Implementing the Scale Invariant Feature Transform(SIFT) Method

YU MENG and Dr. Bernard Tiddeman(supervisor)
Department of Computer Science
University of St. Andrews
yumeng@dcs.st-and.ac.uk

Abstract

The SIFT algorithm[1] takes an image and transforms it into a collection of local feature vectors. Each of these feature vectors is supposed to be distinctive and invariant to any scaling, rotation or translation of the image. In the original implementation, these features can be used to find distinctive objects in differerent images and the transform can be extended to match faces in images. This report describes our own implementation of the SIFT algorithm and highlights potential direction for future research.

1 Introduction & Background

Face recognition is becoming an increasingly important for many applications including human-machine interfaces, multimedia, security, communication, visually mediated interaction and anthropomorphic environments. One of the most difficult problems is that the process of identifying a person from facial appearance has to be performed differently for each image, because there are so many conflicting factors altering facial appearance. Our implementaion focuses on deriving SIFT features from an image and trying using these features to perform face identification.

The approach of SIFT feature detection taken in our implementation is similar with the one taken by Lowe et. [1], which is used for object recognition. According to their work, the invariant features extracted from images can be used to perform reliable matching between different views of an object or scene. The features have been shown to be invariant to image rotation and scale and robust across a substantial range of affine distortion, addition of noise, and change in illumination. The approach is efficient on feature extraction and has the ability to identify large numbers of features.

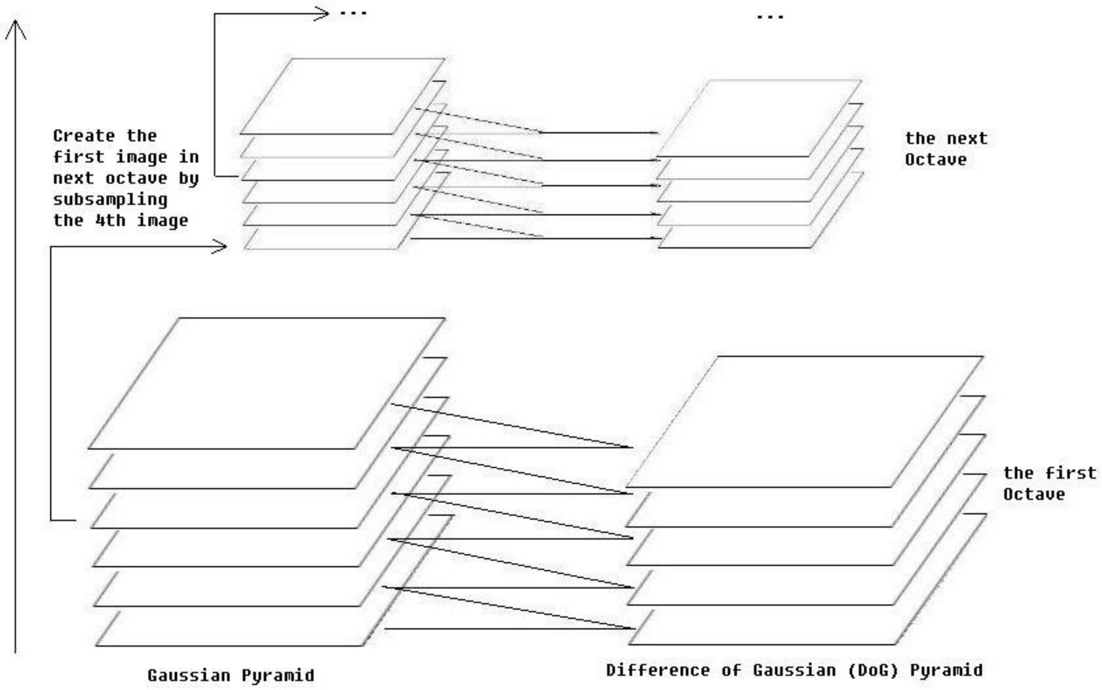

Figure 1: Bottom: On the left is the Gaussian pyramid, with neighbouring images separated by a constant scale factor. These are subtracted to give the DoG pyramid on the right. Top: The Gaussian with σ twice that of the original is subsampled and used to construct the next octave.

