# Assignment 07

*Please refer to MyCourses/Gradescope for assignment deadlines.*

The goal of this assignment is to give you hands-on practice with the following concepts:

- Laplacian pyramids
- SIFT features
- Feature matching
- The Random Sample Consensus (RANSAC) algorithm

## Part 1 of 2

In this part, you will implement a Laplacian pyramid and use it to blend two images together. Recall that a Laplacian pyramid is built from a Gaussian pyramid, and a Gaussian pyramid is built using the `cv.pyrDown()` function.

**Note on sizes:** A k-level Gaussian pyramid divides an image in half `k-1` times. The first level `g[0]` is the original image, and each level `g[i]` is a downsampled version of the previous level `g[i-1]`. Each downsampling cuts the size of the previous level in half for both height and width. If your original image has height and width that is not divisible by $2^{k-1}$, you'll get shape errors! You therefore need to pad the image with enough zeros to make it divisible by $2^{k-1}$ *before* constructing the pyramid, and then crop the pyramid levels to the original size after blending. This is the job of the `pyramid_pad()` function - it adds zeros to the image to make its dimensions divisible by $2^{k-1}$. Make sure you call `pyramid_pad()` inside the `blend_images()` function before you start building the pyramid, and crop the result back down to size at the end.

**Example output:**

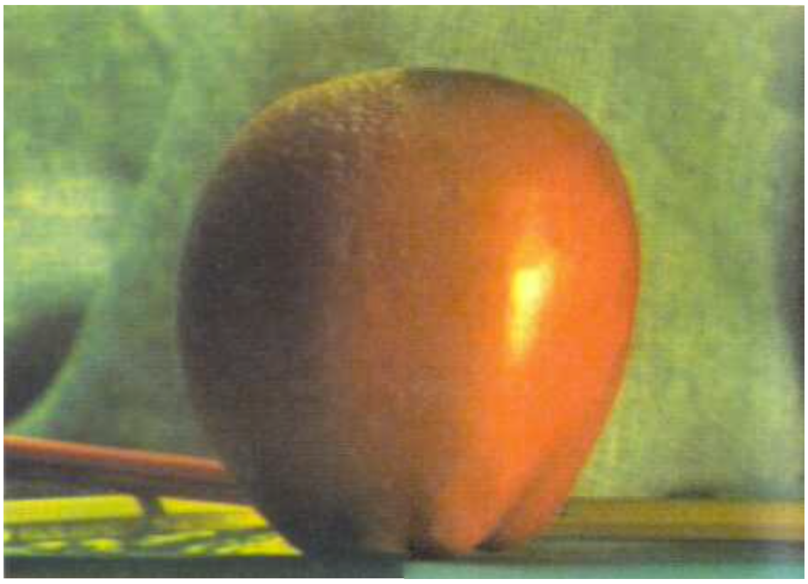

**Figure 1:** Result of `python pyramid_blending.py --image1=src/images/apple.png --image2=src/images/orange.png --mask=src/images/apple_orange_mask.png`

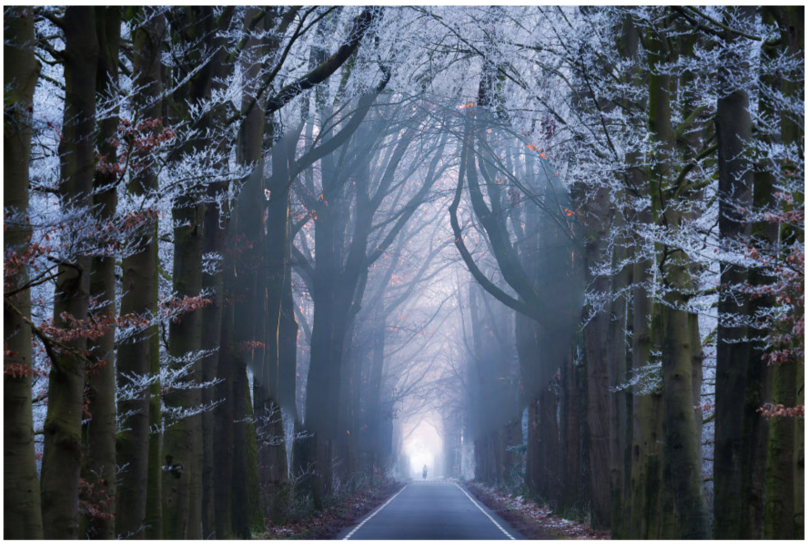

**Figure 2:** Result of `python pyramid_blending.py --image1=src/images/scene01.jpg --image2=src/images/scene02.jpg --mask=src/images/mask-circle.png`

## Part 2 of 2

Go watch **this 8-minute video** on feature-matching for panorama-stitching using the RANSAC algorithm. You may also want to watch **this 12-minute follow-up video** if you want to see some more details on how the underlying `warpPerspective()` function works using a backwards mapping and where the edge weighting comes from.

You've been provided with (a slightly modified) version of the answer key file for `panorama.py` from Assignment 2. Back then, you had to manually click on corresponding points in a pair of images to align them. We're now going to update the procedure using SIFT feature matching, which will also enable us to automatically align more than two images at once.

