OpenIMAJ and ImageTerrier: Java Libraries and Tools for Scalable Multimedia Analysis and Indexing of Images

Jonathon S. Hare
jsh2@ecs.soton.ac.uk

Sina Samangooei
ss@ecs.soton.ac.uk

David P. Dupplaw
dpd@ecs.soton.ac.uk

Electronics and Computer Science, University of Southampton
Southampton, United Kingdom

ABSTRACT

OpenIMAJ and ImageTerrier are recently released open-source libraries and tools for experimentation and development of multimedia applications using Java-compatible programming languages. OpenIMAJ (the Open toolkit for Intelligent Multimedia Analysis in Java) is a collection of libraries for multimedia analysis. The image libraries contain methods for processing images and extracting state-of-the-art features, including SIFT. The video and audio libraries support both cross-platform capture and processing. The clustering and nearest-neighbour libraries contain efficient, multi-threaded implementations of clustering algorithms. The clustering library makes it possible to easily create BoVW representations for images and videos. OpenIMAJ also incorporates a number of tools to enable extremely-large-scale multimedia analysis using distributed computing with Apache Hadoop.

ImageTerrier is a scalable, high-performance search engine platform for content-based image retrieval applications using features extracted with the OpenIMAJ library and tools. The ImageTerrier platform provides a comprehensive test-bed for experimenting with image retrieval techniques. The platform incorporates a state-of-the-art implementation of the single-pass indexing technique for constructing inverted indexes and is capable of producing highly compressed index data structures.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software; I.4.0 [**Image Processing and Computer Vision**]: General; I.5.0 [**Pattern Recognition**]: General; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Algorithms, Design, Experimentation

MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.

1. INTRODUCTION

OpenIMAJ and ImageTerrier are recently released, twinned open-source projects that contain software libraries and tools for experimenting with and developing multimedia analysis, classification and retrieval software. Whilst the codebase has only recently been publicly released, it is quite mature, having been under active development since around 2005 internally in the University of Southampton.

The OpenIMAJ project contains a number of tools and libraries for everything from state-of-the-art computer vision (i.e. SIFT descriptors [1] and salient region detection, face detection, etc.) and advanced data clustering, through to software that performs analysis on the layout and structure of web-pages. The ImageTerrier project implements a highly-scalable architecture for building and searching inverted-indexes of 'visual terms' extracted from images.

The majority[1] of the OpenIMAJ codebase has been released under the New BSD license which makes it freely available to both academic and commercial users. The ImageTerrier project is released under the same licensing terms as the Terrier project on which it is built — namely the Mozilla Public License. This license also allows both academic and commercial use of the code.

2. DESIGN AND PERFORMANCE

OpenIMAJ and ImageTerrier are primarily written in pure Java and, as such are completely platform independent. The video-capture and hardware libraries contain some native code but Linux, OSX and Windows are supported out of the box (under both 32 and 64 bit JVMs). It is possible to write programs that use the libraries in any JVM language that supports Java interoperability, for example Groovy[2][3]. OpenIMAJ can even be run on Android phones and tablets[4].

One of the main goals in the design and implementation of the two projects was to keep all components as modular as possible. For example, the OpenIMAJ difference-of-Gaussian SIFT implementation allows different parts of the algorithm to be replaced or modified at will. Implementations of commonly used algorithms are also made as generic as possible; for example, the OpenIMAJ RANSAC implementation works with generic `Model` objects and doesn't care

[1]Two non-core sub-projects have slightly different licensing conditions as they were originally derived from other open-source code, however all the licenses are business-friendly.
[2]Groovy: `http://groovy.codehaus.org`
[3]Groovy and OpenIMAJ: `http://bit.ly/OIMJGROOVY`
[4]Android and OpenIMAJ: `http://bit.ly/OIMJAND`

