COMP9517

Lab 3, S1 2018

This lab presents a revision of important concepts from week 3 and 4 lectures. Most questions require you to use OpenCV, an open source software package that is widely used in the field. OpenCV v2.4.9.1 is installed on the lab machines and accessible through vlab environment. You are expected to use vlab during the lab sessions.

NOTE: In today's lab we shall be exploring SIFT and SURF algorithms, which are only available in OpenCV's non-free module. These are algorithms patented by their creators but are free for use in academic and research fields. These non-free modules are no longer available in the default OpenCV package and has been moved to the opencv_contrib package. The contrib library is not installed in the lab machine, so please follow the instructions below to setup a virtual environment.

To install:

```
$ virtualenv venv
$ source venv/bin/activate
$ pip install opencv-python
$ pip install opencv-contrib-python
$ python
>>> import cv2
>>> surf = cv2.xfeatures2d.SURF_create()
```

All questions in Section 5 should be attempted during the lab hour.

The questions provided in Section 6 are assessible IN THE LAB. Make sure to show your answer to your tutor before leaving the lab. It will not be assessed later on outside the lab in person or by email.

DATA SAMPLES provided in the previous labs may be used.

SIFT (Scale Invariant Feature Transform)

The goal of this lab is to familiarise you with the details of the Scale Invariant Features Transform (SIFT). SIFT is a well-known algorithm in computer vision to detect and to describe local features in images. Its applications include object recognition, robotic mapping and navigation, image stitching, 3D modelling, video tracking and others.

This lab requires OpenCV(contrib) and numpy libraries.

Sections 1 to 4 provide details of the algorithms. Make sure you understand them before attempting questions in Sections 5 and 6.

1 Overview

A SIFT feature is a salient key point that corresponds to an image region and has associated a descriptor. SIFT computation is commonly divided into two steps:

- detector
- descriptor

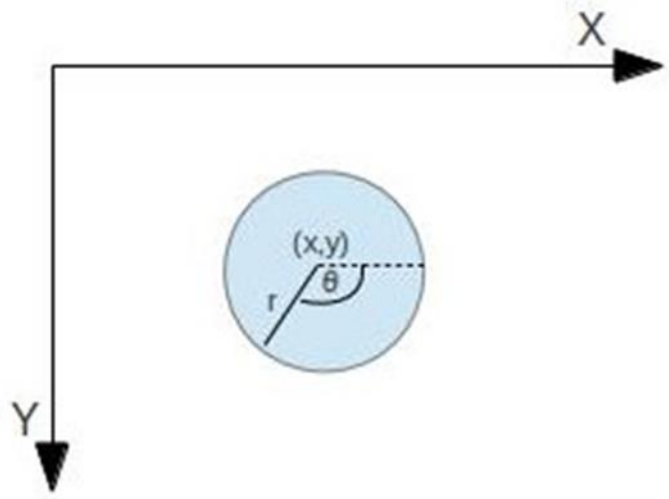
At the end of the detection phase, for each feature detected, the SIFT algorithm establishes:

- key point spatial coordinates (x,y)
- key point scale
- key point dominant orientation

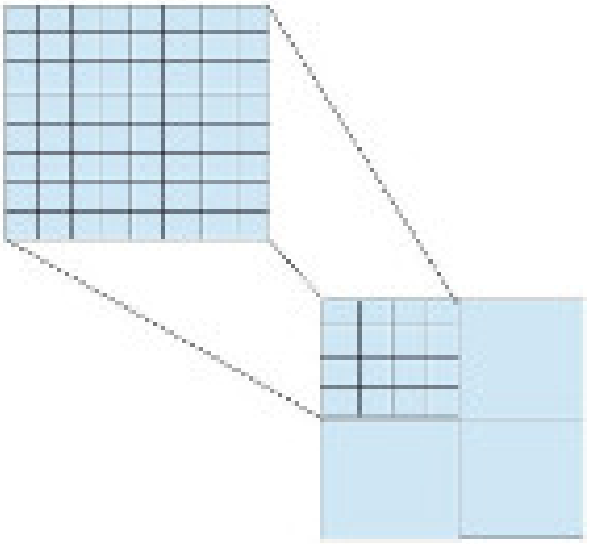
After the detection step, the SIFT descriptor step computes a distinctive fingerprint for each feature. The description obtained is designed to be invariant to scale and rotation. Moreover, the algorithm offers decent robustness to noise, illumination gradients and affine transformations.