Face identification is supposed to discriminate between different faces. In lowe's[1] paper, the nearest-neighbour template matching is presented. The nearest neighbor is defined as the keypoint with minimum Euclidean distance for the invariant descriptor vector. However, different specific stragegies are taken for different problems. If there are multiple training images of the same face, then we define the second-closest neighbor as being the closest neighbor that is known to come from a differnt face than the first.

Because of the variablity in facial appearance statistical methods are often used to do help with the localisation of facial features. Helmer. et.[2] has taken an approach which is time efficient and can be used with complicated models. They use SIFT features grouped using a probabilistic model initialized with a few parts of an object. They learn the parts incrementally and try adding possible parts to the model during the process of training. Then they use the Expectation-Maximization(EM) algorithm to update the model.

Our method is implemented as the following stages: Creating the Difference of Gaussian Pyramid, Extrema Detection, Noise Elimination, Orientation Assignment, Descriptor Computation, Keypoints Matching.

2 Creating the Difference of Gaussian Pyramid

The first stage is to construct a Gaussian "scale space" function from the input image [1]. This is formed by convolution (filtering) of the original image with Gaussian functions of varying widths. The difference of Gaussian (DoG), D(x, y, σ), is calculated as the difference between two filtered images, one with k multiplied by scale of the other.

$$D(x, y, \sigma) = L(x, y, k\sigma)\text{-}L(x, y, \sigma) \qquad (1)$$

These images, L(x, y, σ), are produced from the convolution of Gaussian functions, G(x, y, kσ), with an input image, I(x, y).

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \qquad (2)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2}\exp\{-(x^2+y^2)/2\sigma^2\} \qquad (3)$$

This is the approach we use in the implementation. First, the initial image, I, is convolved with a Gaussian function, G_0, of width σ_0. Then we use this blured image, L_0, as the first image in the Gaussian pyramid and incrementally convolve it with a Gaussian, G_i, of width σ_i to create the i^{th} image in the image pyramid, which is equivalent to the original image filtered with a Gaussian, G_k, of width $k\sigma_0$. The effect of convolving with two Gaussian functions of different widths is most easily found by converting to the Fourier domain, in which convolution becomes multiplication i.e.

$$G_{\sigma_i} * G_{\sigma_0} * f(x) \rightarrow \tilde{G}_{\sigma_i}\tilde{G}_{\sigma_0}\tilde{f} \qquad (4)$$

The Fourier transform of a Gaussian function, e^{ax^2} is given by.

$$F_x[e^{ax^2}](t) = \sqrt{\frac{\pi}{a}}\,e^{-\pi^2(t)^2/a} \qquad (5)$$

Substituting this into equation (4) and equating it to a covolution with a single Gaussian of width $k\,\sigma_0$ we get:

$$e^{-t^2\sigma_i^2}e^{-t^2\sigma_0^2} = e^{-t^2k^2\sigma_0^2} \qquad (6)$$

Performing the multiplication of the two exponentials on the left of this equation and comparing the coefficients of $-t^2$ gives:

$$\sigma_i^2 + \sigma_0^2 = k^2\sigma_0^2 \qquad (7)$$

And so we get:

$$\sigma_i = \sigma_0\sqrt{k^2-1} \qquad (8)$$

This subtle point is not made clear in the original paper, and it is important because after subsampling of the low-passed filtered images to form the lower levels of the pyramid we no longer have access to the original image at the appropriate resolution, and so we cannot filter with

G_k directly.

