16720: Computer Vision Homework 1

Instructor: Martial Hebert
TAs: David Fouhey, Heather Knight and Daniel Maturana

Due: Wednesday, September 19, 2012

A complete homework submission consists of two parts. A pdf file with answers to the questions and the MATLAB files for the implementation portions of the homework.

Please upload your files to Blackboard by the midnight on the day the assignment is due. All questions marked with a **Q** require a submission. For the implementation part of the homework please stick to the function headers described.

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 the same section where you downloaded your assignment. Do not include the handout materials in your submission.

Final reminder: **Start early!** MATLAB is a powerful and flexible language. It means you can do things swiftly by a few lines of code; it also means the program may go wrong without reporting any errors, which can cause a huge amount of time wasted in debugging if you are not careful.

1 Warm up: Homographies (25 points)

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 cameras observe a plane, there exists a relationship between the images captured. This relationship is defined by a 3x3 transformation matrix, called a planar homography.

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.

1.1 Homography in Theory

Suppose we have two cameras $\mathbf{C}_1$ and $\mathbf{C}_2$ looking at a common plane Π in the 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 3x3 matrix $\mathbf{H}$, so that for any $\mathbf{p}$ and $\mathbf{q}$, the

following condition 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 $\equiv$ essentially means $\mathbf{p}$ is proportional to $\mathbf{Hq}$. $\mathbf{H}$ is universal if the locations of the plane Π and two cameras are fixed.

Q1.1 (10pts): Prove that there exists an $\mathbf{H}$ that satisfies (1) given two 3×4 camera projection matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ corresponding to $\mathbf{C}_1$, $\mathbf{C}_2$ and a plane Π. Do not produce an actual algebraic expression for H. All that we are asking is for a proof of existence of 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 (1). You can ignore this case in your answer.

Q1.2 (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.

1.2 Homographies in Matlab

Now it's time to try some homographies in Matlab. Let's start with `curiosity.jpg` and perform some simple image warping using planar homographies. Feel free to use the provided `warpH.m` function.

Figure 1: Unwarped image of Curiosity Rover on Mars (Credit: NASA/JPL).

The warp function should work for any 3×3 homography, but try the following homographies to get familiar with its workings:

$$\mathbf{H}1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{H}2 = \begin{pmatrix} 1 & 0 & 8 \\ 0 & 1 & 8 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{H}3 = \begin{pmatrix} 1 & 1 & 0 \\ 10 & 2 & 0.5 \\ 0 & 3 & 1 \end{pmatrix}$$

$\mathbf{H}1$ will stretch the image in the vertical direction by a factor of two, $\mathbf{H}2$ will shift the image, and $\mathbf{H}3$ will warp and shift the image in an odd way.

Q1.3 (10pts): Save the three warped images as `image1h1.jpg`, `image1h2.jpg`, `image1h3.jpg` and submit them.

QX1 (Extra Credit 10pts): Write a Matlab function that applies the transformation $\mathbf{H}$ to an image so that $\mathbf{p} \equiv \mathbf{Hq}$, where $\mathbf{p} \equiv (u_1, v_1, 1)^T$ is a point in the image expressed in homogenous coordinates.[1] Use the function signature:

```
function warpedImage = warpImage(inputImage, H)
```

Take care to properly generate the warped image without 'holes' in the new image by mapping every point/pixel in the new image to a corresponding value in the old image and not the reverse.

It is important to keep in mind that vectorizing code will speed up repeated pixel operations. Although not required, this function can be implemented without a single for loop (hint `meshgrid`, `inpolygon`, `sub2ind`, `find`, `intersect`). You can use `profile` and `profview` to see what parts of your code are taking the longest.

Again include the three warped images, if there are any differences from **Q1.3**, and include a brief description of your code in your answer sheet.

2 Implementation: Constructing Panoramas (50 points)

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). In this case, corresponding points from two views of the same scene can be related by a homography[2]:

$$\mathbf{p}_1^i \equiv \mathbf{H}\mathbf{p}_2^i, \tag{2}$$

where $\mathbf{p}_1^i$ and $\mathbf{p}_2^i$ denote the homogeneous coordinates (e.g., $\mathbf{p}_1^i \equiv (x, y, 1)^T$) of the 2D projection of the i-th point in images 1 and 2 respectively, and $\mathbf{H}$ is a 3×3 matrix representing the homography.

