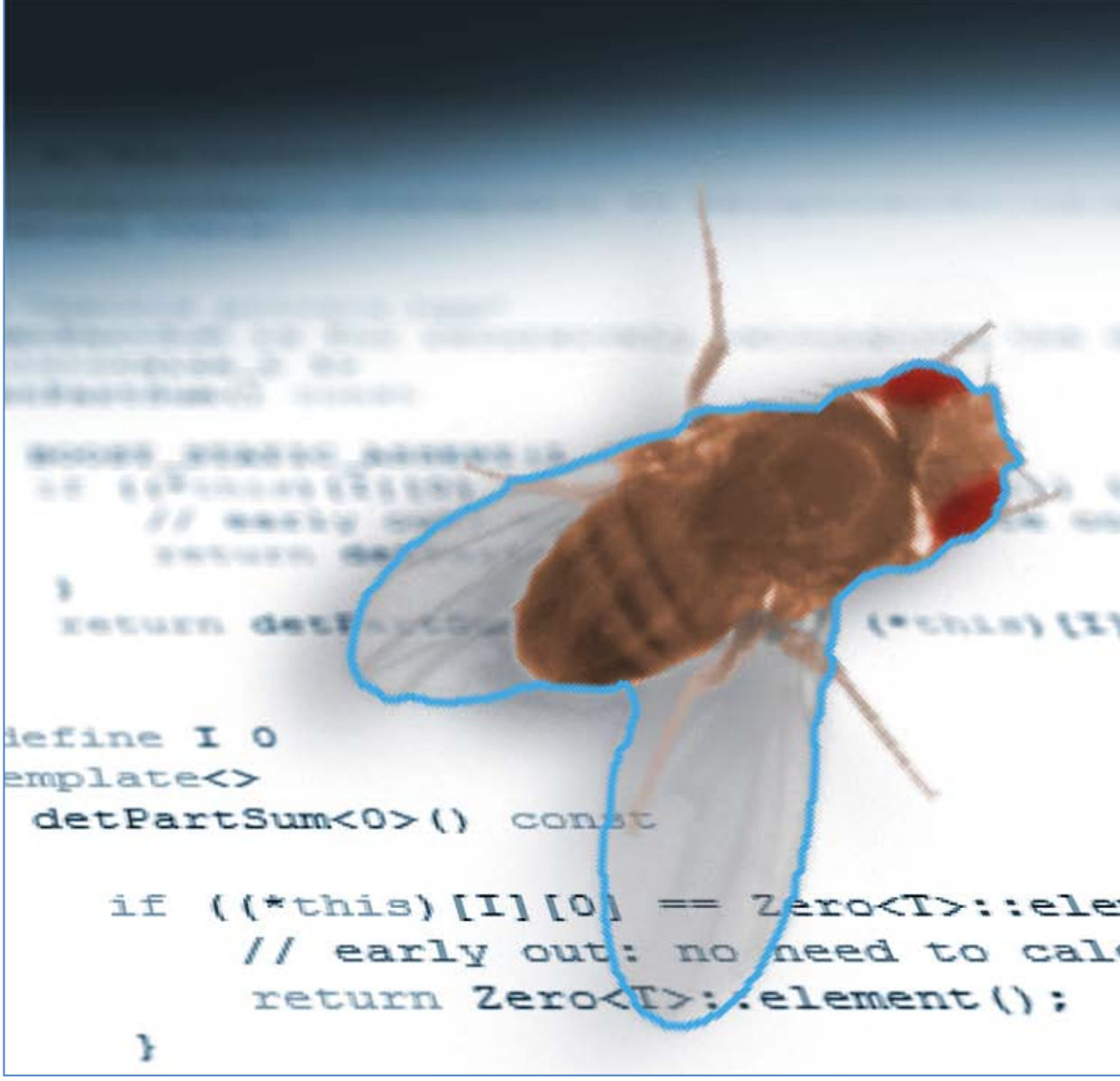


MateBook

Documentation for SVN revision 2141

Contents

1. Introduction and Overview 4
2. User's Guide 4
   - 2.1. Projects 4
   - 2.2. The Main Window 4
   - 2.3. A First Tutorial 5
   - 2.4. Settings 6
     - 2.4.1. Arena Detection 7
     - 2.4.2. Fly Tracking 8
     - 2.4.3. Events 10
     - 2.4.4. Ethograms 10
     - 2.4.5. Behavior Analysis 10
     - 2.4.6. Display 10
     - 2.4.7. System 10
   - 2.5. Measured Attributes 10
     - 2.5.1. Frame-attributes 11
     - 2.5.2. Fly-attributes 14
     - 2.5.3. Pair-attributes 20
3. System Administrator's Guide 22
   - 3.1. Setup for Users 22
     - 3.1.1. Windows 22
     - 3.1.2. Mac 23
   - 3.2. Setup for Cluster Processing 23
   - 3.3. Setup for Software Developers 24
     - 3.3.1. Windows 24
     - 3.3.2. Mac 25
4. Software Developer's Guide 26
   - 4.1. A High-Level Overview 26
   - 4.2. Compiling the Libraries 27
     - 4.2.1. Boost 28
     - 4.2.2. FFMPEG 28
   - 4.3. The GUI Code Structure 29

4.3.1. Tabs 29
4.3.2. Settings 29
4.3.3. Jobs 29
4.3.4. Tables, Trees and Qt Model/View Programming 30
4.3.5. Grapher 31
4.4. The Tracker Code Structure 31
4.4.1. The main function 31
4.4.2. Immutable Attributes 32
4.5. Algorithm Details 33
4.5.1. Background Extraction 33
4.5.2. Arena Detection 33
4.5.3. Arena::track() 34
4.5.4. hofacker() 34
5. Acknowledgements 35

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 mm². Hovering over the graphs you can read off the exact values.

