Computer Vision Assignment 3

Submission Requirements

Format

1. Project materials including template: `proj3.zip`.
2. You should provide a report (pdf file) to describe your algorithm implementation.
3. Pack your `proj3_code` and `report.pdf` into a zip file, named with your student ID, like `[student ID].zip`.

Submission Way

1. Please submit your results to https://docs.qq.com/form/page/DQnRPbmtMaGNWWEdE.
2. The deadline is 23:59 on June 11, 2025. No submission after this deadline is acceptable.

About Plagiarize

DO NOT PLAGIARIZE! We have no tolerance for plagiarizing and will penalize it with giving zero score. You may refer to some others' materials, please make citations such that one can tell which part is actually yours.

Setup

Python Environment

1. Install Anaconda.
2. Download and extract the project starter code.
3. Create a conda environment using following command.

```
conda create -n cv_proj3 python=3.9
conda activate cv_proj3
pip install numpy opencv-python networkx
```

Task: Image Stitching

In this exercise, you will implement an image stitcher that uses image warping and homographies to automatically create an image mosaic.

This assignment consists of two parts. The first part is a warm-up: stitching two images to help you understand key operations like computing homographies and applying image warps. The second part is the main task: designing a robust image stitching pipeline that can handle an arbitrary number of images. You're encouraged to build reusable components in Task 1 and expand them into a full pipeline for Task 2. While Task 1 is not heavily graded, it's highly recommended as a checkpoint to validate your low-level implementations.

1. Two-Image Stitching (Warmup)

We first focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. The example is as follows:

Input Image 1

Input Image 2

Stitched Image

This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps. Here are the main components you will need to implement:

(1) Getting correspondences. Write code to automatically obtain interest points and descriptors, and write code to automatically identify matching descriptors in the two input images. You can use any of the interest point, descriptor, and matching code provided by **opencv (e.g., SIFT, SURF and ORB)** for this portion, or you may implement your own module for matching.

(2) Computing the homography parameters. Write a function that takes a set of corresponding image points and computes the associated 3×3 homography matrix H. This matrix transforms any point $\mathrm{p_i}$ in one view to its corresponding homogeneous coordinates in the second view, $\mathrm{p_i'}$, such that $\lambda \mathrm{p_i} = \mathrm{H p_i'}$. The function should take a list of $\mathrm{n} \geq 4$ pairs of corresponding points from the two views, where each point is specified with its 2d image coordinates. **(refer to findHomography function in opencv)**

(3) Warping between image planes. Write a function that can take the recovered homography matrix and an image, and return a new image that is the warp of the input image using H. Since the transformed coordinates will typically be sub-pixel values, you will need to sample the pixel values from nearby pixels. For color images, warp each RGB channel separately, then stack together to form the output. To avoid holes in the output, use an inverse warp. Warp the points from the source image into the reference frame of the destination, and compute the bounding box in that new reference frame. Then sample all points in that destination bounding box from the proper coordinates in the source image. Note that transforming all the points will generate an image of a different shape / dimensions than the original input. **(refer to warpPerspective function in opencv)**

(4) Create the output mosaic. Once we have the source image warped into the destination image's frame of reference, we can create a merged image showing the mosaic. Create a new image large enough to hold both (registered) views; overlay one view onto the other, **simply leaving it black wherever no data is available**.

2. Multi-Image Stitching (Main Task)

In Task 1, you have implemented basic image stitching for two images. That task gives you hands-on experience with manipulating homogeneous coordinates, computing homography matrices, and performing image warps. In this part, you will extend your implementation to stitch any number of images into a single mosaic. The example is as follows:

Input Image 1

Input Image 2

Input Image 2

Stitched Image

The naive approach to multi-image stitching is to read the images in order and stitch them sequentially—image 1 with image 2, then the result with image 3, and so on. While simple to implement, this method can be fragile: **two adjacent images in the list may not overlap, which may lead to alignment failures or distorted mosaics.**

Here, we provide one possible technical solution that improves robustness: You can **build an image matching graph**, where each node represents an image and edges represent valid pairwise homographies based on feature matching. Then, **choose a reference image** (e.g., the one with the most connections) as the stitching center. By performing a **depth-first search (DFS**) or similar traversal on the graph, you can **determine a stitching order** and compute global homographies for all images relative to the reference. Finally, **warp each image to a common canvas** and blend overlapping regions to form the final mosaic. This approach avoids relying on input order and adapts better to arbitrary image sets.

This is just one feasible strategy—you're encouraged to **design your own algorithm** for determining how images should be aligned and combined. Your solution should be able to handle an arbitrary number of input images and produce a coherent, visually pleasing result.

Evaluation

Project Structure

```
proj_code/
├── data/
│   ├── task1_pairwise/          # Task 1: Two-image stitching
│   │   ├── case1/
│   │   │   ├── 1.JPG
│   │   │   └── 2.JPG
│   │   └── case2/
│   │       ├── 1.JPG
│   │       └── 2.JPG
│   │
│   ├── task2_multiview/         # Task 2: Multi-image stitching
│   │   ├── case1/
│   │   │   ├── 1.JPG
│   │   │   ├── 2.JPG
│   │   │   ├── 3.JPG
│   │   │   └── ...
│   │   ├── case2/
│   │   │   ├── 1.JPG
│   │   │   ├── 2.JPG
│   │   │   └── ...
│   │   └── case3/
│   │       ├── 1.JPG
│   │       ├── 2.JPG
│   │       └── ...
│
├── output/                      # Your results are saved here.
│   ├── task1_pairwise/
│   │   ├── case1.JPG
│   │   └── case2.JPG
│   │
│   ├── task2_multiview/
│   │   ├── case1.JPG
│   │   ├── case2.JPG
│   │   └── case3.JPG
│
├── two_image_stitching.py
├── multi_image_stitching.py
```

Run

```
python two_image_stitiching.py
python multi_image_stitiching.py
```

Evaluation Criterion

The assignment will be evaluated based on the quality of the image stitching, including the accuracy of alignment and the naturalness of the final mosaic without visible artifacts; the originality and computational efficiency of the implemented algorithm; and the clarity, coherence, and conciseness of the submitted report.