

Chapter 1

Segmenting and Tracking Multiple Dividing Targets using *ilastik*

by Carsten Haubold, Martin Schiegg, Anna Kreshuk, Stuart Berg, Ullrich Koethe
and Fred A. Hamprecht

Abstract *Tracking crowded cells or other targets in biology is often a challenging task due to poor signal-to-noise ratio, mutual occlusion, large displacements, little discernibility and the ability of cells to divide. We here present an open source implementation of conservation tracking (Schiegg et al, 2013) in the ilastik software framework. This robust tracking-by-assignment algorithm explicitly makes allowance for false positive detections, undersegmentation and cell division. We give an overview over the underlying algorithm and parameters, and explain the use for a light sheet microscopy sequence of a Drosophila embryo. Equipped with this knowledge, users will be able to track targets of interest in their own data.*

1.1 Introduction

Tracking multiple indistinguishable targets, where each of the tracked objects could potentially divide, is an important task in many biological scenarios. Manually tracking nuclei in microscopy data is tedious. Human annotators need to take temporal context into account to distinguish cells in regions of poor image quality, and thus manual curation is very time consuming. To allow for high throughput experiments, automated tracking software is in great demand, even though detecting cells especially in high density regions of poor image quality poses an even larger challenge to automated methods than to humans. The reason for this is that humans can intuitively detect and link cells in neighboring

time frames by considering e.g. plausible cell motions, size changes, and the number of cells in the neighborhood as well as their relative positions. Unfortunately detection and tracking is like a chicken-and-egg problem. For automated methods, it would be extremely helpful to have detected all cells in each frame so that linking between frames becomes a one-to-one matching problem, or one-to-two in the case of divisions. On the other hand, to facilitate detecting and properly segmenting all cells in high density regions, it would be beneficial if the history and fate of the cells – like their positions in the preceding and subsequent frames – could be considered.

The community has developed a range of tracking tools with graphical user interfaces to allow for easy application by experts of other fields who are not trained in image processing. However, due to the large variety of different use cases – varying microscopy techniques, as well as different types of cells being recorded – most of the tools are limited to specific scenarios. Fiji’s TrackMate (Schindelin et al, 2012; Nick Perry, 2012) for instance builds on a linear assignment problem (LAP) (Jaqaman et al, 2008) that first links detected objects in consecutive frames and then constructs global tracks from these local links. It can handle merging and splitting targets, but cannot deal with dividing cells. CellProfiler (Carpenter et al, 2006) also uses this LAP tracking algorithm or performs greedy nearest-neighbor tracking. Icy (De Chaumont et al, 2012) has several plugins for tracking, the most recent being a *Spot tracking* method building on a multi-hypothesis tracking approach (Chenouard et al, 2013) that can deal with a variety of motion models of non-dividing objects. iTrack4U (Cordelières et al, 2013) is based on a mean-shift algorithm that operates on a fan of triangular regions created around the center of each cell in 2D images, but it cannot handle divisions or merging and splitting of cells.

(Maška et al, 2014) review a broad range of further cell tracking approaches, but these do not necessarily provide graphical user interfaces, which often prevents their adoption by biologists.

This tutorial presents the tracking workflow in *ilastik* (Sommer et al, 2011), which tries to make user input as intuitive as possible, while yielding high quality tracking results for a broad variety of use cases by considering a global model when resolving ambiguities (Schiegg et al, 2013) of cells migrating, merging, splitting and dividing in 2D+t and 3D+t. *ilastik* is an *interactive learning and segmentation toolkit* that uses machine learning techniques to classify pixels and objects by learning from annotations – sparsely placed by the user – to predict the class (e.g. foreground or background) of each unannotated pixel or object. This toolkit contains several workflows, and each workflow is designed to tackle a specific task. We will demonstrate how to obtain a segmentation using *ilastik*’s *Pixel Classification* workflow, and how to track the segmented cells in the *Automated Tracking* workflow. Throughout this tutorial, a crop of a *Drosophila melanogaster* embryo scan ¹ (see figure 1.1), recorded by the

¹The cropped dataset can be obtained from the download page <http://ilastik.org/download.html>, and a larger crop with more information is available at http://hci.iwr.uni-heidelberg.de/Benchmarks/document/Drosophila_cell_tracking/.

Hufnagel group² using selective plane imaging (Krzic et al, 2012), will serve as example.

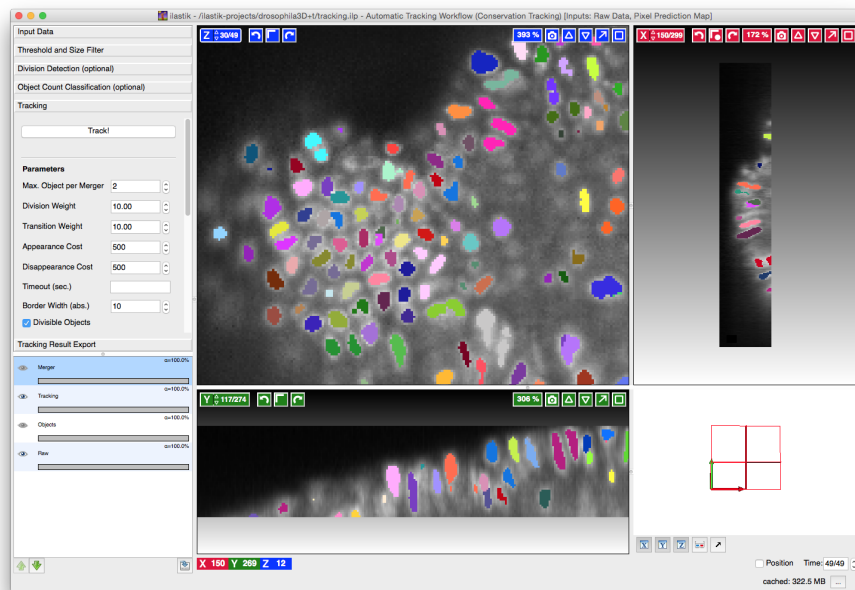


Figure 1.1: The Drosophila dataset opened in *ilastik*, with color coded segmented and tracked cells. Same color indicates common ancestry since the beginning of the video, but not many cells divided during the short time span. The cropped dataset consists of 50 time steps during the gastrulation stage, with the last frame being displayed

TIP: Dedicated *background* sections will give more details on the used algorithms, and *tips* present experienced users' advice on how to best achieve certain goals in *ilastik*, e.g. through useful keyboard shortcuts.

FAQ: This tutorial refers to *ilastik* version 1.1.7.

FAQ: The recommended system specifications for using *ilastik* are a recent multi-core machine with at least 8GB of RAM. When working with larger datasets, especially for 3D volumes, more RAM is beneficial as it allows *ilastik* to keep more images and results cached.

²European Molecular Biology Laboratory, Heidelberg, Germany

1.1.1 Tracking Basics

Many automated tracking tools use a family of algorithms called *tracking-by-assignment*. These algorithms split the work into two steps: segmentation and tracking (Andriluka et al, 2008; Benfold and Reid, 2011; Zhang et al, 2008; Pirsiavash et al, 2011). The segmentation step finds the outline of the targeted objects at each time step (or frame) in the raw data, where each connected component is called a detection, and the tracking step then links detections over time into tracks. Linking cell detections can be done pairwise between consecutive frames (e.g. (Kuhn, 1955)), but can also be approached globally (e.g. (Jaqaman et al, 2008; Bise et al, 2011; Schiegg et al, 2013; Padfield et al, 2011; Magnusson and Jaldén, 2012)). The tracking method in *ilastik* (Schiegg et al, 2013) finds the most probable linking of all detections through a global optimization.

One difficulty of the segmentation step is that the data does not always contain nicely separated objects and is subject to noise or other artefacts of the acquisition pipeline. A good segmentation reduces ambiguities in the tracking step and thus makes linking less error prone. Common segmentation errors are under- and oversegmentation, where the segmentation is too coarse or fine, respectively, as figure 1.2 depicts. The tracking algorithm in *ilastik* (section 1.3) can deal with undersegmentation to some extent, but expects that there are no oversegmentations. Thus when creating a segmentation it is important not to split objects of interest into several segments. In the case of an undersegmentation, a detection that contains N objects is called an N -merger in the remainder of this tutorial.

