Scale-Invariant Feature Transform

Introduction

One of the more interesting problems in the field of computer vision is the task of finding good keypoints in images, i.e., distinct locations in an image that allow for the relation of two or more images. A good set of keypoints forms the basis for a deeper analysis of visual information and plays a crucial role in tasks such as optical flow estimation, object tracking, and even 3D reconstruction based on images. Finding such features is non-trivial and requires a robust computational model with low uncertainty. Several interesting approaches have been proposed over the years, such as Canny's edge detector and Harris' corner detector; however, it is the *Scale-Invariant Feature Transform (SIFT)* that has established itself as one of the most widely used algorithms. Compared to the former two algorithms, which are prone to being affected by noise and may miss information, SIFT robustly detects features across different scales and resolutions. The features detected are known as *blobs*, regions in an image with approximately constant properties compared to the rest of the image. In a specially processed image, a blob looks something like this:

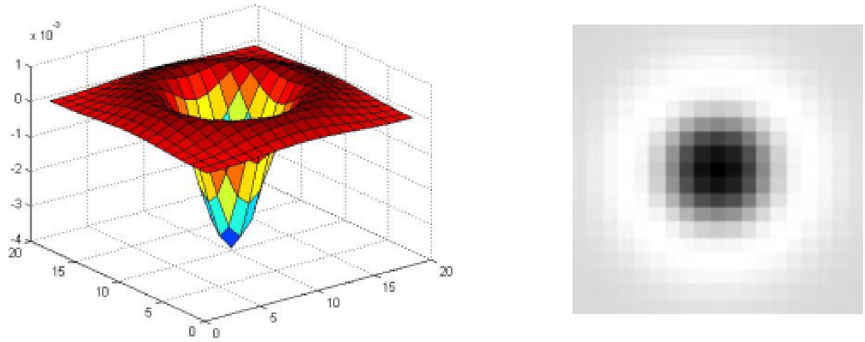

Figure 1: Visualization of a blob feature.

Once the features have been detected, they need to be described in a way that allows matching of said features across images. In other words, using the created description, we wish to find features that are present in both images, also known as *correspondences*, and use this to relate the images. The final outcome of such matching is illustrated in fiugre 2.

While the *OpenCV* library provides an efficient pre-coded version of the SIFT algorithm, and many other feature detectors, it leaves out all the details about the algorithm's inner workings. My aim is thus to understand in detail the process of feature detection and matching, and I am doing so by utilizing the functionality of OpenCV and the C++ programming language to build a working version of SIFT from the bottom up. The goal is to achieve similar detection

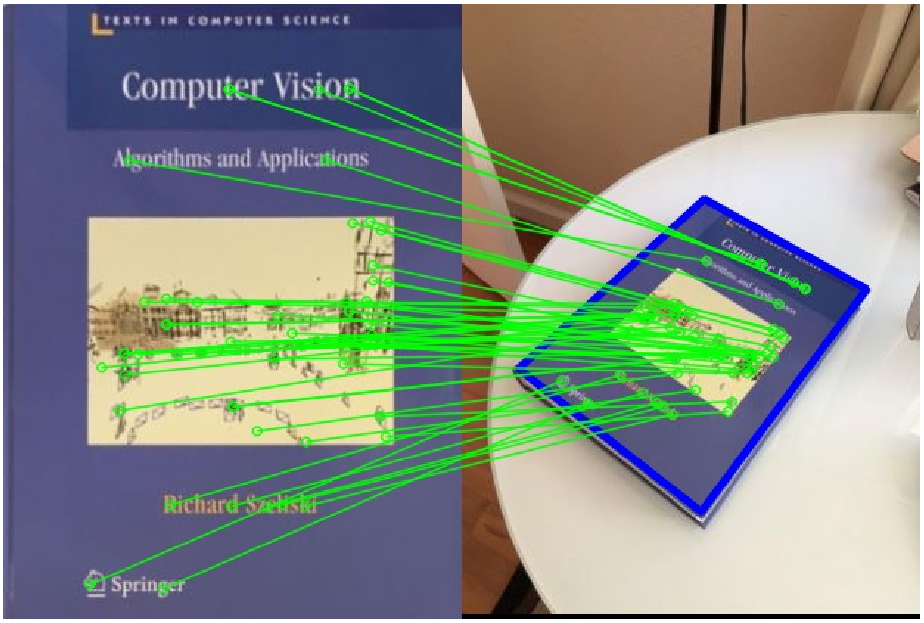

Figure 2: Example of feature matching using SIFT descriptors.

