CHAPTER

20

Mathematical and bioinformatic tools for cell tracking

Peter Hirsch[1], *Leo Epstein*[1] *and Léo Guignard*[2,*]

[1]Max Delbrueck Center for Molecular Medicine in the Helmholtz Association, Berlin, Germany [2]Aix Marseille University, Toulon University, CNRS, LIS 7020, Turing Centre for Living Systems, Marseille, France

Introduction: Cell tracking in biology and medicine: a (very) brief history

Regardless of the system studied or the focus of the study, if one wants to characterize cell position dynamics from a recorded time series of images in a quantitative manner, it is necessary to track the cells and their progeny, the so-called cell lineage. Cell tracking methods report cell positions over time and when and where cells are dividing and potentially dying.

The beginning of cell tracking

Cell trajectories and division patterns have been studied for decades, leading to important biological and medical discoveries. For example, it is by manually tracking every single cell of the worm *Caenorhabditis elegans* during its embryonic development that Sulston et al. [70] discovered that it exhibits a stereotypical lineage, meaning that cells in *C. elegans* embryos always divide around the same time and always follow the same trajectory. Moreover, they were able to show that consistently the same cells were dying, always at the same moments during the development. This observation then led to the discovery of programmed cell death (apoptosis) and to a Nobel Prize that J. Sulston shared with S. Brenner and H. Horvitz. Another example where cell tracking had a crucial role was for the better understanding of the role and the drivers of cell−cell reorganization during tissue shape changes during the 1990s and early 2000s [6,7,19,33,53]. In these works the tracking was still done fully manually. Nonetheless, it led to a significantly better understanding of the mechanisms driving tissue remodeling at the single cell scale.

The rise of faster microscopes

The development of faster and better resolved microscopes together with more efficient cell labeling methods have made live recording of single cells, over a long period of time, easier to acquire. This rapid rise of image acquisition power came with new opportunities for the analysis of cell lineages which in turn led to new biomedical discoveries that were only possible, thanks to novel semiautomatic and automatic tracking methods. For example, the access to longer time series, of better quality, quickly allowed to precisely establish cell number and position over time during the early development of, among others, the fish *Danio rerio* [38,54], the plant *Arabidopsis thaliana* [18], the fly *Drosophila melanogaster* [41], or more recently of limbs of the crustacean *Parhyale hawaiensis* [79].

Cell tracking methods pick up the pace

The accessibility of these modern image acquisition methods increased the importance of the development of better performing cell tracking methods specific for biological and biomedical applications. The ensuing increase of interest in developing cell tracking algorithms is indeed reflected in the number of respective publications (see Fig. 20.1). The improvements of the tracking methods together with the acquisition methods enabled,

* Corresponding author. e-mail address: leo.guignard@univ-amu.fr.

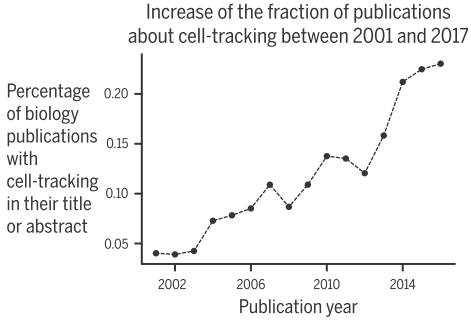


FIGURE 20.1 **Evolution of the appearance of the word cell tracking in biological publications from 2001 to 2017.** *The publication database comes from the CORE dataset [39].*

among others, the refinement of the cell movement quantification during *D. rerio* embryogenesis [64] and allowed the modeling of morphological events such as the cell division patterning during *D. rerio* epiboly [80], the modeling of cell organization during the development of the sea urchin *Paracentrotus lividus* [74], and the modeling of cell–cell communications during the embryonic development of the chordate *Phallusia mammillata* [23]. Moreover, cell tracking methods helped, for instance, to study and build mechanical models of morphogenetic events such as the morphogenesis of the *D. melanogaster* pupal wing [14,15].

Navigating cell tracking methods

As shown in the previous paragraphs, cell tracking allows answering a large variety of biomedical questions. And indeed to be actually able to answer these questions the cell tracks were indispensable. Many different methods are now available to track cells from such time series of images either automatically, semiautomatically, or manually. Each of these methods comes with strengths and weaknesses; each is best suited for different types of questions, organisms, and image acquisition modalities. The variety of existing methods is large and it can be difficult to match the best algorithm and visualization method to the specific characteristics of the question and of the dataset.

This chapter aims at supporting researchers in their choice, less manual work is not necessarily better. Firstly we will cover basic technical aspects about cell tracking that are necessary to make an informed decision about which method to use. Secondly we will present how to choose between the different general categories of methods and what they are best suited for. Finally, we will detail the core principles of each of these categories, putting an emphasis on deep learning methods which have been one of the most developed or most actively improved.

Basic principles of detection, segmentation, and tracking algorithms

To be able to make an informed decision about which method to use, given a dataset and a question to be answered, it is necessary to understand the basic principles of cell tracking.

From a practical perspective the goal of a tracking algorithm is to follow all the cells of interest together with their progeny throughout a time series. This goal can be rephrased, in a technical fashion, as the task of uniquely identifying each cell of interest over a given time series, to associate with each identified cell its position function of time, and to build a hierarchy recapitulating the progeny and ancestry of each cell. To perform all these tasks, cell tracking algorithms can generally be decomposed into two main components:

1. a cell detection or segmentation component[1] that will identify each cell and its position in each frame of the time series
2. a linking component that will reconstruct the progeny of each cell by linking its positions across frames (see Fig. 20.2)

In many tracking methods these two components are applied sequentially and independently. First the detection is applied followed by the linking. Some methods perform these two tasks in parallel allowing to constrain and/or correct each other.

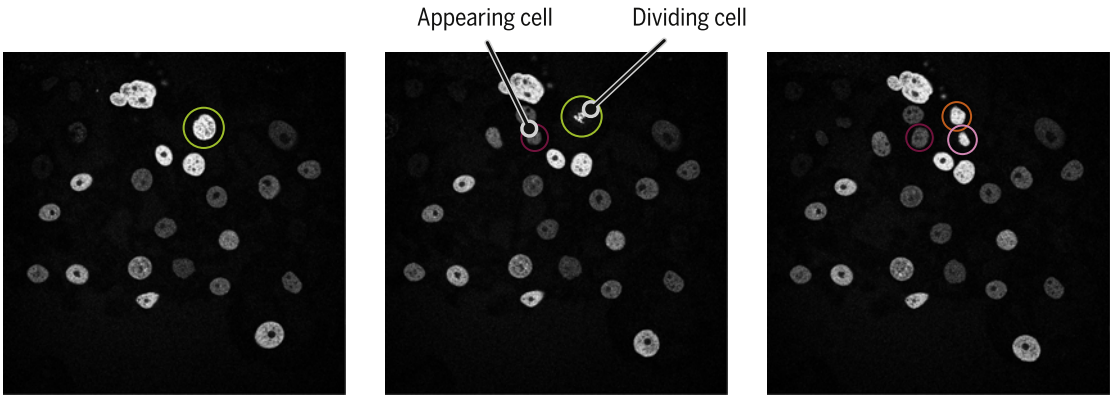
Cell tracking formalization

The result of a cell detection method for a given image is a set of positions $\mathcal{P} = \{p, p \in \mathbb{R}^2\}$ (for 2D images) or $\mathbb{R}^3$ (for 3D images). A cell detection algorithm then aims at building $\mathcal{P}$ such that every element in $\mathcal{P}$ corresponds to a cell of interest.

To properly formalize the result of a cell segmentation method, it is helpful to first define an image. An image I is a function that maps 2D or 3D coordinates onto an intensity value (or a set of intensity values, for example, for RGB images):

$$I : \Omega \subset \mathbb{R}^d \rightarrow \mathcal{I} \subset \mathbb{N}.$$

1 Detection and segmentation are two different tasks. Detection reports the position of an object. Segmentation on the other hand not only allows to have information about the position of the objects detected but also about their shapes. Since this chapter is focused on cell tracking the differences between detection and segmentation will not be discussed in depth. Nevertheless, for each method it will be mentioned whether it is a detection or a segmentation method.