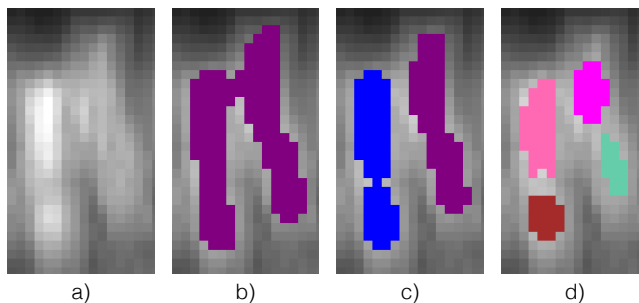


Figure 1.2: From left to right: **a)** Raw data containing two cells. **b)** An undersegmentation of the cells means there are too few segments, here only one detection. The optimization might still decide that two cells merged into this single detection, which is designated as a *2-merger*. **c)** The correct segmentation results in two detections. **d)** The event in which there are more segments than cells is referred to as oversegmentation. Here both cells are split to a false total of four segments. Best viewed in color

Figure 1.3 summarizes the three stages of the pipeline which will be covered in this tutorial. To obtain a segmentation with *ilastik*, the user trains a classifier

by marking a small number of foreground and background pixels. The classifier learns to distinguish the foreground and background class by considering features which are automatically computed on a per-pixel basis, and then predicts the class of all unannotated pixels. A segmentation can then be obtained by thresholding the predicted probability for the foreground class.

The tracking stage takes raw data and segmentation as input. It builds up a graph of all detections and their possible links between consecutive time frames. Each node in this graph can take one of N possible states, representing the number of cells it contains. All these states are assigned an energy, describing how costly it is to take this very state. A candidate hypothesis has high cost if this particular interpretation of the data is unlikely to be correct. The optimization will then find the overall configuration that has the lowest energy, where the locally most plausible interpretations may occasionally be overridden for the sake of global consistency.

The energies for the respective nodes are derived from probabilities predicted by two classifiers. In contrast to the pixel classification step, these classifiers base their decisions on detection-level features. Possible classes are different numbers of contained cells for the *Object Count Classifier*, or dividing and not dividing for the *Division Classifier*. Finally, the result export stage offers two ways of retrieving the tracking result from ilastik for further analysis, namely a relabeled segmentation where each segment is assigned its track ID, and a spread sheet containing linking information as well as the computed features of each segment.

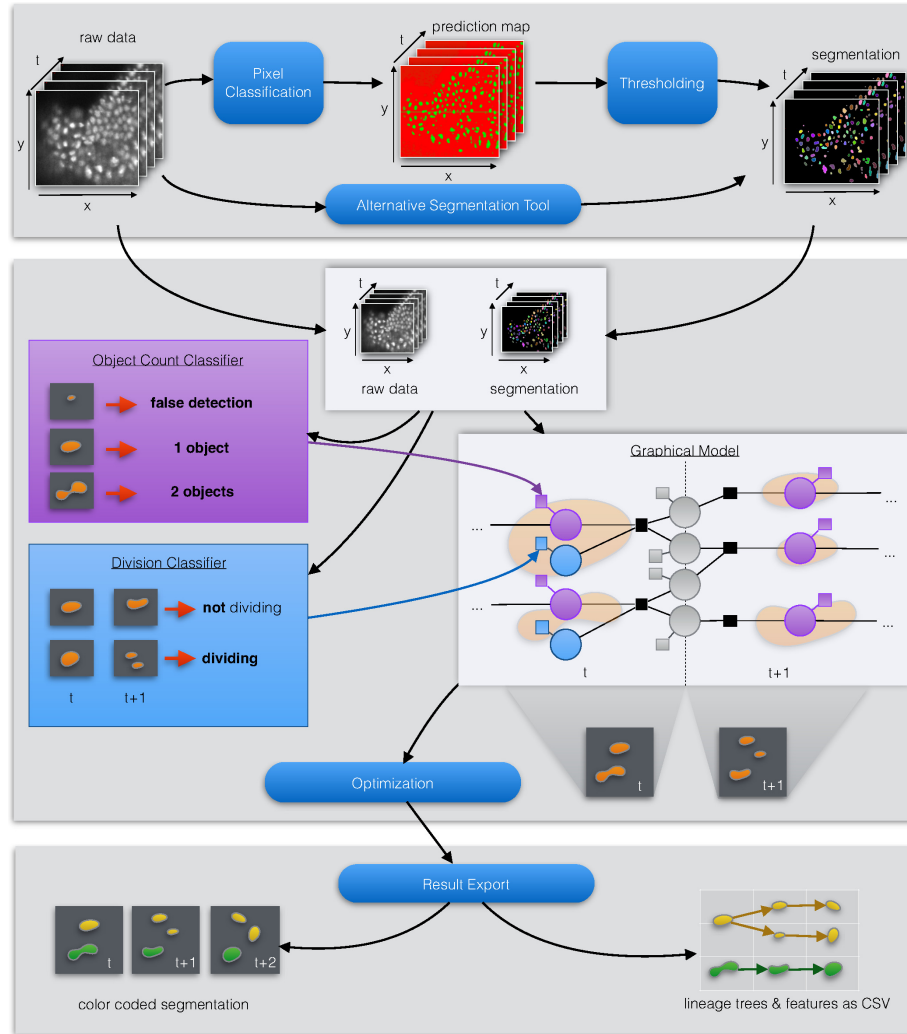


Figure 1.3: Pipeline overview in three stages: segmentation, tracking, and result export. A segmentation of the raw data can be obtained using pixel classification and thresholding, as detailed in section 1.2, or by any other segmentation tool. The tracking algorithm in *ilastik* (section 1.3) builds a global graphical model of all detections in all time frames. Evidence from the input data inserted into the model through an *Object Count Classifier* and a *Division Classifier*. See the *Conservation Tracking Background* box for more information on how the model is constructed and how the most probable configuration is found. Finally, results can be exported as images or CSV tables, as shown in section 1.4

Conservation Tracking Background: One of the important features that distinguish a cell tracker from other multi-object trackers is the ability to detect cell divisions. Candidates for a division are all detections that have at least two possible descendants in the next frame. Figures 1.3(middle) and 1.10 show graphical models where each detection is represented by a (purple) random variable. The nodes are ordered in columns by time step. Possible transitions of an object between two frames are shown as gray nodes linked to the two respective detections through black squared conservation factors. Each detection with at least two outgoing transitions could possibly divide, and is accompanied by a blue division node. If the number of cells entering a detection along its incoming transition nodes exceeds one, the detection represents more than one cell: it is a *merger*. Mergers occur when the segmentation fails to separate cells in clumps. Because we allow mergers, each node in the graph (except for division nodes) can contain more than one cell. The most probable solution of this graphical model will always follow *conservation laws*, meaning that at each detection the number of cells leaving along outgoing transitions is equal to the number of incoming cells. If the detection is no merger, then we allow it to divide, meaning the number of outgoing cells can also be two. These laws are represented by the black squares as factors. The conservation laws are the reason why the algorithm used in *ilastik* is called *Conservation Tracking* (Schiegg et al, 2013). The temporal context is important when the segmentation of a single time frame is ambiguous. By building a graphical model for the complete dataset over all time steps, this context can be incorporated into the optimization procedure to improve the tracking results. The graphical model defines a probability distribution, and the optimization finds the *Maximum-a-posteriori* (MAP) solution, which is the most probable state of the overall system, given all current observations.

The structure of the remainder of this tutorial is as follows: Section 1.2 first presents how to load data (1.2.1), and how to navigate the workflow sidebar (1.2.2) and the data views. Then the segmentation by *pixel classification* and *thresholding* is introduced; this step can be omitted if a segmentation is already available. Section 1.3 covers the *automated tracking* workflow, explains how to train classifiers to detect which cells are dividing and how to give hints as to which detections are mergers, followed by a brief overview of the algorithm that is used for tracking. Lastly the result export capabilities are detailed in section 1.4 and we conclude in section 1.5.

1.2 Pixel Classification and Object Segmentation

ilastik provides the *Pixel Classification* workflow to interactively create a segmentation of multidimensional image data. When starting the application, you will be faced with a workflow selection dialog as in figure 1.4. Select `Pixel`

Classification for now. You will be prompted for a file to save the project to immediately. Save the project as e.g. `pixel-classification.ilp`. As the name implies, this workflow allows you to classify each pixel, e.g. into foreground and background, but also into more classes. To do this, you first have to load the data, create the desired classes, and give some examples for each class by painting brushstrokes in the data. A color coding will then be displayed over the raw data to indicate which pixel probably belongs to what class. You can refine this prediction by drawing additional annotation strokes.

TIP: We will use `monospaced` fonts to indicate the button/text field/... that should be clicked/edited.

