The Scale Invariant Feature Transform

Leopold Gaube

15.01.2017

Abstract

The following paper is supposed to give an overview of how the Scale Invariant Feature Transform, short SIFT, works. It was introduced by David G. Lowe in 1999 and enhanced multiple times until he published his final version in 2004. This algorithm extracts distinctive features from images which can be used for matching objects across multiple images. The first part of this paper introduces the basic concept of SIFT and also presents a range of possible applications. Subsequently, every essential step that leads to the creation of SIFT features is being explained in detail. In contrast to all the advantages of using SIFT, some of its disadvantages are being examined in the third section. Finally, it concludes with a brief summary of the entire paper.

1 Introduction

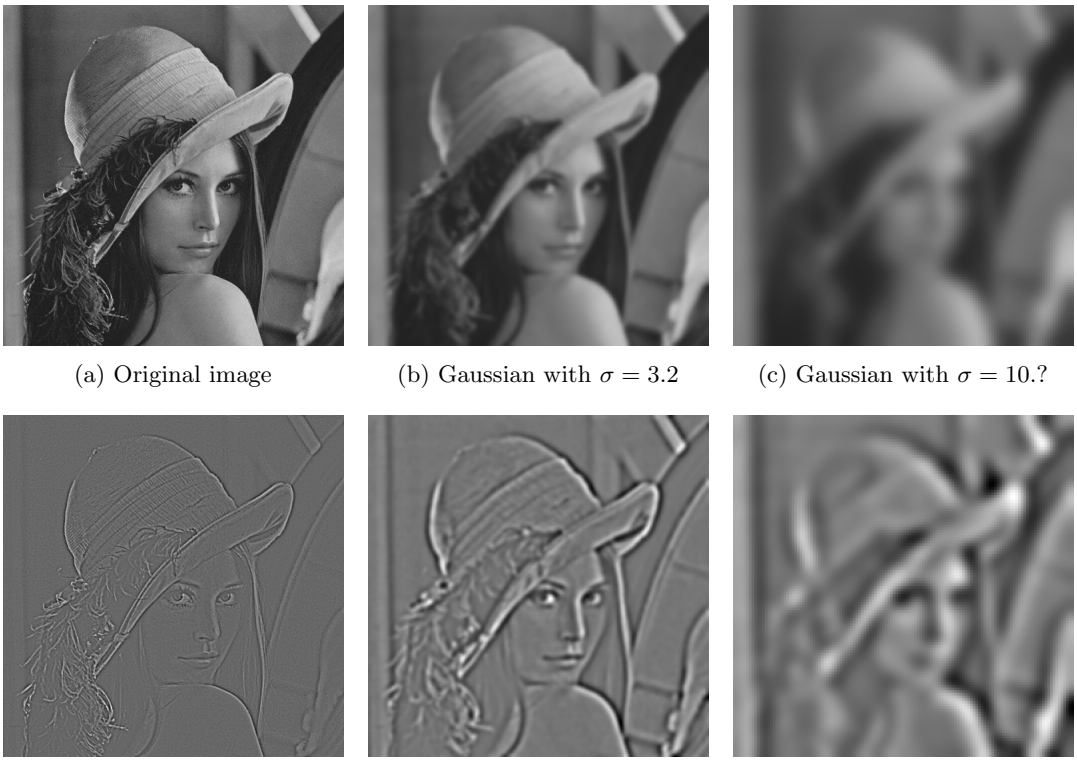
1.1 Motivation and Usage

The main goal of the Scale Invariant Feature Transform is finding distinct points in an image which are likely to appear in a different image, depicting the same object(s). These distinct points are called keypoints and are being used to compute a feature descriptor which summarizes the local neighbourhood around that keypoint's location. The idea behind such a feature-based algorithm is to find many corresponding keypoints across multiple images and trying to match them using their feature descriptors. Even a small amount of correct matches can be extremely useful in a range of computer vision tasks: Lowe proposed a way of how SIFT features can be used to recognize objects even under difficult conditions. A Hough transformation helps to detect objects even if they are partially occluded. Furthermore, SIFT can also be used for tracking objects in videos. It is also possible to use SIFT for recognizing human gestures, which serves to control applications or even tell robots what to do. In 2007 Lowe collaborated with Matthew Brown and together they published a paper on how to use these invariant features to stitch partially overlapping images together, resulting in a single panoramic image. [reference: Automatic Panoramic Image Stitching using Invariant Features]

2 Extracting SIFT Features

2.1 Selecting keypoints from scale-space extrema

The first step of the SIFT algorithm is to select potential keypoints. It would be a bad idea for an image feature-based algorithm to consider every possible pixel location, because computation for a single image would take a long time and most features would be unusable for accurate localization due to lack of image information. This is why we have to choose keypoints based on its recognition value. Therefore, we only want to select keypoints that are highly likely to be found in a different image showing the same scene. This means a good keypoint should be invariant to numerous transformations such as rotation, scale, distortion and brightness, but also to change in illumination, 3D viewpoint and addition of noise. Of course, it would be hard to satisfy all these



(a) Original image (b) Gaussian with $\sigma = 3.2$ (c) Gaussian with $\sigma = 10.?$

(d) DoG with σ_1=1.6?, σ_2=2.0? (e) DoG with σ_1=4?, σ_2=6? (f) DoG with σ_1=12?, σ_2=14?