2 SIFT detector

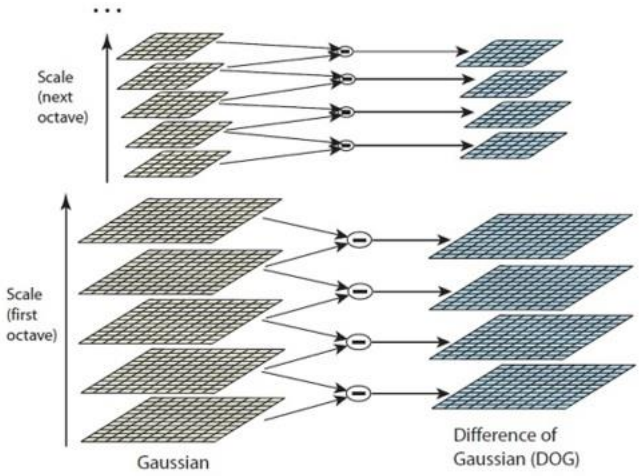
SIFT detector outcomes are SIFT key points. A key point is entirely described by 4 values, namely its x and y key point centre coordinates, scale, that is indicated by the radius of the key point region and orientation, that is an angle expressed in radians that will be explained later on.



SIFT detection starts by building a pyramidal scale space of Difference of Gaussians. For explaining the concept of the DoG pyramid, the octave will be first introduced. An octave is an $n * n$ box of pixels, with n typically set to 16, and an image is divided into a grid of non-overlapping octaves. A square composed of 4 octaves in an image is mapped to one octave in the layer above, if there are any, and it also corresponds to 16 octaves in the layer below, if there are any.



A pyramidal scale space is built by starting from the layer with the highest number of octaves, at the bottom of the pyramid, that may be composed by an octave grid of the size of the original image, or an octave grid of twice the size of the original image (as it happens in Lowe's SIFT).

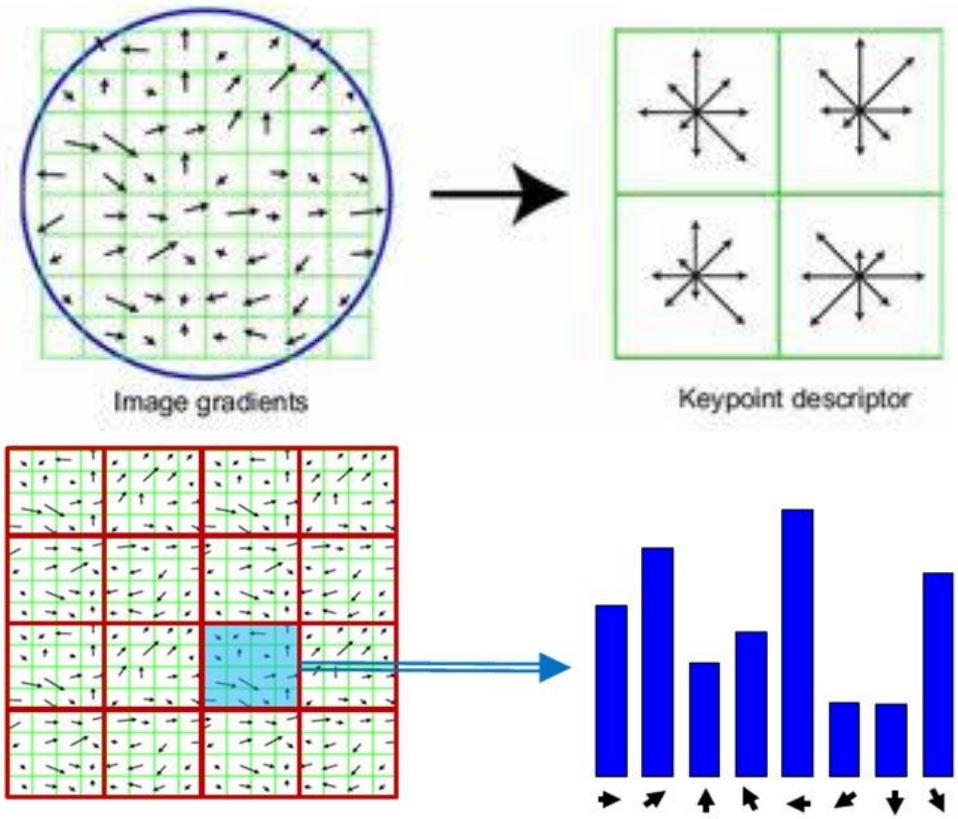


Then, the layers above are iteratively built by successively downsizing the octave grid at the current scale by a factor of 2.

