CS 5670: Computer Vision, Spring 2020
Project 3: Autostitch

Brief

- Assigned: Monday, March 9
- **Code Due: Monday, March 27 by 11:59 pm** (turn-in via CMS)
- **Artifact Due**: Wednesday, March 27 by 11:59 pm (turn-in via CMS)
- Teams: **This assignment should be done in teams of 2 students.**
- Github Link (Skeleton Code)

Synopsis

In this project, you will implement a system to combine a series of horizontally overlapping photographs into a single panoramic image. We give the ORB feature detector and descriptor. You will use ORB to 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 `AutostitchUI` 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

We also post slides to better visualize some steps of the algorithm. These can be found here.

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.

ToDo

Do not modify the code outside the TODO blocks.

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:

a. convert the given spherical image coordinate into the corresponding planar image coordinate using the coordinate transformation equation from the lecture notes
b. apply radial distortion using the equation from the lecture notes

(Note: You will have to use the focal length *f* estimates for the images provided below. **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.)

2. Compute the alignment of image 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. Let us denote transpose of A as A'. From lecture 11: "For equation Ah = 0, the solution h is the eigenvector of A'A with the smallest eigenvalue." Recall that the SVD decomposes a matrix by A = USV' where U and V are unitary matrices and their column vectors 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', since U is unitary. That is, right singular vectors of A are eigenvectors of A'A, and eigenvalues of A'A are the squares of the singular values of A. Returning to the problem, this means that the solution h is the right singular vector corresponding to the smallest singular value of A. For more details, the Wikipedia article 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) as parameters. Then it estimates and returns the inter-image transform matrix M. For this project, the enum *MotionModel* may have two possible values: `eTranslate` and `eHomography`. `AlignPair` uses RANSAC (RAndom SAmpling Consensus) to estimate M. First, it randomly pulls out a minimal set of feature matches (one match for the case of translations, four for homographies), estimates the corresponding motion (alignment)(use `computeHomography` in case of homographies) 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(found using `getInliers`) is used to compute a least squares estimate(found using `LeastSquaresFit`) 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 transformation matrix from image 1 to image 2.

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

3. 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(use `imageBoundingBox`) will be and their absolute displacements in the panorama.(`getAccSize`)

[**TODO 10**] Then, resample each image to its final location (you will need to use inverse warping 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]). For extra credit, you can try other blending functions or figure out some way to compensate for exposure differences. (`accumulateBlend`)

[**Additional hints**] 1) When working with homogeneous coordinates, don't forget to normalize when converting them back to Cartesian coordinates. 2) Watch out for **black pixels** in the source image when inverse warping. You don't want to include them in the accumulation. 3) When doing inverse warping, use linear interpolation for the source image pixels. 4) First try to work out the code by looping over each pixel. Later you can optimize your code using array instructions and numpy tricks (numpy.meshgrid, cv2.remap). You are not required to do this optimization.

[**TODO 11**] Normalize the image with the accumulated weight channel. Pay attention not to divide by zero. Remember to set the alpha channel of the resulting panorama to opaque! (`normalizeBlend`)

[**TODO 12**] In case of 360 degree panoramas, make the left and right edges have perfect seams. 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 necessarily have to use any of these):`

- `np.divide, np.eye, np.ndarray, np.dot, np.linalg.svd`
- `Note: you are allowed to use cv2.warpPerspective in this project.`

Extra Credit

Please see Extra Credit to know where to implement this functionality.

Using the GUI

`You can run the skeleton program by running,`

```
>> python gui.py
```

The skeleton code that we provide comes with a graphical interface, with the module `gui.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 homography 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. Note that you do` **`not`** `need to implement any functions to run this.`
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 labeled 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. Our current code assumes that all images in the panorama have the same width!`

Debugging Guidelines

The `test.py` provides very basic testcases for TODO 1-9. You need to test TODO 10-12 and any extra credit parts with your own testcases.

The supplied tests are very simple and are meant as an aid to help you get started. Passing the supplied test cases does not mean the graded test cases will be passed.

You can also 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. Note that if you use these images with the translation motion model, you might get a bit blurry panoramas in the blending region (as you can also see from the panorama given by us).

