16720: Computer Vision Homework 1

Instructor: Martial Hebert
TAs: Xinlei Chen, Arne Suppé, Feng Zhou
Due: September 18, 2013

1 Instructions

A complete homework submission consists of two parts. A PDF file with answers to the theory questions and the Matlab files for the implementation portions of the homework. You should submit your completed homework by placing all your files in a directory named after your Andrew ID and upload it to your submission directory on Blackboard. The submission location is in the same section where you downloaded your assignment. Do not include the handout materials in your submission.

All questions marked with a **Q** require a submission. For the implementation part of the homework please stick to the function headers described. Your Matlab code should run without us having to adjust paths to any file[1]. Please comment your code to help us understand your thought process and include longer discussions in your report along with any helpful figures.

The problems in this assignment need not be solved sequentially. Don't get stalled on a question when there are other questions to answer. You should read and understand section 3 first, but the proofs are not required to solve later problems. **Q3.X** are theory questions which lend insight to the homography estimation implementation in **Q4.1** and **Q4.2**. Do not underestimate the time required to implement some of these concepts in Matlab. This implementation is necessary for solving the rest of the problems in the assignment. **Q5.1** is is not required for any other part of the homework.

Final reminder: **Start early! Start early! Start early!** Matlab is a powerful and flexible language. It means you can do things swiftly in a few lines of Matlab code; it also means the program may go wrong without reporting any errors. Even for advanced Matlab users, it is often the case that a whole day can be spent debugging.

Questions about this assignment should be directed to: Arne Suppé , **suppe@cs.cmu.edu**

2 Introduction

In this homework, we will be exploring the usefulness of homographies. Robots often deal with planes, whether in the form of walls, ground, or some other flat surface. When two

[1]We will run your scripts/functions with the script's directory as Matlab's current working directory.

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.

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 homogenous coordinates and the equality $\equiv$ means $\mathbf{p}$ is proportional to $\mathbf{Hq}$. It turns out this relationship is also true for cameras that are related by pure rotation without the planar constraint.

3 Planar Homographies: Theory (35pts)

Q3.1 (10pts) 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.

Q3.2 (10pts) 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.

Q3.3 (5pts) 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.

Q3.4 (10pts) We have a set of points $\mathbf{p_1^i}$ in an image taken by camera $\mathbf{C}_1$ and points $\mathbf{p_2^i}$ in an image taken by $\mathbf{C}_2$. Suppose we know there exists an unknown homography $\mathbf{H}$ such that

$$\mathbf{p_1^i} \equiv \mathbf{H}\mathbf{p_2^i} \tag{2}$$

Assume the points are homogenous coordinates in the form $\mathbf{p_j^i} = (x_j^i, y_j^i, 1)^T$. For a single point pair, write a matrix equation of the form

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{3}$$

Where $\mathbf{h}$ is a vector of the elements of $\mathbf{H}$ and $\mathbf{A}$ is a matrix composed of the point coordinates.

How many elements are there in $\mathbf{h}$? How many point pairs are required to solve this system? Show how to estimate the elements in $\mathbf{h}$ using the Rayleigh Quotient Theorem mentioned in class to find a solution to minimize this homogenous linear least squares system.

4 Planar Homographies: Implementation (20pts)

Q4.1 (10pts) Implement a function `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 $\times$ 3 matrix encoding the homography that best matches the linear equation derived above in equation 2 (in the least squares sense). Hint: Remember that a homography is only determined up to scale. The Matlab function `eig` will be useful.

Q4.2 (10pts) Implement a function `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.

(a) Translate the centroid of the points to the origin.

(b) Scale the points so that the average distance from the origin is $\sqrt{2}$.

Normalization improves numerical stability of the solution, as we shall see next. Note that the resulting `H` should still follow equation 2. Hint: Express the normalization as a matrix.

