-normalized results (from `computeH()`) as red dots. Add a legend for the plot as well. Save the figure as `results/q3_1.png` and submit it in your writeup.
Hint: See Matlab `plot`, `hold on/off`, `legend`, and `axis equal`. Figures can be saved by calling the functions `print` or `saveas`. From the GUI, you can use the File→Save As function in the figure menu. In general, we recommend exporting plots in Matlab as `pdf` or `png`. Submit the figure to us in your writeup.

(c) Compute the covariance of the transformed test point from using both the normalized and un-normalized solutions. (see Matlab `cov()`) The largest *Eigen value* of the covariance matrix is the variance σ^2 along the direction (i.e. *Eigen vector*) with the most variance (see Matlab `eig`)
What is the ratio of the *standard deviations*, σ, between the un-normalized and normalized points? What does this tell you about which algorithm is better?

(d) Run this procedure a few times and observe how things change. Given that for each run we use 1000 points, what can you say *qualitatively* about the stability of the normalized and un-normalized result? (We are not looking for anything deep. One or two sentences are sufficient.)

4 Stitching it together: Panoramas (30 pts)

We can also use homographies to create a panorama image from multiple views of the same scene. This is possible for example when there is no camera translation between the views (e.g., only rotation about the camera center), as we saw in **Q1.2**.

First, you will generate panoramas using manually marked (error free) point correspondences between images. In the next section you will extend the technique to use (potentially noisy) keypoint matches.

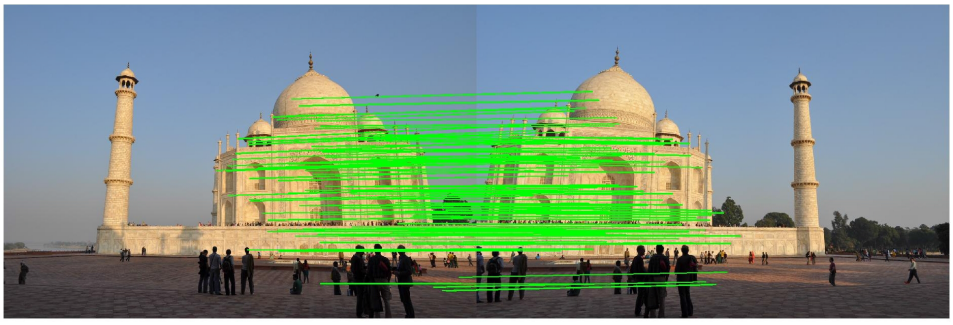
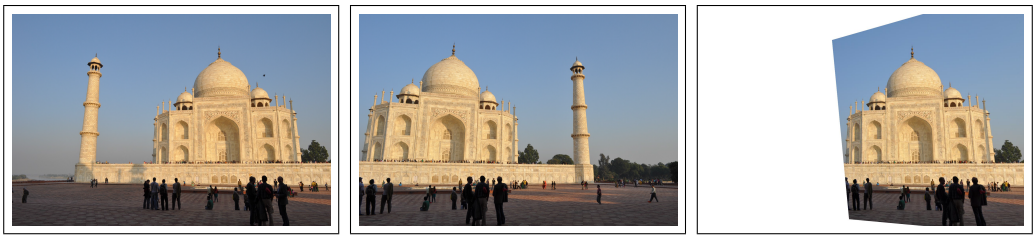


Figure 1: Matched features from `tajPts.mat`.



(a) `taj1r.jpg` (b) `taj2r.jpg` (c) `taj2` warped to `taj1`'s frame

Figure 2: Example output for **Q4.1**: Original images `img1` and `img2` (left and center) and `img2` warped to fit `img1` (right). Notice that the warped image clips out of the image. We will fix this in **Q2.4**

You will need to use the provided function `warp_im=warpH(im, H, out_size)`, which warps image `im` using the homography transform `H`. The pixels in `warp_im` are sampled at coordinates in the rectangle $(1, 1)$ to $(\texttt{out\_size(2)}, \texttt{out\_size(1)})$. The coordinates of the pixels in the source image are taken to be $(1, 1)$ to $(\texttt{size(im,2)}, \texttt{size(im,1)})$ and transformed according to `H`.

