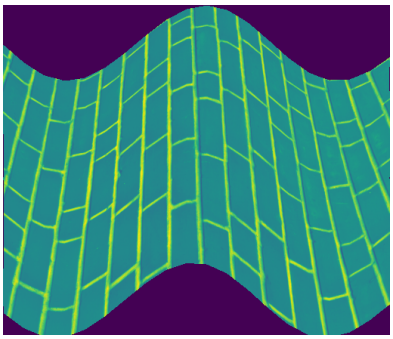
CS 763/CS 764: Lab 03

Take-home Midsem

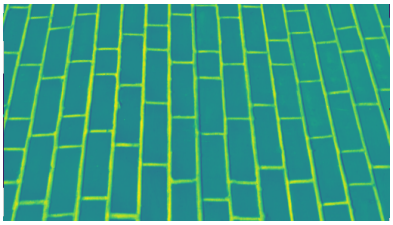
- Announced: 24 Feb. Due 2 March 4PM
- **This is a group assignment**

1 Piecewise Affine transformation

Different parts of an image can undergo different affine transformations. An example appears in Figure 1.



(a) Distorted image



(b) Original Image

Figure 1: Piecewise affine transformation

Image a) was created from image b) using the following steps

- Let us consider the horizontal direction to be the x-axis and the vertical (portrait) direction to be the y-axis.
- The wave seen in figure a) is a sine wave function ($y_{\text{distorted}} = y_{\text{original}} + A\sin(x_{\text{original}} + \phi)$).
- Figure b) was divided into 20 parts along the x-axis after which each part underwent an affine transformation so that we get the image a).

You have to write `piece-affine-trans.py` which reverts back image a) to image b). [Hint: Try to find out the amplitude and number of cycles completed first to get the related function. You might have to use external libraries like `skimage` (for this part of the assignment only).]

Run example :

```
$ python3 piece-affine-trans.py
```

Note: You might get black spaces above and below the image after transformation (depends on how you did the transformation) which must not appear in the final output. Your aspect ratio might not be the same, which need not be corrected.

Question : How to select the grid-point correspondences for this problem? Explain the process that you followed, even if it is elementary.

2 Panoramic Image Mosaicing

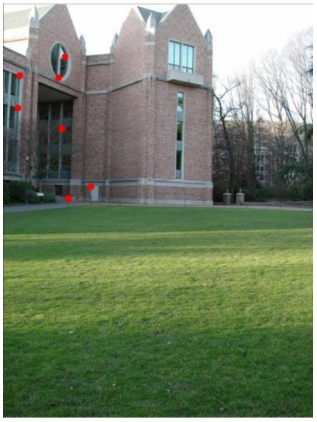
In the above question, you used different piecewise transformations for different parts of the input. Then you obtained the output after mosaicing the results independently.

In this question, image transformations and mosaicing are not independent. Okay, lets start!

Note: In this entire section, you are not to use use external classes such `scikit`. Further, you are not to use the `Stitcher` class from `OpenCV` library. Thus, in addition to `OpenCV` library, you can only use these packages: `numpy` for calculations, `sys` for processing command line arguments and `os` for reading files from a directory. We will be working in a `virtualenv` so if you import any class other than these, the program will straightway crash.

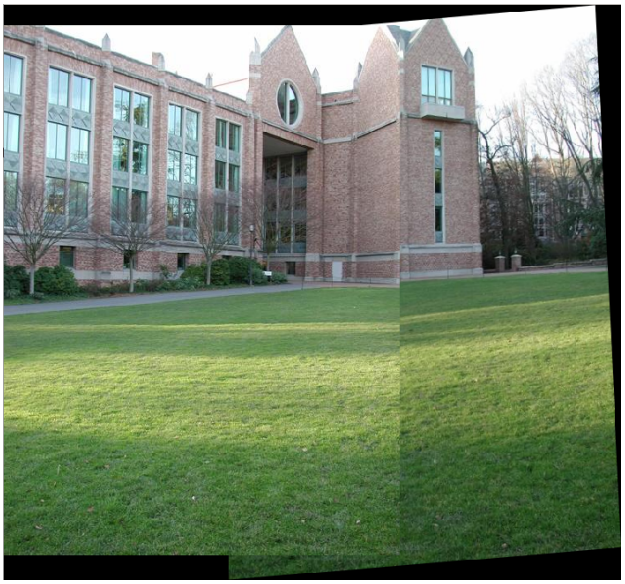
2.1 Manual Mosaicing

This subtask will help you visualize what you have to do eventually:



(a) Input image 1

(b) Input image 2



(c) Result

Figure 2: Manual mosaicing

As you can see, the red points in the figure 2a and 2b, are the correspondences manually chosen by the user.