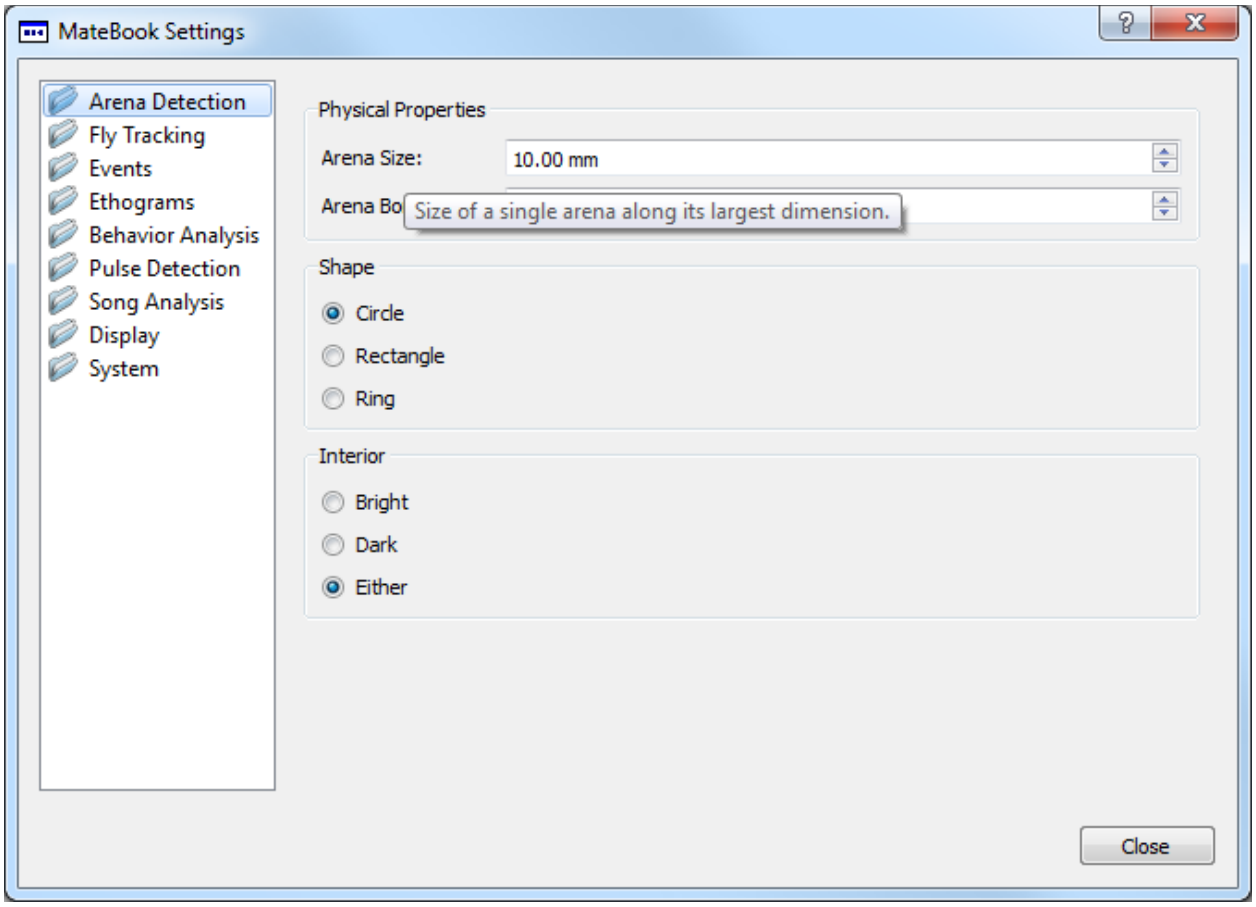
J. Press the play button in the left video player and notice that the player is synchronized with the graphs. The current frame corresponds to the values graphed at the central green line. You can also left-click and drag the graphs. Move your mouse over the graphs and use the mouse wheel (two-finger 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.



Use the mouse to hover over labels and buttons in the settings dialog to get a short description of the setting. The dialog presents the settings in different categories that can be selected on the left.

2.4.1. Arena Detection

Arena Size

For circular arenas, the diameter in mm. For linear, horizontal arenas specify the width in mm.

Arena Border Size

Due to parallax the arena border is sometimes not a sharp line. This setting reduces the area of the arena that is filled with a uniform background color. A border size value that is too low may cause parts of the arena walls to be filled with the background color, leading to false positives during segmentation. A value that is too high may cause motionless flies near borders to show up in the background image, leading to false negatives during segmentation.

Shape

Select *Circle* for courtship assays and either *Rectangle* or *Ring* for walking assays.

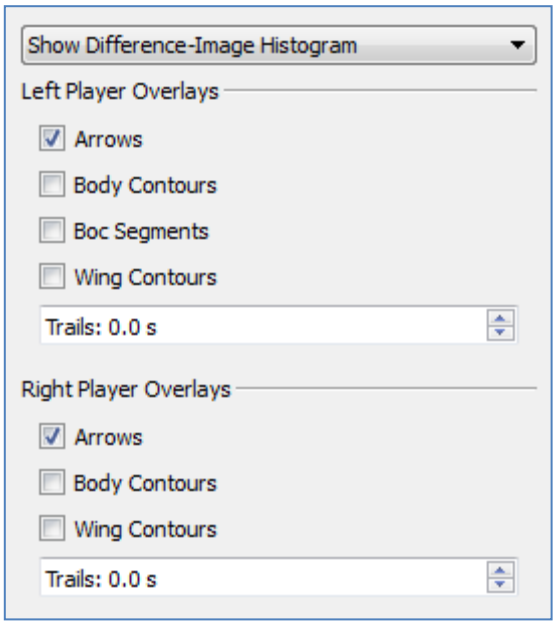
Interior

Arena detection accuracy can sometimes be improved if it's specified whether the interior is *Brighter* or *Darker* than the exterior. The default is *Either* and should work fine in most cases.

2.4.2. Fly Tracking

Flies per Arena

Only 1 or 2 flies per arena are currently supported.



Save Contours

When checked, the contours of the segmented regions are saved for each frame. This requires a lot of storage space, but can be useful to judge the segmentation quality. To view the contours after tracking has completed, select the *Body Contours*, *Boc Segments* and/or *Wing Contours* checkboxes in the action bar of the Arena tab. Boc segments shows the contour-pieces attributed to each fly during an occlusion.

Save Histograms

When checked, histograms of the difference-image (the difference between the current frame and the background image) are saved and can be viewed by selecting *Show Difference-Image Histogram* in the action bar of the Arena tab.