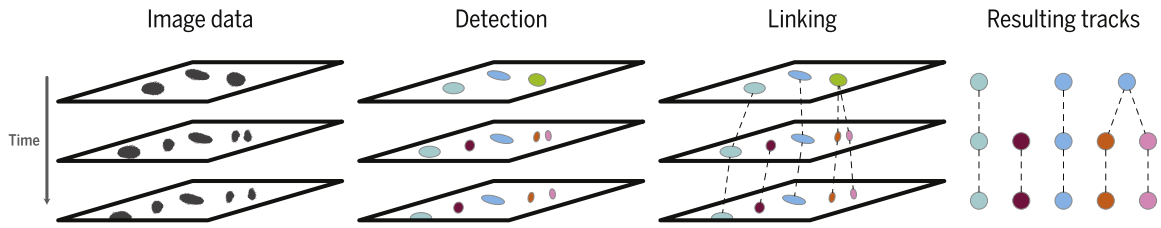


FIGURE 20.2 **Cell tracking.** Top row: Images of HeLa cells where the nuclei have been labeled with fluorescent proteins [47,73]. Examples of cell division and cell apparition are shown. Bottom row: Schematic of the different steps of a cell tracking algorithm.

I associates each point (of dimension d, usually $d = 2$ or 3) in the image definition space Ω with an intensity value $\mathcal{I}$ which is usually a positive integer. The result of a cell segmentation method of a given image I is a segmented image S:

$$S: \Omega \subset \mathbb{R}^d \rightarrow L \subset \mathbb{N}$$

where L is a finite set of labels identifying the region of each segmented object in the image I, in a unique manner. We can then build the set of regions $R = \{r_l\}_{l \in L}$ created by S where:

$$r_l = \{p \in \Omega : S(p) = l\}$$

By definition R is a partition of the image I. Therefore, the following rules are respected:

$$\cup_{r_l \in R} r_l = \Omega$$
$$r_l \cap r_m = \varnothing \quad \forall r_l, r_m \in R, r_l \neq r_m$$

Often, on top of the usual partition rules, a connectivity rule is added (one exception might be if an object can be partly covered by another object, thus potentially visually splitting it into two parts). This rule says that all the regions must be connected components according to a local connectivity rule c. The rule c is equal to 1 if two pixels are connected and equals to 0 otherwise:

$$c\left(p_i, p_j\right) = \begin{cases} 1 & \text{if } p_i \text{ and } p_j \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases}, p_i, p_j \in \Omega$$

The definition of whether two pixels are neighbors is arbitrary and chosen according to the study. For example, it can be that two pixels are neighbors if they share a side (4-connected in 2D) or if they share a side or a corner (8-connected in 2D). Paths between pixels can then be built from the rule c. A path Pa between a pixel p_i and a pixel p_j is a sequence of pixels starting with p_i and ending with p_j such that the $c(p_k, p_l) = 1$ for all consecutive $p_k, p_l \in Pa$[2]. Then, a region r_i is a connected component if there is a path between any couple of pixel in r_i where all the members of the path belong to r_i:

$$\forall \left(p_j, p_k\right) \in r_i, \exists Pa \; s.t. \; \forall p_l \in Pa, \; p_l \in r_i.$$

A segmentation algorithm then aims at building S such that the partition built maps every voxel to the label of the cell it belongs to or to the background.

A cell tracking can be defined as a directed forest $\mathcal{F}$, a disjoint union of directed trees, under the graph theory meaning. A tree $\mathcal{T}$ is defined by a set of vertices and a set of edges that connect the vertices:

$$\mathcal{T} = (V, E \subset V \times V)$$

The vertices ($v_i \in V$) of the trees are the cells and the edges ($e_i \in E$) are the temporal links. V being the set of cells, it can be partitioned in as many subsets as there are time points: $V = \cup_{t \in T} V^t$, where T is the set of time points of the considered time series and V^t the set of detected or segmented cells at time $t \in T$. The trees of

a cell tracking follow the usual rule from the graph theory: there is exactly one path that connects any two vertices in V. Other rules that can be added to constrain the trees are discussed later in Section "Automatic tracking," but one rule is always present: two vertices that exist at the same time point are not connected in $\mathcal{T}$:

$$\forall t \in T, \forall e_i, e_j \in V^t : \left(e_i, e_j\right) \notin E.$$

In $\mathcal{F}$, each tree represents a cell progeny, meaning that if initially only a single cell is tracked and it undergoes multiple rounds of division, the forest will still contain only one tree. The number of trees (i.e., cell lineages) in a forest is at least the number of cells at the beginning of the experiment. Tracking errors or a cell moving into the field of view might cause the start of additional trees.

A tracking algorithm consists of building V, usually using detection or segmentation methods, and of building E using linking methods. As aforementioned, V and E are usually built sequentially but some methods build these two sets in parallel.

A more in-depth explanation of the different methods of detection and tracking are provided in the following sections.

Detection and linking errors

Another important point to keep in mind while assessing which method to use is the kind of errors that can be generated by the different kinds of tracking algorithms, the potential reasons why and when these errors could occur and what can be done to prevent these errors (Fig. 20.3).

Cell detection and linking tasks can fail locally for multiple reasons. Local tracking errors can end up having a dramatic negative effect on the overall analysis. A tracking error can result in the interruption of a track, for example, because the detection method failed to correctly identify a cell. Such an interruption prevents the tracking of the actual path of that cell and will likely prevent any analysis on it, even worse, it could produce erroneous measurements. One error might be sufficient to prevent the analysis of a whole track. Because of this, a cell tracking algorithm that on average mistracks a cell in two consecutive time points 5% of the time would fail to correctly track 75% of the cells after 28 time points (see also Fig. 20.10). Due to the strong dependencies that exist between cell detection and tracking: only the detected cells will be tracked, the errors happening during cell detection will have a strong impact on the tracking. During the detection phase two kinds of errors can be found:

1. the algorithm fails to detect a cell: false negative or underdetection/undersegmentation
2. the algorithm finds a cell where it should not have: false positive or overdetection/oversegmentation

Note that sometimes a given algorithm can split a cell into two or more cells which would also be considered an overdetection. The difficulties that cause detection errors are often related to the quality of the acquisition (e.g., the signal-to-noise ratio is not high enough, the spatial resolution is not high enough). The errors can also stem from the complexity of the system imaged (e.g., the cells are too tightly packed, the cells have a

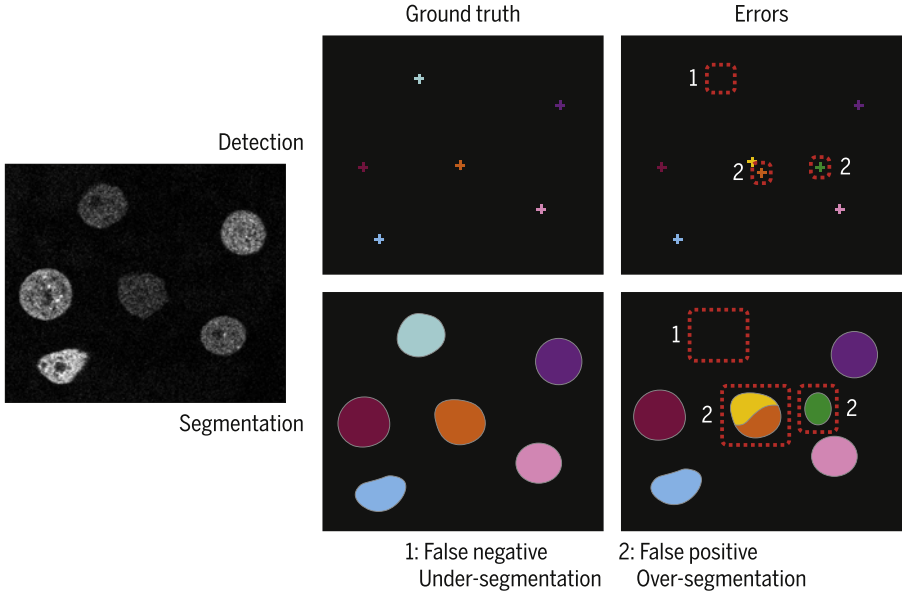


FIGURE 20.3 **Schematics of the different kinds of segmentation errors.** Left: same HeLa cells as in Fig. 20.2. Top row: results of detection algorithms; bottom row: results of segmentation algorithms. Left: expected ground truth; right: detection and segmentation with one false negative (1) and two false positives (2). Note that one false positive for the segmentation is a cell split into two.

