An Open-Source SIFT Library

Rob Hess
School of EECS, Oregon State University
Corvallis, Oregon, USA
hess@eecs.oregonstate.edu

ABSTRACT

Recent years have seen an explosion in the use of invariant keypoint methods across nearly every area of computer vision research. Since its introduction, the scale-invariant feature transform (SIFT) has been one of the most effective and widely-used of these methods and has served as a major catalyst in their popularization. In this paper, I present an open-source SIFT library, implemented in C and freely available at `http://eecs.oregonstate.edu/~hess/sift.html`, and I briefly compare its performance with that of the original SIFT executable released by David Lowe.

Categories and Subject Descriptors

I.4.7 [**Computing Methodologies**]: Image Processing and Computer Vision—*Feature Measurement*; D.0 [**Software**]: General

General Terms

Algorithms

Keywords

Open-Source, SIFT, Library, Keypoints, Image Features

1. INTRODUCTION

Invariant local image features fill a fundamental role in computer vision by facilitating the computation of image correspondences at both the point and patch levels. Due to advances in recent years in the detection and description of robust local features, their use has become prevalent in nearly every area of computer vision research, from 3D vision [12, 5], to object recognition [6, 9], to robot localization and mapping [14, 11], to object tracking [3, 13], and almost everywhere in between.

The scale-invariant feature transform, or SIFT algorithm [7, 8], is today among the most well-known and widely-used invariant local feature methods, and because it was one of the first of these methods to combine invariance to rotation, scale, and a wide range of both affine transformation and illumination change with a robust descriptor that can be reliably matched against a large database, the SIFT algorithm itself played a major role in driving the popularity of invariant local image feature methods in the early part of the last decade.

Unfortunately, despite SIFT's immense popularity, David Lowe, SIFT's creator, released the algorithm only in binary executable format, leaving the need for a general-purpose, linkable library of SIFT routines that could be easily incorporated by developers into computer vision software. As part of my own computer vision research, I implemented in C a version of the SIFT algorithm—based faithfully on Lowe's seminal 2004 paper [8]—using the popular open-source computer vision library OpenCV [10]. Convinced of its potential usefulness to the general computer vision community, I released my SIFT implementation in 2006 as an open-source library. At the time of its release, this was the first open-source version of the SIFT algorithm publicly available, and since its release, it has grown considerably in popularity.[1]

In this paper, I describe in brief detail the SIFT algorithm and my open-source SIFT library's implementation of it, and I briefly compare the performance of the SIFT library with that of the original SIFT executable.

2. THE SIFT ALGORITHM

The SIFT algorithm operates in four major stages to detect and describe local features, or keypoints, in an image:

1. Detection of extrema in scale space
2. Sub-unit localization and filtering of keypoints
3. Assignment of canonical orientations to keypoints
4. Computation of keypoint descriptors

Scale-space extrema detection. The SIFT algorithm begins by identifying the locations of candidate keypoints as the local maxima and minima of a difference-of-Gaussian pyramid that approximates the second-order derivatives of the image's scale space. The interested reader should refer to [8] for a thorough justification of this approach.

Keypoint localization and filtering. After candidate keypoints are identified, their locations in scale space are interpolated to sub-unit accuracy, and interpolated keypoints with low contrast or a high edge response—computed based

[1]The open-source SIFT library described here is available at `http://eecs.oregonstate.edu/~hess/sift.html`.

on the ratio of principal curvatures—are rejected due to potential instability.
Orientation assignment. The keypoints that survive filtering are assigned one or more canonical orientations based on the dominant directions of the local scale-space gradients. After orientation assignment, each keypoint's descriptor can be computed relative to the keypoint's location, scale, and orientation to provide invariance to these transformations.
Descriptor computation. Finally, a descriptor is computed for each keypoint by partitioning the scale-space region around the keypoint into a grid, computing a histogram of local gradient directions within each grid square, and concatenating those histograms into a vector. To provide invariance to illumination change, each descriptor vector is normalized to unit length, thresholded to reduce the influence of large gradient values, and then renormalized.