Gradient Correction

When checked, the tracker tries to expand the segmented body area until it hits a large change in the difference image. This may improve segmentation, but doubles tracking time and is usually not needed.

Discard Missegmentations

When checked, missegmented frames are not used to determine the fly sizes for identity assignment.

Fully Merge Missegmentations

When checked and multiple body areas are detected within the same wing area and those body areas don't have the expected size, they are all merged into a single body area. This may improve wing extension recovery, but also generate more occlusions.

Split Bodies

Attempt to split bodies during occlusions using a competitive flood-fill operation with the boc contours taken as the seed. These split bodies can in turn be used as a seed in the same kind of operation to recover the wing areas, see below.

Split Wings
Attempt to split wing areas if they contain multiple bodies using them as seed for a competitive flood-fill operation.

Threshold Offset
Can be used to bias the body segmentation threshold.

Minimum Fly Body Size
Anything smaller than this is considered to be a missegmented body area. Default: 0.5 mm².

Maximum Fly Body Size
Anything larger than this is considered to be a missegmented body area. Default: 2 mm².

sSize Linear Weight
The factor used to scale the sSize attribute before identity assignment. Higher values mean that more importance is given to fly size differences.

tPos Logistic Regression Coefficient
Coefficient c used to transform tPosScore to tPosProb via $tPosProb = 1 / (1 + e^{-tPosProb * c})$. The optimal value was determined to be c = 6.6 using a set of videos for which the identity assignment was corrected manually. Higher values for c mean that more importance is given to fly position differences before and after occlusions when deciding on fly identities.

tBoc Logistic Regression Coefficient
As above, but the optimal values was determined to be c = 5.32. For higher c, more importance is given to the contours tracked through occlusions when deciding on fly identities.

sMotion Weight
Higher numbers give more importance to a fly's direction of motion when determining which tip of the body ellipse corresponds to the head.

sWings Weight
The weight given to information from the wing segmentation when deciding which tip of the body ellipse is the head.

sMaxMotionWings Weight
This uses the better (larger) of the above information.

sColor Weight
The weight given to body color information during heading assignment.

tBefore Weight
The weight given to heading persistence during heading assignment. For larger values, the algorithm assumes that fast direction changes are less likely.

Use Manual Occlusion Solution

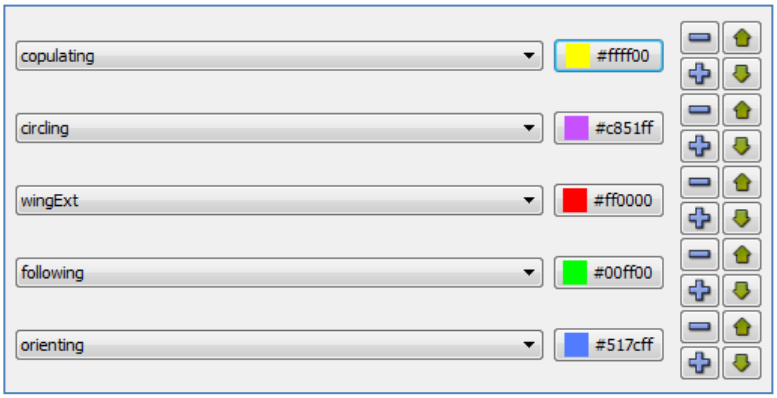
When checked and a manual identity assignment is available, use that instead of the automatic one given by the tracker.

2.4.3. Events

Here, various thresholds can be set to control the event detection. Please use the mouse to hover over the labels to get a description of each one.

2.4.4. Ethograms

Select the events you want to show up in the ethograms. These are color-coded behavior timelines. Click + to add an event and – to remove an event. When multiple events occur at the same time, the top-most list entry overrules the others.



2.4.5. Behavior Analysis

Bin Size and Bin Count

Behavior can be summarized for time-bins in addition to the entire video during step L. of the previously described MateBook workflow (2.3). To disable this feature, set the *Bin Count* to 0.

Heatmaps

Heatmaps can be drawn for all 2D attributes that take on real values. Additionally, frames used in heatmaps can be filtered using thresholds on any of the other attributes.

2.4.6. Display

Fly Colors

Select the colors used to draw arrow, contours and plots. Currently broken. (BUG #36.)

2.4.7. System

See section 4.1 for information about the local processing settings and section 3.2 for information about cluster processing.

2.5. Measured Attributes

The output of the tracker is a set of measurements we call *attributes*. An attribute describes a certain aspect of the arena or the flies within the arena and consists of one or two numbers per video frame. Examples for attributes are *isMissegmented* (an attribute that's 0 or 1 depending on whether the fly bodies were properly detected in this frame or not) and *distanceBodyBody* (the distance between two flies). All attributes can be plotted in the Arena tab. They are also reported in track.tsv files under the MateBook project directories. Microsoft Excel can be used to view these.

Attributes are of different kinds. Some are measured per video frame (like isMissegmented mentioned above), some per fly (like *bodyArea*, the size of the fly's body in the video) and some per fly pair (like distanceBodyBody). We call these *frame-attributes*, *fly-attributes* and *pair-attributes*, respectively. These categories are also used to group the attributes in the drop-down boxes in the GUI. Within each group, the attributes are sorted alphabetically.

In addition, we distinguish between attributes that can take real values, integer values or binary values. The latter we will sometimes call *events*. For the user, the most interesting attributes are typically the *courtship events*, namely *circling*, *copulating*, *following*, *orienting* and *wing extension*.

Most attributes are derived from other attributes; only a few are calculated directly from the pixels of the video frames. For some attributes, a second variant (ending in "_u") is given. It describes the same measurement, only with more natural units like millimeters or seconds rather than pixels or frames.

What follows is an exhaustive list of all attributes and a short description of each.

