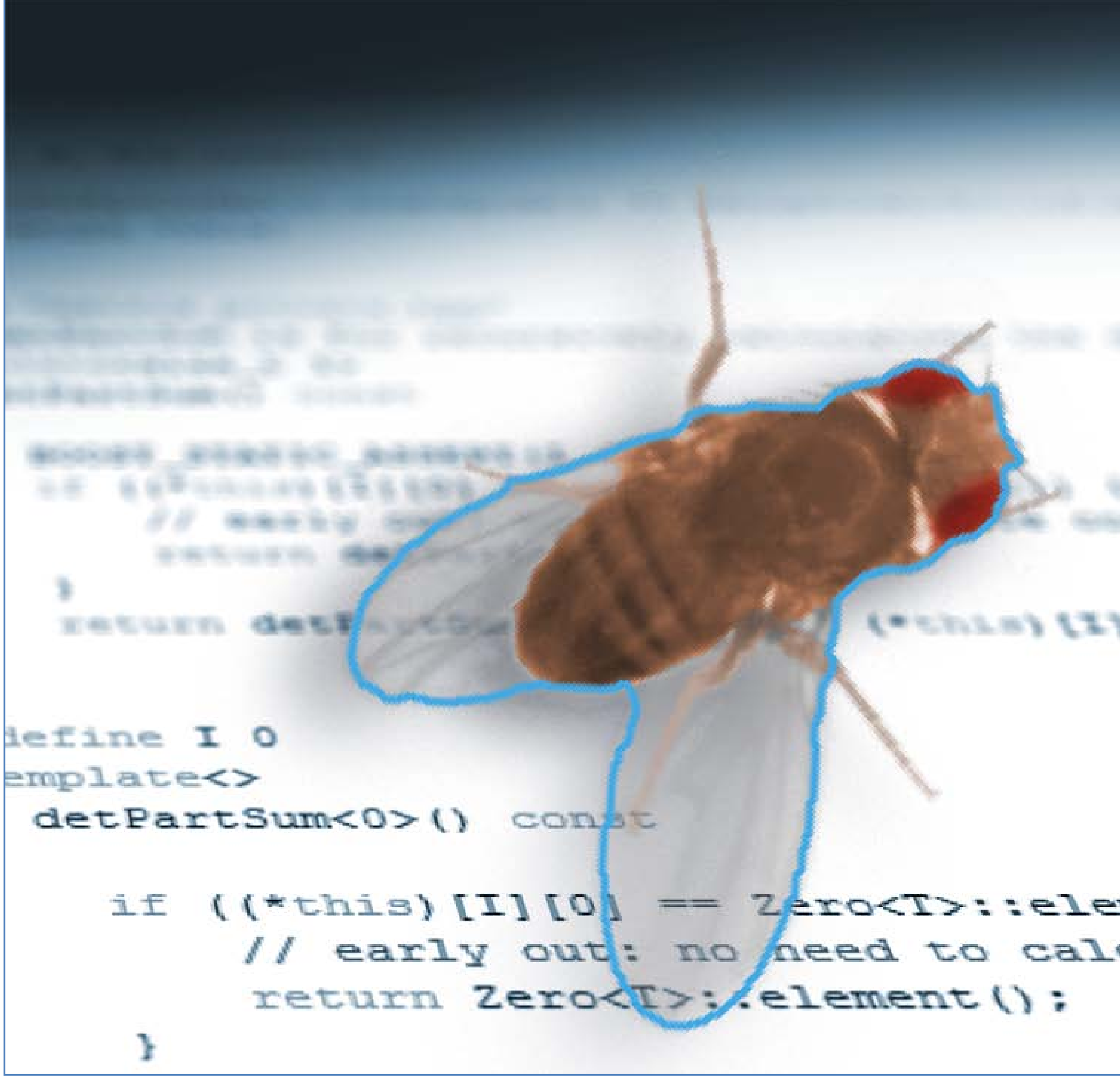


MateBook

Documentation for SVN revision 2141

Contents

1. Introduction and Overview

MateBook is a high performance video analysis software suite designed for quantifying courtship and locomotive behavior of the fruit fly Drosophila melanogaster. It consists of two C++ modules, the *GUI* and the *tracker*, as well as a set of helper scripts. MateBook's key features are:

- Automatic detection and manual refinement of circular, linear and ring-shaped arenas.
- Support for 1 or 2 flies per arena, tracking each fly through occlusions.
- Pre-defined, but adjustable behavioral event detection.
- Visualization of the completed analysis using plots, ethograms and heatmaps.
- GUI support for courtship song analysis.
- Data import scripts for MATLAB.
- Multiprocessor support.
- GridEngine cluster processing and local processing, using the same workflow.

Section 2 of this document aims to guide the user through the workflow required for the video analysis. It explains all settings offered by the GUI and describes the measured attributes in detail. Section 3 is the system administrator's guide and explains how to set up systems for running MateBook, for cluster processing and for software developers wishing to improve MateBook. Finally, section 4 is an introduction to the code structure written for said developers.

2. User's Guide

2.1. Projects

MateBook manages results associated with videos in *projects*. Only one project can be active at any given time and that project's name is displayed in the title bar of the GUI's main window. On start-up, an empty project is created at a system-dependent temporary location. To create a new project, open an existing project or save the current project use the *File* menu. After starting a new project (*File / New Project…*) it is recommended to use *Files / Save As…* and save the project in its final location right away. This way, all results are created at their final location and need not be moved later.

Each MateBook project is stored as an *.mbp* file together with a matching data directory that ends in *.mbd*. It is important that if projects are to be moved, project files and data directories are moved together. Only processing results are kept in the data directory: MateBook does not keep copies of video files in the project! Not saving a project does not remove processing results: only the project settings are forgotten.

2.2. The Main Window

MateBook's main GUI window is organized in tabs, accessible at the bottom. On start-up, the *Files* tab is displayed. The tabs' purpose is as follows:

- Files: Associate videos with the project, enter metadata, start processing and delete results.
- Video: Inspect and correct results from arena detection. View per-arena ethograms.

- Arena: View detailed tracking results, inspect heading and correct the identity assignment.
- Groups: Define groups of arenas across videos and run statistical analyses. [Not implemented.]
- Song: Inspect and correct the pulse detection.
- Statistics: Statistical summary for songs.

Within each tab, the available actions are shown in the *action bar* on the right.

2.3. A First Tutorial

Here we give an example of the typical MateBook workflow.

A. Start MateBook, choose *File / Save As…*, select a location on your hard disk, pick a project name and save the project.

B. In the action bar on the right, press the *Add…* button and select a courtship video with circular arenas. Videos can also be added via drag-and-drop, e.g. by dragging file icons from the Windows Explorer to the MateBook window. Videos will show up in the table along with some metadata like the frame size, the number of frames, the frames per second, the video duration and the file creation date.

C. Double-click the value in the *End* column and set it to 00:00:10 to process only the first 10 seconds. In the action bar on the right click *Arena Detection*. With multiple videos in the project, actions apply to all those videos that are selected. To select any number of videos, simply click the table (any column will do) and drag the mouse while the mouse button is still pressed. When launching a video processing action, the *Video Stage* and *Video Status*, as well as the background color in the table will change. Arena detection should finish within a few seconds and the background should turn green again. Video stage and status in the table should change to *Arena Detection* and *Finished*, respectively.

D. Switch to the *Video* tab to view the arena detection results for the selected video. Uncheck the *Ethograms* checkbox in the action bar on the right: we haven't tracked the flies yet, so the ethograms are currently blank. You should see the first frame from the video with a set of white squares overlaid. At the bottom of the tab there's also a list of the detected arenas. You can select arenas either in the list or by clicking them in the video directly.

E. Select a detected arena and press the [del] key on your keyboard. On Macs without a [del] key press [fn]+[backspace]. This removes the arena.