Again, the interested reader should refer to [8] for a more detailed description of the SIFT algorithm.

3. THE OPEN-SOURCE SIFT LIBRARY

The open-source SIFT library is written in C, with versions available for both Linux and Windows, and it uses the popular open-source computer vision library OpenCV [10]. In particular, the SIFT library's function API uses OpenCV data types to represent images, matrices, etc., making it easy to incorporate SIFT functions into existing OpenCV-based vision code. In addition, all internal operations in the SIFT library are performed using OpenCV functions.

The SIFT library itself contains four main components, each represented by a different header file. I describe these separately below. Afterwards, I describe three simple example applications that are also included with the SIFT library.

3.1 SIFT Library Components

SIFT keypoint detection. The main component of the library is a set of functions for detecting SIFT keypoints. Specifically, the library contains two SIFT keypoint detection functions (located in the `sift.h` header file), one that computes SIFT keypoints using the default parameter settings suggested in Lowe's paper [8] and another that allows the user to set parameters as they desire.

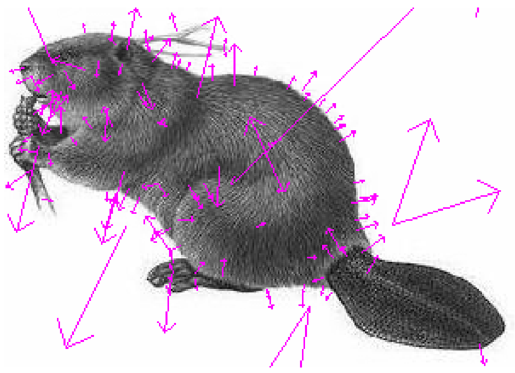
These functions are designed to be easy to call. Specifically, they require no calls to initialization functions and accept both grayscale and RGB images (RGB images are converted to grayscale internally). In particular, the following code snippet is all that is necessary to compute SIFT features in a color image loaded from file.

```
IplImage* img;              /* OpenCV image type */
struct feature* keypoints; /* SIFT library keypoint type */
int n;                      /* feature count */

/* load image using OpenCV and detect keypoints */
img = cvLoadImage( "/path/to/image.png", 1 );
n = sift_features( img, &keypoints );
```

Figure 1 depicts keypoints detected using the SIFT library. For comparison, keypoints detected using David Lowe's executable SIFT software[2] are also depicted in Figure 1.
Kd-tree keypoint database formation. The ability to efficiently match SIFT keypoints from a given image against ones from another image or from a large keypoint database is fundamental. In [1], Beis and Lowe describe a method

[2]`http://www.cs.ubc.ca/~lowe/keypoints/`


(a) Open-source SIFT Library

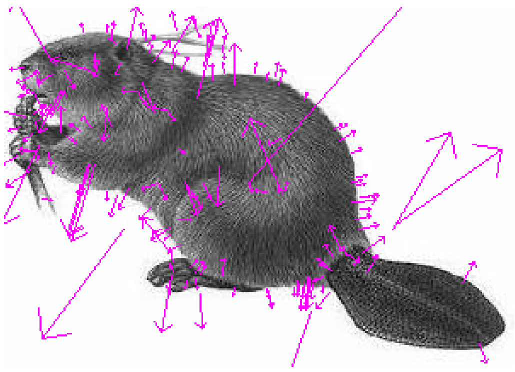
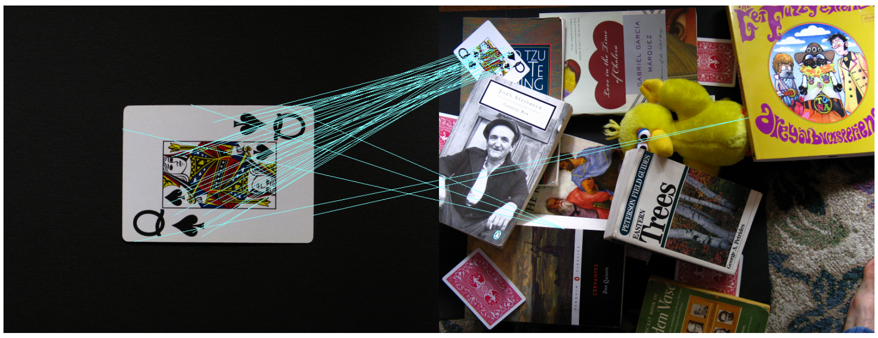