2.5.1. Frame-attributes

averageBodyCentroid
Average position of all body centroids in global 2D pixel coordinates.

bodyContourOffset
Offset into the contour file to access the body contours for this (occluded) frame.

bodyThreshold
The threshold picked for body segmentation.

courtship
Whether any flies are courting in this frame.

idPermutation
In a sorted list of permutations, this is the index of the permutation that was applied to the flies to assign the correct identities.

interpolated
Body attributes are interpolated.

interpolatedAbsolutely
Body attributes are interpolated absolutely.

interpolatedRelatively
Body attributes are interpolated relative to the averageBodyCentroid.

isMissegmented
Whether this frame has been missegmented.

isMissegmentedL
One fly is too large, the other one okay.

isMissegmentedLL
Both flies are too large.

isMissegmentedS
One fly is too small, the other one okay.

isMissegmentedSL
One fly is too small, the other one too large.

isMissegmentedSS
Both flies are too small.

isMissegmentedUnmerged
Missegmentations that have not been merged into occlusions.

isOcclusion
Whether this frame is occluded.

isOcclusionTouched
Whether this frame is occluded due to flies touching.

occlusionInspected
Whether this frame is part of an occlusion that has been been switched by the user at some point.

sCombined
Weighted and combined s values for non-occluded sequences.

sSize
Size-based logodd for occlusion resolution.

sSizeProb
Size-based probability for occlusion resolution.

tBoc
Contour-based logodd for occlusion resolution.

tBocProb
Contour-based probability for occlusion resolution.

tBocScore
Contour-based score in [-1,1] for occlusion resolution.

tCombined
Weighted and combined t values for occluded sequences.

tMov
Motion-based logodd for occlusion resolution.

tMovProb
Motion-based probability for occlusion resolution.

tMovScore
Motion-based score in [-1,1] for occlusion resolution.

tPos
Position-based logodd for occlusion resolution.

tPosProb
Position-based probability for occlusion resolution.

tPosScore
Position-based score in [-1,1] for occlusion resolution.

trackedFrame
Index of the video frame, relative to the beginning of the tracked sequence.

trackedFrameRelative
0.0 is the beginning of the tracked sequence, while 1.0 is the end.

trackedTime
Time passed since the beginning of the tracked sequence in seconds.

videoFrame
Index of the video frame, relative to the beginning of the video.

videoFrameRelative
0.0 is the beginning of the video, while 1.0 is the end.

videoTime
Time passed since the beginning of the video in seconds.

wingContourOffset
Offset into the contour file to access the wing contours for this (occluded) frame.

wingExtCallableDuringOcclusion
Whether wingExt can be called during occlusions.

wingThreshold
The threshold picked for wing segmentation.

2.5.2. Fly-attributes

bocContourOffset
Offset into the contour file to access the boc contour.

bodyArea
Fly body area in pixels.

bodyAreaEccentricityCorrected
Fly body area after correcting for posture in pixels.

bodyAreaEccentricityCorrected_u
Fly body area after correcting for posture in mm².

bodyAreaEccentricityCorrectedTracked
Fly body area after correcting for posture (as reported during tracking) in pixels.

bodyAreaTracked
Fly body area (as reported during tracking) in pixels.

bodyCentroid
Centroid of the ellipse fit to the fly body in pixels.

bodyCentroid_u
Centroid of the ellipse fit to the fly body in mm.

bodyCentroidTracked
Centroid of the ellipse fit to the fly body (as reported during tracking) in pixels.

bodyContourOffset
Offset into the contour file to access the body contour.

bodyContourSize
Number of pixels that are part of the body contour.

bodyEccentricity
Eccentricity of the ellipse fit to the fly body.

bodyMajorAxisLength
Major axis length of the ellipse fit to the fly body in pixels.

bodyMajorAxisLength_u
Major axis length of the ellipse fit to the fly body in mm.

bodyMajorAxisLengthTracked
Major axis length of the ellipse fit to the fly body (as reported during tracking) in pixels.

bodyMinorAxisLength
Minor axis length of the ellipse fit to the fly body in pixels.

bodyMinorAxisLength_u
Minor axis length of the ellipse fit to the fly body in mm.

bodyMinorAxisLengthTracked
Minor axis length of the ellipse fit to the fly body (as reported during tracking) in pixels.

bodyOrientation
Global orientation of the ellipse fit to the fly body, after reinterpolating using the final heading in degrees.

bodyOrientationFlipped
Global orientation of the ellipse fit to the fly body, after determining heading in degrees.

bodyOrientationTracked
Global orientation of the ellipse fit to the fly body (as reported during tracking) in degrees.

bodySplit
Whether the body contour is the result of a split operation.

bottomLeftBodyArea
Potential body area in one of the wing quadrants in pixels.

bottomLeftWingAngle
Potential wing angle in one of the quadrants in degrees.

bottomLeftWingArea
Potential wing area in one of the quadrants in pixels.

bottomLeftWingTip
Potential wing tip in one of the quadrants in pixels.

bottomRightBodyArea
Potential body area in one of the wing quadrants in pixels.

bottomRightWingAngle
Potential wing angle in one of the quadrants in degrees.

bottomRightWingArea
Potential wing area in one of the quadrants in pixels.

bottomRightWingTip
Potential wing tip in one of the quadrants in pixels.

circAboveMinSidewaysSpeed
Fly is moving sideways quickly.

circAboveMinSpeedSelf
Speed is above the threshold set for circling for the active fly.

circBelowMaxSpeedOther
Speed is below the threshold set for circling for the passive fly.

circMovedSideways
Fly is moving sideways.

copulating
Fly is copulating.

courting
Fly is courting.

distanceFromArenaCenter
The distance between the fly body centroid and the center of the arena bounding box in pixels.

distanceFromArenaCenter_u
The distance between the fly body centroid and the center of the arena bounding box in mm.

filteredBodyCentroid
Smoother version of bodyCentroid in pixels.

filteredBodyCentroid_u
Smoother version of bodyCentroid in mm.

filteredHeading
The decision made for heading.

follAboveMinSpeedOther
Speed is above the threshold set for following for the passive fly.

follAboveMinSpeedSelf
Speed is above the threshold set for following for the active fly.

headingFromBefore
Heading score based on the persistence of orientation.

headingFromBody
Heading score based on the shape of the fly body.

headingFromColor
Heading score based on the color distribution within the fly body.

headingFromMaxMotionWings
The more extreme of headingFromMotion and headingFromWings.

headingFromMotion
Heading score based on the direction of motion.

headingFromWings
Heading score based on the position of the wing ellipse relative to the body ellipse (zeroed out for interpolated frames).

headingFromWingsTracked
Heading score based on the position of the wing ellipse relative to the body ellipse (as reported during tracking).

headingSCombined
Weighted and combined s values for heading.