2.1 Homography estimation

Q2.1 (10 pnts): Now we would like to solve for the homography matrix $\mathbf{H}$, given $\mathbf{N}$ point correspondences. Using (2), derive a set of $2\mathbf{N}$ independent linear equations in the form $\mathbf{Ah} = 0$ with the goal of performing least-squares parameter estimation, where `h` is a 9×1 vector containing the unknown entries of $\mathbf{H}$, i.e. $h = (h11, h12, h13, h21, h22, h23, h31, h32, h33)^T$

What are the dimensions of $\mathbf{A}$? How many correspondences are needed to solve for h? Write out an expression for $\mathbf{A}$. Next, step us through the equations to solve for $\mathbf{H}$. Hint: Use the Rayleigh quotient theorem (homogeneous least squares) stated in class.

[1] Please do not use `imtransform` in your solution, implement the transform yourself.

[2] For an intuition into this relation, consider the projection of point $\mathbf{P}^i = (X, Y, Z)^T$ into two views $\mathbf{p}_1^i \equiv \mathbf{KP}^i$ and $\mathbf{p}_2^i \equiv \mathbf{KRP}^i$, where $\mathbf{K}$ is a (common) intrinsic parameter matrix, and $\mathbf{R}$ is a 3×3 rotation matrix. In this case: $\mathbf{p}_1^i \equiv \mathbf{KR}^{-1}\mathbf{K}^{-1}\mathbf{p}_2^i$.

Q2.2 (10 pnts): Implement a function `H2to1=computeH(p1, p2)`. Inputs: `p1` and `p2` should be $2 \times N$ matrices of corresponded $(x, y)^T$ coordinates between two images. Outputs: `H` should be a 3×3 matrix encoding the homography that best matches the linear equation derived above (in the least squares sense). Hint: Remember that, as in the previous question, **H** will only be determined up to scale, and the solution will involve an SVD.

Q2.3 (10 pnts): Select enough corresponding points from images $taj[1, 2].png$ using for example Matlab's `cpselect`. They should be reasonably accurate and cover as much of the images as possible. Use `computeH` to relate the points in the two views, and plot image $taj1.png$ with overlayed points `p1` (drawn in red), and (drawn in green) points `p2` transformed into the view from image 1 (call this variable `p2_t`). Save this figure as `q2_3.jpg`. Additionally, save these variables to `q2_3.mat`:

```
H2to1=computeH(p1,p2);
...
save q2_3.mat H2to1 p1 p2 p2_t;
```

Write an MATLAB expression for the average error (in pixels) between `p1` and the transformed points `p2`. and include it in your answer sheet. This expression should be in terms of **H**, $\mathbf{p}_1$, and $\mathbf{p}_2$. Compare this to the error that was minimized in items 1 and 2.

2.2 Image warping

The provided function `warp_im=warpH(im, H, out_size)` 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`.

The following code should produce a warped image $im2$ in the reference frame of $im1$. Not all of $im2$ may be visible, and some pixels may not be drawn to:

```
H2to1=computeH(p1,p2);
warp_im=warpH(im2, H2to1, size(im1));
```

Inputs: `im` is a `double` matrix of size $height \times width \times nChannels$ matrix[3], with `nChannels` set to 1,3 or 4, depending on whether it is a grayscale, RGB, or RGBA image). `H` is a $3x3$ matrix describing the homography ($\mathbf{p}^i_{warped} \equiv \mathbf{H}\mathbf{p}^i_{orig}$). `out_size` is specified by the dimensions of the warped output image, where `out_size=[out_height,out_width]` of the warped output image.

Outputs: `warp_im` is the warped output image and will be size `out_size(1)` $\times$ `out_size(2)` $\times$ 3.

[3]Images in Matlab are indexed as `im(row, col, channel)` where `row` corresponds to the y coordinate (height), and `col` to the x coordinate (width).

Figure 2: Original images and warped.

(a) *taj*1.*jpg* (b) *taj*2.*jpg* (c) *taj*2 warped into *taj*1

Q2.4 (5 pnts): 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 (a mosaic or panorama image containing all views) and `out_size(2)==1280`. **M** should be scaling and translation only, and the resulting image should fit snugly around the transformed corners of the images. Please include your code to find **M** in your answer sheet.

Note: Use the provided function for `warpH`, rather than any code from **QX1**.

Figure 3: Final mosaic view.