(b) Lowe's SIFT Executable

Figure 1: SIFT keypoints detected using (a) the open-source SIFT library described in this paper, and (b) David Lowe's SIFT executable.

to facilitate efficient keypoint matching using a kd-tree and an approximate (but correct with very high probability) nearest-neighbor search. The SIFT library also contains structures and functions (located in the `kdtree.h` header file) implementing this method, as well as the local keypoint matching method described in [5].
RANSAC transform computation. SIFT keypoints and other local image features are commonly used to compute transforms—fundamental matrices or planar homographies, for example—between images. In particular, once image features are matched between the images, the correspondences thus formed can be used to analytically compute the desired transform. The RANSAC algorithm [2] is widely used to perform this computation under the possible presence of outlier feature matches.

Included with the SIFT library (in the `xform.h` header file) is a set of functions for using RANSAC to compute image transforms from feature matches. These functions are designed to be flexible. In particular, the transform function itself is an argument to the library's RANSAC function. Thus, the developer is free to implement any function he or she wishes for computing transforms from 2D point correspondences. The implementation must only comply with the function prototypes defined in the library. As an example, the library includes functions that can be used in conjunction with RANSAC to compute planar homographies between images.


(a)

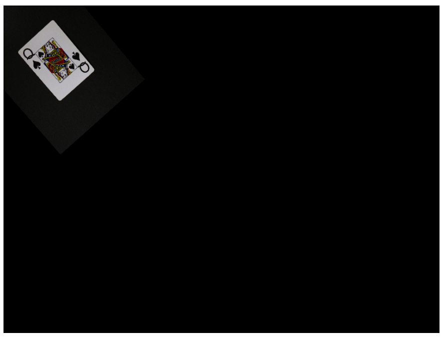

(b)

Figure 2: (a) Matches computed between SIFT keypoint in two images using the SIFT library's kd-tree functions. (b) A transform computed between the two images based on the keypoint matches in (a) using the SIFT library's RANSAC functions.

Figure 2 depicts SIFT keypoint matches computed between two images using the SIFT library's kd-tree functions described above and a transform computed based on the matched keypoints using the library's RANSAC functions.
Invariant image feature handling. Finally, the SIFT library also contains a set of structures and functions for working with invariant image feature data, including data generated by other software. In particular, this component of the library contains a data structure to represent image feature data and functions to import and export keypoints computed using the library's own SIFT functions, as well as SIFT features computed using David Lowe's SIFT executable and the affine-covariant features computed by the Oxford Visual Geometry Group's software[3]. Using this functionality (located in the `imgfeatures.h` header file), the kd-tree and RANSAC functions described above can be applied to any of these types of features.

3.2 Example applications

The SIFT library also contains three very simple example applications—described below—that demonstrate the library's functionality.

- `siftfeat.c`: This application simply computes SIFT keypoints in an image and exports them to file. The keypoints depicted in Figure 1(a) were computed using this application.

- `match.c`: This application computes matches between SIFT keypoints detected in two images using the library's kd-tree functions and optionally computes a transform based on those matches using the library's RANSAC functions. The images in Figure 2 were generated using this application.

- `dspfeat.c`: This application imports and displays image features from any compatible software. The images in Figure 1 depicting SIFT keypoints from the SIFT library and from David Lowe's SIFT executable were generated using this application, as was the image in Figure 3 depicting Harris-affine features computed using the Oxford Visual Geometry Group's software.