F. To add an arena, click into the video. Then press and hold [shift] while moving your mouse. A crosshair with guides should appear. While still holding [shift], left-click, hold the mouse button and drag the mouse to mark an arena. This adds the arena.

G. Switch back to the Files tab, make sure the video is selected and press *Fly Tracking*. Since we're only tracking the first 10 seconds, processing should finish within less than a minute.

H. Still in the files tab, click the small arrow on the left to expand the table view for the video. Notice there is one line per arena. Select an arena and switch to the *Arena* tab.

I. At the bottom of the *Arena* tab, you'll see two drop-down boxes labeled with *<attributes>*. Select *isOcclusion* in the first box and *bodyAreaEccentricityCorrected_u* in the second. The top graph will then mark occlusions (frames in which the flies are touching) and the bottom graph will draw their sizes in mm^2. Hovering over the graphs you can read off the exact values.

J. Press the play button in the left video player and notice that the player is synchronized with the graphs. The current frame corresponds to the values graphed at the central green line. You can also left-click and drag the graphs. Move your mouse over the graphs and use the mouse wheel (two-finger swipe on MacBook touchpads) to zoom in time. Double-click in the graph to make that point the current frame. WARNING: Do not use MateBook's video players for manual scoring (i.e. with a stopwatch)! They are made to display videos frame-by-frame. Where other video players would drop frames to keep the video running at real-time speeds even on slow computers, MateBook's players will display the video at a lower speed instead.

K. Fly IDs are color-coded. The smaller fly (typically the male) is blue, the larger fly (female) is pink. The arena view lets you inspect the video and make sure all IDs are consistent over time. MateBook uses sophisticated algorithms to preserve IDs despite any occlusions. Still, should the ID assignment be wrong after an occlusion, one can switch the IDs manually by selecting that occlusion in the action bar and pressing the |-> button, also in the action bar. This switches the IDs after the current occlusion till the end of the video. Likewise, the <-| button switches identities before the current occlusion and the |<->| button switches identities between the current occlusion and a second one, selected automatically based on the tracker's confidence. Note that when selecting an occlusion, the two video players show a before and after view to help you decide whether the IDs are assigned correctly across that occlusion. Note also that switching IDs updates the graphs immediately, but the values during the affected occlusions are missing. This is because they are interpolated during *postprocessing*, which happens at the end of tracking. Hence, to complete the manual ID change, press *save* in the action bar, switch back to the *Files* tab, make sure the arena is still selected and press Postprocessor in the action bar. (BUG #157: currently, only entire videos can be processed)

L. Select any number of videos in the *Files* tab and press *Behavior Analysis*. A .tsv file with statistics for each arena should open. It is recommended that you associate .tsv files with Microsoft Excel or any other program you're going to use to further analyze the results. Should the file not open, please navigate to your project directory and look for behavior.tsv there.

2.4. Settings

All global settings can be accessed through *File / Settings…* in the menu. Whenever video processing is started (by pressing one of the buttons in the *File* tab's action bar), the current settings are written to disk and used for that run only. Any further changes to the settings will affect only those runs started after the changes are made.

2.5. Measured Attributes

3. System Administrator's Guide

We have kept the source code in a Subversion repository during development. In what follows, any paths starting with *svn* refer to this repository.

3.1. Setup for Users

To run MateBook we recommend at least an Intel Core 2 CPU (or equivalent), 4 GB of RAM, an OpenGL 2.1 compatible graphics processor and a minimum screen resolution of 1280x1024. Our main platform for testing is Microsoft Windows 7 Professional, 64 Bit. Windows XP, Windows Vista and Windows 8 are expected to work as well. MateBook can be built for Mac OSX 10.7; other versions have not been tested.

