Computer Vision/Visual Perception
ESIREM Le Creusot & Master in Computer Vision
Universite de Bourgogne, February 2022

Practical Exercise I - Feature Matching & Mosaicing Images

The goal of this assignment is to practice the detection of keypoints, description and matching seen in the class. We will use the matched visual features for image registration and building mosaic of images. A mosaic is a way of stitching together images which are related either by rotation or by view of a planar surface. Both transformations can be represented by a 3x3 homography matrix (as presented in Chapter 4 of Hartley & Zisserman [1]).

You will also implement RANSAC (and get more familiar with it) to estimate planar homographies, that will be robust to wrong matches between different views of a scene. After stitching the images together you will obtain a mosaic as illustrated in Figure 1. These three input images are provided in the folder **images/**.

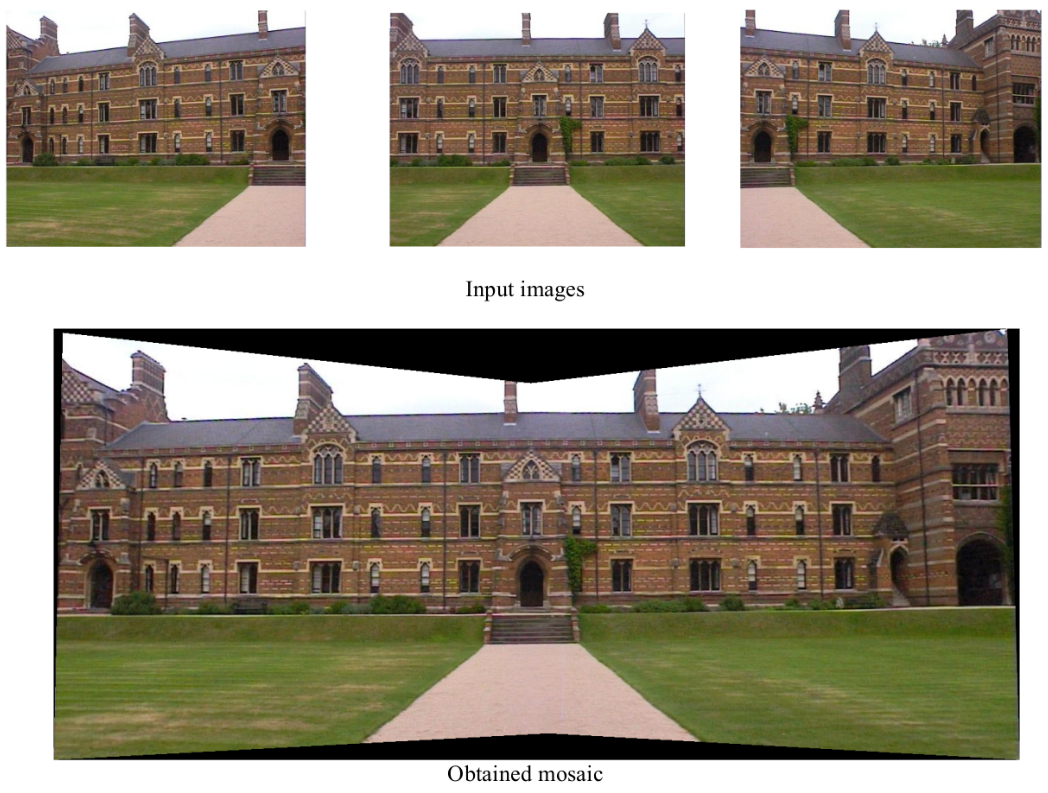


Figure 1: Mosaicing result from the left and right top images using as reference the center image.

The algorithm for merging multiple images into a mosaic can be described as follows:

1. Choose one image as the reference image frame;

2. Estimate a homography between each of the remaining images and the reference one. Each homography estimation is done by:
    - Detecting local features in each image;
    - Matching feature descriptors between two images;
    - Estimating the homography between two images with RANSAC for robustness to wrong keypoint correspondences.
3. Warping each image into the reference frame using the estimated homography and composite the warped images into a single mosaic.

Part I - Environment Setup

Please create a notebook on Colab or follow these steps to configure your conda environment if you are worning in your local machine:

```
>> conda create -n cv_mosaicing
>> conda activate cv_mosaicing
>> conda install jupyter
## Then go to your workspace directory of the course and run jupyter
>> jupyter-notebook
```

Again, we will greatly use Numpy and some OpenCV functions during this exercise. So do not hesitate to go back in the slides of Class 2 (OpenCV/Python) to check some definitions or in this tutorial on Numpy[1]. There are also some helpful “Cheat Sheets” of basic commands for Numpy and Scipy (Linear algebra in Python)[2].