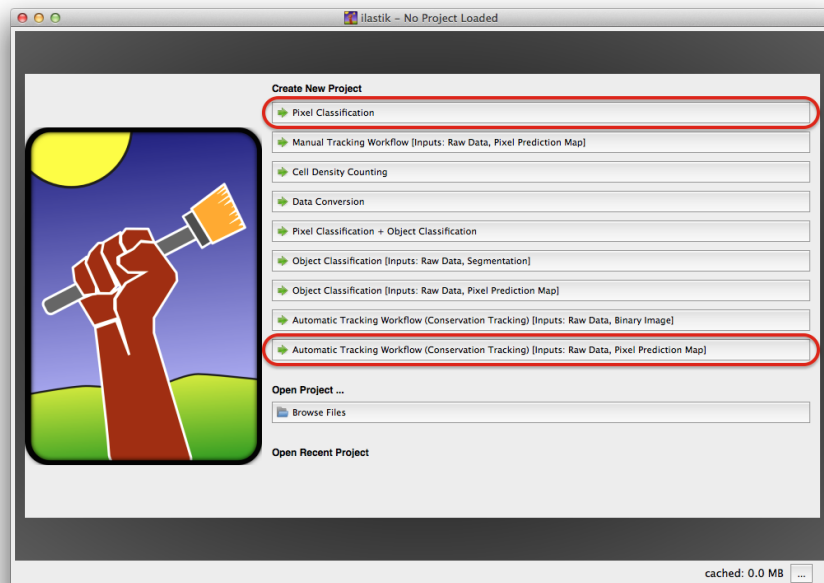


Figure 1.4: Startup screen of *ilastik*, showing all available workflows. Highlighted in red are the workflows covered by this tutorial. In section 1.2 we will use the *Pixel Classification* workflow, and in section 1.3 *Automated Tracking*

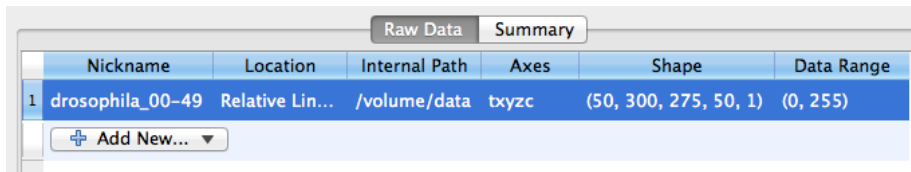
1.2.1 Loading Data

The first step when creating a new project is loading data. The *Input Data* applet should already be pre-selected in the sidebar on the left hand side. To its right is a table containing *Raw Data* and *Summary* tabs as shown in figure 1.5. Select *Raw Data* and click the *Add New* button. *ilastik* can handle most standard image formats like *tiff*, *bmp*, *gif*, *png*, and *jpeg*, as well as *HDF5* files, but no

BioFormats (Linkert et al, 2010) yet. If the data of interest consists of only one frame or is stored as a HDF5 volume in a single file, “Add separate image(s)...” should be selected, but in the case of a stack with one file per time step or z-slice the option “Add a single 3D/4D Volume from Sequence...” should be used. The Drosophila example dataset from figure 1.1 is provided as one HDF5 file, so it is opened via `Add separate image(s)...`. As soon as the data is loaded, the first row in the dataset table will be filled in with information about the dataset’s axes and shape. If the axes and their dimensions in the *Shape* column do not match, **double click** on the row, or **right click** to select **Settings**. In the dialog that pops up, the **Axes** text box allows to arbitrarily swap the axes around by entering a series of characters indicating spatial dimensions **x**, **y**, **z**, as well as the time **t** axes and the color channels **c**. The number of characters has to match the number of dimensions in the *Shape* field. Make sure that **t** is present in this list, as tracking requires a time axis. If the data was loaded correctly, the image content will be shown in the main window of *ilastik*. For time lapse videos, a small spin box titled *Time* will be displayed in the lower right which indicates the current (zero based) frame index and the highest available frame number.

TIP: On Windows, the data needs to reside on the same drive as the project file.

TIP: Loading a volume from a HDF5 file is faster than loading a tiff stack because HDF5 allows blockwise access. If *ilastik* runs very slowly, try converting the data into HDF5. The easiest way to do this is to select **Copied to project file** for the **Storage** option in the dataset properties window, which you already used to specify the axes.



	Nickname	Location	Internal Path	Axes	Shape	Data Range
1	drosophila_00-49	Relative Lin...	/volume/data	txyzc	(50, 300, 275, 50, 1)	(0, 255)

Below the table is a button with a plus sign and the text "Add New..." followed by a dropdown arrow.

Figure 1.5: A dataset has been loaded. The axes description specifies how the shape is interpreted. Here the dataset has 50 frames of 3D volumes with a shape of $300 \times 275 \times 50$ pixels, and only one color channel

1.2.2 Navigating ilastik

Depending on whether a 2D+t or 3D+t dataset was loaded, *ilastik* will display one timeframe as 2D image, or a split view with *xy*, *yz*, and *xz* slicing planes and a 3D window indicating the relative position of those slicing planes as in figure 1.1. In a 3D+t dataset *ilastik* also shows a position marker that indicates

which slice is shown in the other planes (note that they have differently colored GUI elements, this color corresponds to the color of the position marker). The position marker can be disabled by a checkbox in the lower right. Double clicking inside one of the slicing views centers the other views on that position in the data. The best user experience is achieved when using a mouse with scroll wheel. To zoom in and out of any of the views use **ctrl + mouse wheel scrolling** (or **cmd + mouse wheel scrolling** on a Mac). Panning can be achieved by holding the **mouse wheel pressed + move**, or dragging the mouse while pressing the **shift**-key. To navigate through time, press **shift + mouse wheel scroll** or use the time spin box in the lower right. Note how the position gets updated in the upper left of each view, and the timeframe in the lower right.

Each *ilastik workflow* consists of a series of steps, and each step places its important controls inside an *applet*. A list of *applets* occupies the upper half of the left sidebar in the *ilastik* window. The aforementioned *Input Data* applet is typically one of the first applets in each workflow, where consecutive steps are ordered from top to bottom. Clicking on the title of the next applet will let it unfold and show its contents, and also update the image views and update the list of visible *layers* in the lower half of the sidebar on the left.

Layers contain data for each pixel to be displayed, e.g. raw data, a segmentation mask, etc., which might be familiar from advanced photo editing software. The visibility of layers can be toggled by clicking the **eye**, and its opacity can be changed by dragging the intensity slider horizontally. **Right clicking** a layer in this list shows a context menu that allows to export this layer, or, for the raw data, adjust its normalization.

1.2.3 Feature Selection

The next *applet* in the *Pixel Classification* workflow is **Feature Selection**. As mentioned before, *ilastik* will learn from annotations how to distinguish the different pixel classes (e.g. foreground and background) in the data. This decision is based on a set of *features*, where each of these features is a filter applied to a smoothed version of the raw data. The level of smoothing is also called the scale σ of the feature, and the larger this scale, the more information of the neighborhood around a pixel is taken into account. Click on **Select Features** to bring up the dialog shown in figure 1.6, and select a subset of features. Once a set is selected, each chosen feature can be visually inspected in the *ilastik* data viewer by clicking on the respective feature in the lower half of the left side bar.

TIP: Click on a feature in the list on the lower left to display it for the currently visible region of the data. Good features emphasize the structures of interest.

TIP: The time needed to compute a feature increases with its scale and complexity. In the selection matrix, this means the cheapest feature is in the upper left, while scale and complexity grow towards the lower right. Expensive features are not necessarily more expressive.

TIP: In general one could always select all features and let the algorithm find out which ones it needs to distinguish between the classes, but more features also means more computational effort and thus takes longer. Try to remove the ones that look unnecessary but make sure that the prediction remains good. This could also be achieved by iteratively adding or removing a single feature from the list and assessing the quality of the predictions in the *Live update* mode which will be explained in section 1.2.4.

Background: A classifier learns how to discriminate between different classes from a set of examples. For *Pixel Classification* the classes a pixel can belong to are often foreground and background, but could also represent e.g. different tissue types. For the discrimination, the classifier considers the selected features of each pixel. While the classifier is trained, it learns how to combine these features such that it predicts the desired class for the training examples as good as possible. The more informative the set of features is, the better the classifier can predict the correct classes.

The classifier we use is a Random Forest (Breiman, 2001), an ensemble of decision trees. Each bifurcation of a decision tree represents a binary decision based on a feature, for example: is the smoothed intensity value of this pixel greater than some threshold? Each leaf of the tree is assigned the class of all pixels that it contains during training. For a new pixel p , the decision tree assigns p the class of the leaf it falls into. As a Random Forest consists of an ensemble of trees with randomly selected features at the junctions, it allows averaging this decision over all trees to yield a probability estimate per class.

1.2.4 Training

The selected features specify what information *ilastik* is allowed to take into consideration when classifying a pixel. It now needs to know which classes there are and how they look like.