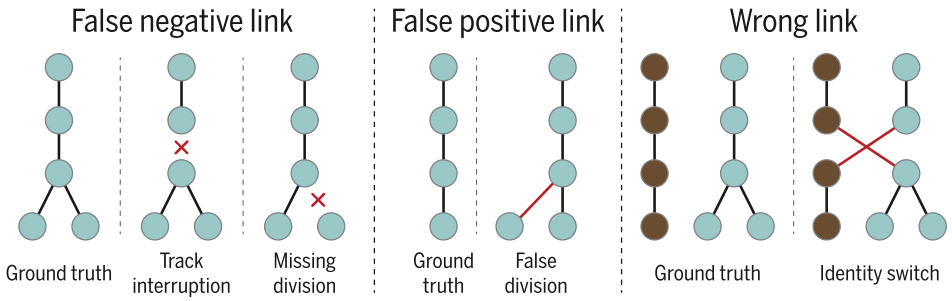


FIGURE 20.4 **Schematics of the different kinds of lineage errors.** Links can be missed (false-negative links) by the tracking algorithm leading to the interruption of a track or the miss of a division. Links can be wrongfully added (false-positive links) which would lead to fake divisions or to missing an apoptosis event, for example. Ultimately, links can be flipped (which is a combination of a false-negative and false-positive edge). This can result in the switch of the identity of a given track.

wide range of shapes or intensity profiles). It is therefore important to keep these sources of errors in mind while planning the imaging protocol. Knowing beforehand the properties of the images and of the system/organism to process helps to determine the correct detection algorithm.

During the linking process multiple other kinds of errors can be found. Fig. 20.4 illustrates the possible kinds of errors:

1. track interruption
2. missing a division
3. adding a division
4. track identity switch

Any linking error is the result of a subset of these four errors (note that missing a division is a kind of track interruption too). There are a large number of reasons why these errors could happen. First, if cell detection is incorrect, it is almost certain that the linking will fail. A missing cell or underdetection can lead to a track interruption (Fig. 20.5C, error 1.1). An underdetection can also lead to false division (Fig. 20.5C, errors 1.2 and 1.3). But even when the cell detection is perfect, the linking task remains difficult and can fail for multiple reasons. When the detection is perfect, linking errors occur if there is an ambiguity in the potential linking. More precisely, if a cell at a given time t has a similar probability to come from several different cells from time $t - \delta t$, δt being the temporal resolution. When this kind of situation arises (Fig. 20.5D, errors 3) a choice has to be made by the algorithm on the basis of its specific implementation, i.e., its internal heuristics or model. More complex heuristics might resolve some tracking errors but might not resolve all of them or might create new ones (Fig. 20.5E). These ambiguity problems are often related to the time resolution used to acquire the time series. If the distance traveled (δx) by a cell c between two consecutive time points is larger than half the distance between two cells there will be an ambiguity in the resulting tracking.

This is the case because the cell c will end up closer to the previous position of one of its neighbors than it is from its own previous position (Fig. 20.6). In other words, the time between two consecutive frames (δt) has to be smaller than half the distance between two cells (D) divided by the cell speed (v)

$$\delta t < \frac{D}{2v},$$

which can be rewritten as follows:

$$\delta t < \frac{D}{2\frac{\delta x}{\delta t}}$$

simplifying to

$$\delta x < \frac{D}{2}. \tag{20.1}$$

This means that the displacement of a cell between two consecutive time frames has to be smaller than half the distance between two neighboring cells. Therefore a direct way to resolve potential ambiguities is to increase the frame rate (resulting in a higher temporal resolution). It is of course not always possible to increase it sufficiently. In these cases it is then critical to carefully choose a well suited set of algorithms to perform cell tracking, for example, a displacement model such as in Bayesian tracking [72] to predict the most likely pairing (Fig. 20.5E). Fig. 20.7 shows an example of tracks obtained from ascidian embryonic development [23].

Quantifying cell detection and tracking errors

In order to have a good idea about the performances of a given method it is important to be able to quantify the errors that are made. For detection algorithms such quantification is done by counting the number of false positives (fp) and the number of false negatives (fn) and comparing these numbers to the number of true positives (tp). The metrics usually computed are: Precision $p = \frac{tp}{tp+fp}$ which informs about the ratio of