5 Sensitivity to Normalization (15pts)

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:

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

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

The homography we want to compute is simply the identity. If you run `computeH(p,p)` on the pristine data, you will get the identity, which is the result you expect.

Q5.1 (15pts) Write a script `normCompare` which performs the following.

- Corrupt one of the point sets that are input to your algorithm with Gaussian noise $\sigma = 1$. (Use Matlab `randn`.) That is, add independent random noise to each of the numbers in p to form *p_corrupt*. 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. You do not need to send us the data, just give us the script.
- Plot the resulting point sets in a single plot with the normalized results as blue dots and the un-normalized results as red dots. (See Matlab `plot`, `hold on/off` and `axis equal`. Figures can be saved using `print` or `saveas`, or using the File→Save As function in the figure menu.) Submit the figure to us in your writeup.
- Compute the covariance of the transformed test point for both the normalized and un-normalized solution. (See Matlab `cov`.) The largest Eigen value of the covariance matrix is the variance along the direction with the most variance. (See Matlab `eig`) What is the ratio of the *standard deviation* between the un-normalized and normalized points? What does this tell you about which algorithm is better?
- 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.)

6 Panoramas (30 pts)

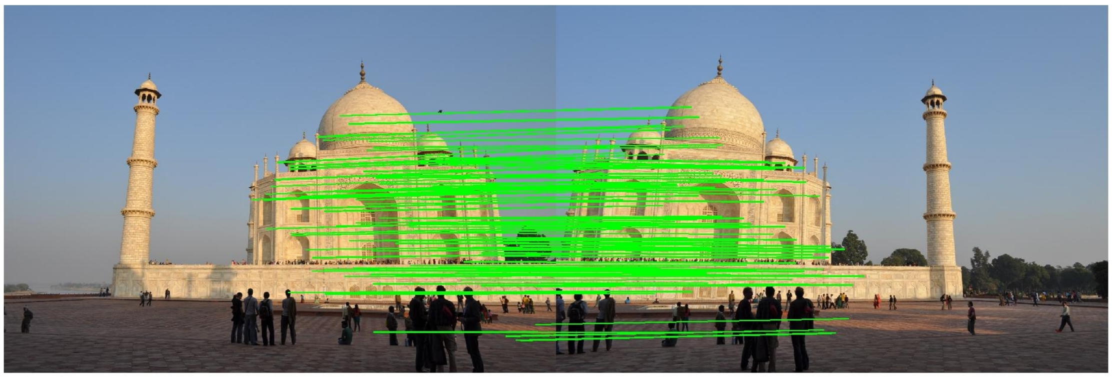
We can also use homographies to create a panorama image from multiple views of the same scene. This is possible for example when there is no camera translation between the views (e.g., only rotation about the camera center), as we saw in **Q3.2**.

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 (`out_size(2)`,`out_size(1)`). The coordinates of the pixels in the source image are taken to be $(1, 1)$ to (`size(im,2)`,`size(im,1)`) and transformed according to `H`.

- **Q6.1 (15pts)** For this problem, use the provided images *taj1r.jpg* and *taj2r.jpg* along with the data in *tajPts.mat*. 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 *taj*1 in the first two rows and matching feature points for *taj*2 in the bottom two rows.

You can visualize a sample of these correspondences using `plotMatches(img1,img2,pts)` Later, we will see how to automatically generate them.

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



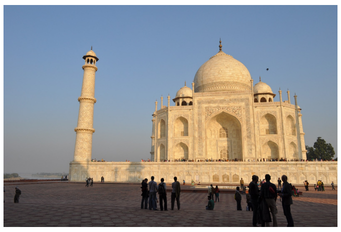
Apply your `computeH_norm` to these correspondences to compute `H2to1`, which is the homography from *taj*2 onto *taj*1. Apply this homography to *taj*2 using `warpH`. Note that since the warped image will be translated to the right, you will need a larger target image. Use `outSize = [size(taj1,1),3000];`

Save this figure as `q6_1.jpg` and save `H2to1` as `q6_1.mat` using Matlab's `save` function. Produce a simple panorama by laying *taj*2 on top of the warped *taj*1. Save this figure as `q6_1_pan.jpg`.