Each octave of scale space in divided into an integer number, *s*, of intervals and we let $k=2^{1/s}$. We produce s+3 images for each octave in order to form s+2 difference of Gaussian (DoG) images and have plus and minus one scale interval for each DoG for the extrema detection step. In this experiment, we set s to 3, following on from experimental results in [1] that suggests that this number produces the most stable key-points. Once a complete octave has been processed, we subsample the Gaussian image that has twice the initial value of σ by taking every second pixel in each row and column. This greatly improves the efficiency of the algorithm at lower scales. The process is shown in Figure 1.

3 Extrema Detection

This stage is to find the extrema points in the DOG pyramid. To detect the local maxima and minima of D(x, y, σ), each point is compared with the pixels of all its 26 neighbours (Figure 2). If this value is the minimum or maximum this point is an extrema. We then improve the localisation of the keypoint to subpixel accuracy, by using a second order Taylor series expansion. This gives the true extrema location as:

$$z=-\left(\frac{\partial^2 \boldsymbol{D}}{\partial \boldsymbol{x}^2}\right)^{-1}\frac{\partial \boldsymbol{D}}{\partial \boldsymbol{x}} \tag{9}$$

where D and its derivatives are evaluated at the sample point and $x=(x\,,y\,,\sigma)^T$ is the offset from the sample point.

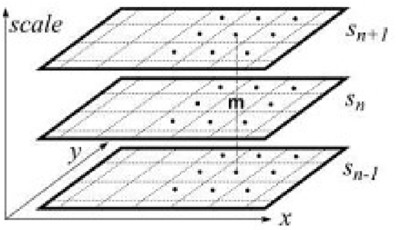

Figure 2: An extrema is defined as any value in the DoG greater than all its neighbours in scale-space.

4 Key points Elimination

This stage attempts to eliminate some points from the candidate list of keypoints by finding those that have low contrast or are poorly localised on an edge.[1]. The value of the keypoint in the DoG pyramid at the extrema is given by:

$$D(z)=D+\frac{1}{2}\frac{\partial D^{-1}}{\partial x}z \tag{10}$$

If the function value at **z** is below a threshold value this point is excluded.

To eliminate poorly localised extrema we use the fact that in these cases there is a large principle curvature across the edge but a small curvature in the perpendicular direction in the difference of Gaussian function. A 2x2 Hessian matrix, H, computed at the location and scale of the keypoint is used to find the curvature. With these fomulas, the ratio of princepal curvature can be checked efficiently.

$$H=\begin{bmatrix} D_{xx}\,D_{xy} \\ D_{xy}\,D_{yy} \end{bmatrix} \tag{11}$$

$$\frac{D_{xx}+D_{yy}}{D_{xx}D_{yy}-(D_{xy})^2}<\frac{(r+1)^2}{r} \tag{12}$$

So if inequality (12) fails, the key point is removed from the candidate list.

5 Orientation Assignment

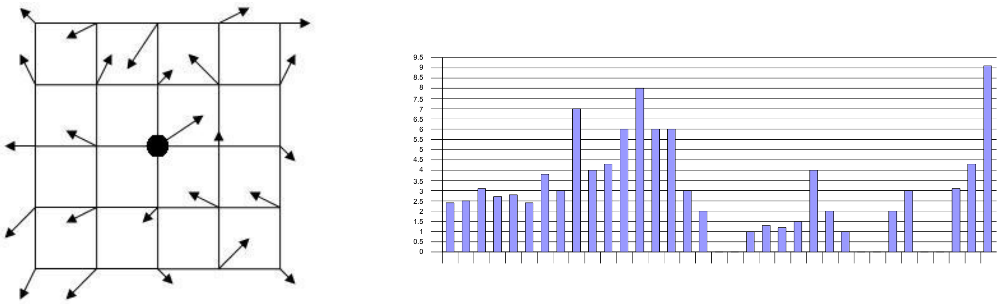

Figure 3: Left: The point in the middle of the left figure is the keypoint candidate. The orientations of the points in the square area around this point are precomputed using pixel differences. Right: Each bin in the histogram holds 10 degree, so it covers the whole 360 degree with 36 bins in it. The value of each bin holds the magnitude sums from all the points precomputed within that orientation.

