1 Introduction

Stasm is a C++ software package for locating facial landmarks in faces. You give it a face image and it returns the positions of the landmarks (Figure 1).

Stasm is designed to work on front views of more-or-less upright faces with neutral expressions. You will see poor fits on faces at angles or with expressions. Download the self-installing Windows program wasm.exe to see what Stasm can do. It has performed well in comparative studies [3].

Stasm uses the OpenCV library and is released under a BSD style license (Appendix A).

The Histogram Array Transform (HAT) descriptors used by Stasm for template matching are similar to SIFT descriptors [9]. Since the SIFT algorithm is patented, this potentially opens a patent issue for commercial use. (I’m not qualified to say if it does.) Section 7 has some technical details.

If you are interested in training new models, please see Building Stasm 4 Models [11], downloadable from the Stasm website.
Figure 1: The 77 Stasm landmarks and their numbers.

Note that Stasm versions prior to version 4.0.0 had a different license (GPL).

2 Executables

The following example programs are provided with the package.

minimal

Minimal program using the stasm_search_single function (Figure 2). This reads an image and show the face landmarks in an OpenCV popup window.

minimal2

Like the minimal program above but uses the stasm_search_auto function, which allows you to locate landmarks in multiple faces in an image.

wasm

Windows utility for demonstrating Stasm. You can scroll through all images in a directory with the PgUp and PgDn keys. Face widths must be at least 25% of the image width.

stasm
// minimal.cpp: Display the landmarks of a face in an image.  
// This demonstrates stasm_search_single.

#include <stdio.h>
#include <stdlib.h>
#include "opencv/highgui.h"
#include "stasm_lib.h"

int main()
{
    static const char* path = "./.data/testface.jpg";
    cv::Mat_<unsigned char> img(cv::imread(path, CV_LOAD_IMAGE_GRAYSCALE));
    if (!img.data)
    {
        printf("Cannot load %s\n", path);
        exit(1);
    }

    int foundface;
    float landmarks[2 * stasm_NLANDMARKS]; // x,y coords
    if (!stasm_search_single(&foundface, landmarks,
        (char*)img.data, img.cols, img.rows, path, "./.data")
    {
        printf("Error in stasm_search_single: %s\n", stasm_lasterr());
        exit(1);
    }

    if (!foundface)
    printf("No face found in %s\n", path);
    else
    {
        // draw the landmarks on the image as white dots
        stasm_force_points_into_image(landmarks, img.cols, img.rows);
        for (int i = 0; i < stasm_NLANDMARKS; i++)
        img(cvRound(landmarks[i*2+1]), cvRound(landmarks[i*2])) = 255;
    }
    cv::imshow("stasm minimal", img);
    cv::waitKey();
    return 0;
}

Figure 2: Example program to read an image and display the facial landmarks.

Command line utility to locate face landmarks in one or more images. Output goes to the text file stasm.log. You can set the minimum face size and so on with command line flags. You can tell it to list the landmarks in Stasm’s default 77 point format, or in the XM2VTS [10], BioID [7], AR [16], or me17 [5] formats, or in the 76 point format used in old versions of Stasm. You can tell it to list the landmarks as a Stasm “shapefile” or as comma separated values.

swas

Command line utility to compare landmarks automatically located to reference land-
marks. (Swas stands for “Stasm with a shapefile”.) Swas creates a text file that lists, for each face, the mean fitness (mean distance between found landmarks and reference landmarks, divided by the reference inter-pupil distance), the me17 fitness (mean fitness over 17 points [5]), and the FM29 fitness (an anisotropically weighted sum of 29 landmarks used for internal development).

`test_stasm_lib` and `test_stasm_lib_err`

Utilities for testing the package in conjunction with the `stasm/tests` directory. See the makefiles for details.

### 3 Building Stasm

Building Stasm is straightforward. You first need to install OpenCV. Stasm has been tested on OpenCV versions 2.3.1 and 2.4.0, but the version probably isn’t important.