- Go to the **Training applet**, and click on **Add Label** twice to create a foreground and a background class (the terms label and class are used interchangeably here).
- Double click on the classes' names to rename, and double click on the colored square in front of the name to assign a different color. Rename the first (red) label to "*background*", and the second (green) label to "*foreground*".

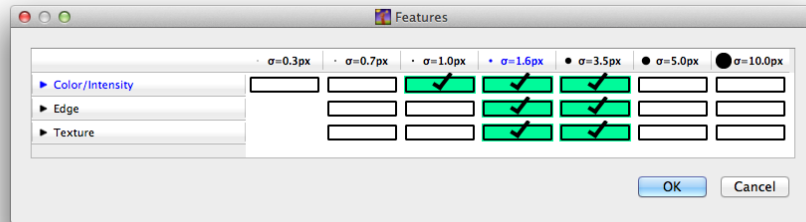


Figure 1.6: Feature Selection Dialog. Color, edge and texture information can be selected on multiple scales σ , indicated by the size of the black disk in the column headers. The size of neighborhood taken into account when computing each feature is also given in pixels, allowing to relate the scale to the image dimensions

- Click on one of the classes in the applet, and make sure that the brush is selected as indicated in figure 1.7.
- Example pixels for the classes are now specified by painting them with the brush in the raw data. This process is called “*labeling*”. For the *Drosophila* dataset, place a few labels, for instance as seen in figure 1.7.
- Use the “*Eraser*” tool to remove wrong labels, and adjust the size of brush and eraser to your needs.

After annotating some pixels, *ilastik*’s **Live Update** mode makes it learn from the placed labels, and predict on the unlabeled visible data. The prediction will be displayed in the same colors as the labels, but with a lower opacity. The plain predictions (full opacity) should look like in figure 1.8.

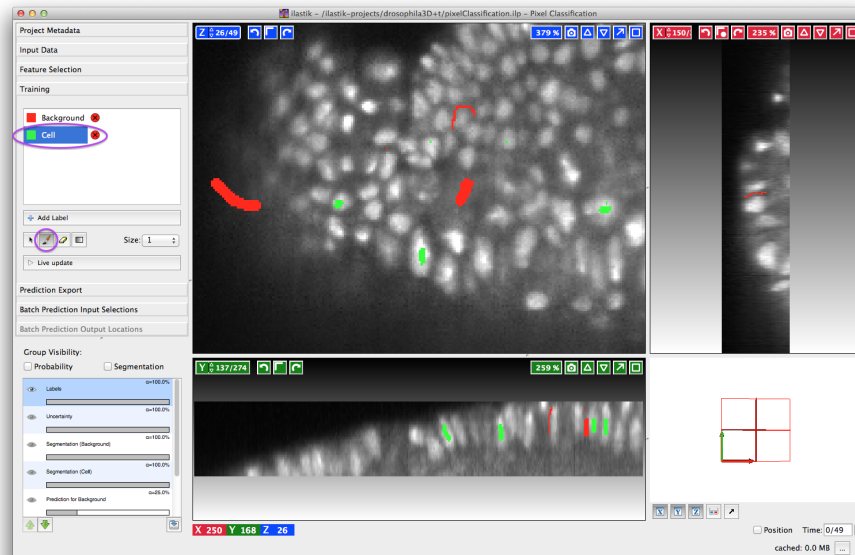


Figure 1.7: Some labels have been drawn in the raw data by selecting a label in the left side bar and using the brush tool

TIP: The predictions are generated using a machine learning algorithm. Such algorithms try to make decisions based on the examples they have been trained on. In machine learning there is always a tradeoff between a perfect fit to the training data and good predictions on unseen data. The latter, also called generalization, usually works better if the algorithm has not been trained too specifically, which is known as overfitting. Transferring this to the labeling step means: make sure not to give too many examples which all look the same. Experienced users would start with a few general labels such as a thin stroke in the background, and by marking a few pixels of one or two cells. When enabling *live update*, the prediction should already look sensible, but probably the boundary between objects is not properly preserved. To fix this, choose a very thin background brush, paint a line between two closely neighboring targets, and inspect the results. Leaving *live update* enabled, the user can now navigate through the volume and add labels where the prediction is wrong.

TIP: Saving the project frequently helps to prevent data loss. By using *Save Copy As* you can also preserve snapshots of the current state.

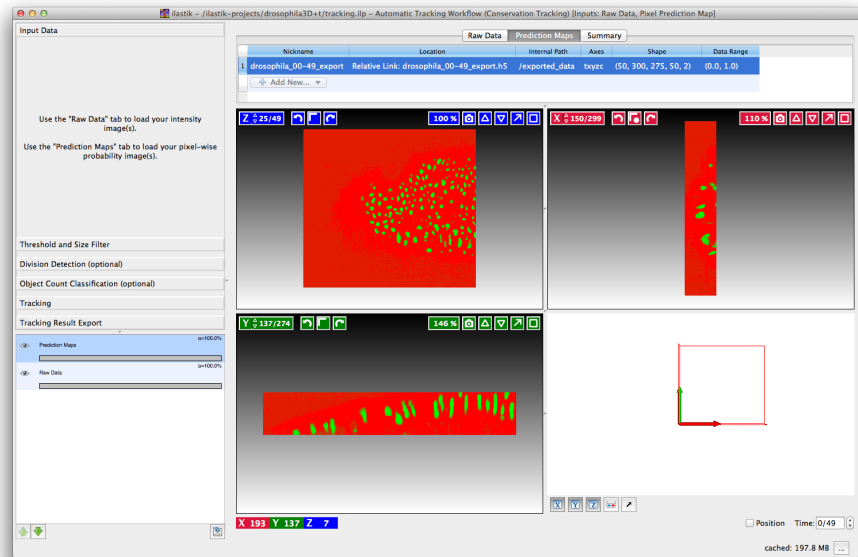


Figure 1.8: Loading the prediction maps, which were exported from the *Pixel Classification* workflow, into the tracking project

TIP: If there are different kinds of objects in the data but only one kind shall be tracked, you can also try to train ‘*target*’, ‘*other object*’ and ‘*background*’ classes instead of just ‘*foreground*’ and ‘*background*’.

TIP: Pressing the *i* key toggles between raw input data and the previous view (e.g. predictions).

By default, *live update* displays the probabilities for a pixel belonging to one of our classes as color mixture. To extract a segmentation from the probabilities one has to make a hard decision. The segmentation coming from choosing the class with highest probability for each pixel individually can be visualized either by clicking on the closed eye of the “*Segmentation for foreground*” and “*Segmentation for background*” layers (depends on the name of the labels), or by using the group visibility checkbox for **Segmentation**. As mentioned before, the tracking algorithm can deal with undersegmentations (when there are more than one cell in one segment), whereas oversegmentation (a cell is falsely split into multiple segments) might lead to erroneous tracking, or only one of the segments is used as cell while the others are deemed false detections. To get a good segmentation for tracking, continue this process until most targets are nicely separated.

TIP: Start by labeling the first frame, but not to perfection; and then go to later time steps where the cell density can be much higher. Also make sure that the segmentation still separates cells.

1.2.5 Export Probabilities

ilastik offers two options to save the segmentation for further processing. On the one hand, the segmentation can be exported as binary mask, by **right clicking** on the respective segmentation layer and choosing **Export**. On the other hand, one can export the probabilities, which facilitates choosing a different decision strategy for the segmentation later on. Export the probabilities by going to the **Prediction Export** applet, and make sure **Probabilities** is selected in the drop down menu. Click on **Choose Export Image Settings** to open up a dialog where the desired output format can be chosen, the exported region can be restricted to a region of interest, the axes can be switched, etc. As the probabilities are floating point values, one for each class for each pixel, the export format of choice is a HDF5 file, which is selected by default, and will be called `our-dataset-name_Probabilities.h5`. Close the dialog box by clicking **OK**, and then press the **Export all** button to actually start the export process. Depending on the dataset size and the speed of the workstation, this might take a couple of minutes, because now all selected features have to be computed for all pixels of the dataset. On a recent laptop this takes around 8 to 10 minutes for the drosophila example dataset. You will need these exported probabilities in the next section.

Background: HDF5 is a container format for numerical data in tables and matrices with arbitrary dimensionality and arbitrary data type.

TIP: To segment multiple datasets using the classifier trained above, these datasets could be added here and exported as well.

FAQ: On Windows, in rare cases an error message pops up complaining that the output file is already used by some other process. Make sure to choose a filename which does not exist yet to prevent this. And make sure that input data, output file, and project file reside on the same windows drive.