Part II - Detecting Features and Matching

To establish correspondences between the image frames, we will detect and match keypoints. Keypoints are points in the image which are expected to be reliably matched, such as corner points or blobs. There are many different keypoint detectors implemented in OpenCV. We will test with two different detectors available: FAST and SIFT detectors.

1. Detection:
    - Write a function that returns a list of detected keypoints either by FAST or SIFT detectors on each frame using their default parameters.
    - Display the detected keypoints on the frames as red dots.
    - How many points do you get in each image for each detector?
    - Is it a way of selecting the “best” keypoints? (hint: please look the fields in cv.Keypoint) Modify your algorithm to return the N best points and then display the best 200 points for each image.
    - What are the differences between the detected keypoints of the two approaches?
2. Description:

[1] https://numpy.org/devdocs/user/quickstart.html
[2] Scipy: http://datacamp-community-prod.s3.amazonaws.com/dfdb6d58-e044-4b38-bab3-5de0b825909b
Numpy: http://datacamp-community-prod.s3.amazonaws.com/ba1fe95a-8b70-4d2f-95b0-bc954e9071b0

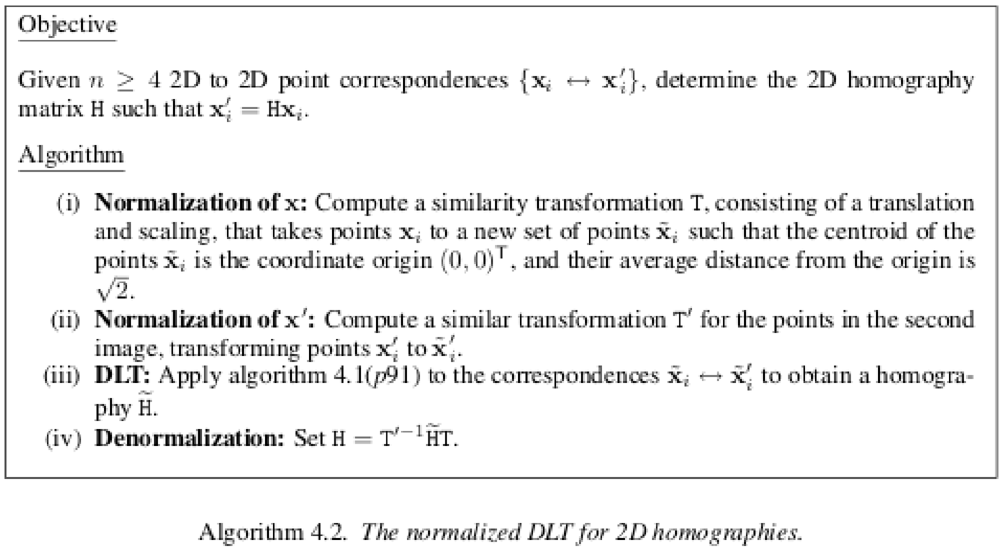
- Write a function that returns the extracted features either by SIFT or ORB descriptors on each frame on the detected keypoints of the previous step.

3. Matching:
   Now that we have done the detection and description of keypoints, we will first find corresponding visual features. For each detector/descriptor:
   - Find the correspondences between the features using brute force matching (one to all). The distances between the descriptors should be the Euclidean distance between the descriptions using SIFT or the Hamming distance for ORB;
   - Use the 1NN/2NN ratio test for removing some ambiguous correspondences (check the slides of the course);
   - Further improve the matching by removing possible wrong matches with cross-validation check (check the slides of the course).

Part III - Robust Homography Fitting & Mosaicing

We will now estimate the homography between two images based on the found matches in the previous step.

1. The homography estimation estimates a 3x3 matrix that handles correspondences between planar points in two images (2D–2D). We will adopt again the DLT algorithm (similar to the one used in the Assignment III Calibration). Implement the algorithm DLT following the recipe of the following algorithm shown in Figure 2. What is the minimal number of correspondences to find the homography transformation?



Objective

Given $n \geq 4$ 2D to 2D point correspondences $\{\mathbf{x}_i \leftrightarrow \mathbf{x}'_i\}$, determine the 2D homography matrix $\mathtt{H}$ such that $\mathbf{x}'_i = \mathtt{H}\mathbf{x}_i$.