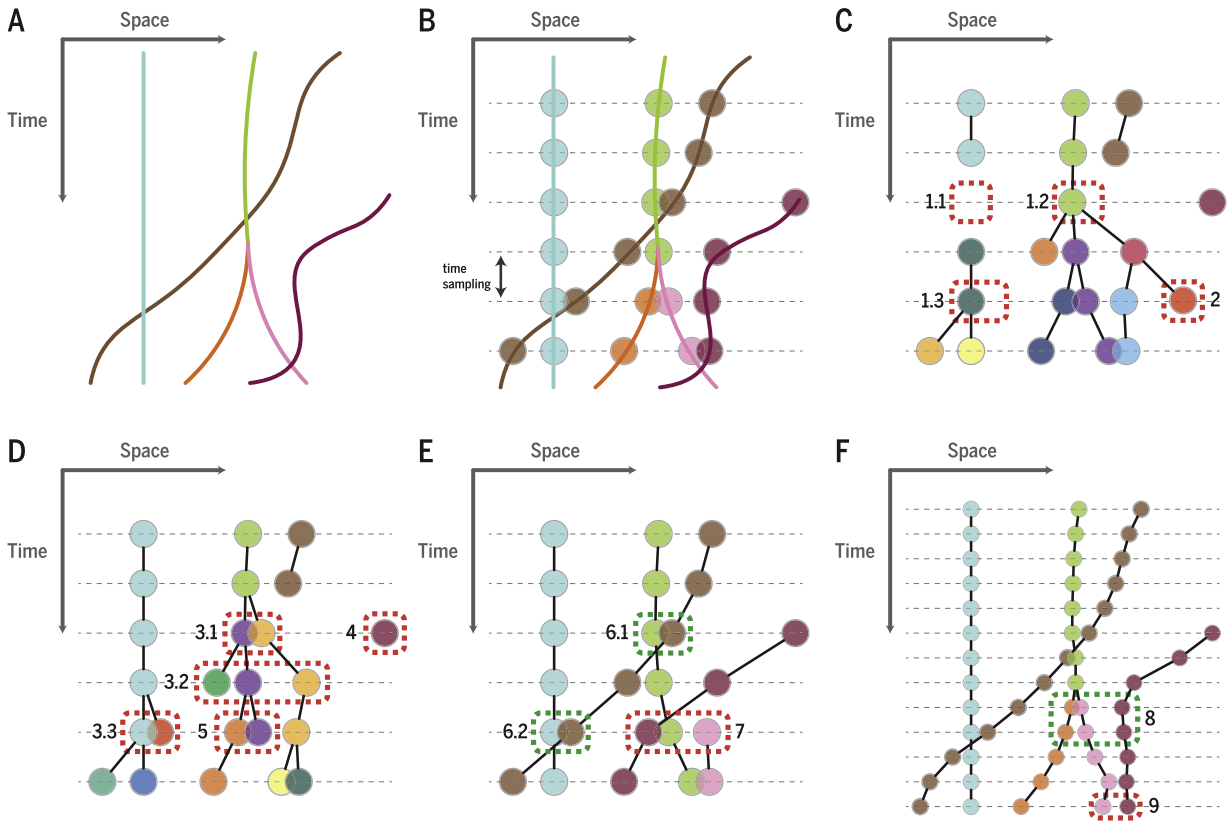


FIGURE 20.5 **Schematic of potential errors in tracking algorithms.** (A) Four cell tracks. The green track divides once, the purple track enters the field of view at a later time. (B) Ground truth for a tracking algorithm for a given sampling of time. (C) Resulting tracking with the given detection, the linking method is a nearest neighbor mapping. The detection errors 1.1, 1.2 and 1.3 are underdetection errors. These detection errors result in an interruption of the light-blue and brown tracks for the errors 1.1 and 1.2. They result in fake divisions in the dark green and green tracks for the errors 1.2 and 1.3. The detection error 2 is an overdetection error, a cell is found while none should have been resulting in a fake division event. (D) Result of a nearest neighbor tracking algorithm given a perfect detection output. Even if the detection is perfect, tracking errors are still present. Because of touching cells, fake divisions are found in 3.1, 3.2, and 3.3. Because of rapid cell displacements, a cell is missing its successors in 4 and a cell is found dying in 5. (E) Using a more sophisticated method, most of the previous tracking errors are corrected. Still, because of abrupt change of displacement, an error is made in 7. (F) By increasing the time resolution, more errors are corrected. But, because of the overlap between cells, a new error of track exchange is made.

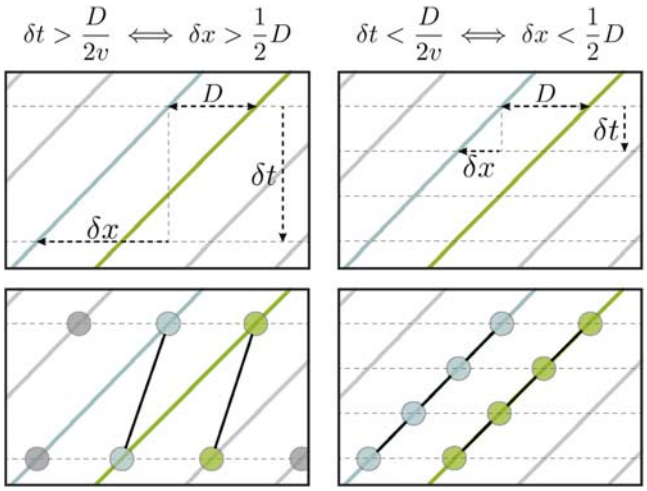


FIGURE 20.6 **Illustration of how having $\delta t < \frac{D}{2v}$ could impact the quality of the cell tracking.**

correctly detected objects to the total number of detected objects and recall $r = \frac{tp}{tp+fn}$ which informs about the proportion of true cells actually found (note that $tp + fn$ is the total number of true cells in the image). These two scores are often aggregated into one, the F-score, $F_1 = \frac{p \cdot r}{p+r}$ which is the harmonic mean of p and r. One difficulty of quantifying detection errors is to decide whether a detection is a true or a false positive. The problem stems from the fact that usually the detection algorithm does not find the cells at exactly the same position as the ground truth (as the ground truth is based on human annotation, both the ground truth and the output of the algorithm exhibit at least some small degree of error or uncertainty). In these cases it is necessary to map cells between the detection and the ground truth. This assignment problem can be resolved with a matching algorithm such as the

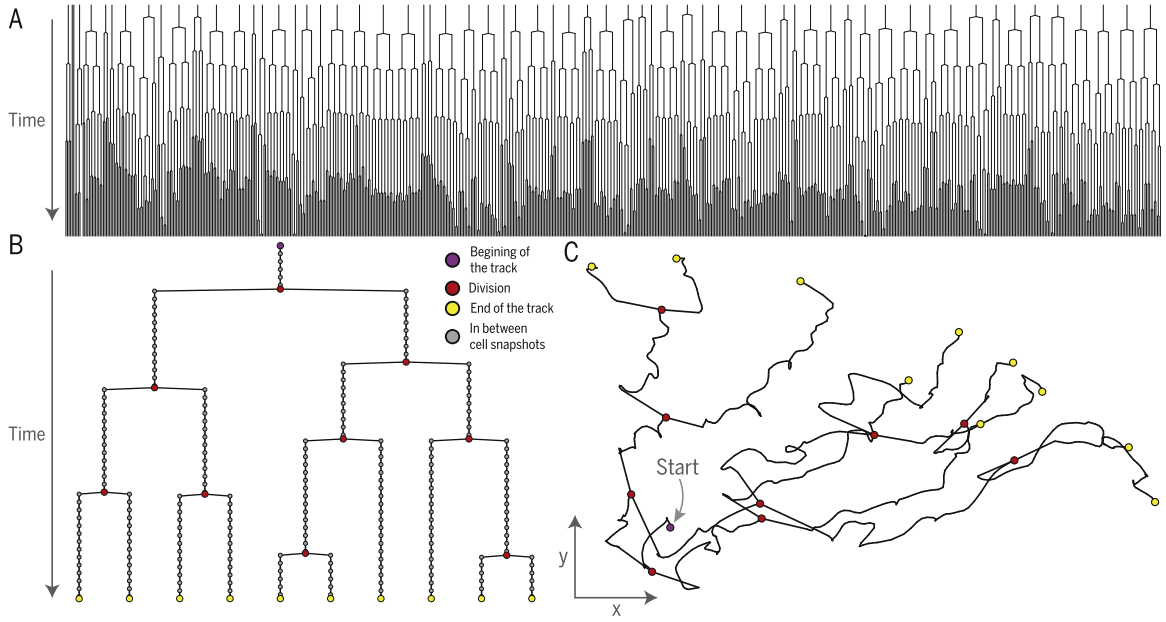


FIGURE 20.7 **Cell tracking output.** (A) Output of the tracking of a *Phallusia mammillata* embryo from the 64-cell stage until the end of the beginning of the tailbud stage. (B) Close-up on one of the tracks from (A). The purple cell is the start of the track, red cells are cell snapshots one time point before division, yellow cells are end of the track cells, and gray cells are intermediary cells. (C) Same tracking as (B) but instead of having time on the y axis, x and y coordinates are plotted. Colors are the same as in (B). *(A) Data from Guignard L, Fiúza U-M, Leggio B, Laussu J, Faure E, Michelin G, et al. Contact area–dependent cell communication and the morphological invariance of ascidian embryogenesis. Science 2020;369:6500.*

Hungarian method [42] using, e.g., the pairwise Euclidean distances between the cells in the ground truth and cells in the detection as costs. Depending on the data a distance threshold has to be chosen up to that a match is considered valid. If the distance value is larger than this threshold, it is usually biologically not plausible to consider the ground truth object and the detected object as the same cell.

Quantifying errors made by segmentation algorithms are similarly done as for detection algorithms by looking at the number of *tp*, *fn*, and *fp*. However, this is done both on the pixel-level and on the object-level. As for the detection it is important to pair cells together. As opposed to detection, usually not the Euclidean distance between cells is used to match cells anymore but the Jaccard index [34] (or Intersection over Union, IoU): $J = \frac{|r_i \cap r_j|}{|r_i \cup r_j|}$ where r_i is a region from the segmented image and r_j a region from the ground truth. J is equal to 0 if the two objects do not overlap and one if they perfectly overlap. Note that the Euclidean distance is a distance, meaning that the closer the distance between two objects is to 0 the more likely it is that they will get paired, while the Jaccard index is a similarity measure meaning that the more similar two objects are (and thus more likely to get paired), the higher the index is. For the application at hand the Jaccard index can be equivalently written as $J = \frac{tp}{tp+fp+fn}$. On the pixel-level, and for a specific potential pair of objects, *tp* is the size of the set of pixels in the intersection of the two objects and $tp + fp + fn$ the size of set of pixels in the union. On the other hand, interpreting *tp*, *fn*, and *fp* on the object-level, *tp* is the number of matched pairs of objects with a pixel-level Jaccard index above a certain threshold τ (*fn* and *fp* analogously). Computing the Jaccard index this way results in an evaluation value for the whole image, often referred to as AP (average precision; the name is unfortunately somewhat imprecise, and its definition dissimilar to the one often used for the evaluation of natural images). Only a single segmented object can have a pixel-level Jaccard index of > 0.5 with a specific ground truth object (as it then covers more than half of it), thus potentially enabling a unique matching. However, especially in 3D and for smaller objects, if an object is segmented with only a single extra pixel in all directions this already has a significant impact on the pixel-level Jaccard index (though maybe less at visual inspection). Thus imprecise yet otherwise correct matches might be undercounted. Moreover, Hirsch and Kainmueller [29] show for 3D nuclei data that the detection performance of using the segmentation masks and a threshold τ of 0.3 to pair cells roughly coincides with using their centers and distances-based matching (as described in the previous paragraph), suggesting it as a good compromise value to balance *fp* and *fn* if one is purely interested in detection. To get a good impression of the overall