3 SIFT Descriptor

Once the key points are detected, the SIFT transform is designed to provide a description of such features as well. The feature description must be highly distinctive, so that it can reduce the probability of feature mismatch. However, descriptors also must be robust to changes in illumination, noise and 3D viewpoint. The principle adopted for features description is very similar to the one adopted for Histogram of Oriented Gradients (HoG). For each key point detected, a neighbourhood of 16x16 pixels is considered. This neighbourhood is split into 4x4 cells, each cell composed of 4x4 pixels. For each cell, a histogram of the oriented gradients relative to the main orientation detected for that key point is computed, to attain invariance to rotation. A histogram of oriented gradients is a histogram of the phase of the gradients for that cell. The histogram has 8 bins, so each bin has a resolution of 45. Since

there are 4x4 cells for each neighbourhood relative to each feature, a 4x4x8 histogram is obtained, for a total of 128 for each feature.



Gradient magnitudes and orientations are calculated using these formulae:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

The magnitude and orientation are calculated for all pixels around the key point and a histogram is created. In this histogram, the 360 degrees of orientation are broken into 36 bins (each 10 degrees). If the gradient direction at a certain point (in the "orientation collection region") is 18.759 degrees, for example, then it will go into the 10-19-degree bin, and the "amount" that is added to the bin is proportional to the magnitude of the gradient at that point.

Once you've done this for all pixels around the key point, the histogram will have a peak at some point.

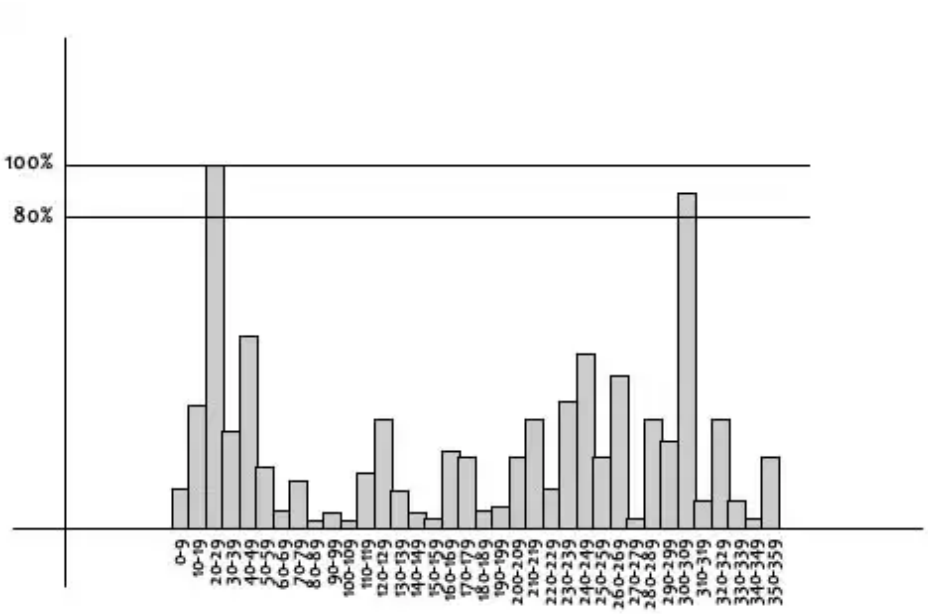
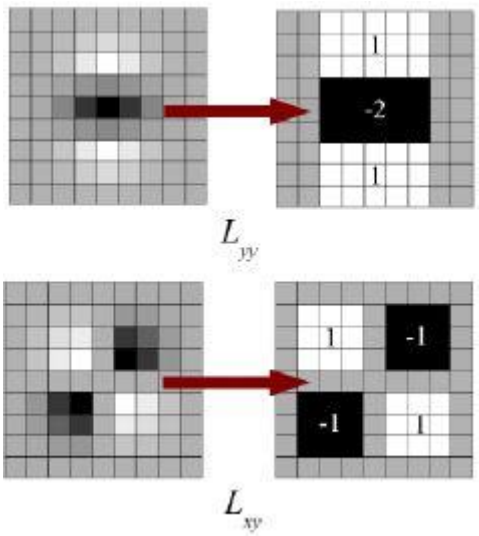


