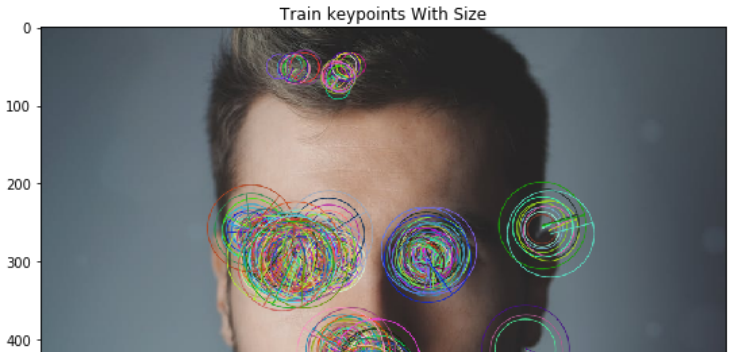
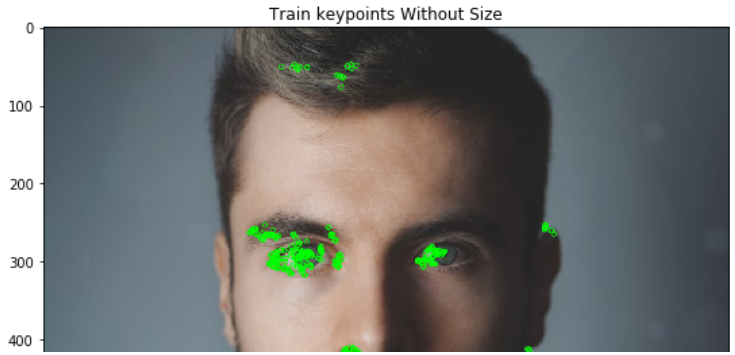
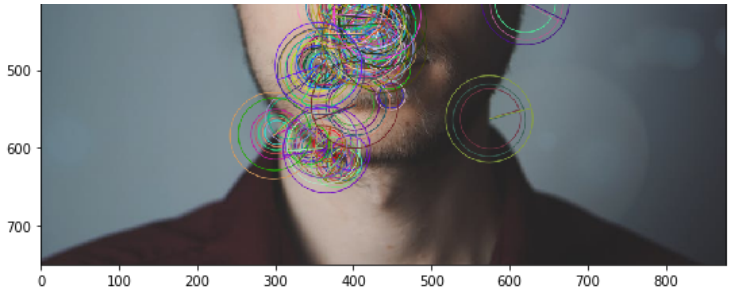
Introduction to ORB (Oriented FAST and Rotated BRIEF)

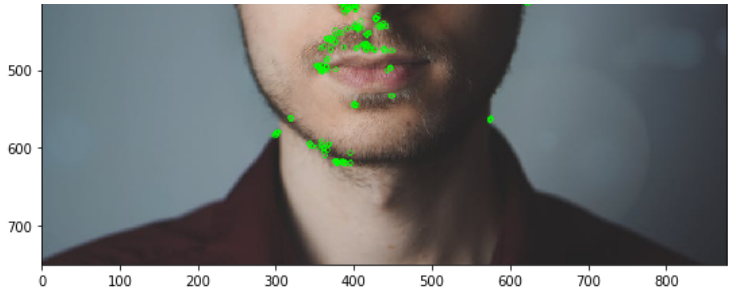
Deepanshu Tyagi Follow
Jan 1 · 6 min read

Oriented FAST and Rotated BRIEF (ORB) was developed at OpenCV labs by Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski in 2011, as an efficient and viable alternative to SIFT and SURF. ORB was conceived mainly because SIFT and SURF are patented algorithms. ORB, however, is free to use. Paper link: http://www.willowgarage.com/sites/default/files/orb_final.pdf









ORB performs as well as SIFT on the task of feature detection (and is better than SURF) while being almost two orders of magnitude faster. ORB builds on the well-known FAST keypoint detector and the BRIEF descriptor. Both of these techniques are attractive because of their good performance and low cost. ORB's main contributions are as follows:

- The addition of a fast and accurate orientation component to FAST
- The efficient computation of oriented BRIEF features
- Analysis of variance and correlation of oriented BRIEF features
- A learning method for decorrelating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications

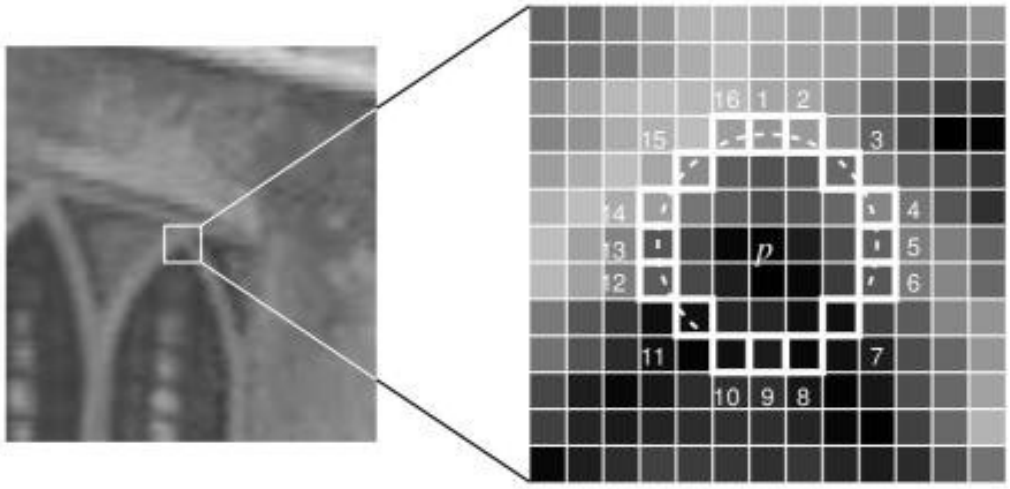
This is part of a 7-series Feature Detection and Matching. Other articles included

- Introduction To Feature Detection And Matching
- Introduction to Harris Corner Detector

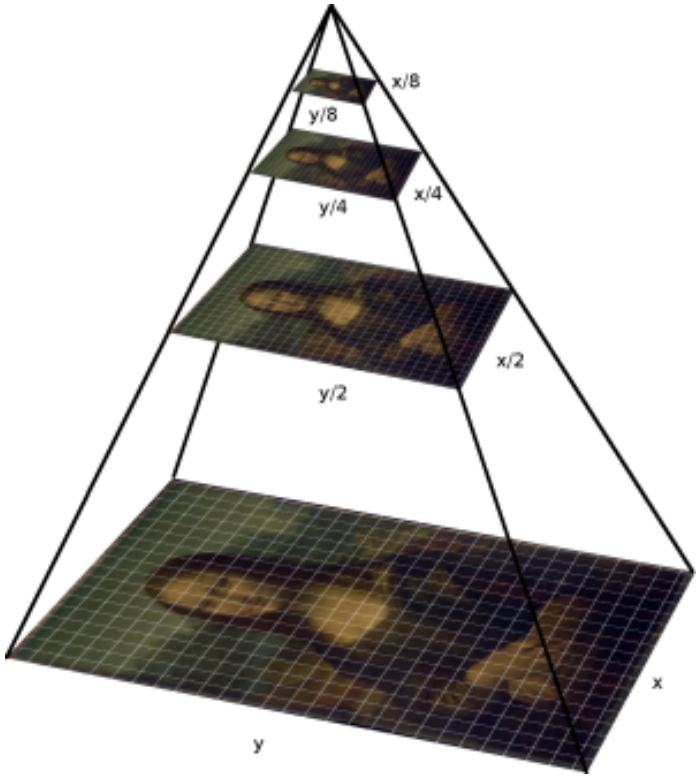
- Introduction to SIFT (Scale-Invariant Feature Transform)
- Introduction to SURF (Speeded-Up Robust Features)
- Introduction to FAST (Features from Accelerated Segment Test)
- Introduction to BRIEF (Binary Robust Independent Elementary Features)

Fast(Features from Accelerated and Segments Test)

Given a pixel p in an array fast compares the brightness of p to surrounding 16 pixels that are in a small circle around p. Pixels in the circle is then sorted into three classes (lighter than p, darker than p or similar to p). If more than 8 pixels are darker or brighter than p than it is selected as a keypoint. So keypoints found by fast gives us information of the location of determining edges in an image. For more information refer to Introduction to FAST (Features from Accelerated Segment Test) Algorithm



However, FAST features do not have an orientation component and multiscale features. So orb algorithm uses a multiscale image pyramid. An image pyramid is a multiscale representation of a single image, that consist of sequences of images all of which are versions of the image at different resolutions. Each level in the pyramid contains the downsampled version of the image than the previous level. Once orb has created a pyramid it uses the fast algorithm to detect keypoints in the image. By detecting keypoints at each level orb is effectively locating key points at a different scale. In this way, ORB is partial scale invariant.



After locating keypoints orb now assign an orientation to each keypoint like left or right facing depending on how the levels of intensity change around that keypoint. For detecting intensity change orb uses intensity centroid. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector may be used to impute an orientation.

