**CS4670/5670: Computer Vision, Spring 2015  
Project 3:  Autostitch**



**Brief**

* **Assigned**: Thursday, March 12, 2015
* **Code Due:** Wednesday, March 25 (**Changed!**), 2015 (by 9:00am)
* **Artifact Due**: Wednesday, March 25 (**Changed!**), 2015 (by 11:59pm)
* This assignment should be done in teams of 2 students.

**Synopsis**

In this project, you will implement a system to combine a series of horizontally overlapping photographs into a single panoramic image. Using the ORB feature detector and descriptors (which you are familiar with through the Features project), you will first detect discriminating features in the images and find the best matching features in the other images. Then, using RANSAC, you will automatically align the photographs (determine their overlap and relative positions) and then blend the resulting images into a single seamless panorama. We have provided you with a graphical interface that lets you view the results of the various intermediate steps of the process. We have also provided you with some test images and skeleton code to get you started with the project.

The project will consist of a pipeline of tabs visualized through autostichUI that will operate on images or intermediate results to produce the final panorama output.

The steps required to create a panorama are listed below. You will be creating two ways to stitch a panorama: using translations (where you'll need to pre-spherically-warp the input images) and homographies, where you align the input images directly. The steps in square brackets are only used with the spherical warping route:

|  |  |
| --- | --- |
|  | Step |
| 1. | Take pictures on a tripod (or handheld) |
| 2. | [Warp to spherical coordinates] |
| 3. | Extract features |
| 4. | Match features |
| 5. | Align neighboring pairs using RANSAC |
| 6. | Write out list of neighboring translations |
| 7. | Correct for drift |
| 8. | Read in [warped] images and blend them |
| 9. | Crop the result and import into a viewer |  |

Finally, to make programming easier, **this assignment will be in Python**, with the help of NumPy and SciPy. Python+NumPy+SciPy is a very powerful scientific computing environment, and makes computer vision tasks much easier. A crash-course on Python and NumPy can be found [here](http://cs231n.github.io/python-numpy-tutorial/).

**Downloads**

* **Skeleton code**

Available through [git](http://git-scm.com/) (this should help make distributing any updates easier). With git installed, you can download the code by typing (using the command-line interface to git)

>> git clone http://www.cs.cornell.edu/courses/cs4670/2015sp/projects/pa3/skeleton.git

This will create the directory skeleton. To get updates to the code you can then simply run

>> git pull

This will fetch any updates and merge them into your local copy. If we modify a file you have already modified git will not overwrite your changes. Instead, it will mark the file as having a conflict and then ask you to resolve how to integrate the changes from these two sources. [Here's](http://www.kernel.org/pub/software/scm/git/docs/v1.7.3/user-manual.html#resolving-a-merge) a quick guide on how to resolve these conflicts.

For those that are already using git to work in groups, you can still share code with your partner by having multiple masters to your local repository (one being this original repository and the other some remote service like github where you host the code you are working on); [here's](http://stackoverflow.com/questions/849308/pull-push-from-multiple-remote-locations) a reference with more information.

* Test sets: [yosemite](http://www.cs.cornell.edu/courses/cs4670/2015sp/projects/pa3/yosemite_test.zip), [campus](http://www.cs.cornell.edu/courses/cs4670/2015sp/projects/pa3/campus.zip)

**Getting Things to Run**

**Skeleton program**

You can run the skeleton program by running,

>> python autostitchUI.py

**Panorama Mosaic Stitching**

You will use the feature detection and matching component  to combine a series of photographs into a 360 degree panorama. Your software will automatically align the photographs (determine their overlap and relative positions) and then blend the resulting photos into a single seamless panorama. You will then be able to view the resulting panorama inside an interactive Web viewer. To start this component, you will be supplied with some test images and skeleton code.

**Taking the Pictures**

1. Take a series of images with a digital camera mounted on a tripod or a handheld camera. For best results, overlap each image by 50% with the previous one, and keep the camera level. You can use your own camera for this or get one from Mann Library. Some cameras have a "stitch assist" mode you can use to overlap your images correctly, which only works in regular landscape mode.  In order to use your camera, you have to estimate the focal length.  The simplest way to do this is through the EXIF tags of the images, as described [here](http://www.cs.cornell.edu/courses/cs4670/2015sp/projects/pa3/focal.html).  Alternatively, you can use a [camera calibration toolkit](http://www.vision.caltech.edu/bouguetj/calib_doc/) to get more precise focal length and radial distortion coefficients.  Finally, Brett Allen describes one creative way to measure rough focal length using just a [book and a box](http://www.cs.washington.edu/education/courses/cse590ss/01wi/projects/project1/students/allen/index.html).
2. Make sure the images are right side up (rotate the images by 90 degree if you took them in landscape mode), and reduce them to a more workable size (480x640 recommended). You can use external software such as PhotoShop or the Microsoft Photo Editor to do this. Or you may want to set the camera to 640x480 resolution from the start.

