EE588 Advanced Image Processing
Project #3 Image Registration
(due Monday, Oct. 22, 2018, 10:30:00 am)

Matlab commands are specified in the text and in `blue Courier font` and python commands in parenthesis and in `red Courier font`; variables and filenames (or commands that are consistent between Matlab and python) are specified in black Courier font. In the following python syntax, I have used `import matplotlib.pyplot as plt`, `import skimage.transform`, and `import os`; additionally, you probably want to use `%matplotlib inline` to include figures inline in your jupyter notebook.

Your code should be well commented, and your submission should include enough narration to lead me through your solution. You should be briefly discussing your decisions, your approach to coding, and your results as you progress through the code. Generally this will mean, for each sub-instruction, a brief narrative, your commented code, and the requested output (printed or images displayed). If it is easier, you may upload a separate pdf document for each part (please name each file according to the part so that I can easily navigate—e.g., part_a.pdf). Please also include the last page of this assignment as a cover page to assist me in grading. You may fill that out and upload it as a separate document, or include it as the first page of your merged pdf.

(a) **SIFT Keypoint Extraction.**

In this problem, we will use the SIFT demo code provided by the original author Lowe. Download the demo code from http://www.cs.ubc.ca/~lowe/keypoints/siftDemoV4.zip and unzip. There are Matlab and c codes provided, but we will be using the binaries and learning how to call command line programs from Matlab (python). Carefully read the `README` file, with particular attention to the "Binaries for detecting SIFT features" section.

(i) This subpart will verify that the binaries provided in the SIFT demo are working on your system before we dive into trying to call those binaries from Matlab (python). Using the syntax provided in the `README` file, from the `siftDemoV4` directory, run the sift command `./sift -display <scene.pgm >result.pgm` in a terminal on a Linux machine (this should also work for a Mac, I think) or `siftWin32 -display <scene.pgm >result.pgm` in a command window on a Windows machine. (For brevity, subsequent instructions will use just the Linux syntax). Take a screenshot of what is printed out in the terminal/command window. It should say something to the effect of

```
Finding keypoints...
1021 keypoints found.
PGM file output.
```

In Matlab (python), read in the screenshot and display it (this is just an easy way for you to include your results for the terminal/command window stuff within publish (jupyter notebook)). Display the original image `siftDemoV4/scene.pgm` and the `siftDemoV4/result.pgm` file generated via the command line. These are grayscale images—please display them accordingly.

(ii) The following instructions assume that you are in the directory just above the `siftDemoV4` directory so that we don't clutter up the original download directory. Copy the `siftDemoV4/scene.pgm` file to the working directory. You can now access the command line directly from Matlab (python) and run the same command as in the previous part using `system('siftDemoV4/sift -display <scene.pgm >result.pgm')` (`os.system('siftDemoV4/sift -display <scene.pgm >result.pgm')`). The `system` (`os.system`) call will return a value of `0` if the command ran successfully. There should now be a `result.pgm` file in your working directory. Read in this image and display it. Note—if you are using Windows, you will want to use a backslash instead of a forwardslash, i.e., `system('siftDemoV4\sift -display <scene.pgm >result.pgm')` (`os.system('siftDemoV4\sift -display <scene.pgm >result.pgm')`). Usually Matlab and python are smart enough to handle this, but the `system` (`os.system`) command literally sends a string to the command window and Windows is not smart enough to handle it. Subsequent instructions will use the forward slash syntax—change it as needed if you are using a Windows machine.

(iii) Without the `-display` option, the `sift` command will output the keypoints in a text file. Carefully read the README file regarding how to interpret the output text file. From Matlab (python) run the command `system('siftDemoV4/sift <scene.pgm >result.key')` (`os.system('siftDemoV4/sift <scene.pgm >result.key')`). Read in the first 9 lines of `result.key` and display them.

(iv) According to the `README` file, the number of keypoints can be controlled by changing the image resolution. For the `scene.pgm` image, determine the number of keypoints detected for the image decimated by factors `res` =1, 2, 4, 8, 16, and 32. In order to do this automatically, you will want to decimate the image using `imresize(I,[M,N]/res)` (`skimage.transform.resize(I,(M,N)/res,order=3,anti_alising=True)`), where `M`, `N` are the dimensions of the original image. Write out the decimated image to an image file. Now you want to point your `system` (`os.system`) command to the decimated image that you just wrote out. Plot the number of detected keypoints versus decimation factor. This code should utilize the text files output from the `sift` command to automatically read in the number of keypoints for each decimated image. What sort of relationship does the number of keypoints have to the image resolution (e.g., is it linear?)?