1.2.6 Thresholding

Because the first step in the tracking workflow differs depending on whether the segmentation is given as binary mask or probabilities, we will cover this first step now before providing details about the general tracking workflow in the next section. To create a tracking project, save the pixel classification project, close the project, and create a new one. This time, select **Automatic Tracking Workflow (Conservation Tracking)** [Inputs: **Raw Data, Pixel**

Prediction Map] and save the project as `tracking.ilp`. In the **Input Data** applet, load the raw data as in section 1.2.1. This time, a **Prediction Maps** tab appears between the *Raw Data* and the *Summary* tab. Select it, choose **Add**, **Add separate image(s)...**, and pick the exported probabilities that were saved as HDF5 file. The *ilastik* window should then look like in figure 1.8.

Now go to the **Threshold and Size Filter** applet, where you can specify how to retrieve a segmentation from the probabilities for all classes. The prediction map stores the probability for each class in a different color channel. To identify which of these channels contains the foreground class, toggle the visibility of the different channel layers in the list of layers in the lower left. The second class in the pixel classification walkthrough was foreground, which was green. It should be stored in channel 1 of the prediction map. Enter the correct channel value into the **Input Channel** field in the thresholding applet. Each channel of the prediction maps stores a probability for each single pixel/voxel to be of the class corresponding to the channel, and all probabilities corresponding to one pixel have to sum to one. These probabilities can be smoothed by a Gaussian filter, specified by the **Sigma** values (allowing for anisotropic filtering).

To extract a segmentation, you can threshold the smoothed probabilities at the value θ which is entered in the **One Threshold** option. Every pixel that has a probability for being of the selected class higher than the threshold θ will be used as foreground. The default values for smoothing and thresholding should work well in most of the cases. Click on **Apply** to see the resulting segmented objects, which should look like figure 1.9.

Noise or clutter in the data often produces small detections in the segmentation, which can already be discarded simply by specifying a minimal number of pixels that a detection needs to span to be considered in the following steps. Adjust the **Size Range** minimum and maximum, followed by clicking **Apply**, until all detections that can be discarded judging by size have disappeared.

TIP: Navigate to densely populated regions in the data and ensure that the chosen threshold yields a sensible segmentation there. Increase the threshold if many objects are merged, or decrease the threshold if there are oversegmentations.

TIP: Hover the mouse over the different options to bring up a tool tip giving more details.

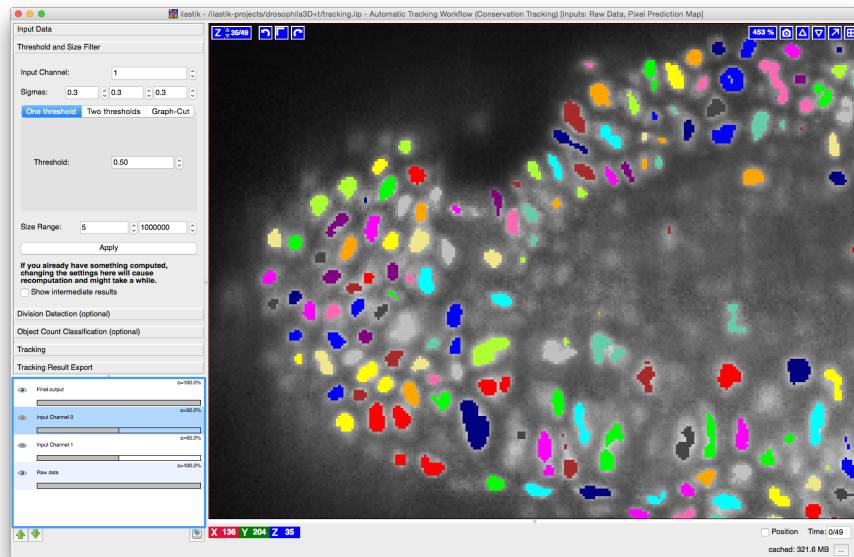


Figure 1.9: After applying the selected threshold and size filter, the resulting connected components of foreground pixels are assigned random colors

1.3 Tracking

Installation: When you download *ilastik* from <http://ilastik.org>, it will not show the Automatic Tracking Workflow immediately. This is because tracking relies on the external library IBM ILOG CPLEX Optimization Studio^a, for which you need your own licence. For academic purposes, this licence can be obtained free of charge on IBM's website^b. This library needs to be installed, and on linux and Mac OS it needs to be copied into the *ilastik* program folder. Instructions for doing so can be found in the documentation^c.

^a<http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud>

^b<https://www-304.ibm.com/ibm/university/academic/pub/page/membership>

^c<http://ilastik.org/documentation/basics/installation.html>

The *Automated Tracking* workflow in *ilastik* allows you to automatically track dividing cells or other objects. To access the tracking workflow, a tracking project must be opened in *ilastik*. This is already the case if you followed the *Pixel Classification* and *thresholding* walkthrough in section 1.2. If you want to use an externally created segmentation of the data, start up *ilastik*, and select *Automatic Tracking Workflow (Conservation Tracking) [Inputs: Raw Data, Binary Image]* from the startup screen. Save the project as `tracking.ilp`. In the *Input Data* applet, load the raw data as described in

section 1.2.1. Between the *Raw Data* and the *Summary* tab of the dataset table is a **Segmentation Image** tab, where the binary images should be loaded the same way as the raw data.

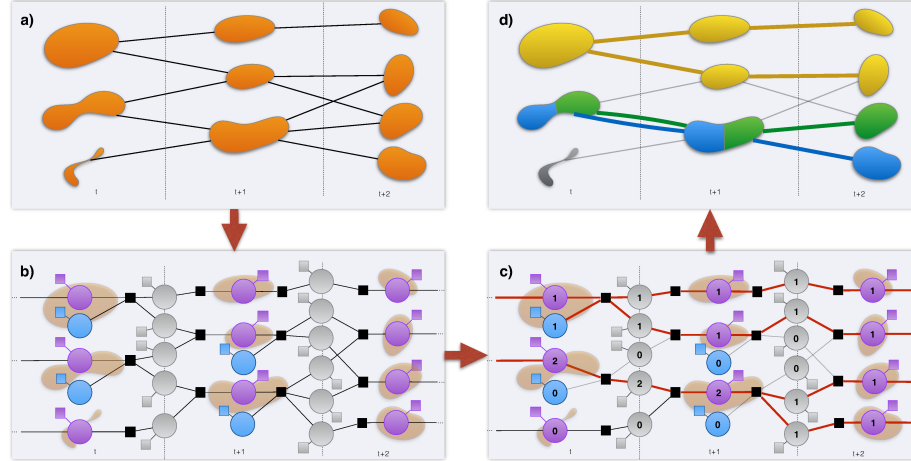


Figure 1.10: From segmentation to lineage using Conservation Tracking. Counterclockwise: **a)** First, all detections in the segmentation of frame t get linked to possible successors in $t + 1$ which lie within a user-defined radius. This builds a graph of tracking hypotheses. Note the lower left detection which does not resemble a cell like the others, as well as the big detection in the lower middle. **b)** A factor graph is constructed by inserting a (round gray) transition node for each linking hypothesis in the hypotheses graph. Division random variables are added (blue circles) whenever there are at least two possible outgoing transitions from a detection (purple). Squares are factors that introduce an energy depending on the state of all connected random variables. The black squares model conservation laws, while the colored factors hold the energies (or costs) for different states of the detections, transitions, and divisions. **c)** The optimization finds the minimum energy configuration of the presented model, which corresponds to the globally most likely solution. Numbers indicate the number of cells in each random variable, while the red edges depict the links which are used. **d)** Extracted lineage trees after tracking and resolving merged detections. Each lineage tree is assigned a unique color. Note that the detection containing the blue and green cell is split into two segments in frames t and $t + 1$. The oddly shaped detection in the lower left is not included in any lineage because the optimization labelled it as false detection. Best viewed in color

This workflow consists of several *applets*³, where each applet has the task of preparing some input for the tracking algorithm. The *Object Count Classifier* and *Division Classifier* help the algorithm in finding single cells, mergers, and

³for details on applets see section 1.2.2

divisions. Lastly the *Tracking* applet holds a couple of other parameters and is the place to invoke the optimization step.

As shown in figure 1.10, which extends on the small illustration in figure 1.3, the tracking algorithm internally builds a graph with several (round) nodes for each detection in every timestep t . For each detection in the spatial proximity in the next timeframe $t + 1$ it adds a transition node with the appropriate connections. The optimization finds the most probable configuration for the whole graph, taking into account the complete temporal context. It determines how many objects were present in each detection, and via which transitions these objects moved to the succeeding frame's detections. Notice also how the detection in the lower left of figure 1.10 a), which does not look like a cell at all, is determined to contain no cells in 1.10 c), and thus it is not part of any lineage in d). Because the tracking in *ilastik* allows objects to merge into one detection, it is necessary to guide the optimization process by providing information on merged objects and divisions. To this end, an “*Object Count Classifier*” and a “*Division Classifier*” is used. Training these classifiers is achieved by specifying a set of examples for each class. The classifiers can then predict the most probable class for each detection, and present visual feedback that facilitates interactive refinement of the classifier's predictions. Finally, the predicted probabilities for each state of every detection are inserted into the graphical model (as squared factors) and steer the tracking optimization to a sensible result. We will now first explain the two classifiers in detail, and then show how to run the automated tracking. It does not matter in which order you train the *Object Count* and *Division Classifier*. Because the division classifier only applies if your dataset contains dividing cells, we will first look at the object count classification step.