openimaj	
core	Submodule for modules containing functionality used across the library.
core	Core library functionality concerned with general programming problems rather than multimedia specific functionality. Includes I/O utilities, randomisation, hashing and type conversion.
feature	Core notion of features, usually denoted as arrays of data. Definitions of features for all primitive types, features with location and lists of features (both in memory and on disk).
audio	Core definitions of audio streams and samples/chunks. Also contains interfaces for processors for these basic types.
image	Core definitions of images, pixels and connected components. Also contains interfaces for processors for these basic types.
video	Core definitions of a video type, and functionality for displaying videos.
video-capture	Cross-platform video capture interface using a lightweight native interface. Supports 32 and 64 bit JVMs under Linux, OSX and Windows.
math	Mathematical implementations including geometric, matrix and statistical operators.
audio	Submodule for audio related functionality.
processing	Implementations of various audio processors (e.g. multichannel conversion, volume change, ...).
image	Submodule for image related functionality.
processing	Implementations of various image, pixel and connected component processors (resizing, convolution, edge detection, ...).
feature-extraction	Methods for the extraction of low-level image features, including global image features and pixel/patch classification models.
local-features	Methods for the extraction of local features. Local features are descriptions of regions of images (SIFT, ...) selected by detectors (Difference of Gaussian, Harris, ...).
faces	Implementation of a flexible face-recognition pipeline, including pluggable detectors, aligners, feature extractors and recognisers.
machine-learning	Algorithms which aid the classification and search of data.
nearest-neighbour	Brute force and KD-Tree implementations of exact and approximate KNN.
clustering	Various clustering algorithm implementations for all primitive types including random, random forest [2], K-Means (Exact, Hierarchical [3] and Approximate [5]), ...
hadoop	Extensions to enable interaction with the Apache Hadoop Map-Reduce implementation.
core-hadoop	Reusable wrappers to access and create sequence-files and map-reduce jobs.
hardware	Various interfaces to hardware devices that we've used in projects built using OpenIMAJ.
serial	Interface to hardware devices that connect to serial or USB-serial ports.
gps	Interface to GPS devices that support the NMEA protocol.
compass	Interface to an OceanServer OS5000 digital compass.
web	Support for analysing and processing web-pages.
core-web	Implementation of a programatic offscreen web browser and utility functions.
analysis	Utilities for analysing the content and visual layout of a web-page.

Figure 1: OpenIMAJ Library Modules as of July 2011

whether the specific model implementation is attempting to fit a homography to a set of point-pair matches or a straight line to samples in a space. ImageTerrier is equally as modular, and for example, it is possible to easily drop in different scoring implementations into the search engine.

The speed of individual algorithms in OpenIMAJ has not been a major development focus, however some decisions have been made during implementation to ensure efficiency (for example making fields public rather than using getters and setters). In order to give the reader some idea of the real-world performance of our Java code, we have compared the time taken to process an image by our SIFT implementation to Lowe's keypoint binary. For our test image, it takes 3.47 seconds to extract the features with Lowe's binary (averaged over 100 tests). If we count the time taken using our implementation and include the time taken to start the JVM, then we get times of around 10 seconds. However, if we discount the time to start the JVM, the averaged time over 100 iterations is just 3.94 seconds, which is agreeably close to the native version. See the OpenIMAJ wiki for a demo showing near-realtime SIFT extraction and matching in pure Java using OpenIMAJ[5]. Whilst the actual algorithm speed has not been a particular design focus, scalability of the algorithms to massive datasets has. Using the OpenIMAJ Hadoop tools on our small three-machine Hadoop cluster, we have extracted visual term features from datasets with sizes in excess of 10 million images. The OpenIMAJ clustering implementations are able to cluster larger-than-memory datasets by reading data from disk as necessary.

ImageTerrier is designed to be fast during searching and memory-efficient during indexing. The fundamental limit on ImageTerrier is the rate at which the inverted index can be accessed from disk. ImageTerrier uses heavy index compression to make the index smaller and thus faster to access. Experiments have shown that an ImageTerrier index of 1 million medium resolution images indexed with difference-of-Gaussian SIFT visual terms (with a one million term vocabulary) can be searched in under 400 milliseconds on a regular PC (i.e. Intel Core 2, 4GB ram, 7200 RPM disk).

[5] http://bit.ly/OIMJVSIF

3. OPENIMAJ

The OpenIMAJ library is structured into a number of modules. The modules can be used independently, so if for instance you were developing data clustering software using OpenIMAJ you wouldn't need the modules related to images. Figure 1 describes the modules and summarises the functionality in each component.

3.1 Tools

OpenIMAJ exposes key functionality through a set of a command-line tools. This allows more casual users to test the abilities of the library more easily as well as allowing use of OpenIMAJ in non-java projects. The main tools currently provided are outlined below.

GlobalFeatureTool. Allows extraction of low-level global features from images. For example: colourfulness, sharpness, number of faces and average brightness. Also allows comparison of extracted features.