Write code to perform the following

1. Obtain point-correspondences between the input images as the input to be given by user. Put the image windows side by side and take inputs by the different mouse clicks. (**Hint:** Look up

`cv2.setMouseCallback()`, `cv2.moveWindow()`, `cv2.resizeWindow()`.)

2. Take image-2a as a reference image, **I1** and image-2b be **I2**. Find the transformation matrix that transforms **I2** to **I1** by using the correspondences found in the above step.

3. Apply the transformation found above to **I2** and mosiac both images. (Again, for uniformity, take the transformation from I2 to I1 i.e. how does I2 appear when seen from I1).

4. Create your own dataset of 2 images.

5. Show the results for the given dataset as well as on your dataset.

Run example :

```
$ python3 pano-manual.py path-to-directory-containing-2-images
e.g. python3 pano-manual.py ../data/manual/campus/
```

Q.2.1: How many point-correspondences did you choose? Which `OpenCV` library function did you use to find the transformation in (2)? and why?

Q.2.2 Explain in the reflection essay how many did datasets you make, and why you chose the one(s) you did.

Q.2.3:

i. Consider a case in which we keep **I2** as the reference image. List the ways or properties in which output after mosaicing will change and, the ways in which it will remain the same.

ii. Consider two **1-D** images **J1** and **J2** of size 100 pixels each. Both share some common points (correspondences). **J1** has correspondences in range of `[10,20]` and **J2** has correspondences in the range of `[85,95]`. Will the size of the output image (after mosaicing **J1** and **J1**) change if we keep **J1** as reference image *vs* if we keep **J2** as the reference image. Explain.

2.2 Auto mosaicing

You selected the point correspondences manually in the previous question. Here, the goal is to automate the process.

1. Use a feature detector (ORB) to find out the point-correspondences automatically. You may want to filter "bad" point-correspondences.

2. Once you have a set of point-correspondences, follow the same procedure as before to mosaic the images.

3. Show the results for the given dataset

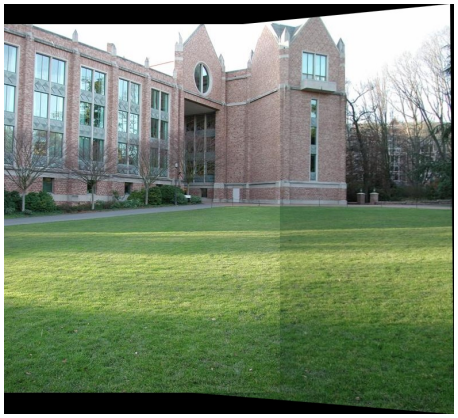
4. Generate your own dataset (of 2 images).

Run example :

```
$ python3 pano-auto.py path-to-directory-containing-2-images
e.g. python3 pano-auto.py ../data/auto/campus/
```

(a) Input image 1

(b) Input image 2



(c) Result

Figure 3: Auto mosaicing

Q.2.2.1: Did you filter? If yes, then how did you filter? how many point correspondences did you find before and after filtering?

Q.2.2.2: How did you choose the values for parameters in the process of finding 'good' point-correspondences?

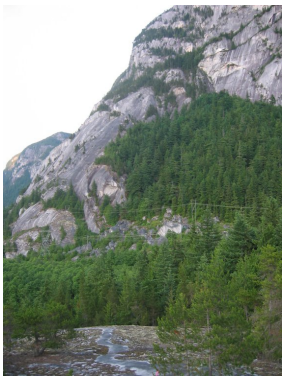
Q.2.2.3: Did the dataset you used earlier work for the manual case work as well? Explain. (Clearly the dataset in this question will work for the earlier manual version, but we are interested in the converse).

2.3 Let's generalize ($3 \leq n \leq 5$ input images)

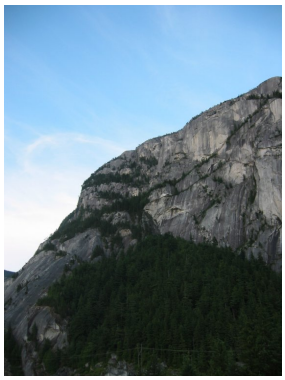
Intention is to create a routine that takes two inputs: 'n' input images from a directory and a reference image. It should return the result after mosaicing all 'n' images such that reference image in the result is intact. Refer figure 4 carefully to understand better.