3.1.1. Windows

GUI and tracker require the *Microsoft Visual C++ 2010 SP1 Redistributable Package (x86)*[1] to be installed. To use the song module, install the *MATLAB Compiler Runtime (MCR)*[2] for the MATLAB version that is used to compile the module. We use R2012a, 64 Bit. Multiple MCR versions can be installed side-by-side. Submitting jobs to a Grid Engine cluster requires Plink of the PuTTY[3] toolset. See section 3.2 for further details regarding the cluster setup. Finally, extract MateBook_2145_MSWin32.zip and double-click MateBook.exe to start the program.

3.1.2. Mac

We're not packaging the Qt libraries with the executable on the Mac, so they have to be installed first. They must match the Qt libraries used for development. We use Qt 4.8.2 for open source projects[4]. The song module is currently not supported on the Mac.

3.2. Setup for Cluster Processing

For cluster processing, the tracker module was built and tested on Debian Lenny, 64 Bit. As is customary for Linux platforms, the system administrator is required to compile the executable from the sources. Paths in *svn/tracker/build.gcc/Makefile* must be changed to match the local environment. Note that the Makefile expects to be run from an SVN repository to make the revision part of the output directory name, but this can easily be changed when not compiling sources from SVN.

The following libraries are needed:

- FFMPEG 0.11.1
- Boost 1.47.0
- OpenCV 2.4.1

They can be downloaded from the project websites. Once that's been taken care of, a simple *make compile* should do the trick. Note that the libraries' interfaces have not stabilized yet, so using different versions may require code changes. Also note that OpenCV itself depends on FFMPEG as well. It's used to compile its HighGUI module. We always aimed to use the same FFMPEG version for compiling OpenCV and the tracker since different versions led to broken binaries.

On both Windows and Mac, the GUI can submit jobs to a GridEngine cluster by connecting to a Linux host via SSH. The host is expected to provide the qsub, qstat and qdel commands. Public key

[1] http://www.microsoft.com/en-us/download/details.aspx?id=8328
[2] http://www.mathworks.de/products/compiler/mcr/
[3] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
[4] http://download.qt-project.org/official_releases/qt/4.8/4.8.2/qt-mac-opensource-4.8.2.dmg

authentication must be set up on the host for every user submitting jobs. In the GUI, open the ***File / Settings…*** dialog and choose ***System*** in the list on the left. The ***Cluster Processing*** section should be checked and set up as follows:

- SSH Client: Full path to *plink.exe* on Windows. Simply *ssh* on Mac. These should be the defaults. On Windows, the GUI will look for plink in PuTTY's standard install path.
- Transfer Host: The Linux host providing the GridEngine environment. At the IMP, that host is called *albert*, and it's the hardcoded default value.
- Host Key: On Windows, the host key of the transfer host in PuTTY format, as used at HKCU/Software/SimonTatham/PuTTY/SshHostKeys in the registry. Added to PuTTY's known hosts list by the GUI. Ignored on Macs where we've disabled host key checking entirely. If that's too much of a security risk, change *svn/gui/source/ClusterJob.cpp* and ask Mac users to ssh to the host once, check the key and add it to the known hosts list manually on every machine they wish to use the cluster from.
- Username: The Linux login for the transfer host, defaulting to the current user.
- Private Key: On Windows, the user's private key file that's used during authentication with the Linux host in PuTTY .ppk format. Use *puttygen.exe* if you need to convert the key from ssh's format. At the IMP we kept the keys in the user's private directory on a share so they'd be accessible from every machine on the intranet. Ignored on Mac, where *~/.ssh/* is used automatically and needs to be set up with the private key accordingly.
- Remote Environment: A script to call right after making the secure connection. The script can be used to make the GridEngine commands available.
- Polling Interval: Jobs running on the cluster are polled to update their status in the GUI.

To process on the cluster, both the video and the project must be stored on a network share that's write-accessible from both the local computer (running the GUI) and the cluster nodes. Note that if ***Local Processing*** is checked, MateBook will process videos locally if their path or the project's path doesn't begin with *//* on Windows or */Volumes/* on the Mac. Users should avoid using mapped network drives when adding videos to a project or opening / saving the project. Instead, the full network path should be used. (BUG #142.)