- **Q4.1 (15pts)** In this problem you will implement and use the function (stub provided in `matlab/pano4_1.m`):

  ```
  [H2to1] = pano4_1(img1, img2, pts)
  ```

  on two images of the Taj Mahal. This function accepts two images and a $4 \times N$ keypoints matrix where the top two rows correspond to coordinates in image 1 and the bottom two rows are the respective points in image 2. This function will:

  (a) Compute the homography **H** between the two images using the feature points (each column of `pts` is a point-pair correspondence),

  (b) Warps `img2` into `img1`'s reference frame using the provided `warp_H()` function

  (c) In one figure: Displays the warped version of `img2` (see Figure 2 for reference output)

  (d) In another figure: Overlays the other image on top, displaying a panorama with the images together

  For this problem, use the provided images `taj1r.jpg` and `taj2r.jpg` along with the data in `tajPts.mat` – all three are provided in the `data/` folder. Use Matlab's `imread()` function to load an image and the `load()` function to load a `mat` file. `tajPts` is a 4x1048 matrix containing the coordinates of features points in `taj1` in the first two rows and matching feature points for `taj2` in the bottom two rows. You can visualize a sample of these given correspondences using `plotMatches(img1,img2,pts`

  Apply your `computeH_norm()` to these correspondences to compute `H2to1`, which is the homography from `taj2` onto `taj1`. Apply this homography to `taj2` using `warpH()`. **Note:** Since the warped image will be translated to the right, you will need a larger target image. Use `outSize = [size(taj1,1),3000]`;

Figure 3: Final mosaic view. Border lines are only for reference and need not be present in your solution.

Visualize the warped image and save this figure as `results/q4_1.jpg` using Matlab's `imwrite()` function and save *only* `H2to1` as `results/q4_1.mat` using Matlab's `save()` function. Produce a simple panorama by laying the warped `taj2` on top of `taj1` (see Matlab's `imfuse()` function). Save this figure as `results/q4_1_pan.jpg`.

- **Q4.2 (15pts)** Notice how the output from Q4.1 is clipped at the edges? We will fix this in this question. Implement a function (stub provided in `matlab/pano4_2.m`)

```
[H2To1] = pano4_2(img1, img2, pts)
```

that takes in the same input types and produces the same outputs as in Q4.1.

To prevent this clipping at the edges, we instead need to warp *both* image 1 and image 2 into a common third reference frame in which we can display both images without having any clipping. Specifically we want to find a matrix M that *only* does scaling and translation such that:

```
warp_im1 = warpH(im1, M, out_size);
warp_im2 = warpH(im2, M*H2to1, out_size);
```

This produces warped images in a common reference frame where all points in image 1 and image 2 are visible. To do this, we will only take as input either the width or height of `out_size` and will have to compute the other one based on the given. For now, assume we only take as input the width of the image in `out_size(2)` and must compute the correct `out_size(1)`.

Hint: The computation will be done in terms of `H2to1` and the extreme points (corners) of the two images (see Matlab's `size()`). Make sure M is only scaling (find the aspect ratio of the full-sized panorama image) and translation only

Again, pass `taj1` as `img1` and `taj2` as `img2`. **Save** the resulting panorama in `results/q4_2_pan.jpg`. Use `out_size(2) = 1280`. (see Figure 3 for example output from Q4.2)

5 RANSAC (25 pts)

The least squares method you implemented for computing homographies is not robust to outliers. If all the correspondences are good matches (like in the previous section), this is not a problem. But even a single false correspondence can completely throw off the homography estimation. When correspondences are determined automatically (using BRIEF feature matching for instance), some mismatches in a set of point correspondences are almost certain. RANSAC can be used to fit models robustly in the presence of outliers.