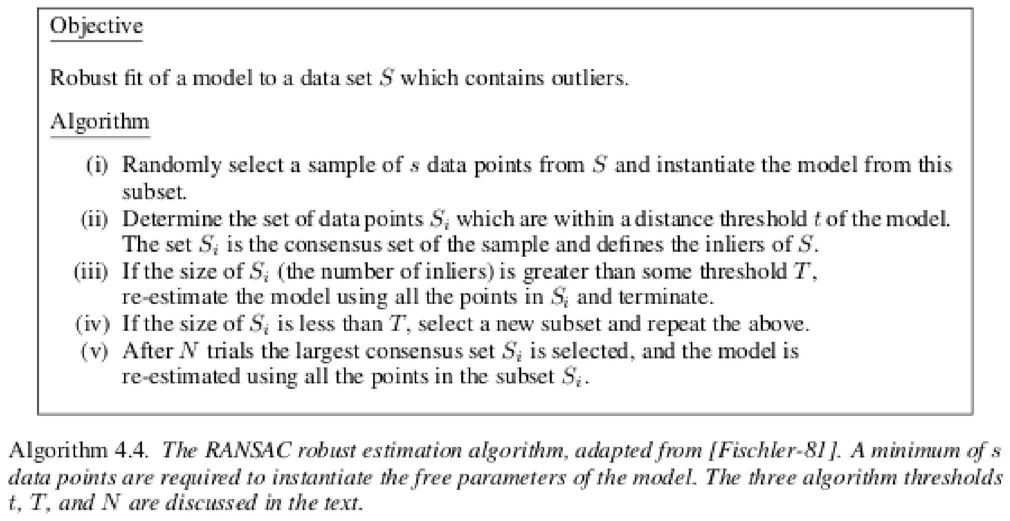
Algorithm

(i) **Normalization of x:** Compute a similarity transformation $\mathtt{T}$, consisting of a translation and scaling, that takes points $\mathbf{x}_i$ to a new set of points $\tilde{\mathbf{x}}_i$ such that the centroid of the points $\tilde{\mathbf{x}}_i$ is the coordinate origin $(0,0)^\mathsf{T}$, and their average distance from the origin is $\sqrt{2}$.
(ii) **Normalization of x′:** Compute a similar transformation $\mathtt{T}'$ for the points in the second image, transforming points $\mathbf{x}'_i$ to $\tilde{\mathbf{x}}'_i$.
(iii) **DLT:** Apply algorithm 4.1(p91) to the correspondences $\tilde{\mathbf{x}}_i \leftrightarrow \tilde{\mathbf{x}}'_i$ to obtain a homography $\tilde{\mathtt{H}}$.
(iv) **Denormalization:** Set $\mathtt{H} = \mathtt{T}'^{-1}\tilde{\mathtt{H}}\mathtt{T}$.

Algorithm 4.2. *The normalized DLT for 2D homographies.*

Figure 2: Homography estimation using DLT recipe.

2. Implement your own RANSAC algorithm for estimating the homography matrix following the RANSAC recipe algorithm shown in Figure 3.



Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

(i) Randomly select a sample of s data points from S and instantiate the model from this subset.
(ii) Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i is the consensus set of the sample and defines the inliers of S.
(iii) If the size of S_i (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_i and terminate.
(iv) If the size of S_i is less than T, select a new subset and repeat the above.
(v) After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.

Algorithm 4.4. *The RANSAC robust estimation algorithm, adapted from [Fischler-81]. A minimum of s data points are required to instantiate the free parameters of the model. The three algorithm thresholds t, T, and N are discussed in the text.*

Figure 3: RANSAC algorithm recipe.

3. Indicate the number of correct and wrong matches obtained after homography fitting with RANSAC for each descriptor.

4. Finally, you can now warp the each image with its computed homographies into the reference one and then build the final composite image. There are several ways of performing the warping such as using *cv.warpPerspective*. Are the results satisfactory for both descriptors?

Submission

Please return a concise PDF report commenting your reasoning and your results. Please provide concrete examples/discussions to support your explanations for the questions when suitable. You should also submit your python notebook (commented with the corresponding implementation, visualizations) inside a [name]_practical_1.zip file (replacing [name] with your name ;-)). The submission should be done via the teams channel of the course using teams assignment.
Deadline: 18/02/2022 at 23:59pm.

Notes:

- Plagiarism copying the work from another source (student, internet, etc.) will be awarded a 0 mark. In case there are multiple submissions with the same work, each one will receive a 0.
- **Credits:** This assignment was based on exercises kindly provided by Désiré Sidibé.

References

[1] Hartley, Richard, and Andrew Zisserman. "Multiple view geometry in computer vision." (2003).