Exercise 10: Image stitching

02504 Computer vision

Morten R. Hannemose, mohan@dtu.dk, DTU Compute

April 11, 2024

Today's exercise is different from the previous weeks in that it has only a few exercises followed by multiple options for how to improve your algorithm. You are going to implement an algorithm that can stitch at least two images together, by using RANSAC to estimate a homography.

Start out with using `im1` and `im2`.

Exercise 10.1

Find SIFT keypoints (`kp1`, `kp2`) in both images and compute their descriptors (`des1`, `des2`).

Match the SIFT features from both images to each other. Make sure to use cross checking.

Exercise 10.2

Implement a RANSAC algorithm for finding the homography between `im1` and `im2`.

What is the minimum number of matches you need to estimate a homography? (*Tip*: it's four) Explain why this is the case.

Use Equation (2.45) from the lecture notes to compute the distance of a match to a homography.

Assume $\sigma = 3$ and use the formulas from the lecture last week to determine the threshold for when a match should be considered an inlier.

While best practice would be to determine the number of iterations while running the algorithm, you can fix it to i.e. 200.

To verify that the inliers of the best model are reasonable, visualize them. Consider using the following code:

```
plt.imshow(cv2.drawMatches(im1, kp1, im2, kp2, np.array(matches)[bestInliers], None))
```

On the provided images you should find in the ballpark of a thousand inliers.

Exercise 10.3

Wrap your code from the previous exercise in a function:
`H = estHomographyRANSAC(kp1, des1, kp2, des2)`.

The function should take SIFT keypoints and descriptors computed on two images and use RANSAC to estimate the best homography between the images.

Make sure that the function ends with fitting a homography to the largest amount of inliers.

Exercise 10.4

Run the function on `im1` and `im2`. You can use the following function for warping the images using your homography.

```
def warpImage(im, H, xRange, yRange):
    T = np.eye(3)
    T[:2, 2] = [-xRange[0], -yRange[0]]
    H = T@H
    outSize = (xRange[1]-xRange[0], yRange[1]-yRange[0])
    mask = np.ones(im.shape[:2], dtype=np.uint8)*255
    imWarp = cv2.warpPerspective(im, H, outSize)
    maskWarp = cv2.warpPerspective(mask, H, outSize)
    return imWarp, maskWarp
```

It takes an image and a homography and returns the image warped with the homography, where `xRange` and `yRange` specifies for which range of x and y values the image should be sampled. The function returns the transformed version of the image, and a mask that is 1 where the image is valid.

Start out by setting

```
xRange = [0, im1.shape[1]]
yRange = [0, im1.shape[0]]
```

Warp one of your images using the estimated homography. Which image you should warp, depends on if you have found the homography going from image one to two or vice versa. This should warp this image to the other, thus cutting off a lot of the content of that image.

Use the warping function on the other image but set the homography to the identity. Change `xRange` and `yRange` so the images are no longer getting cropped by the warp.

Exercise 10.5

Use the mask returned by the warping function, to generate a single image that contains both images. Where the images overlap you can use the intensities from either image.

Optional algorithm improvements

Congratulations! You have now created a panorama! But you can improve your algorithm. Here are some suggestions.

Exercise 10.6

Devise a way to set `xRange` and `yRange` automatically so image content is not lost.

Tip: You can use the homography to warp points, but which ones?

Exercise 10.7

Expand your algorithm so it's able to handle the situation where three or more images are taken in a line.

Exercise 10.8

Expand your algorithm so it's able to handle arbitrarily overlapping images, such as four images taken in a rectangle (i.e. north-east, north-west, south-west, south east). It's fine to manually designate one of the images as the reference image.

Implement non-linear optimization on top, so the homographies fit well to all overlaps.