This step aims to assign a consistent orientation to the keypoints based on local image properties. An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint as illustrated in Figure 3. A 16x16 square is chosen in this implementation. The orientation histogram has 36 bins covering the 360 degree range of orientations[1]. The gradient magnitude, m(x, y), and orientation, θ(x, y), is precomputed using pixel differences:

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \tag{13}$$

$$\theta(x,y)=\arctan((L(x,y+1)-L(x,y-1))/(L(x+1,y)-L(x-1,y))) \tag{14}$$

Each sample is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint.

Peaks in the orientation histogram correspond to dominant directions of local gradients. We locate the highest peak in the histogram and use this peak and any other local peak within 80% of the height of this peak to create a keypoint with that orientation. Some points will be assigned multiple orientations if there are multiple peaks of similar magnitude. A Gaussian distribution is fit to the 3 histogram values closest to each peak to interpolate the peaks position for better accuracy.

This computes the location, orientation and scale of SIFT features that have been found in the image. These features respond strongly to the corners and intensity gradients. The SIFT features appear mostly in the eyes, nostrils, the top of the nose and the corners of the lips for face images. In Figure 4, keypoints are indicated as arrows. The length of the arrows indicates the magnititude of the contrast at the keypoints, and the arrows point from the dark to the bright side.

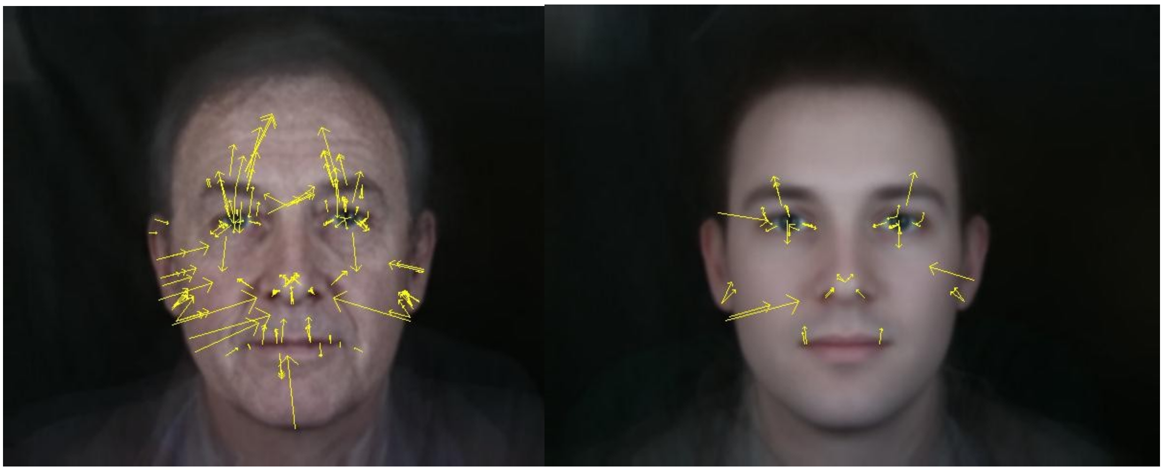

Figure 4: Keypoint vectors are drawn on the two men's face images . Most of the features appear on the eyes, nostrils, the top of nose, the corners of the mouth and the earlobes.

6 Descriptor Computation

In this stage, a descriptor is computed for the local image region that is as distinctive as possible at each candidate keypoint. The image gradient magnitudes and orientations are sampled around the keypoint location. These values are illustrated with small arrows at each sample location on the first image of Figures. A Gaussian weighting function with σ related to the scale of the keypoint is used to assign a weight to the magnitude. We use a σ equal to one half the width of the descriptor window in this implementation. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. This process is indicated in Figure 5. In our implementation, a 16x16 sample array is computed and a histogram with 8 bins is used. So a descriptor contains 16x16x8 elements in total.