First, the moments of a patch are defined as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

ORB descriptor-Patch's moment's definition

With these moments we can find the centroid, the "center of mass" of the patch as:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

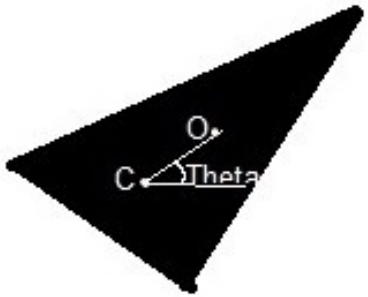
ORB descriptor — Center of the mass of the patch

We can construct a vector from the corner's center O to the centroid -OC. The orientation of the patch is then given by:

$$\theta = \text{atan2}(m_{01}, m_{10})$$

ORB descriptor — Orientation of the patch

Here is an illustration to help explain the method:

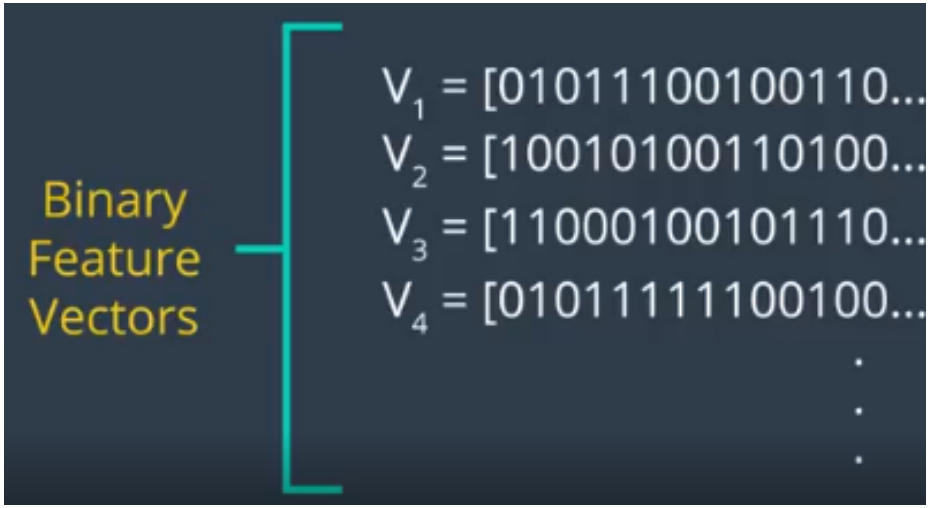


Once we've calculated the orientation of the patch, we can rotate it to a canonical rotation and then compute the descriptor, thus obtaining some rotation invariance.

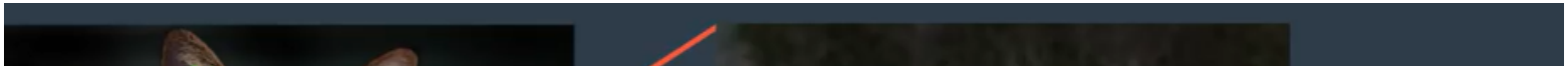
Brief(Binary robust independent elementary feature)

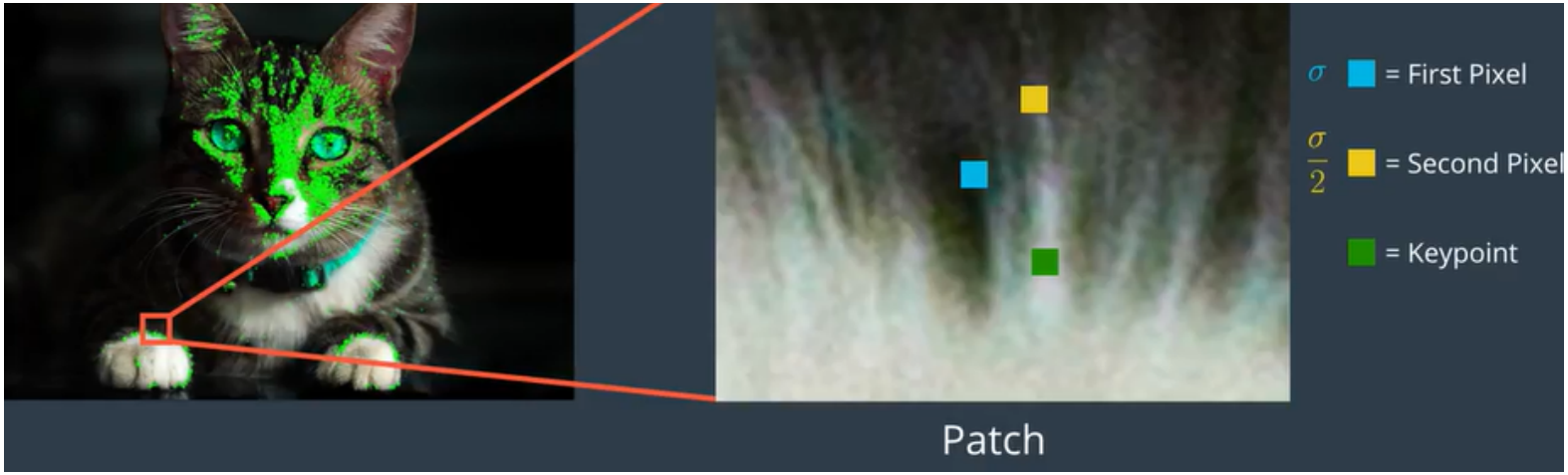
Brief takes all keypoints found by the fast algorithm and convert it into a binary feature vector so that together they can represent an object. Binary features vector also know as

binary feature descriptor is a feature vector that only contains 1 and 0. In brief, each keypoint is described by a feature vector which is 128–512 bits string.