ability within a reasonable amount of time, preferably the same magnitude of runtime as the OpenCV version. In the following section we will take a thorough look at the algorithimc backbone of the SIFT algorithm. Before proceeding, it is assumed that the reader is familiar with the methods used in the field of computer vision.

Method

The SIFT is a method comprised of two main parts: The first part is the detection of blob features in the provided image(-s), and the second step is concerned with creating descriptors for each of the detected features. Both of these steps involve running of several algorithms which form the basis of SIFT. In the following section I will dive into the intuition behind each of the steps in the algorithm. For brevity, matematical notation will be left out, however the reader will be pointed to relevant sources for more details. The terms *pyramid* and *scale-space* will be used interchangeably below.

Step I: Detection of points

Creating the Guassian scale space The capacity of the SIFT algorithm to detect features across scales, encompassing various levels of resolution and image blur, is attributed to the utilization of image pyramids within the detection process. In image processing, an image pyramid denotes a multi-scale image representation, where each image in the pyramid undergoes filtering and downsampling relative to the original image. In the context of the SIFT algorithm, the input image undergoes sequential blurring with a range of Gaussian kernels, generating multiple scales, denoted as m, before it undergoes downsampling, with the blurring process being iteratively applied. With each downsampling step, the final image from the preceding octave (a level within the pyramid comprising images of identical resolution) serves as the input for the subsequent octave. The process of constructing the Gaussian scale space is elegantly depicted in Figure 3. The end result is a scale-space construct with m scales and n octaves.

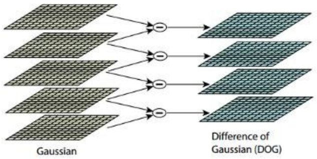

Figure 3: Gaussian scale-space.

Approximation of the Laplacian of the scale-space The original paper on SIFT [Distinctive Image Features from Scale-Invariant Keypoints] defines keypoint features as the extreme points in the normalized Laplacian scale-space, representing the second derivative of the scale-space. However, computing the Laplacian space is inefficient. Fortunately, it can be approximated using the Difference-of-Gaussians (DoG) approach at a lower computational cost. To construct the DoG scale-space, one must first generate a Gaussian scale-space by repeatedly convolving the input image with Gaussian kernels at different scales. Then, the DoG scale-space is obtained by performing element-wise subtraction of two consecutive images in the Gaussian scale-space at each octave and storing the resulting image in a new pyramid. This process is illustrated in Figure 3. The DoG images highlight regions of significant change in intensity, aiding in the identification of potential keypoints.

Detection and refinement of keypoints After creating the DoG scale space, the initial candidate keypoints are identified by scanning each image in the DoG pyramid for extrema points. An extremum is defined as the maximum or minimum pixel value within a 3x3 window across the current and adjacent scales, i.e., the previous and next scales in the current octave. These points are then filtered using a threshold value C_{DoG} to eliminate points with low contrast, thereby removing false detections. The next step involves a second stage of filtering, focused on removing candidate points located at edges. This filtering procedure involves computing a 2D Hessian matrix in the DoG scale space and a measure of edgeness defined as the ratio between the trace and determinant of the Hessian matrix. If the value of edgeness exceeds the following ratio:

$$\frac{(C_{\text{edge}}+1)^2}{C_{\text{edge}}}$$

the point is discarded from the list of candidate points. After this two-stage elimination of false detections, the location of the keypoints in the input image is refined using quadratic interpolation. For a detailed algorithmic and mathematical description, I highly recommend reading the original paper [Distinctive Image Features from Scale-Invariant Keypoints] and the [Anatomy of the SIFT Method].

