BRIEF: Binary Robust Independent Elementary Features

We know SIFT uses 128-dim vector for descriptors. Since it is using floating point numbers, it takes basically 512 bytes. Similarly SURF also takes minimum of 256 bytes (for 64-dim). Creating such a vector for thousands of features takes a lot of memory which are not feasible for resouce-constraint applications especially for embedded systems. Larger the memory, longer the time it takes for matching.

But all these dimensions may not be needed for actual matching. We can compress it using several methods (like PCA, LDA, etc). Even other methods like hashing using LSH (Locality Sensitive Hashing) is used to convert these SIFT descriptors in floating point numbers to binary strings. These binary strings are used to match features using Hamming distance. This provides better speed-up because finding hamming distance is just applying XOR and bit count, which are very fast in modern CPUs with SSE instructions. But here, we need to find the descriptors first, then only we can apply hashing, which doesn't solve our initial problem on memory.

BRIEF comes into picture at this moment. It provides a shortcut to find the binary strings directly without finding descriptors. It takes smoothened image patch and selects a set of n_d (x,y) location pairs in an unique way (explained in paper). Then some pixel intensity comparisons are done on these location pairs. Eg, let first location pairs be p and q. If $I(p)<I(q)$, then its result is 1, else it is 0. This is applied for all the n_d location pairs to get a n_d-dimensional bitstring.

This nd can be 128, 256 or 512. OpenCV supports all of these, but by default, it would be 256 (OpenCV represents it in bytes. So the values will be 16, 32 and 64). So once you get this, you can use Hamming Distance to match these descriptors.

Binary descriptor is composed out of three parts:

1. A sampling pattern: where to sample points in the region around the descriptor.
2. Orientation compensation: some mechanism to measure the orientation of the keypoint and rotate it to compensate for rotation changes.
3. Sampling pairs: the pairs to compare when building the final descriptor.

One important point is that BRIEF is a feature descriptor, it doesn't provide any method to find the features (1). So you will have to use any other feature detectors like SIFT, SURF etc. The paper recommends to use CenSurE which is a fast detector and BRIEF works even slightly better for CenSurE points than for SURF points. But also FAST can be used.

In short, BRIEF is a faster method feature descriptor calculation and matching. It also provides high recognition rate unless there is large in-plane rotation.

ORB: Oriented FAST and Rotated BRIEF

This algorithm was brought up by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in their paper ORB: An efficient alternative to SIFT or SURF in 2011.

As the title says, it is a good alternative to SIFT and SURF in computation cost, matching performance and mainly the patents. Yes, SIFT and SURF are patented and you are supposed to pay them for its use. But ORB is not !!!

ORB is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. First it use FAST to find keypoints, then apply Harris corner measure to find top N points among them. It also use pyramid to produce multiscale-features. But one problem is that, FAST doesn't compute the orientation. So what about rotation invariance? Authors came up with following modification.

It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation. To improve the rotation invariance, moments are computed with x and y which should be in a circular region of radius r, where r is the size of the patch.

Now for descriptors, ORB use BRIEF descriptors. But we have already seen that BRIEF performs poorly with rotation. So what ORB does is to "steer" BRIEF according to the orientation of keypoints.

There are two main differences between ORB and BRIEF:

1. ORB uses an orientation compensation mechanism, making it rotation invariant.
2. ORB learns the optimal sampling pairs, whereas BRIEF uses randomly chosen sampling pairs.

Orientation Compensation

For any feature set of n binary tests at location (xi,yi), define a 2×n matrix, S which contains the coordinates of these pixels. Then using the orientation of patch, θ, its rotation matrix is found and rotates the S to get steered(rotated) version Sθ.

ORB discretize 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.

Let’s see step by step:

ORB uses a simple measure of corner orientation – the intensity centroid. First, the moments of a patch are defined as:

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

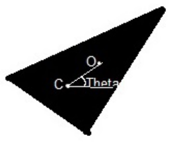
With these moments we can find the centroid, the “center of mass” of the patch as:

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

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

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

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.