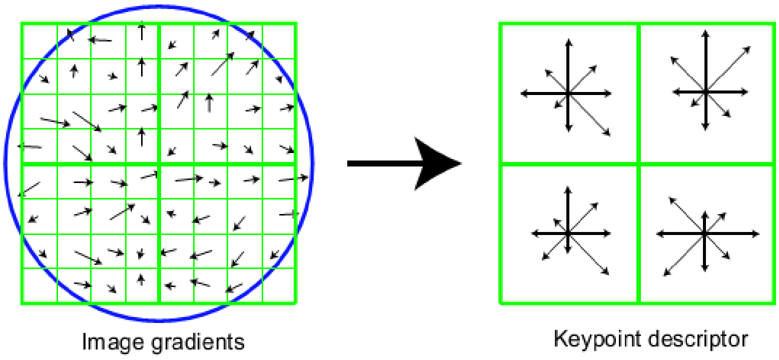

Figure 5: Left: the gradient magnitude and orientation at a sample point in a square region around the keypoint location. These are weighted by a Gaussian window, indicated by the overlaid circle. Right: The image gradients are added to an orientation histogram. Each histogram include 8 directions indicated by the arrows and is computed from 4x4 subregions. The length of each arrow corresponds to the sum of the gradient magnitudes near that direction within the region.

7 Transformation

In this stage, some matching tests are running to test the repeatability and stability of the SIFT features. An image and a transformed version of the image are used as indicated in Figure 6. The features of the two images are computed separately. Then each keypoint in the original image (model image) is compared to every keypoints in the transformed image using the descriptors computed in the previous stage. For each comparison, one feature is picked in each image. f1 is the descriptor array for one key point in the original image and f2 is the descriptor array for a key point in the transformed image. The most likely value for each pair of the keypoints is computed by:

$$error_{ij}=\sum_{i=0}^{i=n1}\sum_{j=0}^{j=n2}|f1_i-f2_j| \tag{15}$$

If the number of features in the two images is n1 and n2, then there are n1*n2 possible pairs altogether. All these data are sorted in ascending order of matching error. Then the first two qualified pairs of the keypoints are chosen to set the transformation.

To apply the transform, the following functions are introduced into the implementation. The transform gives the mapping of a model image point (x, y) to a transformed image point (u, v) in terms of an image scaling, s, an image rotation, θ, and an image translation, [t_x , t_y]: [5]

$$\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}s\cos\theta & -s\sin\theta\\ s\sin\theta & s\cos\theta\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}t_x\\t_y\end{bmatrix} \tag{16}$$