performance of the model and to counterbalance the effect small changes in the segmentation can have and to model the uncertainty in where to place the threshold the object-level Jaccard index or AP is usually averaged across a wide range of thresholds (in 0.05 or 0.1 steps from 0 to 1).

Finally, quantifying the errors made by linking algorithms is significantly more complicated and the metrics used are not yet completely standardized. Methods that have been used are to individually count the number of false-negative links (e.g., track interruption or missing division), false-positive links (e.g., nonexisting tracklet or superfluous division), and otherwise wrong links (e.g., identity switch) (see Fig. 20.4) [46]. Another option is to compare the tracking to the ground truth track in a more global way by counting the minimum number of operations necessary to transform the reconstructed tree into the ground truth tree, a method proposed in Ref. [48].

One must keep in mind that even if the scores show a high level accuracy (99%, for example), it does not mean that only 1% of the tracks are wrong. If the 1% error is spread around all the tracks, it is possible that none of the tracks reconstructed are fully correct.

A solid grasp of the basics of cell tracking methods and the errors that can arise from such methods enables a well-founded selection of the mostsuited method. It can also foster an understanding of why a method might have failed and what can be done to avoid these types of failures in the future.

How to choose the best suited type of method

Cell tracking methods can be grouped into three main categories: manual, semiautomatic, and automatic. Before choosing a specific method, it is important to decide the type of method that is best suited. Each of these three types has both advantages and drawbacks depending on the setting and it is not always easy to decide which one to work with.

Overview

Manual methods are precise with as few errors as humanly possible and require less validation of the quality of the results. The downside is that they are often tedious and time-consuming preventing this type of method to be used for large datasets and the manual work required by the user has to be repeated for every new data sample.

On the other side are fully automatic methods. These methods can be applied almost without any restriction on the dataset size. The downside is that these methods are often both specific to a data type, for example, the image modality (what type of microscope, what labeling method, for example), the type of sample or both, and need to be, often heavily, parameterized and/or trained. To be effective this parameterization and training phase necessitates at least some expertise about the method. Moreover, even with good knowledge of the method, the time spent on parameterization and training can be substantial.

The last category, semiautomatic, is in between manual and automatic. There is no exact definition of what a semiautomatic method comprises. Its main feature is that the user is supported in their tracking effort by automating some aspects of it. This can refer to, for example, human-in-the-loop approaches where iterative tracking proposals are made automatically to be then corrected or verified by the user. In other approaches the user initiates a track by selecting a cell in some frame and the algorithm tries to follow that cell in the adjacent frames repeatedly for as long as it is confident that no error is made. While these kinds of methods often offer a good compromise between the precision and usability of manual methods and the scalability of fully automatic methods they can run short when the datasets are too large. For example, in the second approach, even if the algorithm can complete all tracks over many frames automatically without errors, if there are million cells, the user still has to initiate all of them manually.

The frontier between choosing manual, semiautomatic, and automatic methods can be blurry and often multiple solutions are viable. Choosing which of these categories is best suited for a given tracking task and a given user can be complicated but it mainly boils down to **the time the user has to invest** to get the final set of tracks that can then be analyzed. Note that only user time is mentioned as the computational time necessary to reconstruct tracks once a method is parameterized does not involve the user which can therefore perform other tasks in parallel. This time necessary for the user to reconstruct the tracks from a dataset, function of a given method mainly depends on the complexity of the dataset and the total number of cells to track per time frame and time series.

The following sections explain how to better assess the time necessary to assemble a set of tracks and therefore help toward better choosing a type of method.

Total number of cells per time frame and time series

When manual methods are used, the time required to reconstruct all the tracks from a given dataset depends in part on the number of cells n_c to track in each time frame f. n_o represents the number of cells each

multiplied by the number of times they appear in the time series (i.e., the number of snapshots for each cell); it is the total number of objects to detect, in the case of cell tracking, we call each object to detect a cell snapshot (for example, in Fig. 20.5B, $n_o = 24$). For manual methods the time necessary to track all the cells of a dataset (T_M) can be approximated as the time it takes to identify[2] a cell (t_c) multiplied by the total number of objects to track (n_o) plus the time it takes to link two consecutive cell snapshots (t_l) multiplied by the total number of links. The total number of links can be approximated as the total number of cells to track in each time frame minus the number of cells at the initial time point. If we assume that the number of cells at the initial time point is negligible when compared to the total number of cells then

$$T_M = n_o \cdot (t_c + t_l). \tag{20.2}$$

For automatic methods the user time necessary to produce a cell tracking from a given dataset does not depend on the total number of objects n_o in the dataset, which instead will impact the computational time (see end of Section "Overview"). The time (T_A) can be approximated as the time it takes to generate a ground truth (t_{GT}) that will be used for validating the method and if necessary training it plus the time it takes to parameterize the method (t_p) plus the (computational) time it takes to run the algorithm on the whole dataset. Therefore we approximate the total user time it takes to produce a cell tracking from a given dataset as

$$T_A = t_{GT} + t_p. \tag{20.3}$$

The main benefit for automatic methods in terms of time usage is that once the method is trained and/or parameterized, new similar data that need to be tracked can be processed with negligible required user time.

For semiautomatic methods, the total time necessary to produce a cell tracking from a given dataset (T_{SA}) is a bit harder to define universally. Assuming the second approach defined in Section "Overview" it depends on two components: Similar to manual methods each cell n_c has to be identified (t_c), yet if the linking works well, only once instead of in every frame. However, similar to automatic methods, the method has to be parameterized for the linking to work well $t_{p.sa}$. Note that the time to parameterize semiautomatic methods is typically much lower than for automatic methods, as it usually has a lot fewer parameters. Then

$$T_{SA} = n_c \cdot t_c + t_{p.sa}. \tag{20.4}$$

While the total number of cells has a great impact on the user time for manual and semiautomatic methods and the total number of frames on the user time for manual methods, the complexity of the dataset has an impact on all types of methods.

Data complexity

Manual detection of cells and linking them across time frames is, at least for a subset of the data, always necessary. Its importance is obvious for manual methods. For automatic and semiautomatic methods, manual reconstruction of some of the tracks is usually required to validate the method and sometimes to train it. The more complex a dataset is, the more complex it is to manually reconstruct tracks. A dataset can be complex for multiple reasons, the most common being its spatial dimensionality (two or three dimensions), the quality of the images, the spatial and temporal resolution, and the physical properties of the system studied.

Data dimensionality

The dimensionality of a dataset has a strong impact on the ability of the user to detect cells and link them across consecutive time frames. Visualization and annotation of 3D datasets is known to be complicated. This is due to the human visual system; it is hard to position oneself in 3D when using a 2D screen. This problem is of lesser importance for automatic methods; the impact of 3D versus 2D is mostly that the computational time increases significantly which, as mentioned above, is not considered here. Therefore when dealing with 3D time series, manual reconstruction takes significantly longer while automatic methods are only marginally impacted. Semiautomatic methods might also be significantly impacted since the manual input becomes harder and both the automated processing parts and the feedback from the system require more time, too.