Figure 1: Orientation Histogram

In this example, the histogram peaks at 20-29 degrees. So, the keypoint is assigned orientation 3 (the third bin).

4 SURF (Speeded-Up Robust Features)

The main issue with SIFT was that it was comparatively slow, and a speeded-up version was in demand. SURF approximates Laplacian of Gaussian with Box filters (shown in the image below) rather than the difference of Gaussian as in SIFT. The advantage here is that convolution with box filters may be done easily with the help of integral images and the calculations can be achieved in parallel for different scales [5].



5 Tasks

- Refer to OpenCV documentation
- use SIFT on an image and show detection results
- You may use the code snippet below if you like

```
import numpy as np
import cv2
import sys

def getsize(img):
    h, w = img.shape[:2]
    return w, h

class image_feature_detector(object):

    SIFT = 0
    SURF = 1

    def __init__(self, feat_type,
                 params = None):
        self.detector, self.norm = self.features_detector(feat_type=feat_ty
pe,
                                                          params = params)

    def features_detector(self, feat_type = SIFT, params = None):
        if feat_type == self.SIFT:

            if params is None:
                nfeatures = 0
                nOctaveLayers = 3
                contrastThreshold = 0.04
                edgeThreshold=10
                sigma=1.6
            else:
                nfeatures = params["nfeatures"]
                nOctaveLayers = params["nOctaveLayers"]
                contrastThreshold = params["contrastThreshold"]
                edgeThreshold = params["edgeThreshold"]
                sigma = params["sigma"]

            detector = cv2.xfeatures2d.SIFT_create(nfeatures=0,
                                nOctaveLayers=3, contrastThreshold=0.04,
                                edgeThreshold=10, sigma=1.6)
            norm = cv2.NORM_L2
        elif feat_type == self.SURF:

             if params is None:
                 hessianThreshold = 3000
                 nOctaves = 1
                 nOctaveLayers = 1
                 upright = True
                 extended = False
             else:
                 hessianThreshold = params["hessianThreshold"]
                 nOctaves = params["nOctaves"]
                 nOctaveLayers = params["nOctaveLayers"]
                 upright = params["upright"]
```

```
                extended = params["extended"]

            detector = cv2.xfeatures2d.SURF_create(hessianThreshold = h
essianThreshold,
                                   nOctaves = nOctaves,
                                   nOctaveLayers = nOctaveLayers,
                                   upright = upright,
                                   extended = extended)
            norm = cv2.NORM_L2

        return detector, norm

if __name__ == "__main__":
    sift_detect = image_featuer_detector(feat_type=0)
    surf_detect = image_featuer_detector(feat_type=1)
    # Implement yourself
    ## Refer to opencv documentation,
    ## use SIFT or SURF on the test image and show detection result.
    ### 1. Read Image
    ### 2. Convert the image to greyscale
    ### 3. Initialize an SIFT detector
    ### 4. Feature detection and visualize the detected features
```

6 Questions

1. Run SIFT on at least 2 different images and display the keypoints. How does SIFT ensure rotation invariance?
2. Run SURF on the same images and show the results. How are they different from SIFT?

REFERENCES

[1]. Lowe, David G. "Object recognition from local scale-invariant features. "Computer vision, 1999. The proceedings of the seventh IEEE international conference on. Vol. 2. IEEE, 1999.

[2]. Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." Computer Vision–ECCV 2006. Springer Berlin Heidelberg, 2006. 404-417.

[3]. Mikolajczyk, Krystian, and Cordelia Schmid. "A performance evaluation of local descriptors." Pattern Analysis and Machine Intelligence, IEEE Transactions on27.10 (2005): 1615-1630.

[4]. Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "Speeded Up Robust Features", ETH Zurich, Katholieke Universiteit Leuven

[5]. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html