# **DPS920/CVI620 – Lab 6**

# **Matching Features and Templates**

| Total Mark: | 10 marks (3% of the total course grade)   * 6 out of 10: Learn@Seneca submission (Due: Wednesday October 18th end of day) * 4 out of 10: Lab demo (Due: During Lab of week 7) |
| --- | --- |
| Submission file(s): | * Lab06\_1.py / Lab06\_1.ipynb * Lab06\_2.py / Lab06\_2.ipynb * Lab06.docx |

Please work in **groups** to complete this lab. This lab is worth 3% of the total course grade and will be evaluated through your written submission, as well as the lab demo. During the lab demo, group members are *randomly* selected to explain the submitted solution. Group members who are not present during the lab demo will lose the demo mark.

Please submit the submission file(s) through Learn@Seneca. ALL team members must submit the final work.

***Please paste the resulting images and answers in this document.***

## **Part I: Keypoint Matching**

1. Write code in Lab06\_1.py to detect ORB features on Trillium\_s and Trillium\_t images and show the results for (1) the top 10 and (2) the top 15 matches, using:
   1. BFMatcher

A close-up of a flower

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. FlannBasedMatcher (remove crossCheck argument)

A screenshot of a computer

Description automatically generated

A close-up of a flower

Description automatically generated

1. How are the keypoints matched? What is the difference between these two matchers? Which one works better? Why is that?

The keypoints are matched based on their descriptors, which capture the local characteristics of the area around each keypoint. These descriptors are then compared between the two images, and matches are found based on the similarity of the descriptors.

The difference between the two matchers:

BFMatcher (Brute-Force Matcher): BFMatcher uses a brute-force approach. It takes the descriptor of one feature from the first set and matches it with all the other features in the second set using a specified distance calculation, and returns the best match. The most common distances used are Hamming distance (especially with binary descriptors like ORB) and L2 distance.

FlannBasedMatcher (Fast Library for Approximate Nearest Neighbors Matcher): This matcher uses an algorithm to find approximate nearest neighbors efficiently. It is faster than BFMatcher, especially when dealing with large datasets. FlannBasedMatcher, in contrast to BFMatcher, does not necessarily provide the exact best match but provides an approximation, which is often sufficient and saves computational time.

From the images of the result, both matchers seem to have found similar matches. However, it's important to note that FlannBasedMatcher usually works faster with a large number of keypoints since it doesn't rely on brute-force search.

The quality of matches can vary depending on the nature of the images, the presence of noise, and the type of descriptors used. In some cases, BFMatcher might yield better results because it's exhaustive in its search, while in other scenarios where speed is crucial, FlannBasedMatcher might be preferred.

1. Rotate Trillium\_t by 30 degrees (counterclockwise) around the center of the image. Then repeat question 1 with this rotated image and Trillium\_s and for the top 10 matches. Show the matches using FlannBasedMatcher. How does the rotation affect the matching?

A close-up of a flower

Description automatically generated

Here are some observations and insights into how rotation affects the matching:

Consistency of Keypoints: Many keypoints in the non-rotated image (on the left) align well with the keypoints in the rotated image (on the right). This indicates that the ORB feature detector is able to consistently detect key features in the image regardless of its orientation, which is one of the advantages of ORB as it is rotation invariant.

Quality of Matches: The lines connecting the keypoints indicate matches found by the FlannBasedMatcher. Most of these lines are consistent and point to the correct corresponding features in the rotated image, suggesting a good quality of matches. This is a positive indicator of the robustness of the matching algorithm against rotation.

Positional Shift: While many features have been correctly matched, the physical location of some keypoints in the rotated image has changed relative to the non-rotated image. This positional shift is expected due to rotation.

In conclusion, rotation does introduce challenges for feature matching due to changes in the position of features and potential cropping near the edges.

## **Part II: Template matching**

1. Write code in Lab06\_2.py to:
   1. Open Trillium\_s image as query image, and Trillium\_t image as template image.
   2. Use matchTemplate and TM\_SQDIFF measure to find the best match location. Draw a rectangle around the match in the query image. Also show the matching space. Paste the samples here.

A blurry image of a black spot

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. Repeat matching, using TM\_CCORR measure this time. Similarly, paste the samples here.

A blurry image of a light

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. How are the two results different?

For TM\_SQDIFF I see a dark spot in the area of best match, while for TM\_CCORR we see a bright spot in the area of best match. This is because in the first method we are using the min\_loc (lower value indicates better match), however in TM\_CCORR method we use max\_loc (higher value indicates better match).

* 1. Now resize the query image to double both width and height. Then rotate the query image by 30 degrees around the center of the (resized) image (and keep the template the same). Apply the template matching (with TM\_CCORR).

How is the performance? Are you able to find the location of the flower in the image? Explain.

A blurry image of a cloud

Description automatically generated

A screenshot of a computer screen

Description automatically generated

We are still able to match the flower, however the performance of cv.matchTemplate function on the rotated and resized image is worse than on the original. As we can see on ‘Matching result’ picture, the light area is bigger than in previous examples, which indicates that the result is less precise.

## **Part III: Group Work**

1. Add this declaration to your file:

We, group 5, Liliya Panfilova and Davender Singh, declare that the attached assignment is our own work in accordance with the Seneca Academic Policy. We have not copied any part of this assignment, manually or electronically, from any other source including web sites, unless specified as references. We have not distributed our work to other students.

1. Specify what each member has done towards the completion of this work:

|  | Name | Task(s) |
| --- | --- | --- |
| 1 | Davender Singh | Part I: Keypoint Matching |
| 2 | Liliya Panfilova | Part II: Template matching |
| 3 |  |  |