Image quality and spatial resolution

The quality (both in terms of signal-to-noise ratio and in terms of spatial resolution) of the images forming a time series impacts the ability to efficiently and correctly detect cells. Often a decrease in quality will impact (semi)automatic methods more severely. Both the human visual system and deep learning methods, a common foundation of automatic methods, are known to be great at recognizing patterns. However, unless a large amount of ground truth training data is available and a lot of time is spent on parameterization, human users still have an edge on handling large variability in the data and at drawing conclusions. That is true only up to a certain extent, once below a threshold of quality,

2 Note that the time necessary to segment an object is often order of magnitudes higher than the time necessary to detect an object.

both the time it takes to manually detect and link cells and the time required for training and parameterizing an automatic method will increase vastly.

Temporal resolution

Temporal resolution refers to the frame rate (images per second or minute or time between two images) that is used to record the time series. The value required for a "good" temporal resolution depends on the size and speed of the objects, which in turn depend on the type of cells. As illustrated in Fig. 20.5, a low temporal resolution can negatively impact automatic tracking methods. This is true for manual methods, too, yet to a lesser extent. While especially automatic deep learning systems might be even better at recognizing patterns (inductive reasoning), a lower temporal resolution results in a higher variability in the observed cell arrangements requiring stronger deductive capabilities, something humans excel at and machine learning systems still struggle with. As mentioned in Section "How to choose the best suited type of method", a higher temporal resolution can help the system to avoid certain kinds of errors. Moreover, especially overlap-based linking systems (two objects are linked if their segmentation masks have a large overlap) benefit greatly from it. On the other hand, if the temporal resolution is too low, correct linking might become impossible, even for expert users.

Physical properties of the system

Different systems have different properties, regardless of the acquisition modality. For example, the cells can be more or less packed, divide more or less often, and have similar or a large variety of shapes and behaviors. The more heterogeneous a dataset is, the harder it is for automatic and semiautomatic methods to correctly reconstruct the tracks and therefore the more finely tuned the algorithm needs to be and the larger the training datasets need to be.

All these differences of complexity of the system studied have an impact on the time it takes to either build the full tracking for manual segmentations, or to finely tune the parameters of automatic and semiautomatic methods and to build the partial tracks necessary for the validation of the method. This notion of complexity is carried throughout this chapter.

Deciding on the method

Now, to choose from the set of manual, semiautomatic, or automatic methods, using the presented characteristics, one can estimate the complexity and number of cells of their data and place it in the chart in Fig. 20.8. The needed user time will also vary between users, for example, someone with a stronger computer science background would probably be more efficient at parameterizing a method than one with a stronger biomedical background. On the other hand, a user with a strong biomedical background might be faster at building ground truths and performing manual detection. This will affect the position of the boundaries between the three types of methods and shift them in one direction of the other.

Overall, manual methods are strongly recommended for one-offs, when such tracks are not expected to be routinely reconstructed in the near future. On the contrary, if a large number of time series are expected to be tracked, then automatic methods are strongly recommended. Semiautomatic methods constitute some form of middle ground, very useful for data with not too high complexity and not too many cells or with varying image statistics that only has to be tracked occasionally, making the effort of setting up automatic methods dispensable.

The remaining of this chapter will be dedicated to describing in more detail each type of method and in giving examples of software tools.

The method types in more detail

The previous section describes how cell tracking works and what the usual types of errors are that can be encountered and helps to choose a type of method, manual, semiautomatic, or fully automatic. This section aims at first describing the specific challenges and limitations that one would face using each of these methods, especially for fully automated methods, and second describing a few specific methods that could be used to perform cell tracking.

Manual tracking

Manual tracking methods are usually the methods of choice when the dataset cannot trivially be tracked in an automatic fashion and the number of objects to track is low enough. Note that here, "trivially tracked" will depend on the expertise of the user.

In these approaches the user has to select every object (i.e., cell) in every frame manually and link it to its ancestor (predecessor) and progeny (successor(s)). The resulting cell tracks are therefore usually almost error free as the user himself/herself is building them step by step. The main challenge that manual methods face is to enable the user to easily navigate and annotate the datasets to track its cells. This challenge is particularly increased when dealing with 3D datasets. The following tools (in alphabetical order) are designed to facilitate visualization and annotation of 2D and/or 3D

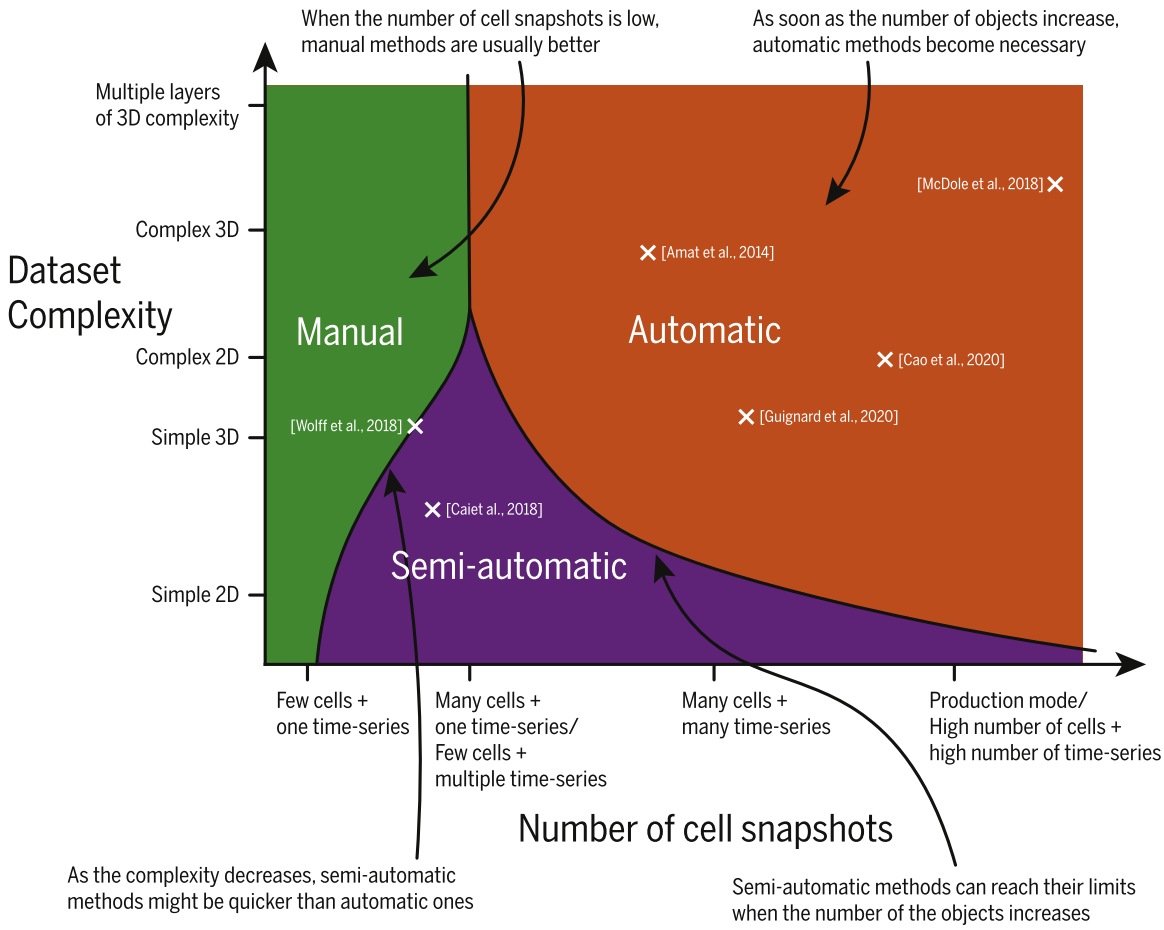


FIGURE 20.8 **Schematics to help choosing a type of method.**

images and therefore allow to perform cell tracking manually:

- CellProfiler (for 2D tracking) [10,50]
- Icy [12]
- Ilastik [5]
- ImageJ/Fiji [61,62]

Note that when dealing with especially large datasets specific visualization tools which optimize memory usage have to be considered. For example, the ImageJ/Fiji plugin named MaMuT [55,71,79] is especially suited for tracking cell positions in terabytes 3D datasets.