Use the following configuration to build the `minimal` example program (the paths below assume your working directory is say `stasm/work`):

---

**Source files:**
- `../apps/minimal.cpp` contains `main()`
- `../stasm/*.cpp` Stasm library files
- `../stasm/MOD_1/*.cpp` frontal pose models

**Include path:**
- `../stasm` dir for `stasm.h` etc.
- `OPENCV_HOME/build/include` dir for `opencv/cv.h` etc.

**OpenCV libraries needed:**
- `core` `imgproc` `objdetect` for Stasm library
- `highgui` for apps that use `imwrite` etc.

---

For details see the make files included with the package. You will need to tweak the make files for the location of your OpenCV directory.

In Windows environments, batch files are also provided for building `minimal.exe` and a few other utilities. Microsoft Solution files are provided for building `minimal.exe` in the Visual Studio IDE. Caveat: the Solution files were created by someone who seldom uses the IDE (precompiled headers aren’t enabled, for instance).

If this is your first OpenCV project, we recommend that you first check your setup independently of Stasm by building `minimal.exe` and running a simple OpenCV program. This holds especially in a Windows environment, where the error messages for missing or mismatched DLLs can be very misleading.

Juan Cardelino has provided cmake files for Linux and other systems. See the Stasm web page.
4 Stasm library functions

The library interface is defined stasm_lib.h. The landmark names listed in stasm_landmarks.h are sometimes also useful.

4.1 Simple interface

The simplest approach is to invoke the function stasm_search_single (Figure 2). This finds the largest face in the image (using the OpenCV frontal face detector) and returns the positions of the landmarks. The face width must be at least 10% of the image width.

If your image is color, convert it to monochrome before passing it to Stasm. In OpenCV you can use cvtColor(colorimage, monoimage, CV_BGR2GRAY).

4.2 More versatile interface

The library also provides a more versatile interface for software that needs to locate the landmarks in multiple faces in an image or to fine tune the interface. The basic idea is that you call stasm_open_image to detect the face(s), and then repeatedly call stasm_search_auto to landmark the faces, one by one. For details see minimal2.cpp and the comments in stasm_lib.h

4.3 The minwidth argument

The minwidth argument of stasm_open_image specifies the minimum face detector box width as a percentage of the image width. It helps reduce false face detections (on the scenery behind the subject for instance). Typical values are 10% or 25%. After scaling by the image width, it is passed to the OpenCV function detectMultiScale.

4.4 The multiface argument

If you set the multiface argument of stasm_open_image to 1, you can call stasm_search_auto repeatedly until there are no more faces in the image. If you set it to 0, stasm_search_auto will return the single “best” face. This will usually be the largest face detected by the OpenCV frontal detector.

To reduce false positives, when three or more faces are detected, any face that is very much bigger or smaller than than the median detected face is discarded internally. An implication is that occasionally the “best” face may not be the largest face, because the largest face has been discarded (treated as a false positive). Another implication is that with multiface set at 1, in a group shot only faces that are roughly similar in size will be returned. In practical applications this approach seems to work fairly well, especially when minwidth is small. It will sometimes make mistakes. Remove the call to DiscardMissizedFaces in facedet.cpp if you don’t want this automatic removal
of missized faces. When debugging, change \#define TRACE\_IMAGES to write images showing the raw face rectangles.

### 4.5 Partial initialization by the user

In some applications, the user manually pins a few points on the face and the machine automatically places the remaining points. This process may be iterated so the user can correct the machine-generated positions. Use `stasm\_search\_pinned` to implement this.

The current implementation was trained mainly for when the user pins five specific points: the outer corners of the eyes, the nose tip, and the corners of the mouth — but `stasm\_search\_pinned` will also work if any two or more points are pinned. Note that the face detector isn’t needed, because the ASM start shape is formed by aligning the mean training shape to the pinned points.

### 4.6 Error handling

The Stasm library functions return 1 on success and 0 on error. If a function returns 0, use `stasm\_lasterr` to get the error string. The functions never throw exceptions. An example error is "Cannot open ../data/haarcascade\_
tonalface\_alt2.xml".

`CV\_Assert` fails inside Stasm are handled in the same way. The library functions will catch the assert internally and return 0. Use `stasm\_lasterr` as usual to get details of the failure. (TODO `CV\_Assert` is currently not working like this in Mingw builds, although the Stasm error function `Err` is working.)