Step II: Comptuing the decriptor of the keypoints

Computing the keypoint reference orientation The Sift approach computes descriptors with the property of being of being rotation-invariant. To achieve this, it uses the approach of computating a local dominant gradient angle for a small image patch around hte keypoint as the reference orientation of the keypoint. The SIFT algorithm assigns this dominant gradient orientation to each keypoint in three steps: extracting a normalized patch, creating and smoothing an orientation histogram, and identifying local maxima in the histogram to select reference orientations. This process ensures robust, rotation-invariant descriptors by analyzing the image gradient distribution around keypoints in two Gaussian neighborhoods of different sizes. An illustration of the image gradient can be found in figure 4.

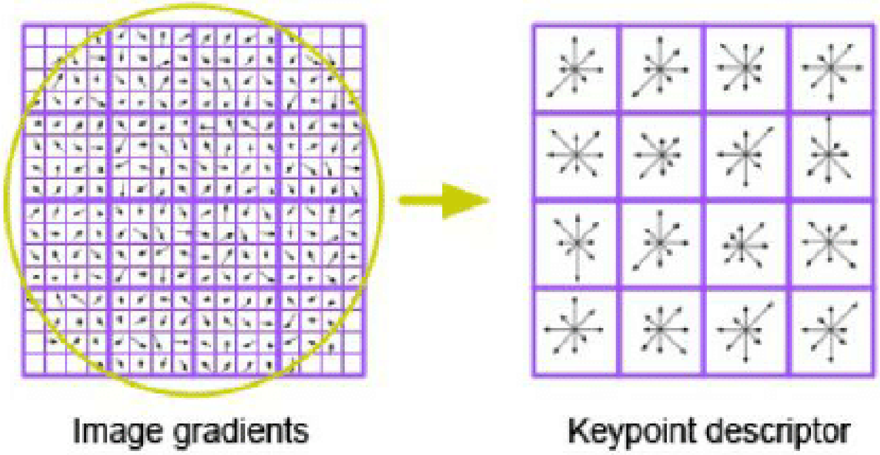

Figure 4: Visual depiction of the image gradients used for the computation of descriptors

Computing the normalized keypoint descriptor At each detected keypoint, the algorithm analyzes a small patch surrounding the point to assess gradient distribution. This patch is subdivided into smaller segments, and histograms of gradient orientations are constructed for each segment. These histograms are then concatenated into a single feature vector, creating a compact representation of the keypoint. This meticulous process ensures feature recognition resilience against lighting variations and noise, rendering SIFT descriptors a crucial tool for various computer vision applications.

Matching of the keypoints To match keypoints between images, descriptors, which are numerical representations of keypoints, are compared by minimizing their Euclidean distances. This comparison helps identify pairs of keypoints that are similar across images. However, not all matches are equally reliable. Thresholds, either absolute or relative using the second nearest neighbor, are employed to determine the reliability of matches. This ensures that only matches meeting certain similarity criteria are considered valid, improving the accuracy of the matching process.

Results

The results achieved are quite impressive with the version I programmed detecting similar keypoints to the SIFT algorithm provided by the OpenCV library, except for several points that could be identified as outliers, deviating by a small margin from the other points. A comparison of achieved results between my version and OpenCV is presented below in images titled *Sift_result1.png* and *Sift_result_opencv.png* included with this report. Additionally, the result of image matching between two images is presented in iamge file titled *Sift_result_matching.png.* Concerning the achieved speed, my algorithm demonstrated a running time of around 3.9 to 4.1 seconds for the detection of keypoint features in the image presented in figure **??**, compared to an elapsed time of 0.03 seconds achieved by OpenCV

Conclusion

"While I successfully programmed a SIFT algorithm capable of producing feature detections almost similar to the OpenCV version upon visual inspection, the running time of my implementation was approximately two orders of magnitude higher than the OpenCV code. This significant disparity suggests a considerable potential for numerical optimization, particularly in applications such as visual odometry or structure from motion. Additionally, I could have measured the deviation in the coordinates of keypoints detected by my SIFT version against those detected by OpenCV using techniques such as Mean Squared Error."

References

Anatomy of the SIFT Method - Rey-Otero, I., & Delbracio, M. Distinctive Image Features from Scale-Invariant Keypoints - Lowe, D. G.