The GUI passes three paths to the tracker: the video path, the output directory path and the path to the settings file. These have to be translated from the Windows or Mac paths to the Linux paths. The GUI does not call qsub directly, but uses the *svn/tracker/binaries/Linux/Release/qsub_track.sh* script to launch the *track.sh* script on the cluster nodes. It's important to edit qsub_track.sh and change the sed expressions to make the path translation work for the network being used. (See *test.sh* in the same SVN directory for an attempt to simplify the script so the translation would have to be specified only once for all three paths. It's not finished.) The scripts should be installed together with the tracker executable. At the IMP they are installed at */projects/DIK.screen/tracker/<version>/*, where <version> is the SVN revision number. Multiple tracker versions are installed in parallel and the GUI picks the matching one. Unfortunately, this path to qsub_track.sh is hardcoded in the GUI. See FileItem::createJob and ArenaItem::createJob in *svn/gui/source/FileItem.cpp* and *ArenaItem.cpp*, respectively, to change it.

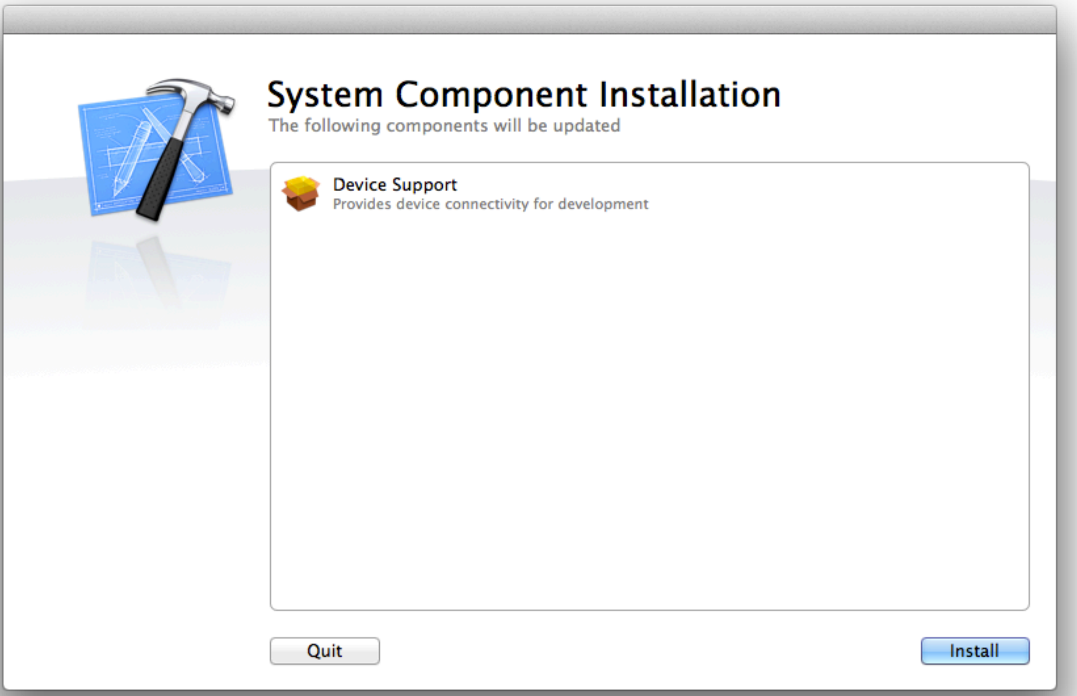
3.3. Setup for Software Developers

3.3.1. Windows

In addition to the software installed for users (Section 3.1.1), developers require Microsoft Visual Studio 2010 Professional SP1 and the open source Qt libraries, version 4.8.4[5], as well as the Qt Visual Studio Add-In for Qt4[6]. All software should be installed to the default directories if possible. Developers may have to select the correct Qt version in Visual Studio's ***Qt / Qt Options*** menu. Install the OpenCV 2.3.0 Windows superpack[7] to *C:\OpenCV\2.3.0\* and make a symbolic link *C:\OpenCV\current* refer to that directory by running *C:\OpenCV>mklink /D current 2.3.0* from an admin command prompt. If you can open *C:\OpenCV\current\build\* it's installed correctly. This is the path used in the Visual Studio projects.

