CSCI 677: Advanced Computer Vision - Fall 2024

Instructor: Prof. Nevatia

Assignment 2

Due on October 1, 2024 before 15:00 PDT

1 Panorama Stitching

This is a programming assignment. You are asked to write a program that stitches multiple images into a panorama by computing homography between consecutive images, All required functions are available in OpenCV version >= 4.4.0. Please follow the instructions below to complete this process. You may find several online tutorials for this task; you are free to consult those but please follow the steps we have outlined below. Please do not use language tools such as ChatGPT to write code for you.

1.1 Data Preparation

We provided an example set of images here, but we encourage you to also take photos with your own camera or smartphone. Take two or more photos of the same scene, ensuring that there is enough overlap between consecutive photos for matching features and the viewpoint change is approximately that of rotation only.

1.2 Feature Detection

Load the images using `cv.imread()`. Convert them to grayscale images. Create a SIFT feature detector. Detect the keypoints on both images and display them with size and orientation using `cv.drawKeypoints()`. Here's an example of how you can use these functions:

```
img = cv.imread('home.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
sift = cv.SIFT_create()
kpts, dpts = sift.detectAndCompute(gray, None)
```

You can also follow the tutorial here.

1.3 Feature Matching

Create a brute force matcher with `cv.BFMatcher()`. Use `bf.knnMatch` to find matches among the descriptors you just detected on the two images. This function returns the top-k matches by filtering out weak matches according to the ratio between the best

and the second-best matches. Set `k=2` for the ratio test to filter matches, but you can experiment with other `k` values to achieve the best matching results. Display resulting matches between the two images using `cv.drawMatchesKnn()`. Here's an example code snippet:

```
bf = cv2.BFMatcher()
matches = bf.knnMatch(src_dpts, dst_dpts, k=2)
```

You can also follow the tutorial here.

1.4 Compute Homography

Using the matched features, compute the homography matrix for each pair of consecutive images with RANSAC. Print out the homography matrix. You can use `cv.findHomography()` for this. Example usage:

```
H, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
```

You can follow the tutorial here.

1.5 Stich into a Panorama

Before stitching the images to compose a panorama, you need to determine the size of the final stitched image. Since the panorama is larger than each individual image, we need to define a rectangle that covers all warped images. We provide the code snippet below for reference. In this code, `cv.perspectiveTransform()` is used to apply a perspective transformation to a set of points, allowing us to calculate the minimum and maximum coordinates (`min_x, min_y, max_x, max_y`) to define the size of the output stitched image.

```
min_x = min_y = max_x = max_y = 0.0
for i in range(count):
    # Get the height and width of the original images
    h, w, p = images[i].shape
    # Create a list of points to represent the corners of the
    # images
    corners = np.array([[0, 0], [w, 0], [w, h], [0, h]],
            dtype=np.float32)
    # Calculate the transformed corners
    transformed_corners = cv.perspectiveTransform(
        corners.reshape(-1, 1, 2), cumulative_homography[i]
        )
    # Find the minimum and maximum coordinates to determine the
    # output size
    min_x = min(transformed_corners[:, 0, 0].min(), min_x)
    min_y = min(transformed_corners[:, 0, 1].min(), min_y)
    max_x = max(transformed_corners[:, 0, 0].max(), max_x)
    max_y = max(transformed_corners[:, 0, 1].max(), max_y)

# Calculate the width and height of the stitched image
```

```
output_width = int(max_x - min_x)
output_height = int(max_y - min_y)

# define an offset for translation due to negative coordinates
offset_matrix = np.array([[1, 0, - min_x], \
                          [0, 1, - min_y], \
                            [0, 0, 1]], dtype=np.float32)

# define the output image
output = np.zeros((output_height, output_width, 3),
        dtype=np.uint8)
```

Now, you can proceed to stitch the images. First, select an image as an anchor and transform other images onto this anchor image. The transformation between any image and this anchor image is the composition of a series of homographies. Compute the transformations and map all other images onto the anchor image. You can use the `cv.warpPerspective()` function to warp individual images `img` to the anchor image perspective using a homography matrix `H`. Example usage:

```
warped_img = cv.warpPerspective(img, H, (width, height))
```

Note that, by default, this function does *bilinear interpolation* to calculate the pixel color values for the warped pixels (there is an option to use nearest neighbor instead but we recommend using the default).

As you warp each image, place it onto the panorama canvas (*i.e.*, the empty "`output`" we defined before). The basic logic is to layer the images such that latter images are placed on top of former ones. This means that as you add each warped image to the panorama, it will overlap and blend with the previous images.

After you obtain the panorama, display it along with each transformed image.

What to Submit

As this assignment is mostly implemented using built-in functions, you are expected to display some intermediate results to show the internal workflow of the program. A sample visualization is shown here.

1. SIFT features: show the detected features overlaid on the images. Also give out the number of features detected in each image.

2. Graphically show the top-10 scoring matches found by the matcher before the RANSAC operation. Provide statistics of how many matches are found for each image pair.

3. Show total number of inlier matches after homography estimations. Also show top-10 matches that have the minimum error between the projected source keypoint and the destination keypoint. (Hint: check the mask value returned by the function estimating the homography).

4. Output the computed homography matrix. (The built-in homography finder also applies a non-linear optimization step at the end; you can ignore or disable this step if you wish.)

5. Show the final panorama along with each transformed image.

You should submit a single PDF file including your source code (should be well-commented) and report. The report should include:

1. A brief description of the programs you implemented.
2. Show the results of intermediate steps as listed above.
3. Write down your observations regarding the results you obtained throughout this process.