The main differences from Assignment 2 are:

- The `get_clicked_points_from_user()` function is gone
- All the buggy A2 code is now correct, so you don't need to touch any of the old functions.
- Three new functions have been added:
  - `get_matching_points()` - this function uses SIFT feature matching to find corresponding points between two images. It is (should be) bug-free and provided for you.
  - `my_find_homography()` - this function is essentially pure-numpy implementation of the `cv.findHomography()` function. It is (should be) bug-free and provided for you, since we will be asserting in test cases that `find_homography_ransac()` makes no calls to any `cv` functions.
  - `find_homography_ransac()` - this function is where you will implement the RANSAC algorithm to find the best homography between two sets of matching points. This is the only function you need to implement for this part of the assignment.

The main steps of the overall panorama stitching algorithm are:

1. Choose a 'reference' image. We'll let the user specify this by providing a list of image files and an index like `-r=1` such that the `images[r]` image is the reference.
2. For each `images[i]`, use SIFT feature matching to find a set of `xy` points whose descriptors match between `images[i]` and the reference image `images[r]`. This is provided for you in the function `get_matching_points()`. Importantly, the returned points are such that `points_ref[i]` is some `xy` point in the reference image and `points_q[i]` is the "best matching" `xy` point in the second "query" image, *but the "match" only takes into account the descriptors.* Thus, we still need another step of the algorithm to figure out which subset of the matching points are 'inliers' and which are 'outliers.' This is where RANSAC comes in.
3. For each pair of `(points_src, points_dst)` points, find the best homography that lines up the greatest number of corresponding points in the two images. This will return a homography

matrix that maps *from* the src coordinate system *to* the dst coordinate system. **You need to implement this part in the `find_homography_ransac()` function.** As in past assignments, this is something that would normally be done by a call to OpenCV functions, but you will be implementing RANSAC yourself using numpy and python. The pseudocode for this algorithm is as follows:

```
for N iterations:
    sample 4 random indices `idx` [0..n-1] and select points_src[idx] and po
    compute the homography H that maps between them using my_find_homography
    use H to map *all* points_src to points_dst
    compute the distance for all `n` pairs of points
    count the number of 'inliers' (i.e., points whose distance is less than
    if the inlier count is greater than the best inlier count so far:
        record the indices of all the inliers
recompute H using all the inliers from the best set of inliers
return H
```

4. Pass the resulting list of images and homographies into the stitching code from Assignment 2

**Example output:**

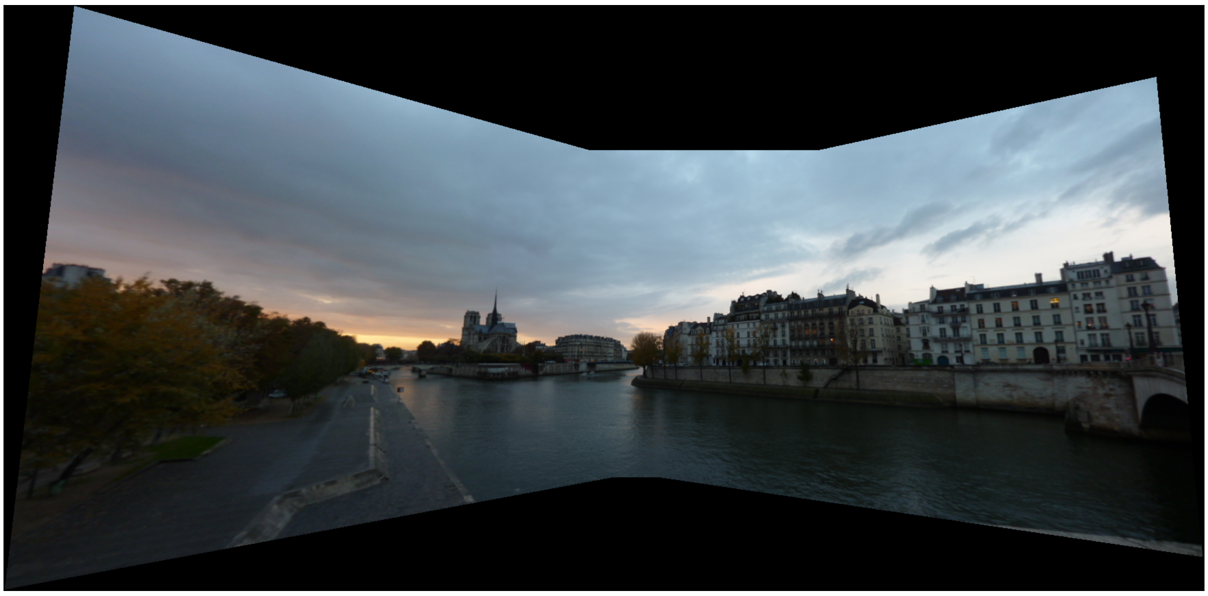

**Figure 3:** Result of `python panorama.py src/images/paris_a.jpg src/images/paris_b.jpg src/images/paris_c.jpg --reference-index=1`

## Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did in the `collaboration-disclosure.txt` file.

## Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?
- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Give your reflections in the `reflection.txt` file if you choose to give us feedback.

## Submitting

Use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline. (This is to prevent us from having to grade your work twice. Let us know so that we don't start grading files that will be replaced later).