COMP9517: Computer Vision

2023 T2 Lab 2 Specification

Maximum Marks Achievable: 2.5

This lab is **worth 2.5% of the total course marks**.

The lab files should be submitted online. Instructions for submission will be posted closer to the deadline. **Deadline for submission is Week 4, Friday 23 June 2023, 18:00:00.**

Objective: This lab revisits important concepts covered in the lecture of Week 3 and aims to make you familiar with implementing specific algorithms.

Materials: The sample images to be used in the tasks of this lab are available on WebCMS3. You are required to use OpenCV 3+ with Python 3+.

Submission: The tasks are assessable after the lab. Submit your source code as a Jupyter notebook (.ipynb) which includes all output (see coding requirements below) by the above deadline. The submission link will be announced in due time.

SIFT: Scale-Invariant Feature Transform

A well-known algorithm in computer vision to detect and describe local features in images is the scale-invariant feature transform (SIFT). Its applications include object recognition, mapping and navigation, image stitching, 3D modelling, object tracking, and others.

A SIFT feature of an image is a salient keypoint with an associated descriptor. SIFT computation is commonly divided into two steps:

1) detection,
2) description.

At the end of the detection step, for each keypoint the SIFT algorithm computes the:

- keypoint spatial coordinates (x, y),
- keypoint scale (in the scale space),
- keypoint dominant orientation.

The subsequent description step computes a distinctive 128-dimensional feature vector for each keypoint. SIFT is designed in such a way that this descriptive feature vector is invariant

to scaling and rotation. Moreover, the algorithm offers decent robustness to noise, illumination gradients, and affine transformations.

RANSAC: Random Sample Consensus Algorithm

The random sample consensus (RANSAC) algorithm is an iterative approach to estimate the parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. Therefore, it also can be seen as an outlier detection method.

Task 1 (0.5 mark): Compute the SIFT features of the given image **House.png**.

a) Extract SIFT features with default parameters and show the keypoints on the image. You are allowed to use existing library functions for this (see suggestions below).
b) To achieve better visualization of the keypoints, reduce the number of keypoints. Hint: Vary the parameter contrastThreshold or nfeatures so that the number of keypoints becomes about 10% of all default keypoints.

Show the result images obtained in a) and b) in your Jupyter notebook and include a brief description of the approach you used for b).

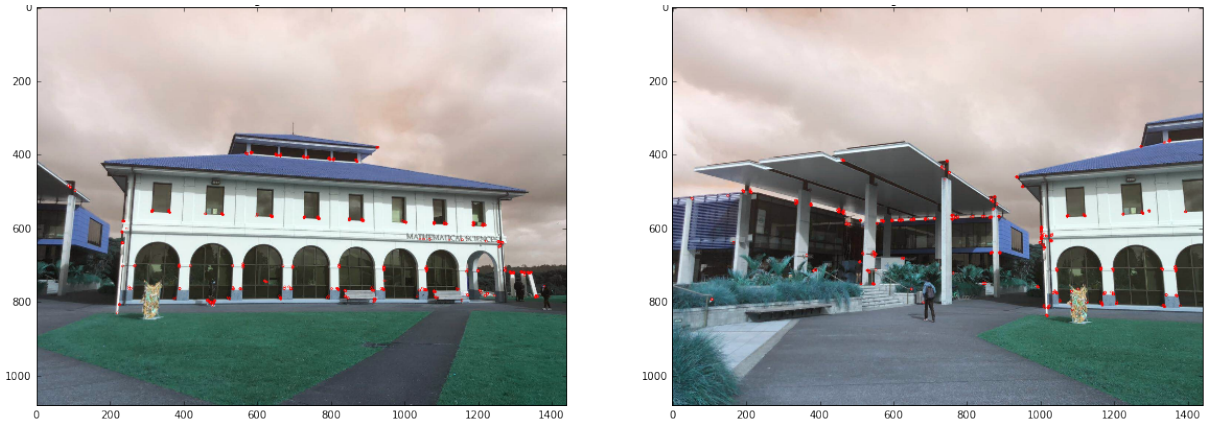
Task 2 (0.5 mark): Recompute the SIFT features for a noisy version of **House.png**.

a) Add pepper noise to the given image. Hint: The scikit-image library has a utility function to add random noise of various types to images.
b) Extract the SIFT features and show the keypoints on the noisy image using the same parameter setting as for Task 1 (for the reduced number of keypoints).
c) Inspect the keypoints visually: Are the keypoints of the noisy image roughly the same as those of the original image? What does this mean?

