Image Feature Extraction Using Speeded-up robust features

Brajraj Panwar (0801CS171021)

Sumit Pachaha(0801EC171082)

Pritesh Makwana (0801CS171059)

Bspanwar22010@gmail.com

sumitpachaha14@gmail.com

December 20, 2020

Contents

Chapter 01

Introduction

1.1 SURF

The SURF method (Speeded Up Robust Features) is a fast and robust algorithm for local, similarity invariant representation and comparison of images. The main interest of the SURF approach lies in its fast computation of operators using box filters, thus enabling real-time applications such as tracking and object recognition.

SURF is composed of two steps-

1.Feature Extraction

2.Feature Description

1.2 Feature Extraction

The approach for interest point detection uses a very basic Hessian matrix approximation.

Integral images

The Integral Image or Summed-Area Table was introduced in 1984. The Integral Image is used as a quick and effective way of calculating the sum of values (pixel values) in a given image — or a rectangular subset of a grid (the given image). It can also, or is mainly, used for calculating the average intensity within a given image.

Hessian matrix-based interest points

Surf uses the Hessian matrix because of its good performance in computation time and accuracy. Rather than using a different measure for selecting the location and the scale (Hessian-Laplace detector), surf relies on the determinant of the Hessian matrix for both.

Scale-space representation

Scale spaces are usually implemented as image pyramids. The images are repeatedly smoothed with a Gaussian and subsequently sub-sampled in order to achieve a higher level of the pyramid. Due to the use of box filters and integral images, surf does not have to iteratively apply the same filter to the output of a previously filtered layer but instead can apply such filters of any size at exactly the same speed directly on the original image, and even in parallel.

1.3 Feature Description

The creation of SURF descriptor takes place in two steps. The first step consists of fixing a reproducible orientation based on information from a circular region around the keypoint. Then, we construct a square region aligned to the selected orientation and extract the SURF descriptor from it.

Orientation Assignment

In order to be invariant to rotation, surf tries to identify a reproducible orientation for the interest points. For achieving this:

1.Surf first calculate the Haar-wavelet responses in x and y-direction, and this in a circular neighborhood of radius 6s around the keypoint, with s the scale at which the keypoint was detected. Also, the sampling step is scale dependent and chosen to be s, and the wavelet responses are computed at that current scale s. Accordingly, at high scales the size of the wavelets is big. Therefore integral images are used again for fast filtering.

2.Then we calculate the sum of vertical and horizontal wavelet responses in a scanning area, then change the scanning orientation (add $\pi/3$), and re-calculate, until we find the orientation with largest sum value, this orientation is the main orientation of feature descriptor.

Descriptor Components

Now we to extract the descriptor

1.The first step consists of constructing a square region centered around the keypoint and oriented along the orientation we already got above. The size of this window is 20s.

2.Then the region is split up regularly into smaller 4×4 square sub-regions. For each sub-region, we compute a few simple features at 5×5 regularly spaced sample points.

Chapter 02

Mathematical Formulation

2.1 Integral images

The entry of an integral image I_∑ (x) at a location x = $(x,y)^T$ represents the sum of all pixels in the input image I within a rectangular region formed by the origin and x.

$$I_{\Sigma}(\mathbf{x}) = \sum_{i=0}^{i \leqslant x} \sum_{j=0}^{j \leqslant y} I(i,j)$$

With I_Σ calculated, it only takes four additions to calculate the sum of the intensities over any upright, rectangular area, independent of its size.

2.2 Hessian matrix-based interest points

Given a pixel, the Hessian of this pixel is something like:

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

For adapt to any scale, we filtered the image by a Gaussian kernel, so given a point X = (x, y), the Hessian matrix H(x, σ) in x at scale σ is defined as:

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

where Lxx(x, σ) is the convolution of the Gaussian second order derivative with the image I in point x, and similarly for Lxy (x, σ) and Lyy (x, σ). Gaussians are optimal for scale-space analysis but in practice, they have to be discretized and cropped. This leads to a loss in repeatability under image rotations around odd multiples of π /4. This weakness holds for Hessian-based detectors in general. Nevertheless, the detectors still perform well, and the slight decrease in performance does not outweigh the advantage of fast convolutions brought by the discretization and cropping.

In order to calculate the determinant of the Hessian matrix, first we need to apply convolution with Gaussian kernel, then second-order derivative. After Lowe's success with LoG approximations(SIFT), SURF pushes the approximation(both convolution and second-order derivative) even further with box filters. These approximate second-order Gaussian derivatives and can be evaluated at a very low computational cost using integral images and independently of size, and this is part of the reason why SURF is fast.

Now we can represent the determinant of the Hessian (approximated) as:

$$\det(\mathcal{H}_{\text{approx}}) = D_{xx}D_{yy} - (wD_{xy})^2.$$

2.3 Scale-space representation

the scale space is analyzed by up-scaling the filter size(9×9 → 15×15 → 21×21 → 27×27, etc) rather than iteratively reducing the image size. So for each new octave, the filter size increase is doubled simultaneously the sampling intervals for the extraction of the interest points(σ) can be doubled as well which allow the up-scaling of the filter at constant cost. In order to localize interest points in the image and over scales, a nonmaximum suppression in a 3 × 3 × 3 neighborhood is applied.