3.3.2. Mac

Mac developers need Xcode version 4. Start Xcode and install *Device Support* if the dialog comes up.



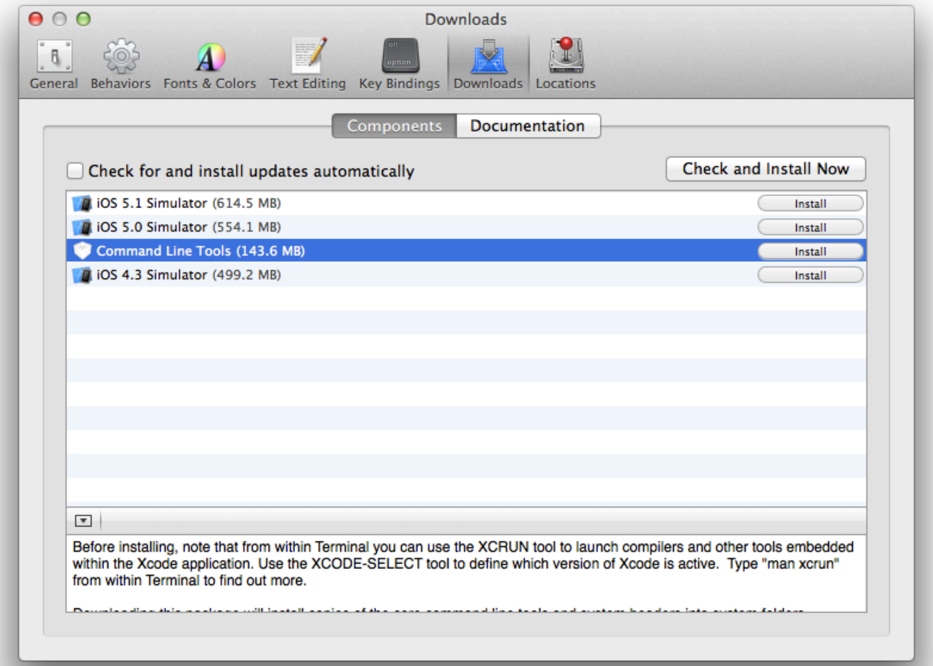
System Component Installation

The following components will be updated

Device Support
Provides device connectivity for development

Go to ***Xcode / Preferences / Downloads*** and under the ***Components*** tab install the *Command Line Tools*:

[5] http://download.qt-project.org/official_releases/qt/4.8/4.8.4/qt-win-opensource-4.8.4-vs2010.exe
[6] http://download.qt-project.org/official_releases/vsaddin/qt-vs-addin-1.1.11-opensource.exe
[7] http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3/OpenCV-2.3.0-win-superpack.exe/download



Check for and install updates automatically

iOS 5.1 Simulator (614.5 MB)
iOS 5.0 Simulator (554.1 MB)
Command Line Tools (143.6 MB)
iOS 4.3 Simulator (499.2 MB)

Before installing, note that from within Terminal you can use the XCRUN tool to launch compilers and other tools embedded within the Xcode application. Use the XCODE-SELECT tool to define which version of Xcode is active. Type "man xcrun" from within Terminal to find out more.

Downloading this package will install copies of the core command line tools and system headers into system folders

Installing Qt debug libraries[8] is optional, but recommended.

