Homework 4
Computer Vision, Fall 2017
Due Date: November 1, 2017
Total Points: 24

This homework contains one written assignment and one programming challenge. All submissions are due at the beginning of class on **November 1, 2017**. The assignment should be submitted according to the instructions in the document **Guidelines for Programming Assignments.pdf** before the beginning of class. Note that there is no credit for late submissions. So please start working on the homework early.

Written Assignments

Problem 1: The two squares in Figure 1 are related by a transformation T. c_1 is located at (x_1, y_1), and c_2 is located at (x_2, y_2). Express T as a chain of transformation matrices such that c_1 is mapped to c_2, and a_1 is mapped to a_2. In addition, comment on the function of each transformation matrix (e.g., scale, rotation, etc.). Note that c_1, c_2 do not represent the center of the squares. **(4 points)**

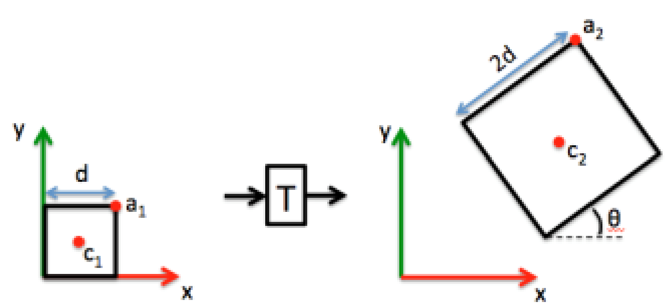

Figure 1

Programming Assignments

This programming assignment has one challenge with a subset of milestones or unit tests. Instructions and summary corresponding to these are given below. **runHw4.m** will be your main interface for executing and testing your code. Parameters for the different programs or unit tests can also be set in that file. Before submitting, make sure you can run all your programs with the command `runHw4('all')` with no errors.

MATLAB is optimized for operations involving matrices and vectors. Avoid using loops (e.g., for, while) in MATLAB whenever possible—looping can result in long running code. Instead, you should "vectorize [1]" loops to optimize your code for performance. In many cases, vectorization also results in more compact code (fewer lines to write!). If you are new to MATLAB, refer to these articles [1] [2] on techniques to optimize MATLAB code.

Challenge 1: Your task is to develop an "Image Mosaicking App" that stitches a collection of photos into a mosaic. Creating image mosaics and panoramas have become one of the most popular features on smart phones. Outside of the realm of smart phones, similar applications have also been developed to create stunning panoramas seen on gigapan.org. With the concepts and algorithms presented in the Image Alignment lecture, you too can create an image mosaic.

Before we create the mosaicking app, we will create the individual tools required to build it. Each tool is a separate program you need to write and submit.

a. In this part, the goal is to develop a program to calculate homography between a pair of images. We will also develop two additional small programs to help verify whether the calculated homography is correct.

   First, write a program named `computeHomograpy` that calculates the homography between two sets of corresponding points in two images:
   `H = computeHomography(src_pts, dest_pts)`.
   You can use the MATLAB function `eig` to compute the homography matrix.

   A pair of images, related by a homography, are provided to help test your program. You will choose at least four corresponding points manually (you can use the MATLAB function `ginput`) and use them to compute the homography. **(2 points)**

   Second, write a program named `applyHomography` that applies homography to a set of points:
   `dest_pts = applyHomography(H, test_pts)`.
   Choose a new set of points `test_pts` from the first image and apply the computed homography to them. **(1 points)**

Third, write a program named showCorrespondence to annotate the corresponding points in the two images:

`result_img = showCorrespondence(src_img, dest_img, src_pts, dest_pts)`.