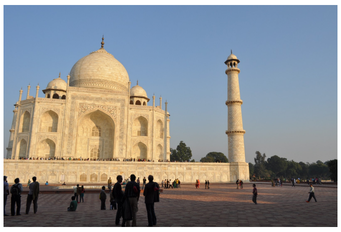
Save your code in a Matlab function `[H2to1] = q6_1(img1, img2, pts)` which accepts the images and feature points and computes the homography and also displays the warped image in one figure and the panorama in another.

Figure 2: Original images and warped.

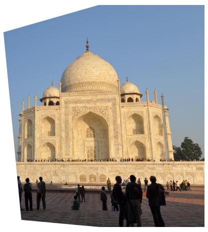
(a) *taj1.jpg*



(b) *taj2.jpg*



(c) *taj*2 warped into *taj*1



- **Q6.2 (15pts)** Suppose we want to display all the visible pixels in the panorama. Write a few lines of Matlab code to find a matrix `M`, and `out_size` in terms of `H2to1` and `size(im1)` and `size(im2)` such that,

```
warp_im1=warpH(im1, M, out_size);
```

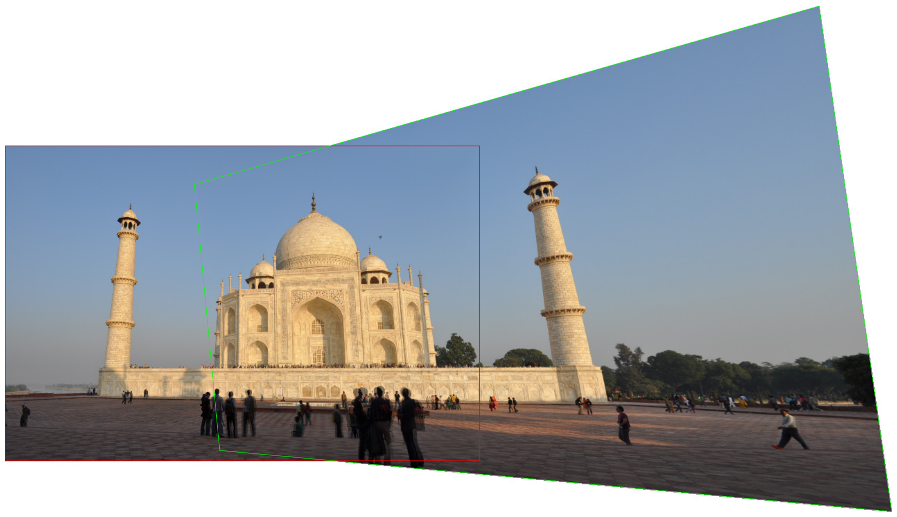
```
warp_im2=warpH(im2, M*H2to1, out_size);
```

produces images in a common reference frame where all points in `im1` and `im2` are visible and `out_size(2)=1280`. **M** should perform scaling and translation only, and the resulting image should fit snugly around the corners of the panorama. Save this image as `q6_2_pan.jpg`. Save the code used to generate and display the panorama as

```
[H2To1] = q6_2(img1, img2, pts)
```

where `img1` is $taj1$ and `img2` is $taj2$.

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



6.1 Image Blending

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:

```
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. What is a simple function 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.

7 Extra Credit (10pts)

Take pictures with your own camera, separated by a pure rotation, and construct a panorama. Be sure that objects in the images are far enough away so that there are no parallax effects. You can use Matlab's `cpselect` to select matching points on each image or some automatic method. Submit the original images as `ec1.jpg`, `ec2.jpg` etc., and a script that loads the images and assembles a panorama. Save the panorama as `ec_pan.jpg`. Save your code as `ec.m`.