Assignment 8

This exercise is part of the course assignment. **Deadline for the assignment 30.11.2022 at 23:59**

The topic of this assignment is image alignment. For this assignment you should return

- The files main8.m, affine_fit.m, homography_fit.m and ransac_homography.m. Each file should contain your name and student number (of both students if you work in pairs).
- Your answers to the questions in the analysis part. At the end of the course, you should return a single pdf containing the answers to all questions of the assignments. The report should contain also your name and student number (of both students if you work in pairs).

Coding part (5pt)

1. First, run demo8.p to get a taste of the assignment.
2. Implement the function [M, t] = affine_fit(x1, x2) which estimates the affine tranformation between the correspondences $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$. The array $\mathbf{x}_1$ is a $2 \times N$ matrix where each column contains the coordinates of a point in the first image. The array $\mathbf{x}_2$ is a $2 \times N$ matrix containing the coordinates of the corresponding points in the second image. Your task is to form the linear system described on slides 91-92 of lecture 9.
3. When you are done, run the first cell of main8.m and check that the function works (transformed and target image should be equal).
4. Implement the function H = homography_fit(x1, x2), that computes the homography matrix $H \in \mathbb{R}^{3\times3}$ given correspondences $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$. Form the homogeneous system as described on the slides 94-95 of lecture 9 and solve using homogeneous least squares.
5. To check your function works, run the second cell of the main.
6. So far we have used clean data with exact matches. However, features matching is a challenging task and in real life you should expect that some of those matches are incorrect. For this reason we want to implement the function H = ransac_homography(x1, x2, t, p), which estimates the homoography in a RANSAC loop. x_1 and x_2 contain the

coordinates of corresponding points, t is the threshold to determine inliers (see below) and p is the probability to pick an outlier free sample at least once. This exercise is very similar to the bonus exercise of this week. Inside the RANSAC loop you should

- Pick the minimum number of points (how many?) and use those to estimate the homography, use the function homography_fit you wrote before.
- Determine the inliers. Suppose $\mathbf{x}_1$ is a point in the first image and $\mathbf{x}_2$ the corresponding point in the second image. Let $\mathbf{x}_1'$ the Cartesian coordinates of the point obtained after applying the homography $\mathbf{H}$ to $\mathbf{x}_1$. The pair $\mathbf{x}_1, \mathbf{x}_2$ are inliers if and only if $\|\mathbf{x}_2 - \mathbf{x}_1'\|^2 \leq t$.
  Hint: To compute $\mathbf{x}_1'$, you may want to convert $\mathbf{x}_1$ to homogeneous coordinates, apply the homography and transform the result back to Cartesian coordinates.
- Compute the outliers ratio and update the maximum number of iterations as in this week bonus exercise
- After the RANSAC loop has terminated, estimate the homography $\mathbf{H}$ one more time using all the inlier points you have found.

7. when you are done, run the last cell of main8.m to check your function works. This cell computes the homography between two images and uses it to construct a panorama image.

Analysis part (5pt)

- The homography matrix $\mathbf{H}$ has 9 elements but only 8 degrees of freedom, why? (1pt)
- Consider the image alignment problem, when is it enough to fit an affine transformation between the images and when do you need to fit a homography? (2pt)
- In image alignment we use RANSAC because it is probable that several matches will be outliers. Give at least 2 reasons that can cause false matches. (2pt)

Bonus: Image warping (5pt)

To get points from this assignment return the file apply_homography.m In Matlab, to apply a homography **H** to an image I we can use the built-in function imwarp, as the following example shows

```
% assuming you have an image I and
   homography matrix H
tform = projective2d(H')
J = imwarp(I, H)
```

Note! The function projective2d builds a transformation object from the transformation matrix. Observe that this function takes the **transpose** of the matrix!

The purpose of this exercise is to implement ourselves a function apply_homography, which will behave like the matlab code above. Obviously, using the matlab function imwarp in this assignment is not allowed.

1. Run bonusdemo.p to get a taste of the assignment.

2. Let's start building our apply_homography(I, H) function. I is an image of height H and width W, i.e. a $H \times W$ matrix if it is a grayscale image or a $H \times W \times 3$ matrix if it is a color image. H is a 3×3 homography matrix. As a first thing, it is good to convert the image to double.

3. Create the meshgrids x and y from the image. The matrix x will have size $H \times W$ and the element $x(i, j)$ will contain the x coordinate of the pixel (i, j) in the image; y is similar. You can use the matlab function meshgrid.

4. From the meshgrids x and y, construct an array X of size $3 \times WH$ containing the homegeneous coordinates of each pixel in the image.

5. Apply the homography H to X and save the result into the variable Y, convert Y to Cartesian coordinates.

6. Reshape the x- and y-coordinates of the transformed points into the matrices xx and yy of size $H \times W$. The same logic applies, $xx(i, j)$ should have the transformed x-coordinate of the pixel (i, j) in the original image. Note that since these points are scattered, technically this is not a meshgrid.

7. The new set of points will probably not be a rectangle. Since images are rectangle shaped, we need to find the *minimum bounding rectangle*,

as shown in Figure 1. Use the functions min and max to find the corner points of the minimum bounding rectangles. Round the result to integers, since these points will represent pixels.

Figure 1: Minimum bounding rectangle of transformed image

8. Using the function meshgrid and the result of the previous step, create the new meshgrids $xnew$, $ynew$ of the transformed image.

9. Create a new variable J which will contain the transformed image and initialize it to zeros. Note that the new image should have the same number of channels of the original one, i.e. your code should be something like J = zeros(something, something, size(I, 3)).

10. The pixel $I(i, j)$ has the intensity value at coordinates $xx(i, j), yy(i, j)$. To obtain the final image, we need to interpolate the intensity values on the points in the meshgrids ($xnew$, $ynew$). To do so, you will need the function gridddata and store the interpolated values into the new image J. Note that you will need to process one color channel at the time, i.e. your code should look like

```
for ii=1:size(J, 3)
J(:,:,ii) = griddata(...)
end
```

11. Finally, convert your image to uint8.

12. When you are ready, run demo8.m. If you want, you can experiment different transformations, e.g. the ones presented in the exercise session.

13. You can also compare your function to the matlab imwarp built-in function, the result should be (almost) identical. Congratulations! You have implemented a small piece of Matlab from scratches!