**ToDo**

1. Warp each image into spherical coordinates. (file: *warp.py,*routine: *computeSphericalWarpMappings)*

[**TODO 1**] Compute the inverse map to warp the image by filling in the skeleton code in the *computeSphericalWarpMappings* routine to:

* 1. convert the given spherical image coordinate into the corresponding planar image coordinate using the coordinate transformation equation from the lecture notes
  2. apply radial distortion using the equation from the lecture notes

(Note: You will have to use the focal length *f* estimates for the half-resolution images provided above (you can either take pictures and save them in small files or save them in large files and reduce them afterwards) .**If you use a different image size, do remember to scale *f* according to the image size.)**

(Note 2: This step is not used when estimating homographies between images, only translations.)

1. Compute the alignment of the images in pairs. (file: *alignment.py*, routines: *alignPair*,*getInliers*, *computeHomography*, and *leastSquaresFit*)

[**TODO 2, 3**] computeHomography takes two feature sets from image 1 and image 2, f1 and f2 and a list of feature matches matches and estimates a homography from image 1 to image 2.

(Note 3: In computeHomography, you will compute the best-fit homography using the Singular Value Decomposition. From lecture 11: "the solution h is the eigenvector of A'A with smallest eigenvalue. " Recall that the SVD decomposes a matrix by A=USV' where U and V are the left and right singular vectors, and S is a diagonal matrix of singular values, conventionally ordered from largest to smallest. Furthermore, there is a very strong connection between singular vectors and eigenvectors. Consider: A'A = (VSU')(USV') = V(S^2)V'. That is, right singular vectors of A are eigenvectors of A'A, and eigenvalues of A'A are the squares of singular vectors of A. Returning to the problem, this means that the solution h is the right singular vector corresponding to the smallest singular value. For more details, the [wikipedia article](http://en.wikipedia.org/wiki/Singular_value_decomposition) on the svd is very good.)

[**TODO 4**] AlignPair takes two feature sets, f1 and f2, the list of feature matches obtained from the feature detecting and matching component (described in the first part of the project),  a *motion model*, m (described below), and estimates and inter-image transform matrix M.   For this project, the enum *MotionModel* takes two possible values: eTranslate and eHomography. AlignPair uses RANSAC (RAndom SAmpling Consensus) to pull out a minimal set of feature matches (one match for the case of translations, four for homographies), estimates the corresponding motion (alignment) and then invokes getInliers to get the indices of feature matches (indexing into matches) that agree with the current motion estimate.   After repeated trials, the motion estimate with the largest number of inliers is used to compute a least squares estimate for the motion, which is then returned in the motion estimate *M.*

[**TODO 5**] getInliers computes the indices of the matches that have a Euclidean distance below RANSACthresh given features f1 and f2 from image 1 and image 2 and an inter-image tranformation matrix from image 1 to image 2.

[**TODO 6, 7**] LeastSquaresFit computes a least squares estimate for the translation or homograpy using all of the matches previously estimated as inliers.  It returns the resulting translation or homography output transform *M*.

1. Stitch and crop the resulting aligned images. (file: *blend.py*, routines: *imageBoundingBox, blendImages, accumulateBlend, normalizeBlend*)

[**TODO 8**] Given an image and a homography, figure out the box bounding the image after applying the homography.(*imageBoundingBox.*)

[**TODO 9**] Given the warped images and their relative displacements, figure out how large the final stitched image will be and their absolute displacements in the panorama.(*blendImages.*)

[**TODO 10**] Then, resample each image to its final location (you will need to use inverse mapping here) and blend it with its neighbors. Try a simple feathering function as your weighting function (see mosaics lecture slide on "feathering") (this is a simple 1-D version of the distance map described in [[Szeliski & Shum](http://www.cs.cornell.edu/courses/cs4670/2015sp/projects/pa3/szeliskiShum97.pdf)]).  For extra credit, you can try other blending functions or figure out some way to compensate for exposure differences. (*accumulateBlend.*) Remember to set the alpha channel of the resulting panorama to opaque! (*normalizeBlend.*)