headingTCombined
Weighted and combined t values for heading.

leftBodyArea
Actual area of the body within the left wing quadrant in pixels.

leftWingAngle
Actual angle of the left wing in degrees.

leftWingArea
Actual area of the left wing in pixels.

leftWingTip
Actual tip of the left wing in pixels.

moved
Motion vector since the last frame in pixels.

moved_u
Motion vector since the last frame in mm.

movedAbs
The distance moved since the last frame in pixels.

movedAbs_u
The distance moved since the last frame in mm.

movedDirectionGlobal
Global motion direction since the last frame in degrees.

movedDirectionLocal
Local motion direction since the last frame in degrees.

oriBelowMaxSpeedOther
Speed is below the threshold set for orienting for the passive fly.

oriBelowMaxSpeedSelf
Speed is below the threshold set for orienting for the active fly.

rayEllipseOriBelowMaxSpeedOther
Speed is below the threshold set for orienting for the passive fly.

rayEllipseOriBelowMaxSpeedSelf
Speed is below the threshold set for orienting for the active fly.

rightBodyArea
Actual area of the body within the right wing quadrant in pixels.

rightWingAngle
Actual angle of the right wing in degrees.

rightWingArea
Actual area of the right wing in pixels.

rightWingTip
Actual tip of the right wing in pixels.

topLeftBodyArea
Potential body area in one of the wing quadrants in pixels.

topLeftWingAngle
Potential wing angle in one of the quadrants in degrees.

topLeftWingArea
Potential wing area in one of the quadrants in pixels.

topLeftWingTip
Potential wing tip in one of the quadrants in pixels.

topRightBodyArea
Potential body area in one of the wing quadrants in pixels.

topRightWingAngle
Potential wing angle in one of the quadrants in degrees.

topRightWingArea
Potential wing area in one of the quadrants in pixels.

topRightWingTip
Potential wing tip in one of the quadrants in pixels.

turned
The change of orientation since the last frame in degrees.

turnedAbs
The absolute value of the change of orientation since the last frame in degrees.

weightedCourting
Weighted combination of courtship events.

wingArea
Fly wing area, which includes the body area in pixels.

wingCentroid
Centroid of the ellipse fit to the fly wings in pixels.

wingContourOffset
Offset into the contour file to access the wing contour.

wingContourSize
Number of pixels that are part of the wing contour.

wingConvexArea
Area of the convex hull of the wings in pixels.

wingEccentricity
Eccentricity of the ellipse fit to the fly wings.

wingExt
Any wing extended.

wingExtAngleLeft
Left wing fulfilling the angle criterion.

wingExtAngleRight
Right wing fulfilling the angle criterion.

wingExtAreaLeft
Left wing fulfilling the area criterion.

wingExtAreaRight
Right wing fulfilling the area criterion.

wingExtBoth
Both wings extended.

wingExtEitherOr
Exactly one wing extended.

wingExtLeft
Left wing extended.

wingExtLeftOccurred
Whether wingExtLeft has occurred in any frames up to and including this one.

wingExtOccurred
Whether wingExt has occurred in any frames up to and including this one.

wingExtRight
Right wing extended.

wingExtRightOccurred
Whether wingExtRight has occurred in any frames up to and including this one.

wingMajorAxisLength
Major axis length of the ellipse fit to the fly wings in pixels.

wingMinorAxisLength
Minor axis length of the ellipse fit to the fly wings in pixels.

wingOrientation
Global orientation of the ellipse fit to the fly wings in degrees.

2.5.3. Pair-attributes

angleToOther
Angle between the major axis and the line connecting the body centroids in degrees.

changeInDistanceHeadBody
Speed at which distanceHeadBody changes in pixels/frame.

changeInDistanceHeadBody_u
Speed at which distanceHeadBody changes in mm/s.

circAngle
Circling angle criterion fulfilled.

circDistance
Circling distance criterion fulfilled.

circling
This fly is circling the other fly.

distanceBodyBody
Distance between body centroids in pixels.

distanceBodyBody_u
Distance between body centroids in mm.

distanceHeadBody
Distance between this fly's head and the other fly's body centroids in pixels.

distanceHeadBody_u
Distance between this fly's head and the other fly's body centroids in mm.

distanceHeadTail
Distance between this fly's head and the other fly's tail in pixels.

distanceHeadTail_u
Distance between this fly's head and the other fly's tail in mm.

follAngle
Following angle criterion fulfilled.

follBehind
The distane between the following fly's head and the followed fly's tail (distanceHeadTail) is smaller than the distance between the followed fly's head and the following fly's tail.

follDistance
Following distance criterion fulfilled.

following
This fly is following the other fly.

followingOccurred
Whether following has occurred in any frames up to and including this one.

follSameMovedDirection
The flies move in roughly the same direction.

follSmallChangeInDistance
The changeInDistanceHeadBody attribute is below the threshold for following.

oriAngle
Orientation angle criterion fulfilled.

oriDistance
Orientation distance criterion fulfilled.

orienting
This fly is orienting itself towards the other fly.

rayEllipseOriAngle
Orientation angle criterion fulfilled.

rayEllipseOriDistance
Orientation distance criterion fulfilled.

rayEllipseOrienting
This fly is orienting itself towards the other fly.

rayEllipseOriHit
This fly's viewing ray hits the other fly's body.

vectorToOtherLocal
Vector in this fly's local coordinate frame pointing to the other fly's body centroid in pixels.

vectorToOtherLocal_u
Vector in this fly's local coordinate frame pointing to the other fly's body centroid in mm.

wingExtAway
This fly is extending its wing away from the other fly.

wingExtTowards
This fly is extending its wing towards the other fly.

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.

[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

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 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.