Limitations

In general, tools for manual tracking are simple to set up and no parameters have to be tuned. Therefore for smaller datasets they are almost always going to be a good choice. But ultimately manual tracking can be very time-consuming. For larger datasets, the number of objects to be tracked grows rapidly and so does the time to manually track the cells. This quickly becomes prohibitive and more automated methods have to be considered. It is also important to remember that for easier data, semiautomatic methods are often similarly straightforward to set up and take off significant burden from the annotator; and for harder data, especially in dense 3D data, it gets difficult to keep track visually of what goes where without specialized visualization tools. The sweet spot is therefore reasonably difficult data that are not too large.

Semiautomatic tracking

Semiautomatic tracking methods come in different flavors but the main universal characteristic is that all have some form of back-and-forth feedback mechanism between the user and the tracking system. The tracking algorithm suggests a solution, potentially highlighting positions of uncertainty. The user then inspects the resulting tracks and partially or fully corrects the proposed solution.

In some systems the user feedback is used to learn better values for a set of internal parameters and then refine the tracking output (the concept of learning

parameters is a common property of automatic methods; for more on it, see Section "Automatic tracking"). This process can be repeated until the tracking results reach a satisfying quality. Feedback loops of this kind between the user and the algorithm are referred to as iterative or active learning. Tools such as Ilastik [5] and JAABA [35] [3] support this principle. The advantage of iteratively tuning an algorithm using this type of feedback loop between the user and the algorithm is that both the algorithm itself and its tuning can be kept simpler. The algorithm does not have to be designed to allow for every possible kind of complication as the user is there to guide it or fix mistakes interactively. This makes it more accessible to less technically versed users. Moreover, similarly to the automatic methods in the next section, once the algorithm is tuned, it can be used as is on new datasets that have the same characteristics as ones the algorithm was tuned on.

In a different semiautomatic tracking approach the user initiates a track by manually selecting a cell in some frame and the algorithm tries to follow that cell according to certain, usually built-in, heuristics in the adjacent frames repeatedly for as long as it is confident that no error is made. The track can then again be inspected and potentially corrected. These approaches typically have even fewer parameters (that are usually manually tunable but not automatically learnable) and, if applicable to the data at hand, are even easier to use. An exemplary tool for this kind of system is MaMuT [79]; Ilastik supports it as well.

One big challenge that semiautomatic methods face is similar to the one of manual methods: enabling the user to seamlessly navigate and annotate the dataset. For this reason tools that allow for manual tracking often allow for semiautomatic methods, too. This is, for instance, the case for both MaMuT and Ilastik.

Limitations

If the data becomes more complex, semiautomatic methods might struggle to reach an ideal tuning giving an ideal tracking, even with the feedback from the user. In these cases switching to more intricate automatic methods that use heuristics that are well suited for the system studied or are based on deep learning methods might be necessary. Moreover, the back and forth between the user and the algorithm does not necessarily scale well with the dataset size. If the time it takes for the algorithm to give feedback to the user becomes too long or the data too large, fully automatic methods will start to outperform semiautomatic ones.

Automatic tracking

Manual and semiautomatic methods already go a long way in supporting researchers in their tracking problems for data with small or medium numbers of cells. Yet more and more datasets are outside of the scope of these methods. This is in part due to the recent trend, in research in general and in biomedical research in particular, toward larger and larger datasets. As an example, recordings of developing mouse embryos can contain millions of cells and reach sizes of multiple terabytes [49]. Moreover, to be able to make statistically significant statements, often multiple such samples are necessary. Manually tracking every cell of such a dataset becomes practically impossible and even semiautomatic methods quickly reach their limits, especially with growing data complexity. For instance, the recording of an embryo with 15,000 cells over 500 frames contains 7.5 million cell snapshots. To track all the cells of such a dataset manually with an average of one cell every 4 s this would require over 8000 h of work, which corresponds to 4 years of working 40 h per week, 50 weeks per year. Thus the goal becomes to automate the process as completely as possible. This comes with a lot of benefits but also with its own set of unique challenges. The biggest challenge for the user is to correctly parameterize and, when using machine learning methods, to successfully train the algorithm.

There are many different approaches to automatic tracking, but in one way or another, implicitly or explicitly, cells still have to be detected in all frames and linked over time. In the following sections we will describe different types of automatic tracking methods.

Two-step methods

As stated before, tracking methods are often split into two steps. First the detection or segmentation step that identifies each cell snapshot in each time point of the dataset. Second the linking step that links cell snapshots between consecutive frames reconstructing the lineage. Several approaches exist for the first step, cell detection and/or segmentation. These methods are mentioned but not described here:

- Thresholding [2,4,38,45]
- Watershed [18,23,66]
- Deformable models: level set [22], deformable mesh [13]
- Laplacian of Gaussian [37,58]
- Graphical models [5,36,59]
- Gaussian mixture model [3]
- Machine learning, especially neural networks [5,9,57,63,78]

3 JAABA was initially designed for fly pose recognition and tracking and might not be well suited for cell tracking.

For the second step, the linking, the goal is to find a mapping $\mathcal{T}$ between objects in consecutive time points. This mapping matches the snapshot of a cell c at time t to its corresponding snapshot at time $t+\delta t$. Formally $\mathcal{T}$ is often described as a graph. A graph is a mathematical structure that is defined by its set of nodes (or vertices, here a node is a cell snapshot) and its set of edges that link the nodes (here the ancestry/progeny relationship). Because of the nature of the ancestry/progeny relationship specific to cells, this graph has the particular substructure of a set of trees (forest). $\mathcal{T}$ is usually even more constrained because of the biological properties of the system tracked together with the properties of the imaging modality. For example, most of the time, because there is no cell fusion, the snapshot of a cell c can have at most one antecedent at the previous time. Another example relating to the imaging modality is that if the time resolution is high enough to capture all division events, the snapshot of a cell c can have 0, 1, or 2 successors: 0 in case of apoptosis, 2 in case of cell division, and otherwise it has 1 successor. All these constraints can be taken into account when trying to build such a mapping $\mathcal{T}$. Different types of methods exist when it comes to cell tracking algorithms.

Nearest antecedent

A first approach to building the cell tracks is to iteratively build $\mathcal{T}$ by comparing the cells in consecutive time points and by matching the ones that correspond best. It is interesting to note that, computationally speaking, if given two sets of points P_1 and P_2 (that in our case would be the sets of cell snapshots at two consecutive time points), matching all the elements in P_1 to its closest element in P_2 could result in a different output than matching all the elements of P_2 to their closest element in P_1. A trivial example for that is showcased in Fig. 20.9. Because of that property and the fact that most of the time cells can divide but not fuse, computational methods usually are designed to calculate and find the antecedents rather than the successors. To find the best matching cell snapshot to another cell it is necessary to be able to do pairwise comparisons between cells. To compare two cells in two consecutive time frames, a similarity measure S between them is computed. In the simplest case this measure can be the (negative) physical Euclidean distance between the (center of the) two cells. It can be further refined for improved matching. For instance, Keller et al. [38] or Brown et al. [8] add a correlation analysis of the shape of the nuclei, in StarryNite [4] a constraint on the maximum number of successors is added. Cell motion can be exploited in order to improve it by giving a probability distribution for the position of a cell at some time t given its previous positions. These and others can be then combined to form the final similarity measure S. Its goal is that snapshots of the same cell have a high similarity while snapshots of two different cells have a low one. This measure associates a numeric value to any pair of cells from two consecutive time points. Finally, $\mathcal{T}$ can then be built by mapping every cell at a given time $t+\delta t$ to its nearest one at time t (in terms of the previously computed similarity measure S). In other words, the mapping $\mathcal{T}$ of a cell $c_{t+\delta t}$ at time $t+\delta t$ is the cell c_t at time t that is the closest to $c_{t+\delta t}$ according to the measure S:

$$\mathcal{T}(c_{t+\delta t}) = \underset{c_j \in t}{\operatorname{argmax}} S\left(c_{t+\delta t}, c_j\right) = c_t. \qquad (20.5)$$