1.3.1 Object Count Classification

As already depicted in figure 1.2, the tracking algorithm allows for N -mergers, where N is the number of cells that are merged into the largest detection in the dataset. Go to the **Object Count Classification** applet. This applet tries to classify the segmented objects (connected components) to predict the number of cells within each segment. This classifier is again a Random Forest as for *Pixel Classification*, but it bases its decisions on features of a detection, such as size (pixel/voxel count), mean and variance of its intensity, and shape information. The selection of these features can be adjusted by opening the **Select Features** dialog, but the preselected set of features captures information that helps to describe the differences between 1 or 2-mergers for a large variety of cell types. The default set of features (names in the list in *italics*) are the pixel *Count* within an object, the lengths of major axes (*RegionRadii*), as well as the *Mean* and *Variance* of the intensity inside the segment.

TIP: If you can filter your objects based on very special properties like their position (*RegionCenter*), the *Minimum*, *Maximum* or *Sum* of the intensity inside the segment, you can add these from the list of available features in the **Select Features** dialog. As long as that is not the case, use the ones that are preselected.

- To create classes for all mergers, add new labels using the **Add Label** button, until the list of labels contains the type “*N Objects*”, where *N* is an estimate on the number of cells combined in the largest merger in the data. Using the provided segmentation of the Drosophila dataset, the largest merger contains two cells, thus the **Add Label** button is pressed three times and $N = 2$.
- For each of these labels, click the label (make sure the **Brush** tool is selected), and then click on detections to mark them as examples for selected class.
 - If there are small objects left after size filtering, or there are objects which are not cells, these are **False detections**.
 - It is also possible not to give examples for one of the labels, e.g. do not mark any object as *false detection* if there is no clutter in the segmentation.
 - Find and label around 10 detections for each class, until the predictions shown in the **Live update** mode are consistent.
- In the **Live update** mode, the classifier will predict for each detection which label it belongs to; indicate this by coloring the detections in the color that corresponds to the label.
- It is a good idea to check and refine the predictions, so jump to other positions in the data and other time slices (as described in section 1.2.2), and correct the label on detections where the classifier is mistaken.

TIP: Giving more labels in general yields better predictions, so we suggest to specify at least 10 examples per class. Still, as in *Pixel Classification*, too many examples might not generalize well to other data. Do not label thousands of objects.

TIP: Try to balance the number of labels for the different classes. The classifier tries to reduce the number of incorrect predictions with respect to your labels. Consequently, if you label 100 detections as *1 Object*, but only 2 as *2 Objects*, it can achieve only 2% error by always predicting *1 Object*.

TIP: It is desirable that the segmentation contains many detections of type `1 Object`, and only a few mergers. Thus it will be harder to find examples for the mergers. Try to find a few nevertheless, and use `Live update` to see whether the predictions look sensible.

TIP: Using the provided prediction map of the example dataset, and with a threshold of 0.5, a 2-merger can be found at $115 \times 105 \times 36$ at $t = 30$, as seen in figure 1.3.1.

TIP: You can also `right click` an object and assign any label to it from the context menu.

TIP: Toggle the visibility of the `Objects` layer in the lower left list and use the randomly assigned colors per detection to ensure that multiple cells are merged into one detection (thus have the same color). Alternatively `right clicking` on objects shows the per-frame unique ID for the selected detection.

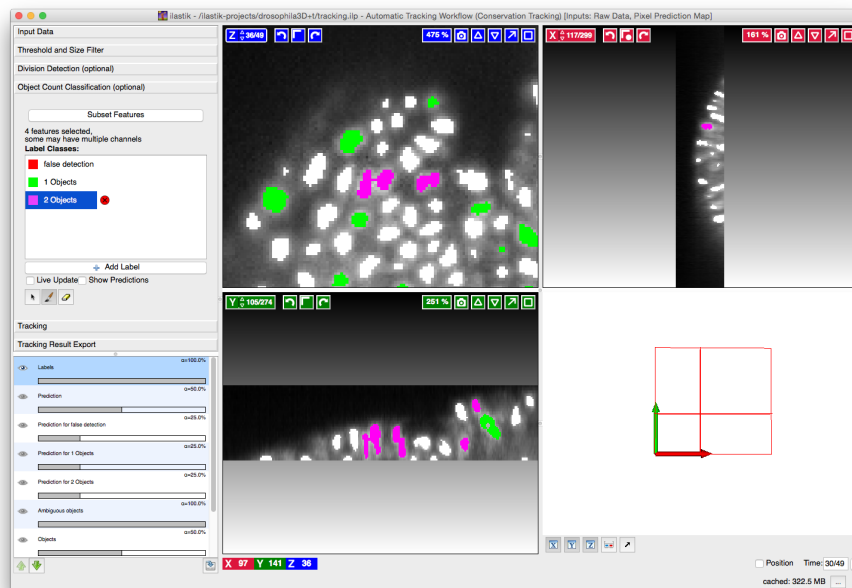


Figure 1.11: In the *Object count classification* applet, label some detections as *false detection*, *1 object* or mergers

TIP: Disable `live prediction` to make scrolling through data faster.

1.3.2 Division Classification

If the data contains dividing cells, a division classifier needs to be trained as well. Go to the **Division Classification** applet, which looks very similar, and behaves the same way as the *object count classification* applet. Only the predefined labels “*Dividing*” and “*Not Dividing*” are available for division classification. As mentioned before, a division can only occur where a detection has at least two possible successors in the next time frame. Due to this requirement, the classifier can also take additional features into account. These are for instance the size and intensity ratio of the children, as well as the angle that parent and connection lines to children cells span.

To place the labels, navigate to a frame where a parent cell is still one detection, but is split up into two children in the next frame, as shown in figure 1.12. Use the **Dividing** label for the parent cell. As before, label roughly 10 occurrences for dividing and non-dividing objects, enable *live prediction*, and browse through the data to proof read and correct the predictions.

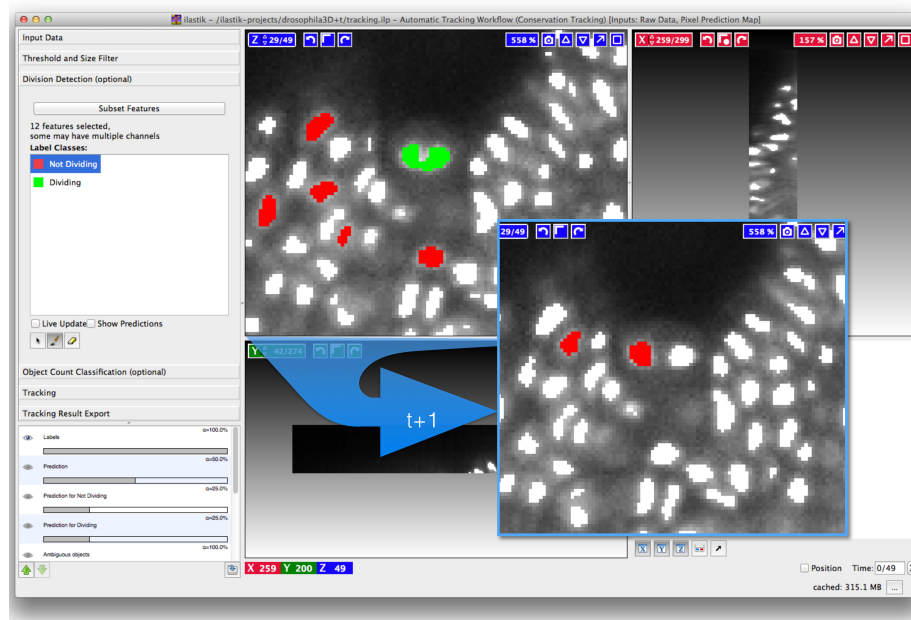


Figure 1.12: To train the division classifier, a parent cell at time t is labeled as dividing, when in the next time frame $t + 1$ there are two child detections present. It can be useful to label the children as “*Not dividing*”, but it is not mandatory to do so

TIP: 2-mergers which de-merge in the next frame often look similar to divisions to the classifier. By labeling these events properly as *non-dividing* while training both classifiers, the optimization has higher chances to disambiguate them later.