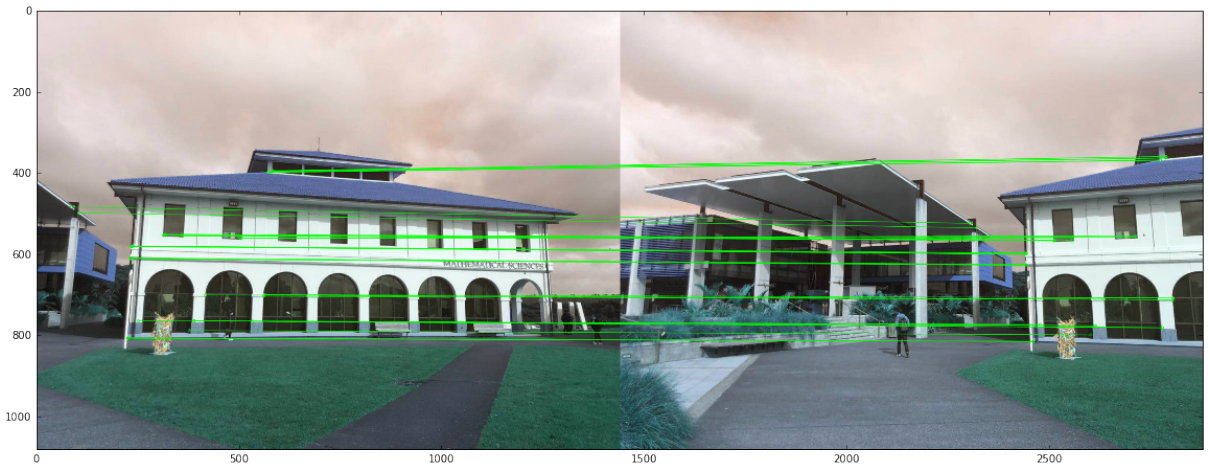
Show the result images obtained in a) and b) in your Jupyter notebook and include your answers to the questions in c).

Task 3 (1.5 mark): Match and stitch two given images **Scene1.png** and **Scene2.png**.

a) Extract the SIFT features and show the keypoints on each image. Below, we show example results just to give you an idea of the sort of output we are looking for, but your results may look somewhat different depending on the specific implementation of the SIFT algorithm and the parameter settings used. As long as the calculations in a) and b) produce a good final result in c), the precise intermediate results are less important.



b) Find the keypoint correspondences between the images and draw them. Hints: First, use OpenCV's brute-force descriptor matcher (BFMatcher) to find matching keypoints. Then, use its kNN-based matching method (knnMatch) to extract the k nearest neighbours for each query keypoint. Use your own criteria based on the keypoint distances to select the best keypoint correspondences between the two images.



c) Use the RANSAC algorithm to robustly estimate the mapping of one the two images to the other based on the selected best keypoint correspondences and then apply the mapping and show the final stitched image. Hints: There are existing OpenCV functions to find the mapping (findHomography) between sets of points using various methods, as well as functions to apply this mapping to sets of points (perspectiveTransform) and warp images accordingly (warpPerspective). You may need to crop the result to get a nicely stitched image. The red line drawn in the example below indicates the stitching boundary.

Coding Requirements and Suggestions

Check the OpenCV documentation for various built-in functions to find SIFT features, draw keypoints, and match keypoints in images, as well as apply RANSAC to estimate a mapping function. You should understand how the algorithms work, what parameters you can set in these built-in functions, and how these parameters affect the output. For your reference, below are links to relevant OpenCV functions.

2D Features Framework
https://docs.opencv.org/4.6.0/da/d9b/group__features2d.html

Drawing Functions of Keypoints and Matches
https://docs.opencv.org/4.6.0/d4/d5d/group__features2d__draw.html

Descriptor Matchers
https://docs.opencv.org/4.6.0/d8/d9b/group__features2d__match.html

OpenCV SIFT Class Reference
https://docs.opencv.org/4.6.0/d7/d60/classcv_1_1SIFT.html

In your Jupyter notebook, the input images should be readable from the location specified as an argument, and all output images and other requested results should be shown in the notebook environment. All cells in your notebook should have been executed so that the tutor/marker does not have to execute the notebook again to see the results.

Refer to the following page to understand image features and various feature detectors:
https://docs.opencv.org/4.6.0/db/d27/tutorial_py_table_of_contents_feature2d.html

Also, refer to the following example of computing SIFT features and showing the keypoints:
https://docs.opencv.org/4.6.0/da/df5/tutorial_py_sift_intro.html

And finally see this page for an example of feature matching:
https://docs.opencv.org/4.6.0/dc/dc3/tutorial_py_matcher.html

---

Reference: D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, November 2004. https://doi.org/10.1023/B:VISI.0000029664.99615.94

Released: 16 June 2023