COMP9517: Computer Vision

2025 T3 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 10 October 2025, 18:00:00 AET.**

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

Software: You are required to use OpenCV 3+ with Python 3+ and submit your code as a Jupyter notebook (see coding and submission requirements below). In the tutor consultation session this week, you can ask any questions you may have about this lab.

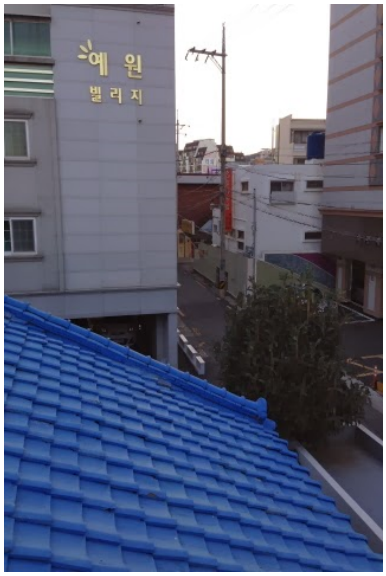
Materials: The image pair to be used in this lab is available via WebCMS3 and you are also asked to capture one pair of pictures yourself. For the latter, use your smartphone or digital camera to take the pictures required for the tasks below.

Submission: All code and requested results are assessable after the lab. Submit your source code as a Jupyter notebook (.ipynb file) that includes all output and answers to all questions (see coding requirements at the end of this document) by the above deadline. The submission link will appear on WebCMS3 in due time.

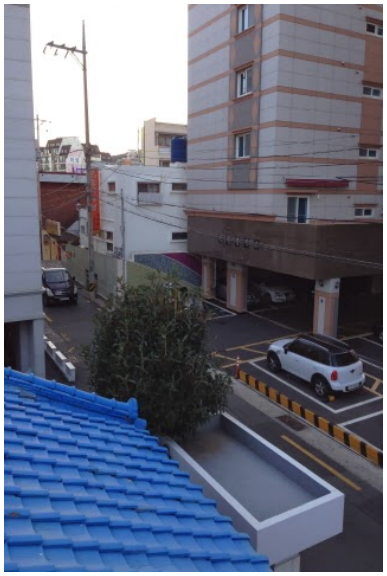
Preparation: For the pictures to be taken by yourself, choose any scene with clear structures (e.g. buildings) on campus or somewhere in your neighborhood and take two pictures of it. The two pictures should have some amount of overlap but neither of them should capture the whole scene. Below is an example pair of pictures (not to be used in your own results). To save disk space and computation time, you may downscale your pictures (e.g. to 1,000 pixels wide) before carrying out the tasks below.

Complete Tasks 1–3 using the image pair provided on WebCMS3 for this lab. Then repeat Task 3 for your own picture pair to see if your code generalizes easily to new images.

Example Picture 1



Example Picture 2



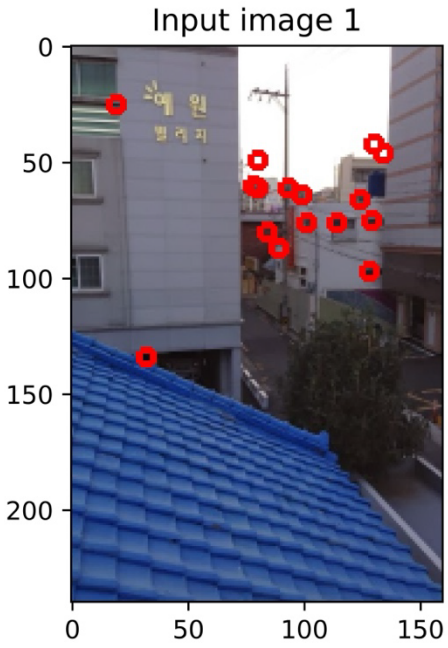
Task 1 (0.5 mark)

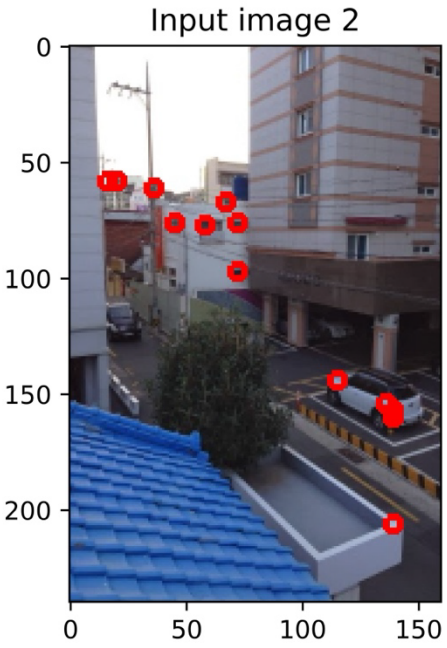
Compute the SIFT features of the two pictures.

a) Extract the SIFT features with default parameters and show the keypoints on the pictures. Hint: Use existing library functions for this (see suggestions at the end).

b) To achieve better visualization of the keypoints, reduce their number to include only the ~20 most prominent ones. Hint: Vary the parameter contrastThreshold or nfeatures.