[4] http://download.qt-project.org/official_releases/qt/4.8/4.8.2/qt-mac-opensource-4.8.2.dmg

- 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

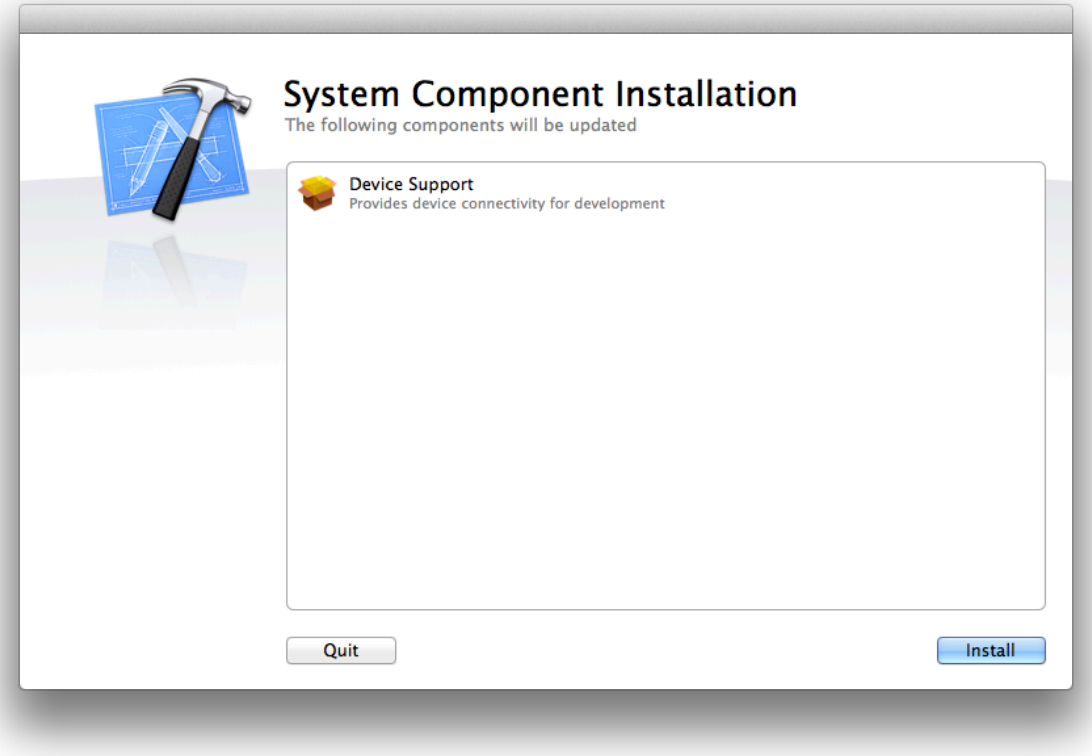
3.3.1. Windows

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.

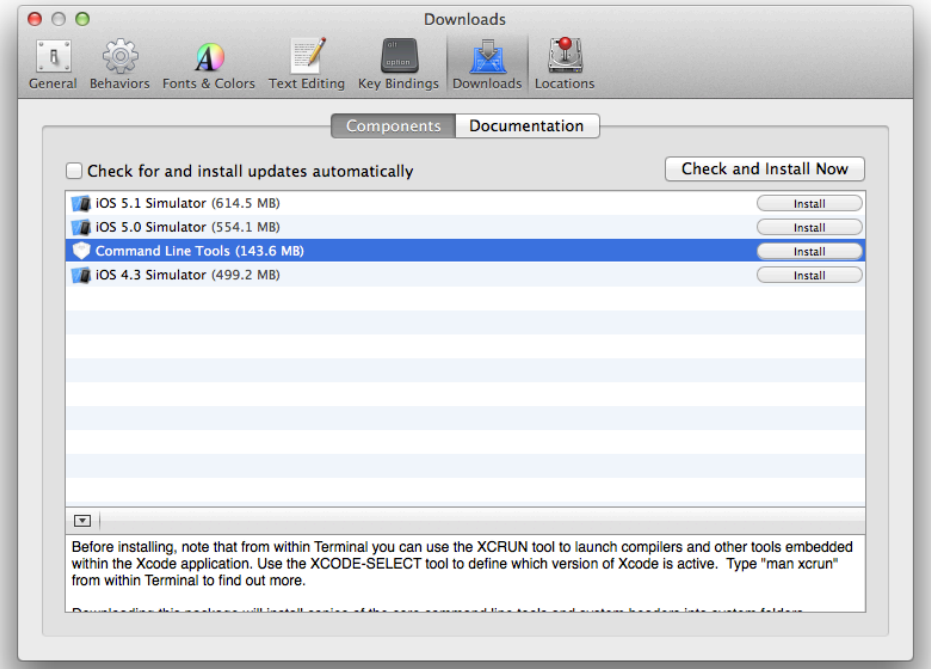
[5] http://download.qt-project.org/official_releases/qt/4.8/4.8.4/qt-win-opensource-4.8.4-vs2010.exe
[6] http://download.qt-project.org/official_releases/vsaddin/qt-vs-addin-1.1.11-opensource.exe
[7] http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3/OpenCV-2.3.0-win-superpack.exe/download

3.3.2. Mac

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



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



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 only on Mac, if at all.

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

The main entry point is in svn/gui/source/main.cpp and is typical for a Qt application. Note that we haven't used Qt Designer – the GUI is hand-coded. The main window and all application-wide functionality like loading or saving projects and settings is initiated in MateBook.cpp and MateBook.hpp.

4.3.1. Tabs

As you have seen, the GUI is organized in tabs. These are implemented in *Tab.cpp and *Tab.hpp, where AbstractTab is the base class. Concrete classes must implement a few virtual functions to get notified when a different project is loaded, a different item (video or arena) is selected by the user or when the active tab is changed. They may also override functions to get a chance to save or load tab-specific data.

4.3.2. Settings

The settings window (shown when the user selects *File/Settings…* as detailed in section 2.4) is implemented in ConfigDialog.cpp and ConfigDialog.hpp. Each page is implemented in one of the *Page.cpp and *Page.hpp files. ConfigPage is the base class. The ConfigDialog class provides access to some of the widget's values to the GUI. Note that the constructor is passed a *Settings& trackerSettings* reference, which is also passed on to the *Page constructors. The referenced object is a member of the MateBook class. Pages can register their widgets with this object. For example, the following code registers a QCheckBox:

```
saveContours = new QCheckBox(this);
saveContours->setText("Save Contours");
saveContours->setToolTip("Fly contours require a lot of storage space, but are useful to
judge the quality of segmentation.");
trackerSettings.add<bool>("contours", boost::bind(&QCheckBox::isChecked, saveContours),
boost::bind(&QCheckBox::setChecked, saveContours, _1));
```

The Settings object can then serialize all registered widgets' values in one go and write them into a file that's passed to the tracker process whenever it is launched. On the other hand, readGuiSettings() and writeGuiSettings() are used to store per-user per-system settings in the registry. See for example SystemPage.cpp.