Chapter 03

Algorithm :

3.1 Speeded-Up Robust Features Algorithm

Import resources and display image

Step 1. Load the image

Step 2. Convert the training image to RGB

Step 3. Convert the training image to gray scale

Step 4. Create test image by adding Scale Invariance and Rotational Invariance

Step 5. Display training image and testing image

Detect keypoints and Create Descriptor

Step 6. Create SURF

Step 7. Display image with and without keypoints size

Step 8. Detect the number of keypoints detected in the training image

Matching the Keypoints

Step 9. Perform the matching between the SURF descriptors of the training image and the test image

Step 10. Matches with shorter distance

Step 11. Display the best matching points

Step 12. Print total number of matching points between the training and query images

Chapter 04

Documentation of API

4.1 Implementation

Using python and opencv 2.4.6.1-

```python
import cv2
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

image1 = cv2.imread('test.jpg')
training_image = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
training_gray = cv2.cvtColor(training_image, cv2.COLOR_RGB2GRAY)
test_image = cv2.pyrDown(training_image)
test_image = cv2.pyrDown(test_image)
num_rows, num_cols = test_image.shape[:2]

rotation_matrix = cv2.getRotationMatrix2D((num_cols/2, num_rows/2), 30, 1)
test_image = cv2.warpAffine(test_image, rotation_matrix, (num_cols,
num_rows))

test_gray = cv2.cvtColor(test_image, cv2.COLOR_RGB2GRAY)
fx, plots = plt.subplots(1, 2, figsize=(20,10))

plots[0].set_title("Training Image")
plots[0].imshow(training_image)

plots[1].set_title("Testing Image")
plots[1].imshow(test_image)
surf = cv2.xfeatures2d.SURF_create(800)

train_keypoints, train_descriptor = surf.detectAndCompute(training_gray,
None)
test_keypoints, test_descriptor = surf.detectAndCompute(test_gray, None)

keypoints_without_size = np.copy(training_image)
keypoints_with_size = np.copy(training_image)

cv2.drawKeypoints(training_image, train_keypoints, keypoints_without_size,
color = (0, 255, 0))

cv2.drawKeypoints(training_image, train_keypoints, keypoints_with_size,
flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

fx, plots = plt.subplots(1, 2, figsize=(20,10))

plots[0].set_title("Train keypoints With Size")
plots[0].imshow(keypoints_with_size, cmap='gray')
plots[1].set_title("Train keypoints Without Size")
plots[1].imshow(keypoints_without_size, cmap='gray')

print("Number of Keypoints Detected In The Training Image: ",
len(train_keypoints))
```

```
print("Number of Keypoints Detected In The Query Image: ",
len(test_keypoints))
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck = False)
matches = bf.match(train_descriptor, test_descriptor)
matches = sorted(matches, key = lambda x : x.distance)

result = cv2.drawMatches(training_image, train_keypoints, test_gray,
test_keypoints, matches, test_gray, flags = 2)
plt.rcParams['figure.figsize'] = [14.0, 7.0]
plt.title('Best Matching Points')
plt.imshow(result)
plt.show()
print("\nNumber of Matching Keypoints Between The Training and Query
Images: ", len(matches))
```

4.2 Methods

Imread() – reads the image file

cvtColor() – converts the color of the image

pyrDown() - constructing an image pyramid

SURF_create() – create SURF object with a threshold value passed in parameter

detectAndCompute() - Detects keypoints and computes the descriptors

drawKeypoints() – Draw Keypoints on image

cv2.warpAffine() – takes 2x3 matrix as input and performs affine transformation

cv2.getRotationMatrix2D() - rotates the image by 90 degree with respect to center without any scaling

cvtColor() –

Syntax: cv2.cvtColor(src, code[, dst[, dstCn]])
Parameters:
src: It is the image whose color space is to be changed.
code: It is the color space conversion code.
dst: It is the output image of the same size and depth as src image. It is an optional parameter.
dstCn: It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.
Return Value: It returns an image.

Cv2.xfeatures2d.SURF_create(	[, hessianThreshold[, nOctaves[, nOctaveLayers[, extended[, upright]]]]]	)

Parameters

hessianThreshold	Threshold for hessian keypoint detector used in SURF.
nOctaves	Number of pyramid octaves the keypoint detector will use.
nOctaveLayers	Number of octave layers within each octave.
extended	Extended descriptor flag (true - use extended 128-element descriptors; false - use 64-element descriptors).
upright	Up-right or rotated features flag (true - do not compute orientation of features; false - compute orientation).

5.0 Learning Outcomes

We learned that speeded up robust features (SURF) is a local feature detector and descriptor. It can be used for tasks such as object recognition, image registration, classification, or 3D reconstruction. It detects keypoints on the image. To detect key points, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed with 3 integer operations using a precomputed integral image.