1.3.3 Tracking

To finally run the tracking optimization, go to the **Tracking** applet, as shown in figure 1.1. This applet offers to specify a set of parameters to control the tracking algorithm, which is invoked by the **Track!** button. The optimization step finds the most probable configuration of the graphical model consisting of all detections and their possible links as depicted in figure 1.10. We derive energies from the predicted classifier probabilities as the negative logarithm. The most probable configuration thus has the minimal energy of the system. The optimization will consider all the energies of detections being mergers or divisions from the classifiers. It also takes into account the distance that an object moved between frames, and penalizes long range transitions. Additionally, objects can appear or disappear, and the optimization incorporates an energy for those events, making them unlikely unless they happen at the border of the dataset. The parameters in the **Tracking** applet, listed in table 1.1, allow to weigh those different energies against each other. The detection energy (purple square in figure 1.10) is always scaled by 10. Thus leaving division and transition weight at the default value 10 will weigh all classifier output equally. The default settings are well balanced and should work for most datasets.

TIP: Hover the mouse over the parameter names to bring up a tool tip that explains the parameter's meaning.

TIP: The quality of tracking results is most influenced by the parameters *Division Weight*, *Transition Weight*, *Appearance Cost* and *Disappearance Cost*. For instance if you notice that new tracks are started where a cell should have migrated, you could increase the appearance cost and reduce the transition weight to make longer migrations less costly. On the other hand, if cells should appear after the first frame of the sequence, and you have segmented them, but they are not tracked, then you would want to reduce the appearance cost.

Leave all settings as they are, given that *Max Objects per Merger* is set to two (because the highest class was *2 Objects* in *Object Count Classification*). Go to the top of the *Tracking* applet and click on **Track!**. The progress bar at the bottom of *ilastik* will keep bouncing for a while, depending on the size of the dataset and the kind of machine *ilastik* is run on. For the *Drosophila* example dataset this takes around one to two minutes on a recent laptop.

Parameter	Meaning
Max Objects per Merger	Corresponds to the highest number N of objects inside a detection, the biggest N -merger. The complexity of the problem solved by the optimizer grows quadratically with N , which implies much longer runtimes and more RAM usage with higher N . $N > 4$ should be avoided.
Division Weight	Scales the influence of the energy for division (blue square in figure 1.10) of each detection. The higher this parameter, the more costly it is to disagree with the prediction of the <i>division classifier</i> .
Transition Weight	Scales the transition energy (gray square in figure 1.10) with respect to other energies. Transitions are penalized exponentially with the distance the cell had to move between frames, and this weight scales the penalty linearly.
Appearance Cost	Sets the energy of a cell appearance. For instance, this should be very high if no new cells can enter the field of view in a microwell.
Disappearance Cost	Energy of cell disappearance. Cell deaths are captured as disappearance as well, so if your cells are allowed to die, choose a lower value.
Timeout	Restrict the runtime of the optimization algorithm. If this is set too low the algorithm might not find a feasible solution at all. No value or 0 indicate no timeout.
Border Width	If a cell appears or disappears within this pixel-distance to the border, the penalty chosen above will linearly diminish towards the outside, such that an appearance at the border of the image is still plausible. Choosing 0 disables this behaviour.
Divisible Objects	Untick this, if there are no divisible objects in the data and the division classifier is not trained. As long as it is enabled, training the division classifier is required.
Filter	Restricting the field of view to less time steps or a smaller volume may lead to significant speed-ups of the tracking optimization. Coordinates are in pixels.
Size range	Restrict tracking to objects with a pixel/voxel count in the specified range. Especially useful if the segmentation was created externally and contains small debris.

Table 1.1: These parameters can be set in the *Tracking* applet to configure the behaviour of the tracking algorithm. Remember that *energy* and *cost* are used interchangeably

TIP: Restrict the field of view to only a couple of time frames for tracking, to get quick feedback for assessing whether the chosen parameters yield sensible tracking results. by scrolling down in the *Tracking* applet to the **Filter** section and changing the **From** and **To** values of **Time**. Click on **Track!** above to run the tracking, and inspect the selected time range for tracking errors.

As soon as the progress bar disappears, the optimization is done, and objects within the selected field of view should inherit the color from their predecessor in the previous frame when scrolling through time. Pay attention to the children of a division, which get assigned the same color as the parent cell. Mergers will be assigned as many colors as objects are contained by re-segmenting the detection. A “*Mergers*” layer can be made visible to highlight those original detections that contained more than one object.

FAQ: The tracking parameters that get stored when you save your tracking project are always the ones that you used for the last run of tracking. This is to ensure that the saved parameters are consistent with the stored tracking result.

1.4 Exporting Results

There are two parts of tracking results that get exported, which complement each other. The first part is a set of two spread sheets, one that contains a list of links of objects between frames, as well as some features that were computed for each object per frame, and a separate table that contains information about each division. The other part is the segmentation, where each object has a unique ID per frame, or the ID of the lineage tree it belongs to. We will now look at both parts separately, but cross reference where they are meant to be used together. All export options can be found in the **Tracking Result Export** applet (see figure 1.13). The two buttons for configuring the export types will bring up extra dialogs for the settings. To dispatch the actual export, click on “Export All”.

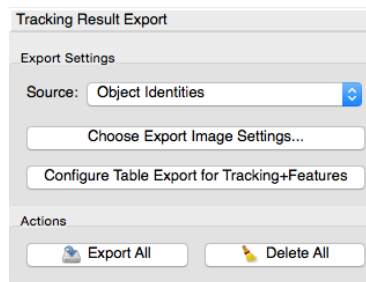


Figure 1.13: The **Tracking Result Export** applet offers to export the relabeled *Object Identities*, *Tracking Result*, and *Mergers* from the **Export Source** drop down menu. The volume to be exported, as well as the export format can be specified in the dialog (figure 1.14) that pops up when selecting **Choose Export Image Settings...**. A table containing all linkings, divisions and features is exported by default, but can be configured by clicking **Configure Table Export for Tracking+Features**

1.4.1 Spread Sheet Export

To configure the spread sheet export, click the **Configure Table Export for Tracking+Features** button at the bottom of the **Tracking Result Export** applet. In the drop down menu for the export format, select **CSV** for *comma separated value*, and specify a path and file name. CSV files can be read by all spreadsheet software as well as most analysis tools. **Features** allows you to choose which additional features to export that have been computed for each detection. Click on **OK** to save the settings. To actually export the tables and the segmentation, you will have to click on “Export All“. But before doing so, however, also read about the relabeled segmentation export in section 1.4.2. The export will create one or two files, one called `{dataset_dir}/{nickname}-exported_data_table.csv` and, in case *divisible objects* were enabled during tracking, a file called `{dataset_dir}/{nickname}-exported_data_divisions.csv`.

TIP: The suffixes `_table` and `_divisions` are added automatically, and the placeholders `{dataset_dir}` and `{nickname}` will be filled in depending on the dataset you loaded. The `{nickname}` is the beginning of your loaded dataset filename. If you leave the values at their defaults, the exported files will be created in the same folder as your dataset.

This `yourSelectedName_table.csv` file is one large table that holds linking information between consecutive frames for the whole dataset. A tracked object is assigned a unique *object identifier* (oid) in each frame, which refers to the segmentation (or label image) gray value. However, it also has a *track identifier* (track_id), where a track is a part of a lineage tree between two events like appearance, disappearance, and division. The information on tracks being linked together by divisions can be found in the divisions CSV file. Finally each detection has a *lineage identifier* (lineage_id) which is the same for all objects and tracks that are descendants of the root cell of the lineage tree. Table 1.2 lists the important columns of the tracking CSV file, and table 1.3 specifies which columns are given per division.

TIP: *ilastik* provides more information on the object features in the *Select Features* dialog of *Object Count Classification*.

1.4.2 Relabeled Segmentation Export

The visual representation of the tracking results which is displayed after running the optimization in the **Tracking** applet can be exported as grayscale segmentation, where black is background, and each object is assigned a value that either corresponds to its lineage, or its per-frame object ID. For this, the **Export Source** drop down menu in the **Tracking Result Export** applet offers three choices: *Object Identities* colors each object with its unique ID in that frame (`labelimage_oid` in the table above), *Tracking Result* exports the segmentation

Column	Content
object_id	a globally unique running identifier
timestep	the frame number
labelimage_oid	the object identifier, unique for each object in each frame
track_id1	unique track identifier (started by appearance or cell division; terminated by disappearance or cell division)
Count	the number of voxels assigned to this object (its size)
Coord<Minimum>_N	the lower left corner of the 3D bounding box around the object ($N = 1, 2, 3$ for dimensions x, y, z)
RegionCenter_N	the center of the segmented object. ($N = 1, 2, 3$ for dimensions x, y, z)
Coord<Maximum>_N	the upper right corner of the 3D bounding box around the object ($N = 1, 2, 3$ for dimensions x, y, z)
additional selected features	e.g. mean intensity, variance of intensity, etc.