Show the results obtained in a) and b) in your Jupyter notebook (like the examples below) and include a brief description of the approach you used for b).





Task 2 (1 mark)

Recompute the SIFT features for the following processed versions of the two pictures:

a) Scaled with a factor of 120 percent.
b) Rotated clockwise by 60 degrees.
c) Contaminated with salt and pepper noise. Hint: The scikit-image library has a utility function to add random noise of various types to images.

For each of these three versions of the pictures, extract the SIFT features and show the keypoints on the processed pictures using the same parameter settings as for Task 1 (for the reduced number of keypoints).

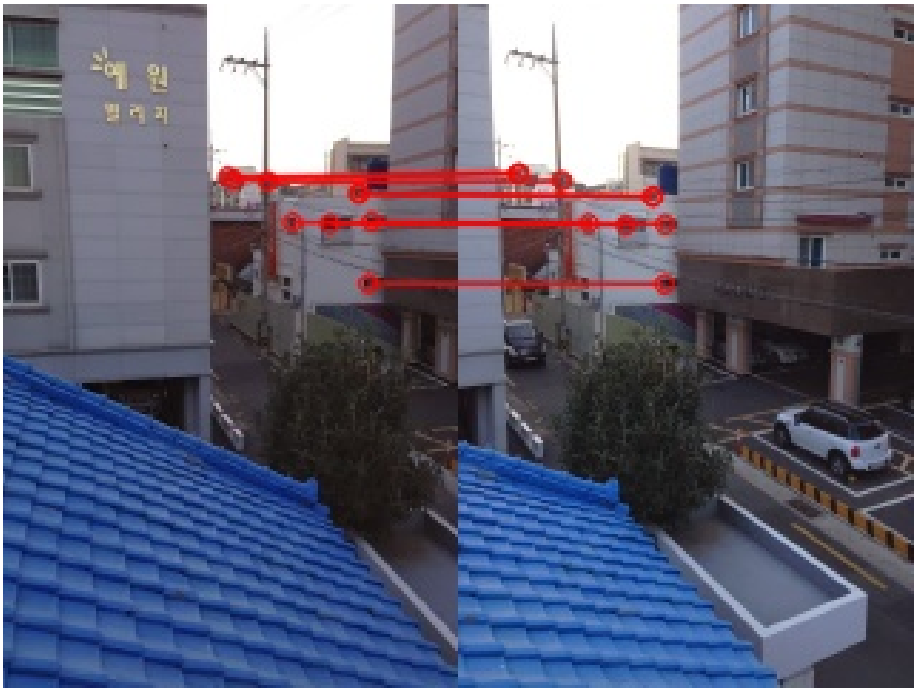
Inspect the keypoints visually: Are the keypoints of the processed pictures roughly the same as those of the originals? What does this say about the robustness of SIFT in each case? To which of the three types of processing is SIFT most robust?

Show the results obtained for each of a), b), and c) in your Jupyter notebook and include your answers to the questions stated above.

Task 3 (1 mark)

Match and stitch the two pictures to create a single composite picture.

a) Find the keypoint correspondences between the pictures 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 pictures.



b) Use the RANSAC algorithm to robustly estimate the mapping of one of the two pictures to the other based on the selected best keypoint correspondences and then apply the mapping and show the final stitched picture. 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 to warp pictures accordingly (warpPerspective). You may need to crop the result to get a nicely stitched picture. The red line shown in the example below indicates the stitching boundary, but it is not necessary to draw the boundary in your result.

Coding Requirements

A general goal in computer vision is to develop methods that work for a wide range of images. So here, too, the goal is to write code that works for both image pairs (the one we provided and your own), ideally using the same parameter values, not requiring careful manual tuning of the parameters for each image pair separately to get good results. If your code does require some tuning, include clear comments explaining this in your code.

Make sure that in your Jupyter notebook, the input pictures are readable from the location specified as an argument, and all outputs and other requested results are displayed 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.

Coding 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

See 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, see 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: 29 September 2025