(v) (Extra Credit): Using the text file that the `sift` command outputs, write code to visualize the SIFT keypoints in a similar fashion to what the `result.pgm` file demonstrated. Display the `scene.pgm` image with the keypoints visualized on top.

(b) **SIFT Keypoint Matching I**

In this problem, we will use the SIFT keypoints returned by the SIFT code to match keypoints between an image `I` and an affine transformed image `I_t`. Carefully read the `siftDemoV4/README` file, with particular attention to the "ASCII file output for keypoints" section. For use in this part, there is a file `cameraman.pgm` file available on canvas (this is a standard image provided with the Matlab image processing toolbox which has been converted to `.pgm` format).

(i) Create a function `[xyso,F]=create_feature_matrices(key_filename)` (`xyso,F=create_feature_matrices(key_filename)`). Input `key_filename` is a string with the key filename (e.g., `'result.key'` from part (a-iii) above). Output are two matrices: `xyso` is $K \times 4$ and `F` is $K \times 128$, where K is the total number of keypoints detected by the SIFT algorithm (note that number of keypoints is the first entry on the first line of the key file). Columns of matrix `xyso` should be the x-location, y-location, scale, and gradient orientation of each keypoint. Columns of matrix `F` should be the 128 features for each keypoint.

(ii) Create a function `D=pairwise_distance(F1,F2)` to compute the Euclidean distance between each 128-D feature vector in feature matrix `F1` with each 128-D feature vector in feature matrix `F2`. Inputs `F1` and `F2` are the $K_1 \times 128$ and $K_2 \times 128$ feature matrices returned by your `create_feature_matrices` function. Output `D` is a $K_1 \times K_2$ matrix, where K_1 is the number of keypoints detected in `I` and K_2 is the number of keypoints detected in `I_t`. Thus, `D(i,j)` (`D[i,j]`) is the Euclidean distance between the `i`-th keypoint in `I` and the `j`-th keypoint in `I_t`.

(iii) Create a function `P=paired_keypoints(D,thresh)` to compute keypoint pairs. Inputs are `D`, the pairwise distance matrix returned by your `pairwise_distance` function, and `thresh` which will control the ratio between the closest and second closest match (`0.8` per the `lowe2004.pdf` paper or `0.6` per the `siftDemoV4/README` file). Output is a length- K vector `P` which contains the index to the keypoint match (or 0 if it does not satisfy the second nearest neighbor check), where K is the number of keypoints in the reference image `I`. Follow the description in Section 7 of the `lowe2004.pdf` paper on how to determine keypoint matches. The interpretation of vector `P` is that if `P(i)=j` (`P[i]=j`), then the `j`-th keypoint in `I_t` is a match to the `i`-th keypoint in reference image `I`. If `P(i)=0` (`P[i]=-1`), then there was no valid match in `I_t` for the `i`-th keypoint in `I_t`.

(iv) Read in the `cameraman.pgm` file as reference image `I`. Define transformed image `I_t=imrotate(I,10,'crop')` (`I_t=skimage.transform.rotate(I,10)`) which rotates image `I` by `10` degrees in a counterclockwise manner. Write out image `I_t` to filename

`I_t.pgm`. Use the SIFT program to generate a key file `I.key` for image `cameraman.pgm` and `I_t.key` for `I_t.pgm`. Create feature matrices `[xyso,F]=create_feature_matrices('I.key')` (`xyso,F=create_feature_matrices('I.key')`) and `[xyso_t,F_t]=create_feature_matrices('I_t.key')` (`xyso_t,F_t=create_feature_matrices('I_t.key')`). Print out the first row of `xymo`, `F`, `xymo_t`, and `F_t`. Create the pairwise distance matrix `D=pairwise_distance(F,F_t)`. Print out the first row of `D`. Create the paired keypoint vector `P=paired_keypoints(D,0.6)`. Print out `P`.