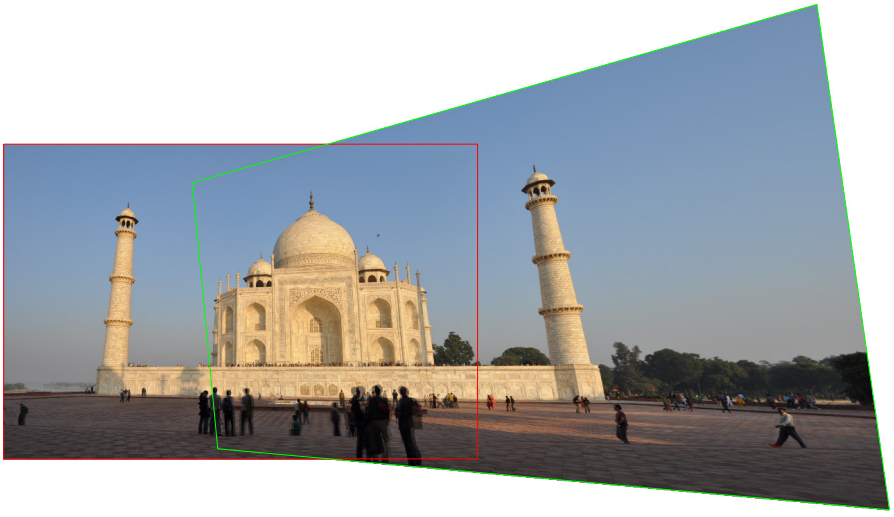

Image 1 Image 2

Figure 4: A pair of images used to detect the transparent objects

2.3 Synthesis and Blending

Q2.5 (15 pnts) Use the functions above to produce a panorama or mosaic view of the two images, and show the result. Save this image as `q2_5.jpg` and the code as a script `q2_5.m`. For overlapping pixels, it is common to blend between the values of both images. You can simply average the values. Alternatively, you can obtain a blending value for each image:

```
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.

QX2 (Extra Credit 5 pnts) Research and implement a more complex blending technique and save it as `blending.m` Please write a brief description of your methodology, observations on its strengths and limitations and include images saved as `blending*.jpg` that demonstrate its output.

3 Challenge: Detecting Transparent Objects (25 points)

Consider two images $\mathbf{I}_1$ and $\mathbf{I}_2$, files `mug.jpg` and `mug2.jpg` which include views of a transparent mug in front of a flat monitor screen, as shown in Fig. (4). Because the screen is planar, the two monitor images are related by a 3×3 homography $\mathbf{H}$, as shown in the previous question.

However, the mug and water inside it cause refraction such that the pixels inside the mug do not agree with the mapping $\mathbf{H}$. Therefore, *in principle*, one should be able to locate the mug by

1. Estimating the planar homography $\mathbf{H}$ between the two screens in $\mathbf{I}_1$ and $\mathbf{I}_2$.
2. Finding the image region in which the positions of the corresponding pixels between the two images do not agree with the planar homography $\mathbf{H}$.

The goal is to implement such an algorithm using Matlab, such that you will be able to detect the transparent mug given the two images $\mathbf{I}_1$ and $\mathbf{I}_2$.

After obtaining the homography $\mathbf{H}$, which encodes where a pixel in the second image $\mathbf{I}_2$ comes from the first image $\mathbf{I}_1$, we can 'warp' the image $\mathbf{I}_1$ to generate a new image as if the camera viewpoint physically changed. If both images contain only the planar scene, then this warping will precisely transform $\mathbf{I}_1$ to $\mathbf{I}_2$; for any object not in the planar scene, it will stand out (e.g. compare pixel intensities) – and we get a detection!

Q3.1 (5 pnts): Use your code from Section 2 to create a panorama of the two images and save it as `mug12.jpg`. What do you notice? No new code needed.

Q3.2 (20 pnts): Create a function called `detectTransparent(image1, image2)`, that overlays the original images with gray to indicate where the estimated transparent regions are located and displays them in /verb!imshow!. Please turn in `detectTransparent.m` and any supporting code files, the image files generated, a written description of your method and describe how you chose your threshold. Run your code for both $\mathbf{I}_1$ to $\mathbf{I}_2$ and $\mathbf{I}_2$ to $\mathbf{I}_1$, such that you generate two image files displaying detected transparencies overlaid on each source file.

3.1 Extra Credit: Make your own Image

Detect a transparent object OR create a panorama using two (or more) images of your making, i.e., use a physical camera to capture the images, not an online image database. Small prizes (chocolate and bragging rights) will be awarded for the three most creative or computationally impressive scenes and solutions.

QX3 (Extra Credit 10 pnts): What parameters does one need to keep in mind to create images for which one can use homographic transformations? Did you face additional challenges in creating your own image set? Please create a stitched panorama or estimated transparency images, as in the previous sections. Optionally include any additional visualization you find relevant to demonstrate your technique and solution. Include copies of the images you have created, e.g. `myimage1.jpg`, `myimage2.jpg`, a written description, and any new code you have created.