Figure 1: This figure shows the impact of the Gaussian blur with different scales σ on an image in its top row. Every image at the bottom depicts a normalized version of the difference between such two Gaussian images. A gray pixel value indicates no difference between the two Gaussians at that location, whereas black and white pixels reveal their disparity. [original image taken from https://upload.wikimedia.org/wikipedia/en/2/24/Lenna.png]

properties, but the Scale Invariant Feature Transform satisfies most of them which will be further examined in this paper.

In order to select keypoints from an image, Lowe uses a so-called scale-space representation of that image. This means the image will be smoothed with different scales σ of the Gaussian kernel. Smoothing an image with a low σ value will suppress fine structures, whereas smoothing with a large σ only leaves rough contours. The results of the Gaussian blurring can be seen in figure 1. The resulting images stack-up to form a so-called Gaussian pyramid. Lowe has found that using an initial $\sigma = 1.6$ and gradually adjusting each ensuing scale of the Gaussian blur by a constant factor of $k = 2^{1/3}$ will achieve the best results [reference paper]. After every three blurred images - which also means after each doubling of the σ scale - the image can be downsampled by a factor of two. All images of the same size in the pyramid are considered to form an octave. The accuracy of sampling relative to σ stays roughly the same for all octaves, but results in major computational performance boosts. Subtracting one Gaussian blurred image pixelwise from another gives us a Difference of Gaussians, short DoG. Doing this for every two neighbored images of the same octave in the Gaussians pyramid forms another pyramid of multiple Difference of Gaussians with one image less in each octave then before. In order to compensate for this lost image, we previously need to compute one more Gaussian blurred image at the end of each octave. Keep in mind that the resampling for the first image of the next octave should to be done on the same image as before. To finally retrieve the keypoints for our original image, each pixel of every Difference of Gaussian is compared to its eight pixel neighbors as well as its 18 neighbors from the DoGs laying

directly above and below it in the pyramid [see figure 2.x]. A pixel's location is taken into the set of our potential keypoints only if its value is either a minimum or a maximum in its surrounding neighborhood. Searching for extrema cannot be done on the first and last DoG image in each octave, since they have only one adjacent DoG of the same size, so they are omitted. Again, we need to generate two more Gaussians at the end of each octave, because it is favourable to extract extrema on three different sizes of σ before moving on to the next octave. As mentioned before smoothing an image with different strengths results in images with different amount of detail. This is exactly what we need in order to find distinctive features no matter how large or small they are depicted in the picture. A minimum or maximum in scale-space means that a structure is still visible in one Gaussian image, but has been "smoothed away" by the stronger Gaussian kernel of the next. This leads to a large pixel value difference along the structure between these two Gaussians. A Difference of Gaussians has therefore a strong response to edges, as explicated in "Gaussian-based edge-detection methods-a survey" by M. Basu[reference]. If we look at two images depicting the same object, but one taken from further away, we will still find corresponding keypoints, but on different scales in the DoG pyramid. This emphasizes the scale invariance property of SIFT features.

2.2 Keypoint refinement

Our next step in the SIFT algorithm is to predict the accurate location and scale of our previously selected keypoints on a subpixel precision level. It is done by fitting a 3D quadratic function to the sample points that we have chosen from the scale-space extrema.

2.3 Keypoint filtering

Some of the keypoints we obtained earlier might not serve for good features due to a lack of contrast or unstable localization along an edge, so we need to reject such keypoints. A low contrast also means that we might not find corresponding feature in an image that has slightly been modified from the original. Contrast can easily be measured by the Tayler expansion of the scale space function shifted by the precise location $\hat{x}$ of the extremum:

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x}$$

Lowe found that if the absolute value of that function is less than 0.03, then there is just not enough contrast for a stable feature to be computed at that location. Even if there is enough contrast the algorithm might still struggle to accurately localize a keypoint along an edge. This is why we rather have corner-like keypoints than only edge-like. To further illustrate this problem, we can simply look at an arbitrary thick white line on black background as shown in Figure 2(a). If we were to cut a patch from the middle section of the line 2(b), it would be impossible to tell at which exact location on the line the patch originated from. A patch of one of the end sections, as shown in 2(c), can easily be matched to its correct location at the corresponding end of the line. In order to mathematically determine whether a given keypoint is corner-like and therefore stable or not, we can use a 2x2 Hessian matrix H, where D_{xx}, D_{xy}, D_{yx} and D_{yy} are approximated derivatives using neighbouring sample points at the keypoints location and scale:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$$

Given H, we can look at the ratio of its two eigenvalues. Basically speaking, if both eigenvalues are of the same magnitude, then we are dealing with a keypoint which shows a strong curvature. However, if one eigenvalue is significantly bigger than the other, we will discard this keypoint from our set, because it is unusable for our purposes due to its similarity to an edge.

