

# Homework 3

## M1522.001000 Computer Vision (2016 Spring)

Due: Wednesday April 27, 11:59PM

The goal of this homework is to explore Transformation Matrix and Image Warping.

## 1 Submitting Your Assignment

Your submission for this assignment consists of answers to one theory questions, the code for your MATLAB implementation and a short writeup describing any thresholds and parameters used, interesting observations you made or things you did differently while implementing the assignment. You need to zip the following items, name it as M1522001000\_HW3\_(your student ID).zip, and send it to TA's email(ta.cv@vision.snu.ac.kr).

Your submitted zip file should include the files arranged in this layout:

- `theory.txt` or `theory.pdf` : answers to the theory questions
- Folder `result`
  - `points_used.jpg` : jpeg image showing the set of corresponding points you selected for the Washington DC images plotted on the original images (use subplots to display the images side by side) (Part 1)
  - `wdc1_warped.jpg` : jpeg images showing the results of warping applied to `wdc1.jpg` (Part 2)
  - `crop1_warped.jpg` : jpeg images showing the results of warping applied to `crop1.jpg` (Part 2)
  - `wdc_merged.jpg` : jpeg images showing the results of combining `wdc1.jpg` and `wdc2.jpg` (Part 2)
  - `crop_merged.jpg` : jpeg images showing the results of combining `crop1.jpg` and `crop2.jpg` (Part 2)
  - `writeup.doc` or `writeup.pdf` : the writeup describing your experiments
- Folder `matlab`
  - `points.mat`, containing two variables, `points1` and `points2`, which contain lists of the corresponding points you used for the Washington DC images. The list should be  $N \times 2$  matrices, where each row is an x-y coordinate (Part 1)
  - `computeH.m`, `warpImage.m`
  - You can also include any extra helper functions.

Refer to Section 4 for the details of the above files.

Your zip file should be sent before the due(Wednesday April 27, 11:59PM). Later than that, you will use one late day.

## 2 Warm Up

This section is meant to provide a short intro to get you used to working with images in Matlab. It is recommended that you do this section searly and so that you would not be overwhelmed by what the rest of the assignment asks you to do.

Let us start with the toys image (*toys.jpg*) and perform some simple image warping. Let us begin with a simple operation that will stretch the image in the horizontal direction. One possible method that can be used is to apply a transformation matrix to each point/pixel in the image. Apply the transform  $T_h$  to the image (ie.  $T_h P_{i,j}$ , where  $P_{i,j} = [i, j, 1]^T$  is a point in the image expressed in homogeneous coordinates).

$$T_h = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The transformation matrix  $T_h$  will stretch the image in the horizontal direction by a factor of 2. 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. (This means that your loop should iterate for every pixel in the warped image which has twice as much pixels as the original image in the horizontal direction and the same number of pixels in the vertical direction). It is important to observe the relationship between the transformation matrix  $T_h$  and the resultant image. Further explore this relationship by modifying the transformation matrix to stretch the image in the vertical direction.

We can now check to see if the image warping worked as expected by displaying the two images (original and warped) in Matlab to see if their scales are correct. This can be done using one of many Matlab commands (try using `imshow`, `image`, etc. and identify the difference between each function). To ensure that the scaling was done properly you can use the command `size` on each image to check their dimensions. We can perform yet another check to ensure that the image warping was done correctly. Start by selecting points in the original image (try using the `ginput` command). Then transform and map these points onto the warped image and plot both sets of points (selected and transformed) on the corresponding images to check the accuracy of the warping. **Note that `ginput` and `plot` commands use the (horizontal,vertical) format to refer to each point/pixel, while an image matrix is referenced using the (row,column) format.** Therefore, pay careful attention to which format you are using while mapping the points to the warped image. If done properly, both sets of points should appear in the same location in their corresponding images.

## 3 Theory Questions

### Question 1: Image Warping

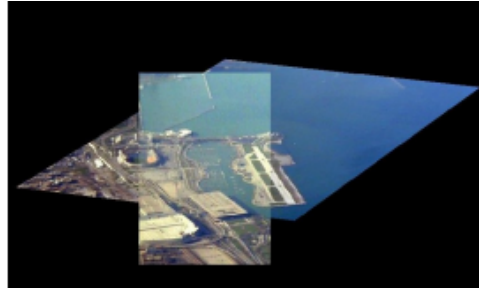
(20 points)

In this problem, we consider a vision application in which a scene is being observed by multiple cameras at various locations and orientations. Figure 1 shows examples of images taken from two different cameras.

Given two cameras, a natural thing to do would be to compute how the scene currently seen from camera 1 would appear from camera 2's point of view. In particular, this would allow one to paste together multiple images which have overlapping regions, even if those images were obtained from different locations. This is also called *Mosaicing*.



**Figure 1:** Views from two cameras



**Figure 2:** Mosaic image

An image mosaic is created by first choosing one camera as the reference frame and its associated image as the reference image. The task then consists of mapping all other images onto the reference frame so that all images can be displayed together with the reference image. In the most general case, there would be no constraints on the scene geometry, making the problem quite hard to solve. If, however, the scene can be approximated by a plane in 3D, a solution can be formulated much more easily even *without* the knowledge of camera calibration parameters. Figure 2 depicts a typical mosaicing result. To solve this section of the homework, you will first derive the transformation that maps one image onto another in the planar scene case. Then you will write Matlab code to find this warping and apply it to two pairs of test images, which are provided on the assignments web page.