Here src_img and dest_img are two images related by a homography, and result_img is an image showing src_img and dest_img side-by-side, with lines connecting src_pts and dest_pts. Figure 2 shows an example output of showCorrespondence. Refer to demoMATLABTricksFun from the previous assignment for tricks to draw lines and save the displayed image along with its annotations. **(2 points)**

Functions not allowed: all `im*, cp*, tf*` functions

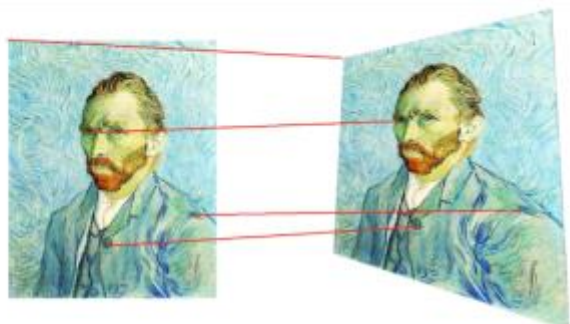

Figure 2

b. Develop a program to warp an image. Use it to warp and paste a portrait of Vincent to the empty billboards in the image shown in Figure 3. To help you get started, a skeleton code has been provided in challenge1b.

Before [3] After

Figure 3

First, you need to estimate the homography from the portrait to the target billboard frame. Fill in the missing parts in challenge1b where appropriate and use the programs developed earlier to compute a homography.

Second, write a program named `backwardWarpImg` that warps an image based on a homography:

```
[mask, dest_img] = backwardWarpImg(src_img,
dest_to_src_homography, dest_canvas_width_height).
```

The input argument `src_img` is the source image (the portrait in this case). Because you will implement **backward warping**, `dest_to_src_homography` is the inverse of the homography that maps source to destination. `dest_canvas_width_height` specifies width and height of the destination image.

Now the output arguments. `dest_img` is the output image. In this case, `dest_img` is a warped portrait on a black canvas. `mask` is a binary mask indicating the area of the warped image on the canvas. `mask` is needed because the warped image is often not the final result, but used as an input to other post-processing tasks (in this case, you will post the warped image to another image).

After generating a warped portrait image and its associated mask, superimpose the image on the given billboard image. A reference code is provided to help you perform this task. You are allowed to use the MATLAB built-in function `interp2`. **(4 points)**
Functions not allowed: all `im*, cp*, tf*` functions

c. Recall the outlier problem discussed in the Image Alignment lecture. Some of the corresponding points, referred to as outliers, computed using automatic interest point detectors may not concur with the actual homography. Write a function named `runRANSAC` to robustly compute homography using RANSAC to address the outlier problem.

```
[inliers_id, src_to_dest_H]= runRANSAC(src_pt, dest_pt,
ransac_n, ransac_eps).
```

To ease your workload, a function `genSIFTMatches` has been provided to compute a set of points (`src_pt`) in the source image and a set of corresponding points (`dest_pt`) in the destination image. Typically, these matches will include many outliers. `ransac_n` specifies the maximum number of RANSAC iterations, and `ransac_eps` specifies the acceptable alignment error in pixels. Use Euclidean distance to measure the error. Your program should return a homography `src_to_dest_H` that relates the `src_pt` to the `dest_pt`. In addition, return a vector `inliers_id` that lists

the indices of inliers, i.e., the indices of the rows in `src_pt` and `dest_pt`. Figures 4 and 5 show the matches between two images, before and after running RANSAC. **(4 points)**
Functions not allowed: all `im*, cp*, tf*` functions