Figure 2: The left image shows a straight line. The other two are upscaled patches taken from the left image. Even for a human, it would be impossible to tell from which exact location the patch in the middle originally came from. However, the patch on the right can easily be localized at the left end of the line.

2.4 Orientation assignment

We have already shown that SIFT-Features are invariant to scale, but rotational invariance is just as important for numerus image processing tasks. For this purpose, Lowe assigns each keypoint one or more consistent orientations, based on its neighborhood's gradients. An image gradient indicates a change of pixel intensity in a specific direction. The most prominent gradient direction of a keypoint seems to be a good pick for assignment of a consistent orientation, because rotating an image by an angle ϕ will only shift all gradients by ϕ as well; thus, maintaining the same relative directions. To find the most prominent gradient direction, we first need to compute all gradients around a keypoint and organize them in a histogram. A histogram shows the distributing of numerical data by dividing the data in disjoint categories, called bins. In our application we will have 36 bins and each bin represents a range of 10° out of all 360° possible angle directions. For every Gaussian blurred image, L, all gradient directions and magnitudes are precomputed for performance purposes. The direction of a gradient at a certain location in L can be calculated with the following formula using pixel differences:

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

The gradient's magnitude shows how strong the pixel intensity changes in the gradient's direction. It can be calculated in a similar manner:

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

The magnitude of each gradient is weighted by a Gaussian function, so that gradients closer to the center of the keypoint have a higher impact than gradients further away which tend to be more unstable to minor transformations. This weighted input will be added to the bin which represents the gradient's direction. The highest peak in that histogram is the direction we are looking for. Any other peaks that holds a value higher than 80% of the highest peak leads to the creation of an additional keypoint at the same location, but with a different orientation. These keypoints with multiple strong gradient directions are scarce, but help to make matching more stable. A parabola is fit to the three closest values of each peak to achieve a more accurate orientation at the real maximum of that peak. (Picking up the patch matching problem of a straight line from earlier (Figure 3). Both end patches would generate similar descriptors after they have been assigned an orientation, thus matching their keyypoints could proof problematic)

2.5 Constructing the feature descriptor

There are many possibilities to describe the area around a keypoint. For instance we could be storing the intensity or color values inside our feature descriptor. However, the disadvantage of this approach is that many images are taken under different lighting conditions. Hence intensity values of corresponding keypoints might vary a lot. Image gradients are a better choice for constructing a feature descriptor, because they only coincide with change of pixel intensity, rather than intensity values itself. SIFT uses all gradients within a 16x16 neighbourhood around each keypoint. These gradients will further be organized in 4x4 cells, so there will be a total of 16 cells. Analogous to section 2.4 each cell accumulates its gradients in an orientation histogram, but this time with only 8 bins. Hence each bin represents 45° which allows for small inaccuracies of the gradients directions, but keeps the final descriptor small... The feature descriptor is actually just a sequence of linking all values of every cell histogram together in a consistent manner. Every keypoint uses 4x4 cells with 8 orientation bins each. Thus accumulating all these values in a single descriptor, results in a 128 dimensional vector. Experiments show that normalizing the vector to unit length helps to cope with illumination changes. By limiting every orientation value in the cell histogram to a maximum of 0.2, we guarantee that no single dominant gradient can suppress the other gradients. Have multiple strong orientations in our final descriptor is more desirable than having a single dominant one, because "non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts" [Zitat 6.1 paper].

3 Problematic situations

We have seen that SIFT features are highly thought-out, but of course they might fail their purpose in some situations. SIFT features can be computed relatively fast, however matching over a vast database might still take a long time. Herbert Bay addresses this problem in his publication "Speeded Up Robust Features" (SURF) from 2006. His SURF features are based on SIFT and can be computed even faster. Their smaller 64 dimensional descriptor allows for faster matching of its features with only minor loss in accuracy. Hence they might be better suited for some applications which require real-time. In one picture a subject might be well lit by sunshine, whereas another picture of the same scene might be taken in cloud weather. Hence the first will have high pixel intensity values and also strong contrast due to shadows, whereas the other will have low intensity values and flat contrast. Contrast... Shadows also tend to cause strong gradients which might corrupt some descriptors.

4 Conclusion

The SIFT algorithm is a great way to extract distinct points from an image and describe their surrounding area by constructing a feature descriptor in an invariant manner. These descriptors can in turn be used to match corresponding points across images. Their invariance and robustness properties allow for stable matching even if the reference images differ in scale, rotation or lighting of the depicted object(s). Slight changes in 3D viewing angle are also accounted for. Correctly matching a small amount of these distinct points is the major foundation of many image processing tasks like object recognition, panorama stitching and gesture recognition. SIFT features are not perfect for every situations as shown in the last section. Nevertheless, the Scale Invariant Feature Transform has been a milestone in computer vision and provided the fundamentals for many following image feature-based algorithms.