CS 763/CS 764: Task 05: Picture in picture

- Announced: 28 Feb. Due: Mar 09 10PM
- Please write (only if true) the honor code. You can find the honor code on the web page. If you used any source (person or thing) explicitly state it.
- **This is NOT an individual assignment**

1 Overview

In Task04, you would have experimented with `SIFT` for keypoint detection and feature extraction. In this task, using what you have learnt in Task04, we will solve the image mosaicing problem which is essentially how the panorama mode in your mobile camera works

Note that when not explicitly specified, you are not to use the built in support of mosaicing in OpenCV.

2 Manual Mosaicing

This subtask will help you visualize what you have to do eventually. As you can see, the red points in the figure 1a and 1b, 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-1a as the source image, **I1** and image-1b as the target image, **I2**. Find the transformation matrix that transforms **I1** to the perspective of **I2** by using the correspondences found in the above step. By 'source' and 'destination', we mean that, in this case, we want to see how I1 appears when seen from the viewpoint of I2). See question below on normalization.
3. Apply the transformation found above to **I1** and mosaic both the images.
4. Create your own dataset of 2 images and repeat.
5. Show the results for the given dataset as well as on your dataset.

(a) Input image 1 (b) Input image 2 (c) Result

Figure 1: Manual mosaicing

Note: In Fig.1c note that there are no black pixels around the borderYou may want to write a function `cropify` with this functionality which will be needed in subsequent tasks too.

Run example :

```
python3 manual.py path-to-directory-containing-2-images
-normalize 1

e.g. python3 manual.py ../data/manual/campus/  -normalize 0
#No normalization
```

Questions Points in the source (`srcPoints`) and those in the destination (`dstPoints`) contain the integer valued coordinates (x, y) which will lie in the range $[0, m-1] \times [0, n-1]$ where the shape of corresponding image is $m \times n$. The general practice is to first normalize the `srcPoints` and, `dstPoints` separately . The transformation is an affine transformation such that the transformed point set becomes centered at the origin (usually the centroid), and scaled. The scaling is done so that the resulting variances s_x^2 and s_y^2 is unity. Next, the homography H_n obtained on the normalized points is used to get the required homography H.

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

Q.3 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 source image *vs* if we keep **J2** as the source image. Explain.

Q.4 Explain your choice of the affine transform.

Q.5 Explain the relationship between H and H_n.

Q.6 Design an error metric and report the percentage error while using normalization and not using normalization.

3 Auto mosaicing

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

3.1 RanSaC subroutine

As a subroutine in mosaicing, you will need to implement the RanSaC algorithm in a file `ransac.py` which will be called by the subsequent tasks.

The routine should look similar to this:

```
def ransac(srcPoints, dstPoints, threshold=1.0):
    .
    .
    return result
```

In this case, the RanSaC algorithm is used to filter out the potential outliers from a set by **Ran**domly **Sa**mpling a **C**onsensus set. Below are the directions for implementing the function. Your main goal is to ensure that you follow the principles mentioned in the class lecture, esp. not hardcoding parameters.

1. To obtain the values for parameter **T** and **N** in the algorithm, you may first assume that the percentage of outliers to be 50%. This will fetch you 80% marks for this step. To get full credits, use the adaptive RanSaC in your implementation as discussed in the slides. For calculating **N**, take the value of **p** to be .99. The `threshold` parameter is assumed to be the standard deviation of measurement errors in the call above.

2. On the termination of the `ransac`, the return `result` is not intended to hold the desired homography but simply the presumed inliers. Use `findHomography()` external to the `ransac` call.

Questions Assume `threshold` is fixed to 1 in the following questions, and we run the non-adaptive ("50%") version. Answer the following questions with respect to the `campus` dataset provided.

Q.1 Explain your value of N.

Q.2 Explain your value of T. What is the percentage of times this parameter was used?

Q.3 Explain your value of t and how you used this for checking whether a sample is an inlier or not.

Q.4 What was the size of the largest consensus set as a function of the inliers?