To begin, we consider the projection transformations of planes in images. Imagine two cameras  $C_1$  and  $C_2$  looking at a plane  $\pi$  in the world. Consider a point  $\mathbf{X}$  on the plane  $\pi$  and its projections  $\mathbf{p} = (u_1, v_1, 1)^T$  in *image1* and  $\mathbf{q} = (u_2, v_2, 1)^T$  in *image2*. Assume that the two cameras have  $3 \times 4$  projection matrices,  $\mathbf{M}_1$  and  $\mathbf{M}_2$ , associated with them, respectively.

(a) Show that there exists a  $3 \times 3$  matrix  $\mathbf{H}$  such that, for any point  $\mathbf{X}$ :

$$\mathbf{q} \equiv \mathbf{H}\mathbf{p}$$

(Recall that  $\equiv$  denotes the equality in homogeneous coordinates, meaning that the left and right hand side are proportional.) Note,  $\mathbf{H}$  only depends on the plane and the projection matrices of the two cameras.

The interesting thing about this result is that by using  $\mathbf{H}$  we can compute the image of  $\mathbf{X}$  that would be seen in camera  $C_2$  from the image of the point in camera  $C_1$  without knowing its three-dimensional location. Such an  $\mathbf{H}$  is a projective transformation of the plane, also referred to as a *homography*.

(b) Given a set of points  $\{(u_1^i, v_1^i)\}, i = 1 \dots N$  in *image1*, and the corresponding set of points  $\{(u_2^i, v_2^i)\}, i = 1 \dots N$  in *image2*, how many sets of points, at least, are needed to recover  $\mathbf{H}$ ? Write and justify your answer.

(c) Design your algorithm for estimating  $\mathbf{H}$ . (It's okay to explain your algorithm roughly. It's warming-up for following part.)


## 4 Programming

**Part 1** Estimating transformations from the image points (15 points)

Write a Matlab function

```
function H = computeH(t1, t2)
```

which computes the matrix  $\mathbf{H}$  using the method you just derived. Input arrays should be  $2 \times N$  matrices, listing a single image points coordinates per column. First, try the Washington DC images (*wdc1.jpg* and *wdc2.jpg*). Manually select two sets of corresponding image points on the images, and apply the transformation  $\mathbf{H}$  to them. (Checkout the function *cpselect* in Matlabs Image Processing Toolbox for help selecting corresponding points.) Next, try the crop circles images (*crop1.jpg* and *crop2.jpg*). For these, use the points that we've provided in *cc1.mat* and *cc2.mat*. Note that, throughout this estimation procedure, camera projection matrices did not come into play at all! Hence, no calibration was necessary.

Draw corresponding points you used for Washington DC images with red dots on *wdc1.jpg* and *wdc2.jpg* and save it as *points\_used.jpg* (use subplots to display the images side by side). Save these corresponding points as *points.mat*. *points.mat* must contain two variables, *points1* and *points2*, which contain lists of the corresponding points you used for Washington DC images. The list should be  $N \times 2$  matrices, where each row is an x-y coordinate. 

*Hint:* Recall that  $\mathbf{H}$  is a projective transformation matrix and hence, defined only up to a scale. A good way to enforce this is by constraining the Frobenius norm of the matrix  $\mathbf{H}$  to be 1.

Here are a few implementation tips:

- You should be able to implement the solution within a few lines of Matlab code. If you are new to Matlab, it may help to think of doing the problem iteratively.

for each point, and then try to convert this algorithm to a vectorized version for a more efficient Matlab implementation.

- Beware of numerical ill-conditions: Your estimation procedure may perform better if image coordinates range from 0 to 2 as opposed to from 1 to 200. Consider scaling your measurements to avoid numerical issues.
- For the estimation to work well, a sufficient number of points should be provided. You should select the corresponding points in the two Washington DC test images manually (or with the help of *cpselect*) and give all your results with your own point set.
- Ignore all differences in illumination between pairs of images, the illumination difference shouldn't make any difference to your algorithm.

## Part 2 Image warping and mosaicing

(5 points)

Write a function

```
function [Iwarp, Imerge] = warpImage(Iin, Iref, H)
```

which takes as input an image *Iin*, a reference image *Iref*, and a  $3 \times 3$  homography, and returns 2 images as outputs. The first image is *Iwarp*, which is the input image *Iin* warped according to **H** to be in the frame of the reference image *Iref*. The second output image is *Imerge*, a single mosaic image with a larger field of view containing both the input images.

In order to avoid aliasing and sub-sampling effects, consider producing the output by **solving for each pixel of the destination image** rather than mapping each pixel in the input image to a point in the *destination* image (which will leave holes). **Also note that the input and output images will be of different dimensions.**

Run `warpImage(Iin, Iref, H)` with `(wdc1.jpg, wdc2.jpg)` and `(crop1.jpg, crop2.jpg)` and save its output as `(wdc1-warped.jpg, wdc-merged.jpg)` and `(crop1-warped.jpg, crop-merged.jpg)`