Assignment 4
Image Alignment and Stitching

Computer Vision 1
University of Amsterdam

Due 11:59pm, October 5, 2019 (Amsterdam time)

General guidelines

Your source code and report must be handed in together in a zip file (`ID1_ID2_ID3_ID4.zip`) before the deadline by sending it to Canvas Lab 4 Assignment. For full credit, make sure your report follows these guidelines:

- Include an introduction and a conclusion to your report.
- The maximum number of pages is 10 (single-column, including tables and figures). Please express your thoughts concisely. The number of words does not necessarily correlate with how well you understand the concepts.
- Answer all given questions (in green boxes). Briefly describe what you implemented. Blue boxes are there to give you hints to answer questions.
- Try to understand the problem as much as you can. When answering a question, give evidences (qualitative and/or quantitative results, references to papers, etc.) to support your arguments.
- Analyze your results and discuss them, e.g. why algorithm A works better than algorithm B in a certain problem.
- Tables and figures must be accompanied by a brief description. Do not forget to add a number, a title, and if applicable name and unit of variables in a table, name and unit of axes and legends in a figure.

Late submissions are not allowed. Assignments that are submitted after the strict deadline will not be graded. In case of submission conflicts, TAs' system clock is taken as reference. We strongly recommend submitting well in advance, to avoid last minute system failure issues.
Plagiarism note: Keep in mind that plagiarism (submitted materials which are not your work) is a serious crime and any misconduct shall be punished with the university regulations.

Contents

1 Image Alignment (60pts)

In this practice, you will write a function that takes two images as input and computes the affine transformation between them. You will work with supplied *boat* images. The overall scheme is as follows:

1. Detect interest points in each image.
2. Characterize the local appearance of the regions around interest points.
3. Get the set of supposed matches between region descriptors in each image.
4. Perform RANSAC to discover the best transformation between images.

Hint

The first three steps can be performed using David Lowe's SIFT. Check out the Docs of SIFT related function for further information in the following link: `https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html` and `https://docs.opencv.org/3.4.9/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html`

For some patent issue, the newest version of OpenCV does not contain SIFT related function. However, you can install a old version (for example: opencv-python==3.4.2.17 and opencv-contrib-python==3.4.2.17)

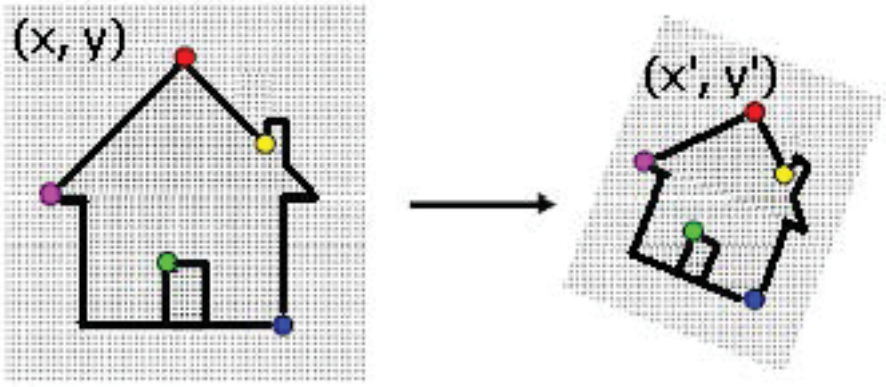
The final stage, running RANSAC, should be performed as follows:

- Repeat N times:
- Pick P matches at random from the total set of matches T.
- Construct a matrix A and vector b using the P pairs of points and find transformation parameters $(m1, m2, m3, m4, t1, t2)$ by solving the equation $Ax = b$.

  Hint

  Equation $Ax = b$ can be solved using pseduo-inverse: $x = (A^T A)^{-1} A^T b$, or `x = np.linalg.inv(A)*b` in Python.

- Using the transformation parameters, transform the locations of all T points in image1. If the transformation is correct, they should lie close to their counterparts in image2. Plot the two images side by side with a line connecting the original T points in image1 and transformed T points over image2.
- Count the number of inliers, where inliers are defined as the number of transformed points from image1 that lie within a radius of 10 pixels of their pair in image2.



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \tag{1}$$

It can be rewritten as

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{2}$$

or

$$Ax = b,\ A = \begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \end{bmatrix},\ x = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix},\ b = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{3}$$

Figure 1: Affine transformation.

- If this count exceeds the best total so far, save the transformation parameters and the set of inliers.
- End repeat.
- Finally, transform image1 using this final set of transformation parameters. If you display this image, you should find that the pose of the object in the scene should correspond to its pose in image2. To transform the image, implement your own function based on nearest-neighbor interpolation. Then use the OpenCV built-in function `cv2.warpAffine` and compare your results.

Hint

Nearest neighbors does not mean you have to classify points. The problem is that if you have a transformation, then the transformed points may not be at perfect pixels (e.g. 0.3px). Instead of linear interpolation, which requires more work to implement, we can just use nearest neighbors which means simply rounding the coordinates.

Question - 1 (55-*pts*)

1. Create a function that takes image pairs *boat1.pgm* and *boat2.pgm* as input, and return the keypoint matchings between the two images. Name your script as **keypoint_matching.py**.

2. Take a random subset (with set size set to 10) of all matching points, and plot on the image. Connect matching pairs with lines. You can assign a random color to each line to make them easier to distinguish.

3. Create a function that performs the RANSAC algorithm as explained above. The function should return the best transformation found. For visualization, show the transformations from image1 to image2 and from image2 to image1. Name your script as **RANSAC.py**.

Include a demo function to run your code.

Question - 2 (5-*pts*)

1. How many matches do we need to solve an affine transformation which can be formulated as shown in Figure 1?

2. How many iterations in average are needed to find good transformation parameters?

2 Image Stitching (40pts)

In this part, you will write a function that takes two images as input and stitch them together. The method described in the previous section will be used to stitch two images together by transforming one of them to the coordinate space of the other. You will work with supplied images *left.jpg* and *right.jpg*. The overall scheme can be summarized as follows:

1. As in previous task you should first find the best transformation between input images.
2. Then you should estimate the size of the stitched image.

   Hint

   Calculate the transformed coordinates of corners of the *right.jpg*

3. Finally, combine the *left.jpg* with the transformed *right.jpg* in one image.

Question - 1 (40-*pts*)

1. Create a function that takes an image pair as input, and return the stitched version. Name your script as **stitch.py**.
2. Visualize the stitched image alongside with the image pair.

Include a demo function to run your code.