Brief start by smoothing image using a Gaussian kernel in order to prevent the descriptor from being sensitive to high-frequency noise. Than brief select a random pair of pixels in a defined neighborhood around that keypoint. The defined neighborhood around pixel is known as a patch, which is a square of some pixel width and height. The first pixel in the random pair is drawn from a Gaussian distribution centered around the keypoint with a stranded deviation or spread of sigma. The second pixel in the random pair is drawn from a Gaussian distribution centered around the first pixel with a standard deviation or spread of sigma by two. Now if the first pixel is brighter than the second, it assigns the value of 1 to corresponding bit else 0.





Again brief select a random pair and assign the value to them. For a 128-bit vector, brief repeat this process for 128 times for a keypoint. Brief create a vector like this for each keypoint in an image. However, BRIEF also isn't invariant to rotation so orb uses rBRIEF(Rotation-aware BRIEF). ORB tries to add this functionality, without losing out on the speed aspect of BRIEF.

Consider a smoothed image patch, p. A binary test τ is defined by:

$$Where\ \tau(p;x,y)\ is\ defined\ as:$$

$$\tau(p;x,y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases}$$

$$p(x)\ is\ the\ \text{int}ensity\ value\ at\ pixel\ x.$$

where p(x) is the intensity of p at a point x.The feature is defined as a vector of n binary tests:

$$f(n) = \sum_{1<i<n} 2^{i-1} \tau(p; x_i, y_i)$$

The matching performance of BRIEF falls off sharply for in-plane rotation of more than a few degrees. ORB proposes a method to steer BRIEF according to the orientation of the keypoints. For any feature set of n binary tests at location (xi, yi), we need the 2 x n matrix:

$$S = \begin{pmatrix} x1, \ldots x_n \\ y1, \ldots y_n \end{pmatrix}$$

It uses the patch orientation θ and the corresponding rotation matrix Rθ, and construct a steered version Sθ of S:

$$S_\theta = R_\theta S$$

Now, the steered BRIEF operator becomes:

$$g_n(p,\theta) = f_n(p) \mid (\mathbf{x}_i, \mathbf{y}_i) \in S_\theta$$

Than it discretizes the angle to increments of 2π/30 (12 degrees), and construct a lookup table of precomputed BRIEF patterns. As long as the keypoint orientation θ is consistent across views, the correct set of points Sθ will be used to compute its descriptor.

ORB specifies the rBRIEF algorithm as follows:

1) Run each test against all training patches.

2) Order the tests by their distance from a mean of 0.5, forming the vector T.

3) Greedy search:

- Put the first test into the result vector R and remove it from T.

- Take the next test from T, and compare it against all tests in R. If its absolute correlation is greater than a threshold, discard it; else add it to R.
- Repeat the previous step until there are 256 tests in R. If there are fewer than 256, raise the threshold and try again

rBRIEF shows significant improvement in the variance and correlation over steered BRIEF.

Implementation

I was able to implement ORB using OpenCV. Here's how I did it:

In [5]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Load the image
image1 = cv2.imread('./images/face1.jpeg')

# Convert the training image to RGB
training_image = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

# Convert the training image to gray scale
training_gray = cv2.cvtColor(training_image, cv2.COLOR_RGB2GRA
Y)

# Create test image by adding Scale Invariance and Rotational
 Invariance
```

```
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_row
s/2), 30, 1)
test_image = cv2.warpAffine(test_image, rotation_matrix, (num_
cols, num_rows))

test_gray = cv2.cvtColor(test_image, cv2.COLOR_RGB2GRAY)
```

orb.ipynb hosted with ❤️ by GitHub view raw

ORB Implementation

Github link for the code: https://github.com/deepanshut041/feature-detection/tree/master/orb

References

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html#orb
- https://in.udacity.com/course/computer-vision-nanodegree--nd891
- https://gilscvblog.com/2013/10/04/a-tutorial-on-binary-descriptors-part-3-the-orb-descriptor/
- https://www.oreilly.com/library/view/computer-vision-with/9781788299763/

Thanks for reading! If you enjoyed it, hit that clap button below and follow Software Incubator for more updates

Computer Vision | Opencv | Image Recognition | Image Processing