Learning the sampling pairs

There are two properties we would like our sampling pairs to have. One is uncorrelation – we would like that the sampling pairs will be uncorrelated so that each new pair will bring new information to the descriptor, thus maximizing the amount of information the descriptor carries. The other is high variance of the pairs – high variance makes a feature more discriminative, since it responds differently to inputs.

The authors of ORB suggest learning the sampling pairs to ensure they have these two properties. A simple calculation [1] shows that there are about 205,000 possible tests (sampling pairs) to consider. From that vast amount of tests, only 256 tests will be chosen.

The learning is done as follows. First, they set a training set of about 300,000 keypoints drawn from the PASCAL 2006 dataset. Next, we apply the following greedy algorithm:

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:
    1. Put the first test into the result vector R and remove it from T.
    2. 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.
    3. Repeat the previous step until there are 256 tests in R. If there are fewer than 256, raise the threshold and try again.

Once this algorithm terminates, we obtain a set of 256 relatively uncorrelated tests with high variance.

To conclude, ORB is binary descriptor that is similar to BRIEF, with the added advantages of rotation invariance and learned sampling pairs.

You're probably asking yourself, how does ORB perform in comparison to BRIEF? Well, in non-geometric transformation (those that are image capture dependent and do not rely on the viewpoint, such as blur, JPEG compression, exposure and illumination) BRIEF actually outperforms ORB. In affine transformation, BRIEF perform poorly under large rotation or scale change as it’s not designed to handle such changes. In perspective transformations, which are the result of view-point change, BRIEF surprisingly slightly outperforms ORB.

For descriptor matching, multi-probe LSH which improves on the traditional LSH, is used. The paper says ORB is much faster than SURF and SIFT and ORB descriptor works better than SURF. ORB is a good choice in low-power devices for panorama stitching etc.

OpenCV implementation

C++:

ORB(int nfeatures=500, float scaleFactor=1.2f, int nlevels=8, int edgeThreshold=31, int firstLevel=0, int WTA_K=2, int scoreType=ORB::HARRIS_SCORE, int patchSize=31)

Parameters:

nfeatures – The maximum number of features to retain.

scaleFactor – Pyramid decimation ratio, greater than 1. scaleFactor==2 means the classical pyramid, where each next level has 4x less pixels than the previous, but such a big scale factor will degrade feature matching scores dramatically. On the other hand, too close to 1 scale factor will mean that to cover certain scale range you will need more pyramid levels and so the speed will suffer.

nlevels – The number of pyramid levels. The smallest level will have linear size equal to input_image_linear_size/pow(scaleFactor, nlevels).

edgeThreshold – This is size of the border where the features are not detected. It should roughly match the patchSize parameter.

firstLevel – It should be 0 in the current implementation.

WTA_K – The number of points that produce each element of the oriented BRIEF descriptor. The default value 2 means the BRIEF where we take a random point pair and compare their brightnesses, so we get 0/1 response. Other possible values are 3 and 4. For example, 3 means that we take 3 random points (of course, those point coordinates are random, but they are generated from the pre-defined seed, so each element of BRIEF descriptor is computed deterministically from the pixel rectangle), find point of maximum brightness and output index of the winner (0, 1 or 2). Such output will occupy 2 bits, and therefore it will need a special variant of Hamming distance, denoted as NORM_HAMMING2 (2 bits per bin). When WTA_K=4, we take 4 random points to compute each bin (that will also occupy 2 bits with possible values 0, 1, 2 or 3).

scoreType – The default HARRIS_SCORE means that Harris algorithm is used to rank features (the score is written to KeyPoint::score and is used to retain best nfeatures features); FAST_SCORE is alternative value of the parameter that produces slightly less stable keypoints, but it is a little faster to compute.

patchSize – size of the patch used by the oriented BRIEF descriptor. Of course, on smaller pyramid layers the perceived image area covered by a feature will be larger.