Table 1.2: Explanation of the columns found in the tracking export CSV table. All coordinates are given in pixels

Column	Content
timestep	the frame number just before mitosis
parent_oid	the object identifier of the parent cell in the given timestep
track_id	the track identifier of the parent cell in the given timestep
child1_oid	the object identifier of one child cell in the given timestep+1
child1_track1_id	the track identifier of this child cell in the given timestep+1
child2_oid	as above
child2_track1_id	as above

Table 1.3: Columns in the division export table and their meanings.

where each object is assigned a gray value corresponding to its lineage ID, and *Merger Result* will export only the detections where the optimization decided that it contains more than one object. For visual inspection, *Tracking Result* is usually the most helpful. However, for further analysis in conjunction with the CSV tables from above, the *Object Identities* export provides more valuable information.

Select the desired format as **Export Source** in the applet, then click the **Choose settings** button. The dialog in figure 1.14 opens, allowing to select a subregion (note the parentheses: “[*start*” means inclusive, “[*stop*” means exclusive) of the data for exporting. The lower part of the dialog is devoted to the export file format, which should be some kind of *sequence* for tracking results. For the 3D example dataset, this could e.g. be **hdf5** or a **multipage tiff sequence**. Select your choice and close the dialog box with OK. Then click on **Export All** in the applet, or **Export** in the dataset table at the top of the main window to start the actual export process.

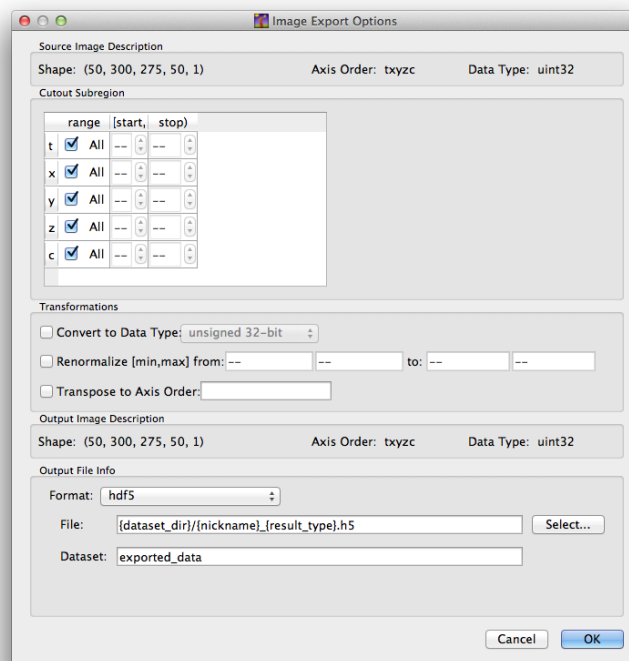


Figure 1.14: The export settings dialog allows to select a subregion and specify the output file format

1.5 Conclusions and Outlook

This tutorial presented how *ilastik* can be used to generate a segmentation through the intuitive *Pixel Classification* workflow, and how to apply the *Automated Tracking* workflow. By sparsely annotating the data, the user can train several classifiers which allow to segment the data and support the tracking optimization step in disambiguating falsely detected objects and divisions. To facilitate further analysis of the resulting lineage, we have presented details on *ilastik*'s CSV export file format. *ilastik* emphasizes intuitive and interactive training to reduce the need of parameter tweaking, while giving state of the art results.

The open source software framework *ilastik* is under active development, and some changes that we are working on are improving the overall performance, learning the remaining tracking parameters from user annotations, as well as a bridge to KNIME (Berthold et al, 2007) to make analysis easier.

Further information about workflows and features, as well as source code and binaries for all major operating systems can be found on <http://ilastik.org/>.

Acknowledgments We thank all *ilastik* developers for providing this open source software: Bernhard Kausler, Thorben Kroeger, Christoph Sommer, Christoph Straehle, Markus Doering, Kemal Eren, Burcin Erocal, Luca Fiaschi, Ben Heuer, Philipp Hanslovsky, Kai Karius, Jens Kleesiek, Markus Nullmeier, Oliver Petra, Buote Xu, and Chong Zhang.

Partial financial support by the HGS MathComp Graduate School, the SFB 1129 for integrative analysis of pathogen replication and spread, the RTG 1653 for probabilistic graphical models, and the CellNetworks Excellence Cluster / EcTop is gratefully acknowledged.

Bibliography

- Andriluka M, Roth S, Schiele B (2008) People-tracking-by-detection and people-detection-by-tracking. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), URL <http://www.mis.tu-darmstadt.de/node/382>
- Benfold B, Reid I (2011) Stable multi-target tracking in real-time surveillance video. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp 3457–3464
- Berthold MR, Cebon N, Dill F, Gabriel TR, Kötter T, Meinel T, Ohl P, Sieb C, Thiel K, Wiswedel B (2007) KNIME: The Konstanz Information Miner. In: Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007), Springer
- Bise R, Yin Z, Kanade T (2011) Reliable Cell Tracking by Global Data Association. In: IEEE International Symposium on Biomedical Imaging (ISBI), pp 1004–1010
- Breiman L (2001) Random forests. *Machine learning* 45(1):5–32
- Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, Guertin DA, Chang JH, Lindquist RA, Moffat J, et al (2006) Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology* 7(10):R100
- Chenouard N, Bloch I, Olivo-Marin JC (2013) Multiple hypothesis tracking for cluttered biological image sequences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35(11):2736–3750
- Cordelières FP, Petit V, Kumasaka M, Debeir O, Letort V, Gallagher SJ, Larue L (2013) Automated cell tracking and analysis in phase-contrast videos (itrack4u): development of java software based on combined mean-shift processes. *PLoS one*
- De Chaumont F, Dallongeville S, Chenouard N, Hervé N, Pop S, Provoost T, Meas-Yedid V, Pankajakshan P, Lecomte T, Le Montagner Y, et al (2012) Icy: an open bioimage informatics platform for extended reproducible research. *Nature methods* 9(7):690–696

- Jaqaman K, Loerke D, Mettlen M, Kuwata H, Grinstein S, Schmid SL, Danuser G (2008) Robust single-particle tracking in live-cell time-lapse sequences. *Nature methods* 5(8):695–702
- Krzic U, Gunther S, Saunders TE, Streichan SJ, Hufnagel L (2012) Multiview light-sheet microscope for rapid in toto imaging. *Nature methods* 9(7):730–733
- Kuhn HW (1955) The hungarian method for the assignment problem. *Naval research logistics quarterly* 2(1-2):83–97
- Linkert M, Rueden CT, Allan C, Burel JM, Moore W, Patterson A, Loranger B, Moore J, Neves C, MacDonald D, et al (2010) Metadata matters: access to image data in the real world. *The Journal of cell biology* 189(5):777–782
- Magnusson KE, Jaldén J (2012) A batch algorithm using iterative application of the viterbi algorithm to track cells and construct cell lineages. In: *International Symposium on Biomedical Imaging (ISBI), IEEE*, pp 382–385
- Maška M, Ulman V, Svoboda D, Matula P, Matula P, Ederra C, Urbiola A, España T, Venkatesan S, Balak DM, et al (2014) A benchmark for comparison of cell tracking algorithms. *Bioinformatics* 30(11):1609–1617
- Nick Perry JS Jean-Yves Tinevez (2012) Fiji trackmate. <http://fiji.sc/TrackMate>
- Padfield D, Rittscher J, Roysam B (2011) Coupled minimum-cost flow cell tracking for high-throughput quantitative analysis. *Medical image analysis* 15(4):650–668
- Pirsiavash H, Ramanan D, Fowlkes CC (2011) Globally-optimal greedy algorithms for tracking a variable number of objects. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE*, pp 1201–1208
- Schiegg M, Hanslovsky P, Kausler BX, Hufnagel L, Hamprecht FA (2013) Conservation tracking. In: *IEEE International Conference on Computer Vision (ICCV), IEEE*, pp 2928–2935
- Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, Pietzsch T, Preibisch S, Rueden C, Saalfeld S, Schmid B, et al (2012) Fiji: an open-source platform for biological-image analysis. *Nature methods* 9(7):676–682
- Sommer C, Straehle C, Kothe U, Hamprecht FA (2011) Ilastik: Interactive learning and segmentation toolkit. In: *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI), IEEE*, pp 230–233
- Zhang L, Li Y, Nevatia R (2008) Global data association for multi-object tracking using network flows. In: *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR), IEEE*, pp 1–8