(v) Define a side-by-side image `sbs=[I,I_t]` (`sbs=np.concatenate((I,I_t),axis=1)`, assuming both `I` and `I_t` are ndarrays). Note—make sure that both `I` and `I_t` are scaled the same (i.e., both in [0,255] or both in [0,1]) since the `imrotate` (`skimage.transform.rotate`) command may change intensity scaling. Display image `sbs`. Using your paired keypoint vector `P` and matrices `xymo` and `xymo_t`, draw lines from keypoints in `I` to the paired keypoints in `I_t`. Recall that the (x,y) locations of the `i`-th keypoint in image `I` are in `xyso(i,1:2)` (`xyso[i,0:1]`) and the (x,y) locations of the `j`-th keypoint in image `I_t` are in `xyso_t(j,1:2)` (`xyso_t[j,0:1]`). You will need to offset the column location for the `I_t` keypoint in order to draw a line between `I` and `I_t` in `sbs`. For example, if there are `N` columns in image `I`, `plot([xyso(1,2),xyso_t(1,2)+N], [xyso(1,1),xyso_t(1,1)])` (`plt.plot([xyso[0,1],xyso_t[0,1]+N], [xyso[0,0],xyso_t[0,0]])`) would plot a line from the first keypoint in `I` to the first keypoint in `I_t`. It is your job to figure out how to leverage `P` to connect the matched keypoints for `I` and `I_t` as defined above. Do the SIFT keypoints appear to be well-matched? Are there specific keypoints that do not appear to be well-matched?

(c) **SIFT Keypoint Matching II**

This part will leverage the basic code you developed in part (b) and will study the effects of different image transformations on the ability to match SIFT keypoints. In addition to the `cameraman.pgm` file used above, there are two additional `.pgm` files available on canvas (`hestain.pgm` and `circlesBrightDark.pgm`, both of which are also standard Matlab image processing toolbox images converted to pgm format). These three images are chosen to illustrate the performance of SIFT for images with man-made objects (`cameraman.pgm`), images with biological entities like stained cells under a microscope (`hestain.pgm`), and images with very little structure (`circlesBrightDark.pgm`). Unless otherwise specifed, you may assume a distance ratio of `0.6` for computation of your paired keypoints vector `P`.

(i) For `I` as the original image and `I_t` as the same image rotated by `10` degrees, match keypoints between `I` and `I_t` for `cameraman.pgm` and visualize as in (b-v) above. Repeat for `hestain.pgm`. Repeat for `circlesBrightDark.pgm`. Discuss the implications of using SIFT for each of these images. What potential issues do you foresee using SIFT for image registration for images similar to these three examples?

(ii) Repeat (c-i), but before rotating by `10` degrees, add random Gaussian noise with a standard deviation of `0.1` (assuming an image intensity range of `[0,1]`) using `randn` (`np.random.randn`). The addition of Gaussian noise means that you may end up with some pixels with intensity outside of the range `[0,1]`, in which case you should clip any intensities less than `0` to `0` and any intensities greater than `1` to `1`. Discuss implications of using SIFT for image registration for these rotated and noisy images. How do the keypoint matches compare here compared to those in part (c-i)?

(iii) Repeat (c-i), but define the transformed image as half the resolution in each dimension of `I`, i.e., `imresize(I,[M,N]/2)` (`skimage.transform.resize(I,(M,N)/2.,order=3,anti_aliasing=True)`), where `M`, `N` are the dimensions of the original `I`. Then rotate the resized image by 10 degrees. Note that you will need to concatenate `M/2` rows of zeros to the bottom of `I_t` prior to defining the side-by-side image in order to keep dimensions consistent. Discuss implications of using SIFT for image registration for these resized and rotated. How do the keypoint matches compare here compared to those in part (c-i)?

(iv) Define image `I` as `cameraman.pgm` and image `I_t` as `hestain.pgm` and match keypoints between the images. Since `I_t` will have fewer rows than `I`, you will need to concatenate an appropriate number of rows of zeros to the bottom of `I_t` prior to defining the side-by-side image. How do the keypoint matches compare here compared to those in part (c-i)?

(v) If we choose distance ratio `thresh=1`, we can study the effects of using *all* keypoint matches. Use `I` as the `cameraman.pgm` image and `I_t` as a `10` degree rotation of `I` and display the keypoint matches associated with `thresh=1`. How do the keypoint matches compare here compared to those in part (c-i)?

(d) **Estimation of Affine Transformation Parameters**

In this part, you will implement the least-squares estimation problem described in Section 7.4 of the `lowe2004.pdf` paper. You will be working with your paired keypoints vector `P` and computing and using the affine transformation matrix `A` as defined in the `lowe2004.pdf` paper.