LocalFeatureTool. Allows extraction of local features using internal implementations of various detectors and descriptors.

ClusterQuantiserTool. Provides methods for both creation and usage of clusters. Various kinds of clusters can be trained on various data sources and data sources can also be quantised using clusters. Currently random, random forest, RAC and KMeans (exact, approximate and hierarchical) clusters are supported. This tool can be used to create visual-term representations of images through SIFT features extracted with the LocalFeatureTool.

FlickrCrawler. A set of tools for the targeted downloading of images from Flickr in order to create experimental datasets.

3.2 Hadoop

OpenIMAJ was written with scalability in mind. Using the Hadoop framework a series of tools were written which implemented existing OpenIMAJ functionality as map-reduce tasks. This allows for scalability to large datasets limited only by the size of the Hadoop cluster. Together these tools form a tool chain which allows for the end to end construction of efficiently searchable indexes from images in a scalable way.

SequenceFileTool. Allows the easy creation, examination and extraction of Hadoop sequence-files. Sequence-files are a form of archive file containing many smaller files, used by the Hadoop framework. The tool may be used to construct a sequence-file containing a large number of images as a precursor step to extracting image features.

HadoopClusterQuantiserTool. Multithreaded map-reduce implementation of vector quantisation. Given an existing quantiser definition (usually created by the ClusterQuantiserTool), compute nodes read the quantiser into RAM and quantise data points in parallel.

HadoopFastKMeans. Iterative map-reduce implementation of the exact and approximate K-Means algorithms.

HadoopGlobalFeaturesTool and HadoopLocalFeaturesTool. Distributed feature extraction from large volumes of images.

HadoopImageDownload. Given a file containing a large number of URLs, download a the images in parallel. Used to download all the images from image-net for example.

SequenceFileIndexer. Construct ImageTerrier indexes using sequence-files as the source.

4. IMAGETERRIER

ImageTerrier is a platform for building efficient inverted indexes of visual-term information extracted from images. ImageTerrier is an extension of the Terrier text search platform [4]. ImageTerrier is capable of building compressed augmented inverted indexes using the highly efficient, state of the art, single-pass indexing technique. The index format supports term payloads which allows the optional specification of a given visual term's geometric information, such as a SIFT visual term's position, orientation and scale. The payload information is easily accessed when the index is being searched. ImageTerrier contains implementations of a number of geometric consistency algorithms that make use of this information in order to improve search precision. ImageTerrier is easily extended and it is possible to have complete control over both the payloads that get placed in the index and the operations involved in retrieval from the index.

4.1 Tools

ImageTerrier currently has two complete tools which wrap the entire process of building and querying an image database:

BasicIndexer. Generation of searchable inverted indexes from a directory of images. Allows specification of feature extraction, codebook generation and index options. The image-specific operations use the OpenIMAJ library.

BasicSearcher. Allows both one off and interactive querying of an index using images. Allows for the specification of searching scheme and consistency measures.

5. USAGE EXAMPLES

The tools implemented in OpenIMAJ and ImageTerrier and their applications are many; the reader is encouraged to experiment with the codebase and follow some of the many demos available on the websites[6]. In this section we attempt to give a taste of the capabilities of the two projects by showing two self contained examples, demonstrating some of the capabilities.

5.1 Example 1: Detecting interest points in an image

For OpenIMAJ we present a code example using the JVM language Groovy. Using Groovy's dynamic remote library loading technology, these demos can be directly copied into the `groovyConsole` and run without the need for OpenIMAJ installation. In Figure 2 we show an example of the capabilities of the OpenIMAJ library. In the example an image is loaded and converted to greyscale. At this point a few of the available interest point detectors are initiated and used to detect interest points in the greyscale version of the loaded image. Once detected, the interest points are drawn on the original colour image using the drawing functionality and finally the detected interest points are displayed.