Q5.1 (15pts): Write a function that uses RANSAC to compute homographies automatically between two images (stub provided in `matlab/ransacH.m`):

```
[bestH, bestError, inliers] = ransacH(matches, locs1, locs2, nIter, tol)
```

The input and outputs of this function should be as follows:

- Inputs: `locs1` and `locs2` are matrices specifying point locations in each of the images and `matches` is a matrix specifying matches between these two sets of point locations. These matrices are formatted identically to the output of the `briefMatch` function you implemented in the last assignment.
- Algorithm Input Parameters: `nIter` is the number of iterations to run RANSAC for, `tol` is the tolerance value for considering a point to be an inlier. Define your function so that these two parameters have reasonable default values.
- Outputs: `bestH` should be the homography model with the most inliers found during RANSAC. Its error `bestError` is a scalar and is also returned. `inliers` is a vector containing a 1 at matches which were marked as inliers and 0 at the other positions.

Q5.2 (10pts): You now have all the tools you need to automatically generate panoramas. Write a function that accepts two images as input, computes keypoints and descriptors for both the images, finds putative feature correspondences by matching keypoint descriptors, estimates a homography using RANSAC and then warps one of the images with the homography so that they are aligned and then overlays them. For the keypoint computation and matching you can try reuse the BRIEF code you implemented in the last assignment, or you can download and use a `MATLAB` implementation of any other popular interest point detector and matcher (SIFT or SURF for example). (stub provided in `matlab/generatePanorama.m`)

```
im3 = generatePanorama(im1, im2)
```

Run your code on the image pair `data/taj1h.jpg`, `data/taj2h.jpg` (to keep running time down, these images are scaled down versions of the ones your used in the previous section). Save the resulting panorama as `results/q5_2.jpg`. Include the figure in your writeup.

6 Extra Credit (Upto 10pts)

Take pictures with your own camera, separated by a pure rotation, and automatically construct a panorama *with 3 or more* images. Be sure that objects in the images are far enough away so that there are no parallax effects. Submit the original images as `results/ec1.jpg`, `results/ec2.jpg` etc., and a script `matlab/ec.m` that loads the images and assembles a panorama. Save the panorama as `ec_pan.jpg`. We may even showcase some of the best panoramas in class! :)

7 What to Submit

Your submission should consist of only 1 file, a zip file named `<AndrewId>.zip`. This zip file should contain

- a folder `matlab` containing all the `.m` and `.mat` files you were asked to write and generate
- a `pdf` named `writeup.pdf` containing the results, explanations and images asked for in the assignment along with to the answers to the questions on homographies.
- If you reused your old BRIEF implementaion for feature matching, make sure the `.m` files BRIEF needs to run are in your `matlab` folder.

Submit all the code needed to make your panorama generator run. Make sure all the `.m` files that need to run are accessable from the `matlab` folder without any editing of the path variable. If you downloaded and used a feature detector, include the code with your submission and mention it in your writeup. You may leave the `data` folder in your submission, but it is not needed. Please zip your homework as usual and submit it using blackboard.

Appendix: Image Blending

Note: This section is not for credit and is for informational purposes only.

For overlapping pixels, it is common to blend the values of both images. You can simply average the values but that will leave a seam at the edges of the overlapping images. Alternatively, you can obtain a blending value for each image that fades one image into the other. To do this, first create a mask like this for each image you wish to blend:

```
mask = zeros(size(im,1), size(im,2));
mask(1,:) = 1; mask(end,:) = 1; mask(:,1) = 1; mask(:,end) = 1;
mask = bwdist(mask, 'city');
mask = mask/max(mask(:));
```

The function `bwdist` computes the distance transform of the binarized input image, so this mask will be zero at the borders and 1 at the center of the image. You can warp this mask just as you warped your images. How would you use the mask weights to compute a linear combination of the pixels in the overlap region? Your function should behave well where one or both of the blending constants are zero.