3.2 Mosaicing Implementation

1. Use a feature detector to find the point-correspondences automatically.

2. Once you have a set of point-correspondences, you would like to remove the outliers that may be present in the filtered point-correspondences. Perform the following to remove the outliers (make sure that you have outliers, at least for your own dataset!)

   (a) Use your custom RanSaC code and tinker with `threshold` to get acceptable results.

   (b) Now compare your results with the `OpenCV` version of RanSaC in `findHomography` function.

3. Show/Save the results for the given dataset

4. Compare the previous results with that obtained using `Stitcher` api in `OpenCV`.

5. Generate your own dataset (of 2 images) and repeat the above steps.

Run example :

```
python3 auto.py path-to-directory-containing-2-images -mode
custom-ransac  -normalize 1# your ransac implementation
python3 auto.py path-to-directory-containing-2-images -mode
auto-ransac -normalize 1
# opencv's ransac
python3 stitcher.py path-to-directory-containing-2-images
# defaults
```

Questions Answers these questions by comparing the custom version and the built in version (except the last question which relates to `stitcher`.

Q.1 Report the number of feature points used for `campus` and the percentage of pruning due to your method

Q.2 Consider the personalized ("group") dataset you used earlier for the manual case where you handtweaked the correspondences. Does auto-mosaic work with this dataset? Can you think of a dataset that breaks auto-mosaic but will (obviously) work with the manual case.

4 Let's generalize

This section is intended to be the challenge section (see the questions at the end).

At a basic level, the intention is to create a routine that takes two inputs: n $(3 \leq n \leq 5)$ 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 Fig. 2 carefully to understand better.

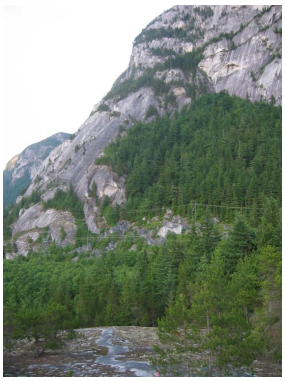
Your code should look similar to this:

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

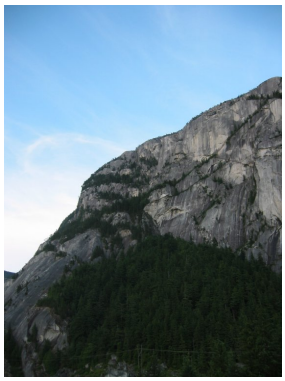
Assume that images present in the directory are named according to their sequence number. In other words, image names gives you the order of mosaicing.

Run example :

```
python3 general.py --path images --idx index --mode custom--ransac --normalize 1
```



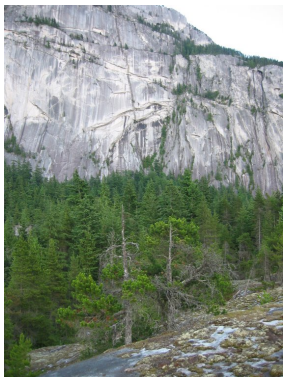
(a) Input image 1



(b) Input image 2

(c) Input image 3

(d) Input image 4



(e) Input image 5

(f) Result

Figure 2: Example 1: (a), (b), (c), (d), (e) are the input images. Output (f) is obtained, when (c) is the reference image.

```
python3 general.py -path ../data/general/mountain/ -idx 3 -mode auto-ransac
-normalize 0
```

Questions

Q.1 Explain the method you followed in accomplishing this task.

Q.2 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).

Q.3 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.4 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.5 How does the functionality you are providing (or could provide) differ from the `Stitcher` API.

Q.6 List items that you think need fixing. For example, the mosaicing results show a "seam". What techniques can be used to remove the seam?

5 Submission Guidelines

Submission guidelines generally remain the same. However there are some parts that can change, so be flexible. Your lab submission folder should look something like this:

```
130010009_140076001_150050001_Task05
├── ReflectionEssay.pdf
├── code
│   ├── manual.py
│   ├── ransac.py
│   ├── crop.py
│   ├── auto.py
│   ├── general.py
│   └── stitcher.py
├── data
├── results
├── answers.pdf
├── convincingDirectory
└── readme.txt
```