Note that “not finding a face” isn’t an error in the sense we are using it here. If the search function runs successfully but doesn’t find a face, it returns 1 for success, but with the `foundface` argument set to 0.

### 4.7 Utility functions

A few utility functions are also provided.

`stasm\_convert\_shape` converts the default 77 point shape to a shape with the specified number of points. It can convert to the XM2VTS, BioID, AR, or me17 formats, or to the 76 point format used in old versions of Stasm. The landmark definitions often don’t coincide exactly, particularly on the side of the forehead and jaw, and on the nostrils. The position of these landmarks is estimated from nearby landmarks.

`stasm\_force\_points\_into\_image` forces landmarks into the image boundary. In normal use, if say the forehead is cut off by the edge of the image, Stasm will position the forehead landmarks above the image boundary.

`stasm\_printf` prints to stdout like `printf` but also prints to the file `stasm\_log`, if it is open. (It will be open if `stasm\_init` was called with `trace\_1`.) This function was
added primarily for the programs that test the package, but may be otherwise useful.
5 Visual FAQ

In this section we show some images where Stasm performed poorly, and give a brief explanation. (Most of the images are from the BioID set [7].)

Figure 3:

The face detector did not find the face.

(Stasm uses the OpenCV frontal face detector.)

Figure 4:

The face detector mistook the door for a face.

Figure 5:

The face detector mistook the nostril and mouth for a face.
Figure 6:

*Stasm struggled with the non-frontal face.*

Figure 7:

*Stasm struggled with the non-frontal face.*

Figure 8:

*Stasm mistook the collar for the jawline.*

Figure 9:

*Stasm mistook the vertical edge of the door for the left jawline.*
Figure 10:

The glasses caused the face detector to overestimate the size of the face. The start shape was too big, and Stasm didn’t recover, mistaking the chin for the mouth.

Figure 11:

Stasm was confused by the mustache.

Figure 12:

Stasm does not work very well with open mouths.

Figure 13:

Stasm mistook the bottom of the bottom lip for the top of the lip. Probably the face rectangle was too big and Stasm did not make a full recovery.
6 Stasm version 4 and 3

This section describes the differences between Stasm version 4 and version 3.

Stasm 4 is released under a BSD style license; Stasm 3 was released under a GPL license.

Stasm 4 has 77 landmarks (Figure 1); Stasm 3 has 76 landmarks.

Both Stasm 4 and 3 are designed for frontal faces, but Stasm 4 is a bit more flexible and can handle a wider variety of faces. Figure 14 compares the me17s [5] for the two different Stasm versions. The figure is admittedly a bit cluttered, but Stasm 4 does better than Stasm 3. Details. The BioID [7] and PUT [8] curves include the entire set. The DEF set is a difficult test set we use internally. The poor performance on the right of the DEF curve is principally due to the many non-frontal faces in the set, a challenge that requires heavier (and slower) machinery than either version of Stasm.

Figure 14:

*Stasm version 4 gives better fits than version 3.*

*Compare the solid lines to the dashed lines.*

Figure 15:

*Version 4 is faster than version 3.*

*The graph is for VC10 32 bit builds without OpenMP on a 3.4 GHz i7.*
Faces not found by the OpenCV frontal detector are included in the curves, but with \texttt{me17} set to infinity. They reduce the height of the curves at the right of the plot.

Stasm 4 is faster (Figure 15). The ASM search itself is faster, and there is also less internal overhead converting image formats and so on before starting the search.

Building Stasm 4 is easier than Stasm 3. The code has been cleaned up and simplified. Stasm 4 supports both 64 and 32 bit builds.

Stasm 4 uses standard OpenCV image coordinates (0,0 is the top left of the image); Stasm 3 uses an internal coordinate system where 0,0 is the center of the image.

Stasm version 4 uses the OpenCV libraries throughout — which makes operations like image scaling trivial, using functions familiar to OpenCV users. Stasm 3 has its own routines for image scaling, reading images, etc.

In Stasm 3 the models are read off disk during runtime initialization. In Stasm 4 the models are built-in (meaning that C++ model files were created during model training, and these files get compiled into Stasm). Thus the only external files needed by Stasm 4 are the OpenCV face and feature detection files (\texttt{haarcascade_frontalface_alt2.xml} and friends). A benefit is that Stasm 4 starts up more quickly.