4. Software Developer's Guide

4.1. A High-Level Overview

Each module or library is kept in a subdirectory under the SVN root. They are:

- boost: Version 1.49.0 of the boost libraries[9] used by both the GUI and the tracker. (For Linux we used 1.47.0 from outside of SVN, but 1.49.0 should work as well.)
- common: C++ utility functions used by both the GUI and the tracker.
- ffmpeg: Version 0.11.1 of the FFMPEG libraries are used by both the GUI and the tracker to decode (or encode) videos.
- glew: The OpenGL Extension Wrangler Library[10] version 1.6.0, needed only on Windows. Used in the GUI and the grapher. We use the prebuilt binaries from the website.
- grapher: A hardware-accelerated interactive graph drawing widget used by the GUI to display tracked attributes over time.

[8] http://download.qt-project.org/official_releases/qt/4.8/4.8.2/qt-mac-opensource-4.8.2-debug-libs.dmg
[9] http://www.boost.org/
[10] http://glew.sourceforge.net/

- gui: The MateBook GUI. A Qt application for Windows and Mac. We compile it as a 32 Bit executable on Windows, using the prebuilt Qt libraries. Compiling a 64 Bit version works, but requires 64 Bit Qt libraries that have to be built also.
- images: Currently contains a high-res MateBook logo that's not being used in the software.
- lame: MP3 decoder library needed for compiling FFMPEG on the Mac.
- mediawrapper: A high-level C++ wrapper around FFMPEG for easy video reading and writing.
- opencv2.2: OpenCV .dylibs and include files for Mac. See Section 3.3.1 for how to install OpenCV on Windows. OpenCV is used only by the tracker.
- test: Currently contains the command line build script test.pl that can build, package and deploy new SVN revisions of MateBook for Windows, Mac and Linux automatically. We planned to turn this into a test script to compare and report tracking results from various revisions that are committed.
- tracker: The tracker module for Windows, Mac and Linux. On Windows we build 32 Bit executables. 64 Bit executables can be built, but are broken. The tracker does not depend on Qt.
- zlib: Required on Mac?

Most of the development is carried out by opening *svn/gui/build.vs10/gui.sln* and *svn/tracker/build.vs10/tracker.sln* on Windows or *svn/gui/build.xc3/MateBook.xcodeproj* on Mac. Note that the Mac project is actually an Xcode 4 project that builds both the GUI and the tracker, while on windows they're built from two separate solutions.

While working with MateBook, the GUI launches the tracker in a separate process. This is how MateBook makes use of multi-core processor machines: as long as the user has more videos than cores, the machine can be kept 100% busy.

The following assumes you're developing on Windows. Select ***File / Settings…*** and ***System*** in the MateBook GUI. The ***Local Processing*** section offers a few settings that can be very useful for debugging:

- Tracker: The path to the tracker executable that should be invoked by the GUI. Can be used to switch to a debug build of the tracker without restarting the GUI, e.g. to run the arena detection with the fast release build and run fly tracking with the debug build.
- Maximum Number of Jobs: The number of tracker processes to run in parallel.
- Attach Debugger: Calls __debugbreak() in the beginning of the tracker so that the Visual Studio debugger can be attached to it.
- Live Visualization: Opens an OpenCV HighGUI window to display segmentation information while a video is being tracked. This has not been used in a while.

…

4.2. Compiling the Libraries

We have built and committed the necessary libraries to SVN. If they ever need be rebuilt, here's how we did it:

4.2.1. Boost

In our repository, the svn/boost directory has an *svn:externals* property set to *-r77131 http://svn.boost.org/svn/boost/tags/release/Boost_1_49_0 source* to pull the source code from the boost project's SVN server into our *svn/boost/source* directory automatically on checkout. Run *svn/boost/build.win/build.bat* or *svn/boost/build.xc3/build.sh* to compile boost on Windows or Mac, respectively. Note that only the *libboost_filesystem* and *libboost_system* libraries need to be linked against.