2. Testing the alignment routines:

   - Note that the campus images are only suitable for the translational motion model! The yosemite images are suitable for both motion models. 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 overlaid over the left image. This enables visually analyzing the accuracy of the transformation matrix. Note that blending is not performed at this stage.

3. Testing the blending routines:

   - An example panorama is included in the yosemite and the campus test set. Compare the resulting panorama with these images. Note that it's important to use the specified `f, k1, k2` parameters to get the same image. Use 360 degree panorama to get the same result for campus dataset!

4. Additional notes:

   - If you use high resolution images when creating you own panorama, you might run into memory problems. Increasing the allowed memory of the VM might solve these issues.

What to Turn In

First, your source code should be zipped up into an archive called 'code.zip', and uploaded to CMS. 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).

Taking Pictures

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. 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. Alternatively, you can use a camera calibration toolkit to get more precise focal length and radial distortion coefficients.

Downloads

Skeleton code: Available through Github

Test sets: Look inside the resources subdirectory. You will find three datasets: yosemite, campus and melbourne.

Use Python3 for this project.

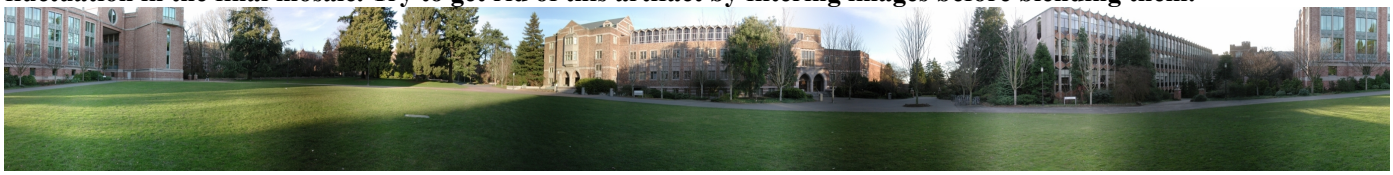
cs5670_python_env: Tutorial on how to set up cs5670_python_env

Virtual machine: As an alternative to cs5670_python_env, the class virtual machine available here has the necessary packages installed to run the project code.

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 gui.py. You will need to use args in line 550 and modify the code as necessary. If we run your program without the flag, it must perform the basic implementation. Here is a list of suggestions for extending the program for extra credit (The following example panoramas are from this page):

- **Remove brightness artifact. Sometimes, there exists exposure difference between images, which results in brightness fluctuation in the final mosaic. Try to get rid of this artifact by filtering images before blending them.**



- **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.**

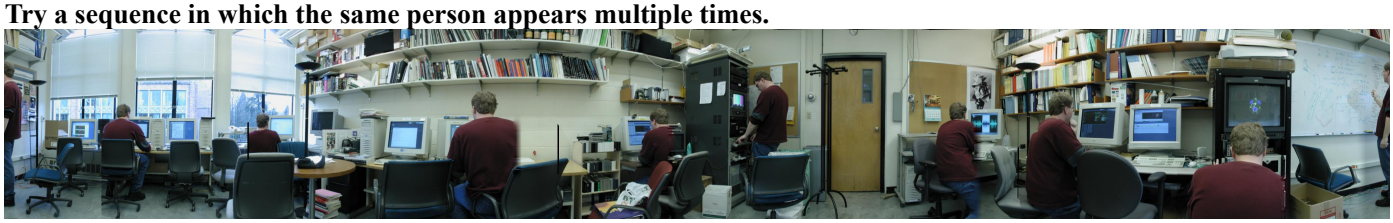


Photo credit: Doug Zongker

- **Implement a better blending technique, e.g., pyramid blending, poisson imaging blending and graph cuts.**

Panorama Links

- **Panoramas.dk: weekly archive of full-screen, high-quality panoramas worldwide**
- **Super high resolution panoramas at GigaPan**
- **VR Seattle: Seattle & Washington panorama**
- **Matt Brown's Autostitch page.**

FAQ

Click here

Acknowledgments

The instructor is extremely thankful to Prof. Steve Seitz for allowing us to use this project which was developed in his Computer Vision class.