4.3.3. Jobs

Every time the user launches the tracker (through the action bar in the Files tab), a Job object is created and inserted into a JobQueue. For local processing, an ExternalJob is created and the JobQueue makes

sure the number of tracker processes running is less than the number of CPUs available (or the number set in the System page of the settings dialog) so that the system stays responsive. For cluster processing, a ClusterJob is generated. The base class for both is *Job*. Upon completion or when encountering an error, jobs signal the new status. The Files tab then updates the background colors for the affected rows and the JobQueue may try to start new jobs.

4.3.4. Tables, Trees and Qt Model/View Programming

Creating tree-like tables, like the ones that make up the majority of the Files tab and the bottom of the Arena tab is non-trivial. The complexity stems from the fact that the data representation in the table (most often text, but in some cases widgets like checkboxes) does not match the internal data representation needed by the applications. For example, text may actually denote a number; checkboxes may actually denote a boolean variable.

Qt supports the programmer by providing customizable table widgets (the *view*) and leaving the internal representation (the *model*) up to the programmer, *as long as they follow the rules of the Qt model/view framework*. Please refer to the *Qt Model/View Tutorial*[12] for the details. Following these rules is hard because mistakes are not immediately obvious. They may cause a delay in the display of updated table data or occasional, difficult to reproduce crashes. The Qt developers recognized this and provide a way to at least exercise a model implementation[13] to uncover bugs sooner.

In MateBook, we use a heavily customized tree-like table, implemented in FancyTreeView.cpp and FancyTreeView.hpp. It's derived from the QTreeView widget, supports sorting and filtering (through the QSortFilterProxyModel member), context menus for cells and the header, as well as copy/pasting both within the table and to other programs like Excel. The *Delegate.cpp and *Delegate.hpp files contain classes that provide special widgets for some of the table columns. For example, the *Arenas Approved* column displays a checkbox rather than text input fields.

On the model-side, take a look at ItemTree.cpp and ItemTree.hpp. The ItemTree class implements the QAbstractItemModel. The actual data is stored in a vector<Item*>, where each Item may have children. The root items must be FileItem objects (that represent a video file), while their children are ArenaItems (representing individual arenas). Both have to implement the Item interface so they can be displayed together in the same table.

Because the view interacts with the model through QModelIndexes (that essentially provide a row and a column number to address the data), but our Item classes implement conventional C++ methods, we need to map column numbers to those methods. This is accomplished by the *Accessor classes, which are instantiated in the ItemTree constructor and kept in a vector<Accessor<Item>*>, one per table column. Accessors provide the interface required by Qt's model/view framework, but when given a particular Item object know which methods to call to get or set the data for the table column they are responsible for. They may also check the user's input and reject values that do not make sense.

[12] http://qt-project.org/doc/qt-4.8/modelview.html
[13] http://qt-project.org/wiki/Model_Test

As a shortcut, in our implementation the first column of the table contains the address of the Item referenced by each row.

4.3.5. Grapher

The grapher is an interactive graph drawing widget that can draw linegraphs, filled graphs, bargraphs, graphs with whiskers, makers via OpenGL. The widget is implemented in QGLGrapher.cpp and QGLGrapher.hpp, which is a misnomer: it's not part of Qt, but implemented by us. The grapher reports all sorts of mouse events through Qt signals. Individual graphs are created with the add* methods. The returned pointers must not be used to delete the objects. Instead, call the QGLGrapher::removeData method. The grapher makes a copy of the given data in graphics memory (using VBOs).

4.4. The Tracker Code Structure

While the GUI is event-based and reacts to user input, the tracker is a non-interactive program running the same sequential calculations every time it is launched. The tracker expects a few commandline parameters:

- --in <path>: path to the input video
- --out <path>: path to the directory where results shall be stored
- --settings <path>: path to the .tsv file with the tracker settings (see also section 4.3.2)
- --preprocess {0,1}: whether to run background extraction and arena detection
- --track {0,1}: whether to track flies
- --postprocess {0,1}: whether to calculate derived attributes that don't require the image data from the video frames
- --begin N: seconds into the video where to begin tracking
- --end N: seconds into the video where to end tracking

4.4.1. The main function

By looking at the main() function in main.cpp, the pipeline should become clear:

A. Commandline parameters are parsed.
B. Settings from the settings file are read.
C. The video file is opened and the metadata undergoes a sanity check.
D. The background image is extracted.
E. Arenas are detected in said background image.
F. Arena sizes and positions are saved to arena.tsv.
G. Arena masks (white for the interior part, black for the exterior part) are saved to mask.png.
H. The video is read frame-by-frame, each frame subtracted from the background and ellipses are fit to the largest connected areas in the resulting image. Primary attribute measurements (those which become available "live" during tracking) are saved to the *frames* member of each Arena object.
I. In normalizeTrackingData(), the primary attributes are transformed from the array-of-structures representation in *frames* to a structure-of-arrays representation in *frameAttributes*, *flyAttributes*

and *pairAttributes* (members of Arena) that lends itself better to deriving further attributes. The *frames* data structure is then freed.

J. In prepareInterpolation() short missegmented sequences with confident pos scores are fixed immediately.
K. We then figure out how we'll be interpolating fly positions during occlusions. If there are no missegmented frames, we use their common centroid as the reference. In other words, their positions are interpolated *relative to the common centroid*, which is itself moving during the occlusion.
L. Missegmentations are detected by comparing the segmented regions to the fly size thresholds and some statistics are written to segmentationStatistics.tsv.
M. In calculateTScores the tBoc*, tMov* and tPos* attributes which are used for identity assignment are calculated.
N. In solveOcclusions the sSize* attributes are calculated and the identity assignment is finalized by calling our dynamic programming optimization algorithm, hofacker(). Values that have been attributed to the wrong fly are swapped.
O. A report is written to occlusionReport.tsv that can be used to judge the quality of the individual scores. During the identity assignment, scores may change their sign. When the identity assignment is finished, a score becomes negative if it disagrees with the final (optimal) identity assignment and positive if it agrees. Scores that rightfully disagree often should be given a higher weight as explained in section 2.4.2.
P. Because identities have been decided, many attributes can now be interpolated through occlusions. The main exception is the flies' orientation. It is decided next.
Q. For the orientation we must determine which tip of the ellipse corresponds to the fly's head. This is done similar to how identities are assigned, using the hofacker() dynamic programming optimization algorithm. During occlusions we have no ellipse data and occluded frames are temporarily removed in this step.
R. Once heading is decided on, the orientation is interpolated for occluded frame sequences, fly quadrants that contain the wings are selected and further attributes that depend on heading are derived.
S. Some attributes are converted to more natural units like millimeters and seconds.
T. Events (following, wing extension, …) are derived.
U. Attributes are exported to both track.tsv and to the binary files in the track directory.
V. Some statistics are written to various .tsv files.