7 Notes on the implementation

The OpenCV frontal face detector often fails if the face is near the edge of the image. To counteract this, Stasm internally adds a 10\% border around each edge of the image (Figure 16). Change \texttt{BORDER_FRAC} to 0 in \texttt{facedet.cpp} if you don’t want the border (it makes face detection slower, 1.2 x 1.2 = 44\% more image area, or maybe you want to exclude cut-off faces).

Stasm can use OpenMP if your compiler supports it. (On multicore machines, Stasm

\begin{figure}[h]
  \centering
  \begin{minipage}[b]{0.4\textwidth}
    \includegraphics[width=\textwidth]{left_face.png}
    \caption{\textit{left} Face is too near the edge — face detection fails.}
  \end{minipage}
  \begin{minipage}[b]{0.4\textwidth}
    \includegraphics[width=\textwidth]{right_face.png}
    \caption{\textit{right} Artificial border added — face is now detected.}
  \end{minipage}
  \caption{Face detect failrate on the BioID set goes from 3.3\% down to 0.7\%.}
\end{figure}
uses OpenMP for template matching on several landmarks in parallel.) Enable OpenMP with the OpenMP command line flag for your compiler (typically -openmp or -fopenmp). In our experience OpenMP can sometimes offer significant speed advantages, although these may depend on the compiler in non-obvious ways.

A hash_map is used in hatdesc.cpp, which may cause some portability issues. You can easily disable references to hash_map by changing a #define, although that will make Stasm slower.

Stasm is based on the Active Shape Model (ASM) developed by Cootes and Taylor [4, 14]. Stasm uses Histogram Array Transform (HAT) descriptors for template matching at landmarks as described in [15].

Here is an overview of HAT descriptors. Like SIFT descriptors [9], HATs are grids of image orientation histograms, with orientations weighted by the gradient magnitude and smoothed out over nearby histogram bins. Stasm prescales the face to a fixed eye-mouth distance of 100 pixels and rotates the face so the eyes are horizontal. Thus the extreme scale invariance of SIFT isn’t required, nor is the SIFT descriptor’s automatic orientation to the local average gradient direction. In that sense HAT histograms are more similar to earlier descriptors such as those in [1, 6]. For readers familiar with SIFT, note that ASMs like Stasm employ descriptors very differently from SIFT. SIFT discovers and matches image-defined keypoints at sparse scale space extrema; the ASM moves the descriptors in a fine grid, after training on a large set of faces where the facial landmarks were manually assigned.

The Stasm models were trained on the MUCT data [13] with landmarks extended to 77 points. The models are defined in .mh files, which are machine generated C++ files.

If you are interested in training new models, please see the manual Building Stasm 4 Models [11]. You can train models for faces, or “non-face data” such as medical images.

You will see references to three-quarter face models in the code. However the current open-source version of Stasm uses only a frontal model (referred to as the yaw00 model in the code). The three-quarter models (referred to as the yaw22 and yaw45 models) may be released in the future, but for now the code for three-quarter views stays inactive in the released code. Our approach to multiview models is described in [12].

Some variables are not thread-safe (because they are static to a function or class). For most people this is not an issue. If it is for you, locate these by enabling the thread safety warnings for your compiler. With the Microsoft compilers, one approach is to use -Wall on the command line and search the compiler warnings for thread-safe.

Define TRACE_IMAGES true in stasm.h for Stasm to create debugging images showing the face detector face(s), ASM search progress, etc. The images are named so they list in the order they were created.

Arguments that may be modified appear first in function argument lists. When the function is invoked, such arguments are often listed on their own line. For example,

```
FaceRoiAndDetPar(face_roi, detpar_roi, 
                 img, detpar, false);
```

indicates that face_roi and detpar_roi will be modified.
Stasm stores shapes as \(x_0, y_0, x_1, y_1, \cdots\) (rather than as \(x_0, x_1, \cdots, y_0, y_1, \cdots\)).

Global variables have a \_g suffix. These are used, for example, for saving command-line flags, where this somewhat unstructured use of globals is clearer than passing flags as arguments through possibly nested functions. All such variables are “file global” (i.e. declared as static), except for print\_g, trace\_g, and imgpath\_g.