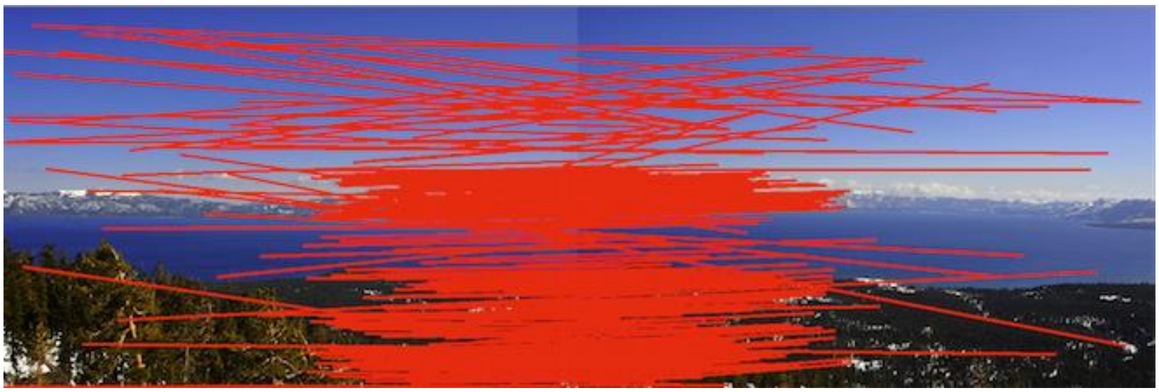

Figure 4

Figure 5

d. Write a program named `blendImagePair` that blends two images into one:
`result = blendImagePair(img1, mask1, img2, mask2, blending_mode)`.
`img1` and `img2` are two input images. `mask1` and `mask2` are the corresponding binary masks indicating the regions of interest.
`blending_mode`, a string, is either `"overlay"` or `"blend"`. In the case of `"overlay"`, copy `img2` over `img1` wherever the `mask2` applies. In the case of `"blend"`, perform weighted blending as discussed in class. You may use the MATLAB function `bwdist` to compute a new weighted mask for blending.
Figure 6 shows example outputs (left: blended, right: overlay). **(2 points)**
Functions not allowed: all `im*, cp*, tf*` functions

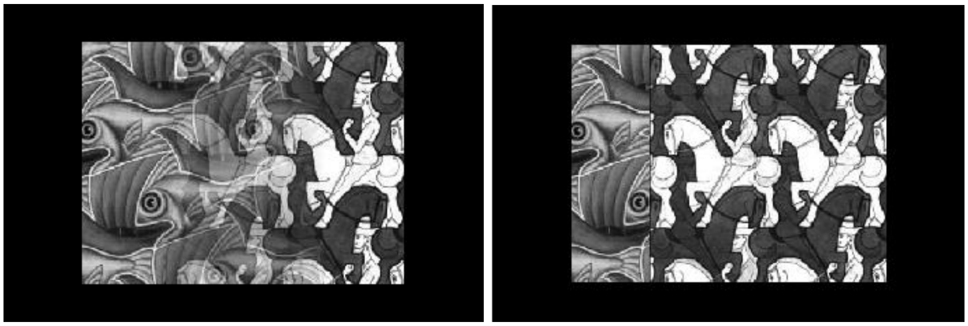

Figure 6

e. Now you have all the tools to build the mosaicking app! Write a program `stitchImg` that stitches the input images into one mosaic.
`stitched_img = stitchImg(img1, img2, …, imgN)`.
Your program should accept an arbitrary number of images (you can use the MATLAB function `varargin` to take variable number of inputs). You can assume the order of the input images matches the order you wish to stitch the images. Also, in this assignment we will only stitch images of a single row/column. Use "`blend`" mode to blend the image when you call `blendImagePair`. **(4 points)**
Functions not allowed: all `im*`, `cp*`, `tf*` functions

When warping multiple images on to a single base image, you may want to first compute the bounding box. This bounding box may extend beyond the size of the base image and can have negative coordinates. How would you warp the images in this case? Hint: Homography needs to be updated with additional translation.

f. Capture images using your own (cell phone) camera and stitch them to create a mosaic. Note that the mosaic need not be a horizontal panorama. Submit both the captured and stitched images. **(1 point)**

References

[1] MathWorks. Vectorization. [Online]. http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

[2] MathWorks. Technique for Improving Performance. [Online]. http://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

[3] Flickr. [Online]. http://www.flickr.com/photos/schmaeche/2545319687/