Instead of D.-G. (the *preprocessing*) the program attempts to load the files mentioned from disk if the command line sets --preprocess to 0. Instead of H.-I. (the *tracking*) the program attempts to load primary attributes from track.tsv if --track is 0.

4.4.2. Immutable Attributes

To keep the postprocessor code manageable, attributes are treated as immutable variables in almost all cases. That is, once calculated they retain their values until the end of the program. Having attribute

changes scattered throughout the program would make it next to impossible for the programmer to understand the dependencies and control flow.

Attributes are therefore made accessible in subsections of the tracker through one of two ways: If the intention is to calculate their values for the first time, we use

```
Attribute<float>& distanceFromArenaCenter =
flyAttributes[flyNumber].getEmpty<float>("distanceFromArenaCenter");
```

If the intention is to read them in order to calculate some other attribute, we use

```
const Attribute<Vf2>& bodyCentroid =
flyAttributes[flyNumber].getFilled<Vf2>("bodyCentroid");
```

That way the code is self-documenting. Note the *const* keyword and the *getEmpty* vs. *getFilled* calls. Const prevents changes at compile-time. The getEmpty and getFilled calls check (at runtime) to make sure the attribute is still empty (i.e. not calculated yet) or already filled (i.e. calculated). Keeping the returned references only for as long as necessary (e.g. by wrapping the calculations in { } blocks) improves code readability even more.

Whenever attribute changes are required, a new attribute with a new name is created. See for example Arena::interpolateAttributes(). The original values have the *Tracked* suffix attached to their names. One notable exception is FrameAttributes::swap and FlyAttributes::swap, where attributes are swapped in-place between flies once the correct identity assignment becomes available.

4.5. Algorithm Details

4.5.1. Background Extraction

See getBackground.cpp. We read a set of evenly spaced frames from the video. Because the exact frame number is not important, we use our mediawrapper labrary's seekApprox() function to get a frame close to the one we're requesting. This can be much faster than an exact seek() to a frame number. A median pixel color is then calculated for each pixel over time using C++'s *std::nth_element* function that takes linear time.

4.5.2. Arena Detection

See findArenas.cpp and findCircles.cpp. We use a grey-scale variant of the circular Hough transform. First we create a gradient image by convolving the background image with Sobel kernels. Then, an accumulation buffer is incremented for each pixel along its gradient, using the gradient magnitude as a weight. The buffer is blurred and local maxima are preselected as circle center candidates.

For all the candidate centers, the gradient magnitudes from their surroundings are accumulated in a 1D table, indexed by the distance. True circle centers will show a clear peak in this table, at the distance of the circle radius. We therefore rank the candidates by the entropy of this 1D distribution, from lowest to highest. For the first 10, we sum up their tables (element-wise) and pick the distance with the highest value. We assume that all arenas are circles with the same radius, given by that distance.

We repeat the first step, where an accumulation buffer was incremented along the gradients of the pixels, this time only incrementing at the distance given by the known radius. Similar to the 1D radius table before, we use a 2D radius/angle table of gradient magnitudes around the circle center candidates now. True centers will have a large gradient near the expected radius in every directions. False positives will not. Therefore, we pick the maximum gradient magnitude for each angle-bin and rank the candidates by the lowest maximum, from largest to smallest. We then accept arenas until there's one that would overlap an already accepted one.

4.5.3. Arena::track()

This method, called for each arena after each frame has been read is the "meat" of the tracking part. All the image analysis, segmentation and ellipse fitting is done here. It is one of the more difficult parts to understand, especially since C++ makes it hard to look at intermediate images in the debugger[14]. Pay close attention to the comments in the source code.

4.5.4. hofacker()

Because cameras provide a discrete, not a continuous record of what happened, and because occlusions lead to missing observations, we have only a limited number of time-limited direct observations available. While our goal is to have correct fly identity assignments and correct heading assignments at all times, deciding them by looking at each frame in isolation is suboptimal. Instead, we use prior knowledge about fly behavior to make educated guesses on what may have happened while we couldn't watch.

Our hofacker() dynamic programming algorithm is a variant of the Viterbi algorithm that integrates both direct observations (given as what we call *s-scores*) and the aforementioned guesses (*t-tcores*) to find the optimal solution over the entire video.

An example of a direct observation in the context of identity assignment is the fly size score *sSize*. Males are significantly smaller than females, so in a typical courtship video the size is a strong indicator for the identy. An example for a guess is the position change during an occlusion, *tPos*. The algorithm will try to keep fly identities sorted by size, while at the same time preferring smaller position changes.

A change (e.g. of identity) for a sequence of frames changes the sign of the t-scores at the beginning and the end of the sequence, as well as the sign of all the s-scores in-between. Rather than trying all possible changes, the algorithm makes use of the fact that *up to a given frame* only two identity assignments are of interest, the best one that leads to the first fly being the male, and the best one that leads to the second fly being the male. Whatever happens in the rest of the video, one of these two is going to be part of the solution. This is referred to as the *optimal substructure* property of the problem[15]. When reaching the final frame, the better of the two is chosen and the identity assignment for the entire video is reconstructed during a backward pass. This is a linear time algorithm.

[14] I'd suggest moving to Visual Studio 2012 and giving Microsoft's *Image Watch* plugin a try: http://visualstudiogallery.msdn.microsoft.com/e682d542-7ef3-402c-b857-bbfba714f78d

[15] http://en.wikipedia.org/wiki/Optimal_substructure

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. :-)