5.2 Example 2: Building and searching an ImageTerrier Index

The ImageTerrier project comes with tools which use OpenIMAJ to allow the generation of a searchable index of image content from a directory of images. This includes extraction of image features, generation of a visual codebook, quantisation of image features and construction of an ImageTerrier index. Each of these stages supports a plethora of options,

[6] `http://openimaj.org` and `http://imageterrier.org`

```
@GrabResolver(name='octopussy-releases', root='http://octopussy.ecs.soton.ac.uk/m2/releases/')
@Grab('org.openimaj:core-image:1.0')
@Grab('org.openimaj:image-local-features:1.0')
import org.openimaj.io.*
import org.openimaj.image.*
import org.openimaj.image.colour.*
import org.openimaj.math.geometry.shape.*
import org.openimaj.image.feature.local.interest.*

img = ImageUtilities.readMBF(getClass().getResource("/org/openimaj/OpenIMAJ.png")) //Load an image
gimg = Transforms.calculateIntensityNTSC(img) //make a grey version
float intScale = 2.5f; float diffScale = 0.6f * intScale; //set the scales (std.dev of Gaussian)
float intScaleVar = intScale**2; float diffScaleVar = diffScale**2; //calculate the variance
ipds = [ //set up a list of detectors
    [ipd: new HarrisIPD(diffScaleVar, intScaleVar), colour:RGBColour.RED],
    [ipd: new HessianIPD(diffScaleVar, intScaleVar), colour:RGBColour.MAGENTA],
    [ipd: new LaplaceIPD(diffScaleVar, intScaleVar), colour:RGBColour.GREEN],
]
ipds.each{rec -> //loop through detectors and draw 100 best points found
    rec.ipd.findInterestPoints(gimg)
    img.drawPoints(rec.ipd.getInterestPoints(100), rec.colour, 3)
}
DisplayUtilities.display(img)//display the result
```

Figure 2: Interest point detectors working in OpenIMAJ

```
$> cat highfield.config
crawler {
    apikey="<key>",secret="<secret>"
    outputdir="highfield"
    queryparams {woeId="43761"}
    images {targetSize=["large", "original", "medium"]}
}
$> groovy FlickrCrawler.groovy highfield.config
$> BasicIndexer -o idx -qt RANDOM -k 100000 highfield
$> BasicSearcher -i idx -q query.jpg -dr
```

Figure 3: Example settings for the FlickrCrawler tool, highfield.config and the tools useage

here we present a basic example. Firstly, an image collection can be downloaded using the provided FlickrCrawler tool in OpenIMAJ. This tool is called using the command in Figure 3 which references a configuration file also shown in Figure 3. This configuration file supports various features including the specification of a Flickr query against which images are downloaded[7]. In this case we download images geo-tagged with coordinates with a specific Flickr Where On Earth ID (woe_id), which in the example is the id for Highfield, the location of the University of Southampton's main campus in the United Kingdom.

Once an appropriate directory of images is collected the indexer tool can be applied. This tool allows specification of: extracted feature type, generated codebook type and various ImageTerrier index options. In the example in Figure 3 we generate the default SIFT features and specify that a Random codebook of size 100000 be generated using all extracted features as training samples. Using the codebook, the features are quantised and used to construct an index. The image set is now searchable.

Once the index and codebook are generated the directory of images is now searchable. The `BasicSearcher` tool can be used to search indexes with images. The tool provides a useful command line interface for multiple queries, but in the example shown in Figure 3 a single query image is used. Elaborations and example query results using these tools can be found on the ImageTerrier Wiki[8]

[7] `http://bit.ly/OIMJFCR`
[8] `http://bit.ly/ITEREXMP`

6. APPLICATIONS IN EDUCATION

The codebase that makes up the core of OpenIMAJ has been used by undergraduate student projects in Southampton for around the last 5 years. Students have used the library in a range of projects, from performing road-sign recognition using local features through to experimenting with K-nearest-neighbour classification of city/landscape scenes using KD-Tree data-structures with large image collections crawled from Flickr. With the software now open-sourced, we expect that our future students will help contribute to the codebase in the coming years.

7. ACKNOWLEDGMENTS

Current development of the OpenIMAJ and ImageTerrier software is funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 270239 (ARCOMEM) and 231126 (LivingKnowledge) together with the LiveMemories project, graciously funded by the Autonomous Province of Trento (Italy). We are also grateful to the Arts and Humanities Research Council ('Bridging the Semantic Gap' - MRG-AN6770/APN17429) and Ordnance Survey, for earlier funding under which a number of the older OpenIMAJ classes were created.

References

[1] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, January 2004.

[2] F. Moosmann, E. Nowak, and F. Jurie. Randomized clustering forests for image classification. *IEEE PAMI*, 2008.

[3] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[4] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proc SIGIR*, 2006.

[5] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007.