[**TODO 11**] Crop the resulting image to make the left and right edges seam perfectly. The horizontal extent can be computed in the previous blending routine since the first image occurs at both the left and right end of the stitched sequence (draw the "cut" line halfway through this image).  Use a linear warp to the mosaic to remove any vertical "drift" between the first and last image.  This warp, of the form y' = y + ax, should transform the y coordinates of the mosaic such that the first image has the same y-coordinate on both the left and right end.  Calculate the value of 'a' needed to perform this transformation. (*blendImages*)

Summary of potentially useful functions (you do not have to use any of these):

* np.divide, np.eye, np.ndarray, np.dot

**Using the GUI**

The skeleton code that we provide comes with a graphical interface, with the module autostichUI.py, which makes it easy for you to do the following:

1. **Visualize a Homography:** The first tab in the UI provides you a way to load an image and apply an arbitrary homograhy to the image. This can be useful while debugging when, for example, you want to visualize the results of both manually and programmatically generated transformation matrices.
2. **Visualize Spherical Warping:** The second tab on the UI lets you spherically warp an image with a given focal length.
3. **Align Images:** The third tab lets you select two images with overlap and uses RANSAC to compute a homography or translation (selectable) that maps the right image onto the left image.
4. **Generating a Panorama:** The last tab in the UI lets you generate a panorama. To be able to create a panorama, you need to have a folder with images labelled in such an order that sorting them alphabetically gives you the order the images appear on the panorama from left to right (or from right to left). This ensures that the mappings between all neighboring pairs are computed.

**Debugging Guidelines**

You can use the GUI visualizations to check whether your program is running correctly.

1. Testing the warping routines:
   * In the campus test set, the camera parameters used for these examples are
     + f = 595
     + k1 = -0.15
     + k2 = 0.00
   * In the yosemite test set, a few example warped images are provided for test purposes. The camera parameters used for these examples are
     + f = 678
     + k1 = -0.21
     + k2 = 0.26

See if your program produces the same output.

1. Testing the alignment routines:
   * To test alignPair, load two images in the alignment tab of the GUI. Clicking 'Align Images', displays a pair, the left and right images, with the right image transformed according to the inter-image transformation matrix and overlayed over the left image. This enables visually analyzing the accuracy of the transformation matrix. Note that blending is not performed at this stage.
2. Testing the blending routines:
   * An example panorama is included in the yosemite test set. Compare the resulting panorama with this image.

**Extra Credit**

Here is a list of suggestions for extending the program for extra credit. You are encouraged to come up with your own extensions. We're always interested in seeing new, unanticipated ways to use this program! Please use the --extra-credit flag in autostitchUI.py. You will need to use args in line 511 and modify the code as necessary. If we run your program without the flag, it must perform the basic implementation.

* Sometimes, there exists exposure difference between images, which results in brightness fluctuation in the final mosaic. Try to get rid of this artifact.
* Try shooting a sequence with some objects moving.  What did you do to remove "ghosted" versions of the objects?
* Try a sequence in which the same person appears multiple times, as in [this example](http://www.cs.washington.edu/education/courses/cse590ss/01wi/projects/project1/students/dougz/siggraph-hires.html).
* Implement a better blending technique, e.g., [pyramid blending](http://www.cs.princeton.edu/courses/archive/fall05/cos429/papers/burt_adelson.pdf), [poisson imaging blending](http://www.cs.jhu.edu/~misha/Fall07/Papers/Perez03.pdf) and [graph cuts](http://www.cc.gatech.edu/cpl/projects/graphcuttextures/)

**What to Turn In**

First, your source code should be zipped up into an archive called 'code.zip', and uploaded to [CMS](https://cms.csuglab.cornell.edu/). . In addition, turn in a panorama as JPG as your artifact. In particular, turn in a panorama from a hand-held sequence. This panorama can be either translation-aligned (360 or not), or aligned with homographies (your choice).

**Panorama Links**

* [Panoramas.dk](http://www.panoramas.dk/): weekly archive of full-screen, high-quality panoramas worldwide
* Super high resolution panoramas at [GigaPan](http://gigapan.com/)
* [VR Seattle](http://www.vrseattle.com/): Seattle & Washington panorama
* Matt Brown's Autostitch [page](http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html).

**Acknowledgments**

The instructor is extremely thankful to [Prof. Steve Seitz](http://www.cs.washington.edu/homes/seitz/) for allowing us to use this project which was developed in his Computer Vision class.

*Last modified on March 10th, 2015*