[3]`http://www.robots.ox.ac.uk/~vgg/research/affine/index.html`

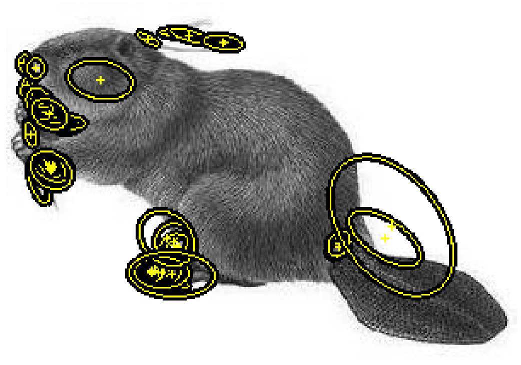


Figure 3: Harris-affine image features computed using the Oxford Visual Geometry Group's software and displayed using functions from the SIFT library.

4. PERFORMANCE

Below I compare the performance of the SIFT library with that of David Lowe's executable in terms of runtime and matching and transform accuracy.

4.1 Runtime

Table 1 compares the runtime for SIFT feature computation for the SIFT library and David Lowe's SIFT executable. The runtimes reported were averaged over the 209-image "people" collection of the Caltech-256 data set [4]. The average area of the images tested was 285350 sq. pixels. The runtimes for the two implementations are comparable.

	Average Runtime
SIFT Library	1.81s
Lowe's Executable	1.68s

Table 1: A runtime comparison between the SIFT library and David Lowe's SIFT executable. Runtimes are averaged over the 209 image "people" collection of the Caltech-256 data set.

4.2 Matching and Transform Accuracy

Table 2 compares the keypoint matching and transform computation accuracy for the SIFT library and David Lowe's SIFT executable. These results were obtained by applying a

	Keypoints Matched	Match Percentage	Avg. Transform MSE (px. sq.)
SIFT Library	858 of 3705	23.2%	0.172
Lowe's Executable	1087 of 4635	23.5%	0.061

Table 2: A comparison of keypoint matching and computed transform accuracy between the SIFT library and David Lowe's SIFT executable. These are computed over a randomly chosen set of ten images from the Caltech-256 datsaet. See the text for more details.

random perspective transform to each of 10 randomly chosen images from the Caltech-256 dataset, computing keypoint matches between the original and transformed images, and then using RANSAC to compute a perspective transform based on those matches. Transform accuracy is reported as the MSE between original keypoint locations transformed by both the computed and ground-truth perspective transforms, averaged over the ten images. Again, the performance of the two implementations is comparable.

5. REFERENCES

[1] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbor search in high-dimensional spaces. In *CVPR*, 2003.

[2] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

[3] H. Grabner, J. Matas, L. Van Gool, and P. Cattin. Tracking the invisible: Learning where the object might be. In *CVPR*, 2010.

[4] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[5] R. Hess and A. Fern. Improved video registration using non-distinctive local image features. In *CVPR*, 2007.

[6] B. Leibe, A. Leonardis, and S. Bernt. Robust object detection with interleaved categorization and segmentation. *IJCV*, 77(1–3), 2008.

[7] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Intl. Journal of Computer Vision*, 60(2):91–110, 2004.

[9] A. Opelt, A. Pinz, M. Fussenegger, and P. Auer. Generic object recognition with boosting. *IEEE TPAMI*, 28(3), 2006.

[10] OpenCV. `http://opencv.willowgarage.com/`.

[11] S. Se, D. G. Lowe, and J. J. Little. Vision-based global localization and mapping for mobile robots. *IEEE T-RO*, 21(3), 2005.

[12] N. Snavely, R. Garg, S. M. Seitz, and R. Szeliski. Finding paths through the world's photos. *ACM TOG (Proceedings of SIGGRAPH 2008)*, 27(3), 2008.

[13] S. Tran and L. Davis. Robust object tracking with regional affine invariant features. In *ICCV*, 2007.

[14] B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalization. In *ICCV*, 2007.