(i) Create a function `t=estimate_affine_transformation(P,xyso,xyso_t)` to estimate the affine transformation parameters. Note—the `lowe2004.pdf` paper uses **x** as the transformation parameter vector, but I am calling it `t` to distinguish it from the x-coordinates used throughout these instructions. Within your `estimate_affine_transformation` function, using your paired keypoint vector `P` and the corresponding (x,y) locations in `xyso` and `xyso_t`, create matrix `A` and vector `b` as described in the `lowe2004.pdf` paper. Note that for `P(i)=j` (`P[i]=j`), this means that you will populate two rows in `A` (`[x,y,0,0,1,0]` and `[0,0,x,y,0,1]`, where `x`, `y` come from `xyso(i,1:2)` (`xyso[i,0:1]`)) and two rows in vector `b` (`x_t` and `y_t` where `x_t`, `y_t` come from `xyso_t(j,1:2)` (`xyso_t[j,0:1]`)). Be careful to be consistent in your interpretation of x- and y-coordinates in an image. Matrix `A` will be $2K \times 6$ and vector `b` will be $2K \times 1$ where K is the number of keypoint matches. The 6×1 transformation parameter vector `t=pinv(A)*b` (`t=np.squeeze(np.matmul(np.linalg.pinv(A),b))`).

(ii) Define image `I` as `cameraman.pgm` and transformed image `I_t=imrotate(I,45)` (`I_t=skimage.transform.rotate(I,45,resize=True)`) Note that we are using slightly different options with the `imrotate` (`skimage.transform.rotate`) command in order to avoid cropping the image when rotating. Using your previously developed functions, compute feature matrices `xyso`, `F`, `xyso_t`, and `F_t`; compute the pairwise distances `D`, and the paired keypoints `P`. Using your `estimate_affine_transformation` function, compute transformation parameter vector `t`. Print out `t`. Now we need to get those estimated affine transformation parameters into a form that we can use them to manipulate `I` and `I_t`. First, create the homogeneous transformation matrix `T=[t(1),t(2),t(5);t(3),t(4),t(6);0,0,1]` (`T=np.array([[t[0],t[1],t[4]],[t[2],t[3],t[5]],[0,0,1]])`). Print out `T`. Use the homogeneous transformation matrix to create a transformation object `tform=invert(affine2d(T'))` where the `invert` is needed since Matlab interprets the homogeneous matrix `T` as transforming from `I_t` to `I` rather than the other way around (`tform=skimage.transform.AffineTransform(matrix=T)`). Apply the estimated transformation `tform` to the original image `I` using `I_tform=imwarp(I,tform)` (`I_tform=skimage.transform.warp(I,tform)`). If transformation parameters `t` were estimated well, then `I_tform` should be similar to `I_t` (your `45` degree rotated `I`). Note, however, that the treatment of the image edges may be different, so you will want to focus on the central portions of the image. Display `I_tform`. Does this image appear to correspond to a `45` degree rotation of image `I`? Discuss similarities and differences between `I_tform` and `I_t`.

(iii) You can compute the inverse transformation of `tform` using using `itform=invert(tform)` (`itform=tform.inverse`). Define `I_tform_itform=imwarp(I_tform,itform)` (`I_tform_itform=skimage.transform.warp(I_tform,itform)`). Display `I_tform_itform`. Since we apply the transform `tform` and then the inverse transform `itform`, `I_tform_itform` should be the original image. How does this image `I_tform_itform` compare to the original image `I`? It will probably be worth enabling the axis on your visualization so you can better estimate the pixel extent of the image.

(iv) Apply `itform` to `I_t`, `I_t_itform=imwarp(I_t,itform)` (`I_t_itform=skimage.transform.warp(I_t,itform)`). Since `tform` is only an estimate of the transformation, application of `itform` to `I_t` may not exactly reverse the operations. How does this image `I_t_itform` compare to the original image `I`? It will probably be worth enabling the axis on your visualization so you can better estimate the pixel extent of the image.

(v) Define image `I` as `cameraman.pgm`, `I_t` as image `I` with a `45` degree rotation and use a distance threshold `thresh=1` for computation of your paired keypoints vector `P`. Repeat the steps above to compute the estimated transform `tform` and inverse transform `itform`. Apply `itform` to `I_t` and display. Discuss how this result is similar and/or different from the results in part (d-iv). It will probably be worth enabling the axis on your visualization so you can better estimate the pixel extent of the image.

EE588 Advanced Image Processing
Project #3 Image Registration
(due Monday, Oct. 22, 2018, 10:30:00 am)

Confidence (number 0 to 100% about your confidence in your performance on this project and optional statement about areas where you are particularly confidence or not):

Difficulty (number 0 to 100%):

Time Spent (hours):

Problem	**Points**
(a) Keypoint Extraction	/20
(b) Keypoint Matching I	/40
(c) Keypoint Matching II	/20
(d) Affine Transformation	/20
TOTAL	/100