Show the results for the given datasets as well as on your own datasets (of 5 images). Mention the results when your code is working and when it is not working for some reference images or some datasets (if that's the case). (Why do we say this?)

Note: Assume that images present in the directory are named according to their sequence number. In other words, image names gives you the order of mosaicing.



(a) Input image 1



(b) Input image 2

(c) Input image 3

(d) Input image 4

(e) Input image 5

(f) Result

Figure 4: Example 1: (a), (b), (c), (d), (e) are the input images. Output (f) is obtained, when (c) is the reference image.

The routine should look similar to this:

```
def function_name(images, referenceImage):
    .
    .
    return result
```

Run example :

```
$ python3 pano-general.py path-to-directory-of-images index-to-reference-image
e.g. python3 pano-general.py ../data/general/mountain/ 3
```

Q.2.3.1: Explain the method you followed. Do you get the result if you choose any image to be the reference image? If no, what issues are you facing? Give your opinion on reason behind these issues.

Q.2.3.2: Here, we assumed that you know the sequence in which you have to stitch the images. How would you modify your approach if you don't know the sequence? Describe the approach briefly.

Q.2.3.3: How does the functionality you are providing (or could provide) differ from the `Stitcher` class.

Q.2.3.4: The mosaicing results show a "seam". What techniques can be used to remove the seam?

3 Submission Guidelines

1. The top assignment directory should include the lab submission as detailed below.

   (a) Do include a `readme.txt` (telling me whatever you want to tell me including any external help that you may have taken). Don't forget to include your honor code here (or your name and roll number). All members of a group are expected to (electronically) sign the honor code and the percentages (see below). This is a text file, not `pdf`, not `docx`.

   The `readme.txt` will contain individual contributions of each of the team members. If the assignment is finally worth 80 marks as graded by the TA, then a contribution of the form `80, 100, 60` (in sorted order of roll numbers) will result in (respectively) marks `64, 80, 48`. Do this for each question separately. A person claiming 100% is basically saying that (s)he can reproduce the entire assignment without the help of the other team members.

   (b) `ReflectionEssay.pdf`: Should contain the explanation for all the questions implicitly and explicitly raised. Provide an output of a sample run. Explain what you learnt in this assignment, and how this assignment improved your understanding. What will someone who reads this gain? Can this be a blog post, which if read end-to-end someone not in your class (but in your batch) will understand?

   (c) A directory called `code` which contains all source files, and only source files (no output junk files). The mapping of code file to questions should be obvious and canonical.

   (d) A directory called `results` to store output that needs to be saved (this will typically be explicitly stated in the question).

   (e) A directory called `data` on similar lines to code, whenever relevant. Note that your code should read your data in a relative manner.

   (f) Source files, and only source files (no output junk files). Mac users please don't include junk files such as `.DS_store`. We are not using MacOS.

   (g) Create a directory called `convincingDirectory` which contains anything else you want to share to convince the grader that you have solved the problem. We don't promise to look at this (especially if the code passes the tests) but who knows? This is your chance.

2. Once you have completed all the questions and are ready to make a submission, prepend the roll numbers of all members in your group to the top assignment directory name and create a submission folder that looks like (for group but you get the idea) this
   `130010009_140076001_150050001_lab0X_description.tgz`

   Please stick to `.tar.gz`. Do not use `.zip`. Do not use `.rar`

3. Your lab submission folder should look something like this:

```
130010009_140076001_150050001_lab03_pano/
├── ReflectionEssay.pdf
├── code
│   ├── piece-affine-trans.py
│   ├── pano-manual.py
│   ├── pano-auto.py
│   └── pano-general.py
├── data
│   ├── piece
│   ├── manual
│   ├── auto
│   └── general
├── results
│   ├── piece-affine-results
│   ├── pano-manual-results
│   ├── pano-auto-results
│   └── pano-general-results
├── convincingDirectory
└── readme.txt
```

4. Submission. Very very important.

   (a) Submit on **Moodle** at the course **CS-763**.

   (b) The lexicographic smallest roll number in the group should submit the entire payload (with all the technical stuff).

   (c) All other roll numbers submit only `readme.txt` as discussed above.