4.2.2. FFMPEG

We are cross-compiling FFMPEG for Windows on a Debian Linux (Lenny) machine called *hutter*. Because lib.exe from Visual Studio has to be run between make and make install, it makes sense to perform the build on a network drive that's accessible from both Linux and Windows:

```
machacek@hutter:/...$ svn checkout svn+ssh://machacek@svn/ffmpeg
```

First we have to build the MinGW toolchain. Build scripts[11] are already in SVN and can be started like this:

```
machacek@hutter:/.../ffmpeg/build.win$ ./mingw-w64-build-2.8.4
```

When asked, build both the 32-bit and the 64-bit toolchain. Note that it's only necessary to build the toolchain once. Future FFMPEG builds can use the same toolchain. Change to the *svn/ffmpeg/source* directory and call *./download.sh 0.11.1* to download the source. Next change back to *build.win* and edit build.sh to fix the absolute path there. It should point to the *lib.exe-intercept* subdirectory of *svn/ffmpeg/build.win*. Run build.sh to compile the given version of FFMPEG:

```
machacek@hutter:/.../ffmpeg/build.win$ ./build.sh 0.11.1
```

This script also intercepts calls to lib.exe (which is not available on Linux) and writes them to *convert-to-lib.bat*. From a Visual Studio 32-bit Command Prompt on Windows, call:

```
U:\...\ffmpeg\build.win\ffmpeg-0.11.1-build-i686>convert-to-lib.bat
```

From a Visual Studio 64-bit Command Prompt call:

```
U:\...\ffmpeg\build.win\ffmpeg-0.11.1-build-x86_64>convert-to-lib.bat
```

Lastly, we go back to Linux and run install.sh:

```
machacek@hutter:/.../ffmpeg/build.win$ ./install.sh 0.11.1
```

The .dll and .lib files can now be found in the ffmpeg/binaries and ffmpeg/lib folders. They will automatically be used for any gui or tracker builds and should be committed to SVN if they prove to be working.

[11] from http://www.zeranoe.com/scripts/mingw_w64_build/

...but since this is an awful lot of work, one can also just pick up the shared and dev packages at http://ffmpeg.zeranoe.com/builds/ (built from http://www.ffmpeg.org/index.html) and use the *lib*, *include* and *bin* directories. Also, one has to add *inttypes.h* and *stdint.h* to the include directory; those two files are already in SVN. The main reason we compiled it ourselves was to test various optimization settings to improve video decoding speed. (BUG #1.) This turned out to be unnecessary as the bottleneck was in the mediawrapper code. Note that it's currently not possible to get FFMPEG debugging symbols for Visual Studio.

4.3. The GUI Code Structure

4.4. The Tracker Code Structure

4.5. Algorithm Details

5. Acknowledgements

Many people have contributed to MateBook over the years.

- Barry Dickson: Has been leading this project since the beginning, always eager to improve small details while not losing sight of the big picture.
- Alexander Gabler: Helped implementing the original prototype, focusing on video quality assessment.
- Herbert Grasberger: Implemented the mediawrapper and grapher modules. Drew the first GUI layout with Qt Designer. Did all of the Mac-specific development and wrote parts of this documentation.
- Ivo Hofacker: Helped with the first correct formulation of the dynamic programming algorithm we use for both occlusion resolution and heading assignment.
- Christian Machacek: Reimplemented the original MATLAB tracker in C++, while optimizing the processing pipeline. Consolidated identity- and heading-assignment. Conceived of and implemented heading assignment based on body color. Designed the GUI code and implemented most of the GUI. Implemented cluster processing. Wrote this documentation.
- Thomas Micheler: Kept the web-based user interface to the original MATLAB tracker prototype running over many years.
- Ines Ribeiro: Did lots of testing during the later stages of development and reported many bugs. Helped with this documentation.
- Christian Schusterreiter: Researched and evaluated most of the algorithms used in the tracker and implemented the original working prototype in MATLAB.
- Alexander Seitinger: Implemented GUI support for fly song analysis and improved other parts of the GUI while doing so.
- Laszlo Tirian: Helped evaluating algorithms in the original prototype.

We would like to thank all those who tried to use MateBook for serious research long before it was ready. :-)