- The print\_g variable is by default false; set it true to allow output to stdout (as well as to the log file).
- The trace\_g variable is also by default false; set it true to trace Stasm’s operation. Both of the variables are set by stasm\_init’s trace argument.
- The imgpath\_g variable holds the path of the current image. It is used only for debugging and tracing (TRACE\_IMAGES must be set; see the above paragraph on TRACE\_IMAGES).

C++ class member names have trailing underscore.

Internal code is in the stasm namespace. The library interface functions aren’t in a name space — instead they have a stasm\_ prefix.

Nearly all .h files are included in the single file stasm\_h, to facilitate precompiled headers to speed the build process. Definitions needed by the example applications (but not in the Stasm library code itself) are in appmisc\_h.

8  Citing Stasm

Cite Stasm as [15]:

@article{Milborrow14,
  author={S. Milborrow and F. Nicolls},
  title={Active Shape Models with SIFT Descriptors and MARS},
  journal={VISAPP},
  year={2014},
  note={\url{http://www.milbo.users.sonic.net/stasm}}
}

Cite this user document as follows (although it’s normally best to cite the Stasm paper):

@book{StasmManual,
  author={S. Milborrow},
  title={Stasm User Manual},
  publisher={\url{http://www.milbo.users.sonic.net/stasm }},
  year={2013}
}

You may need to add \texttt{\usepackage{url}} to your tex file to support the \texttt{\url} in the citation.
9 Acknowledgments

It is a pleasure to acknowledge the following people who provided ideas, code, data, and techniques used in Stasm. The list is in alphabetical order.

Juan Cardelino for his cmake files.

M. Castrillón Santana, O. Déniz Suárez, M. Hernández Tejera, and C. Guerra Artal at the University of Las Palmas de Gran Canaria for their OpenCV eye and mouth detectors [2].

Tim Cootes http://www.isbe.man.ac.uk/~bim

David Cristinacce http://mimban.smb.man.ac.uk

Eugen Dedu for his line drawing code used in earlier versions of Stasm http://lifc.univ-fcomte.fr/~dedu

M. Galassi, J. Theiler, and others for the GSL library used in earlier versions of Stasm http://www.gnu.org/software/gsl

Rob Hess for his opensift library, helpful for getting to grips with the details of SIFT http://blogs.oregonstate.edu/hess/code/sift

GuoQing Hu for his start-shape alignment technique and for an Android demo http://www.androidhat.org


David Lowe the inventor of SIFT http://www.cs.ubc.ca/~lowe/home.html

The manual landmarkers at Manchester for their work http://david.cristinacce.net and http://www.isbe.man.ac.uk/~kob

Aleix Martinez and Robert Benavente for the AR database http://cobweb.ecn.purdue.edu/~aleix/aleix_face_DB.html

Darren Murray and his team for high quality manual landmarking

Ramin Nakisa and others for gslwrap used in earlier versions of Stasm http://sourceforge.net/projects/gslwrap

Fred Nicolls http://www.dip.ee.uct.ac.za/~nicolls

Developers and contributors to the OpenCV library

Henry Rowley, Shumeet Baluja, and Takeo Kanade for the Rowley face and eye detector used in earlier versions of Stasm http://vasc.ri.cmu.edu/NNFaceDetector

Oliver Walker, Elizabeth Walker-Watts, and Gill Andrew for MUCT landmarking

Roger Willcocks for his prototype ASM code http://www.rkww.com
Yan Wong at Bang Goes The Theory http://www.bbc.co.uk/bang

The XM2VTS people at Surrey for the XM2VTS database http://www.ee.surrey.ac.uk/CVSSP/xm2vtsdb

Brad Yearwood and Pierre Moreels for help with the original Ubuntu port

The following people also contributed in one way or another. Thanks guys!

A Stasm License Agreement

Stasm License Agreement

Copyright (C) 2005-2013, Stephen Milborrow
All rights reserved.

Redistribution of Stasm in source and binary forms, with or without modification, is permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.

A SIFT patent restriction may be in conflict with the copyright freedoms granted by this license. This license does not give you permission to infringe patents.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the copyright holder be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Note that Stasm versions prior to version 4.0.0 had a different license.
References