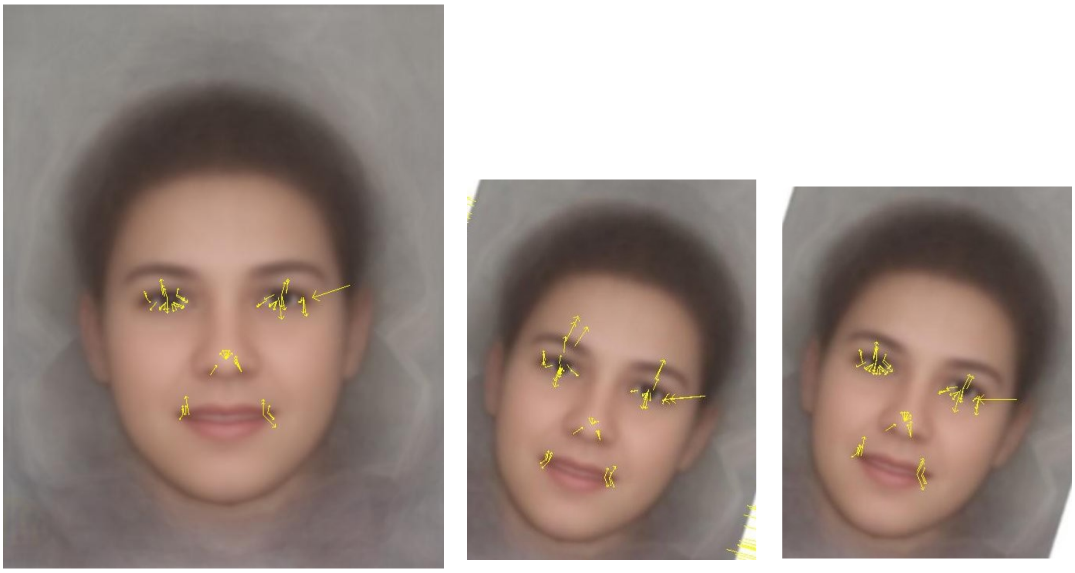

Figure 6: The first image is the original image with it's own features. The second image is the transformation of the original with the features after operation. The third one is the transformation of the original image with the features before the operation.

8 Results

Results from our implementation are shown in Figures 3, 4 and 6. We can currently calculate the SIFT features for an image and have experimented with some simple matching schemes between images. Noise adjustment is a very essential part for our approach which could result in inefficient or false matching. However, we have used parameters which should help the keep the feature matching robust to noise in this implementation.

9 Conclusion and Future Work

The SIFT features described in our implementation are computed at the edges and they are invariant to image scaling, rotation, addition of noise. They are useful due to their distinctiveness, which enables the correct match for keypoints between faces. These are achieved by using our Gradient-Based Edge Detector and the local descriptors presented around the keypoints.

Edges are poorly defined and usually hard to detect, but there are still large numbers of keypoints can be extracted from typical images. So we can still perform the feature matching even the faces are small. Sometimes the images are too smooth to find that many features for a matching, and in that case a small face could be unrecognized from the training images. The recognition performance could be improved by adding new SIFT fetures or varying feature sizes and offsets.

In the next step, we will try to perform some face identification, and we choose the nearest neighbor or second-closest neighbor algorithm which is a good method to do the keypoints matching. There is another useful method to recognize faces by learning a statistical model [2]. In this method, a probalistic model is used to recognize the faces. An Expectation-Maximization(EM) algorithm is used to learn the parameters in a Maximum Likelihood framework. Hopefully, we can limit the model to a small amount of parts which is efficient for matching faces.

With the useful information represented as the feature-based model, we can find many directions as the applications. We may try to track persons through a carema by tracking SIFT features. We may render 3D pictures using the SIFT features extracted from still images. To achieve that, we could try to fit the SIFT features of a specific face to a 3D face model we've created. That's one of the attractive aspect of the invariant local feature method we're using.

References

[1]Lowe, D.G..2004. Distinctive Image Features from Scale-Invariant Keypoints. January 5, 2004

[2]Helmer, S.&Lowe,D.G..Object Class Recognition with Many Local Features.

[3]Gordon, I.&Lowe,D.G..Scene Modelling, Recognition and Tracking with Invariant image Features.

[4]Mikolajczyk, K., and Schmid, C. 2002. An affine invariant interest point detector. In European Conference on Computer Vision (ECCV), Copenhagen, Denmark, pp. 128-142.

[5] Lowe, D.G. 2001. Local feature view clustering for 3D object recognition. IEEE Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii, pp. 682-688.

[6] Matching Images with Different Resolutions.

[7] Mikolajczyk, K. & Schmid, C. 2001. Indexing based on scale invariant interest points. International Conference on Computer Vision, Vancouver, Canada (July 2001), pp. 525--531.

[8] Mikolajczyk, K. & Schmid, C. 2002. An Affine Invariant Interest Point Detector. ECCV.

[9]Forsyth, D.A.&Ponce, J..(2003) *Computer Vision – A Modern Approach*. New Jersey:Prentice Hall