Constrained tracking

One major obstacle during tracking is that the detection performance is often flawed. And even when it is not, the frame rate is rarely high enough to cover all potential cell movements, especially during cell division. If such problems are a concern, it is better to use algorithms that can detect these potential errors and discard the tracks that contain them, correct them, or prevent them from happening when reconstructing the tracks. These methods are implemented as constraints on the mapping (also referred to as

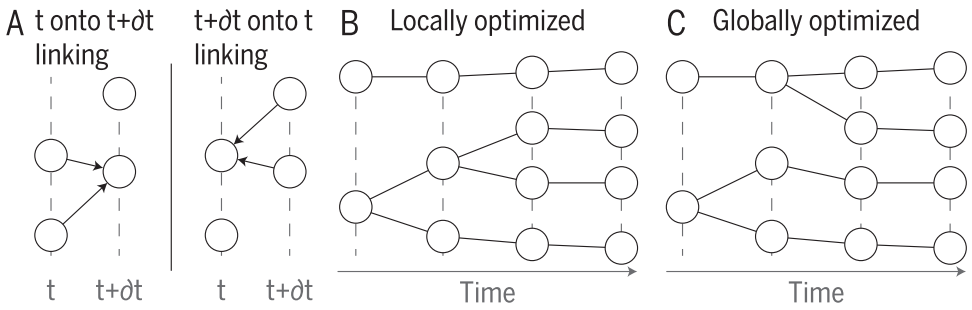


FIGURE 20.9 **Remarkable tracking differences for simple cases.** (A) Forward (left) or backward (right) linking induces a different mapping. (B) Result of tracking algorithm based on local optimization, the bottom cell divides into two consecutive time points which is usually unlikely. (C) Result of a method based on global optimization which prevents consecutive division. The method has the same input as B but the global optimization process results in different tracks that are more likely to be biologically correct.

conservation tracking [60]). The constraints can be split into two categories: logically motivated and biologically motivated ones. Biological constraints are sometimes also referred to as moral constraints; some examples are:

- a cell can only divide into two daughter cells
- there is a minimum amount of time that has to pass between to consecutive division of a cell
- the distance a cell can move between to consecutive time points is limited

Examples for logical constraints are:

- A cell cannot appear out of nowhere; it has to exist in the previous frame, has to be the result of a cell division, or, if that is possible in the setup, has to have moved in from outside the boundary of the view of the microscope
- Depending on the type of data a cell might not disappear into nothing unless, again if that is possible in the setup, it can move out of the view of the microscope or undergo apoptosis.

When these constraints are used to detect errors and discard potential tracks containing them, the final tracks are checked for adherence of these constraints.

If error correction or prevention is the goal, violations of such constraints can be penalized within the method to preclude them from occurring in the final tracks. Such constraints can be enforced either locally or globally.

Local enforcement usually employs so-called greedy methods (such as the nearest antecedent methods described above). These methods are termed greedy because at each step the currently best option that does not violate any constraints is selected and once a choice is made it cannot be undone. As a result, while the matching between cells can be optimal locally it might not be optimal globally, but in general it is significantly faster. An example of a potential difference between local and global matching is showcased in Fig. 20.9.

One common way to enforce such constraints globally is to formulate the problem as a global optimization problem. These problems have to be formulated in a precise way but can then be solved by optimized generic solvers (e.g., SCIP [1] and Gurobi [25]) using methods such as integer linear programming (ILP) [5,46], flow-based approaches [5,18], or simulated annealing [16]. Simplified, each possible choice (e.g., adding or removing a cell snapshot or link) has a (positive or negative) cost associated with it and a solver tries to find the overall lowest possible sum of costs (which is called the objective) while adhering to the constraints. This will result in the globally optimal solution for the given set of predictions, costs, and constraints but can potentially be very slow to the point of being unsolvable in a reasonable time. Analogously to physical conservation laws this has also been dubbed conservation tracking [60] as the system ensures the conservation of certain properties (e.g., concerning the number of objects) through the compiled constraints.

A problem that these methods raise is that if the detection is flawed, a morally or logically sound solution might not exist at all. If a valid solution does not exist, the method should still provide a solution that is as good as possible instead of not providing any solution at all. Some approaches optionally add some semiautomatic/human-in-the-loop aspects (see, for example, Ilastik and Elephant [69]) so that the decision in such cases can be verified by a human operator.

One-step methods

As opposed to two-step methods, one-step methods perform detection and tracking concurrently. This precludes building a detection or segmentation which would not give rise to a valid tracking. In contour evolution methods the objects in a single frame are detected, manually or automatically, and are then propagated forward or backward in time frame by frame. The propagation of the detected cells at a given time t constrains the detection of the cells at the next time $t + \delta t$. In the algorithm named Tracking Gaussian Mixture Model (TGMM) [3] the cells that were detected at time t are used to constrain the number of cells to be detected and their shapes in $t + \delta t$. Similarly in the algorithm Automatic Segmentation and Tracking of Embryonic Cells (ASTECs) [23], the segmented cells are propagated from one time point to the next. The propagated cells are compared to the local, independent segmentation of that next time point. The algorithm then decides what is the correct shape of the cells and whether divisions occurred.

Limitations

If correctly parameterized and trained, automatic methods are extremely powerful. They allow for the reconstruction of cell tracks from huge datasets and therefore enable the large-scale analyses that are usually necessary for cohort or population studies, or for the validation of mathematical models, for example. This high-throughput property of automatic methods does not come for free. The knowledge and time necessary to parameterize and potentially train these methods can be immense and should not be overlooked. This is why it is often better to use manual or semiautomatic methods instead of fully automatic ones if the amount of cells to track is small enough or if the complexity of the dataset is high enough that the parameterization of automatic methods becomes harder (see Fig. 20.8).

Deep learning–based approaches

Thanks to the increasing power of computers and the fact that more datasets are becoming available, more and more deep learning–based image analysis methods have been developed since the early 2010s. This trend has been mainly observed in the area of natural images (e.g., face and object recognition, tracking of cars, pedestrians, and other objects in traffic or public places). In the more recent years this trend of deep learning–based approaches has expanded to biological and medical systems not only for cell detection and tracking as will be discussed below but also, for example, for image denoising [40,77] or for quantifying the morphology of cells [81,82]. These methods are getting more numerous and powerful and, similarly to the situation in other areas, at least for the time being, are starting to overshadow most other automatic tracking methods. The following sections explain the principles of deep learning–based cell tracking algorithms.

From pedestrian and car tracking to cell tracking in biomedical images

While the goal of tracking algorithms for natural images is similar to the one in biomedical images, the use cases pose different challenges and the solutions developed for natural datasets are therefore not directly transferable to biomedical ones. Natural images datasets are mainly 2D, whereas biomedical datasets predestined for deep learning often consist of huge 3D volumes. Natural images datasets usually have significantly fewer objects to track (typically a few tens or hundreds of objects) compared to biomedical datasets (up to tens of thousands of cells). In natural scenes the objects to be tracked often vary significantly (e.g., cars and humans). Cells, and especially nuclei, on the other hand are more similar and sometimes even nearly identical. Moreover, in natural image datasets, typical objects can only leave the view of the camera, but they cannot split, as opposed to cells which split during cell division. And finally, helping significantly with tracking, the acquisition frame rate for natural images is often a lot higher compared to the speed of the tracked objects. This property is not observed in biomedical datasets. Usually the acquisition frame rates for natural data are about 24 frames per seconds or 2880 frames every 2 min while for biomedical datasets it is more often around 1 frame every 2 min. Though on the other hand the speed in pixels per second that an object moves is lower in cell data, it is not enough to compensate for the reduced frame rate.

This can be due to a technical limitation of the microscope or to avoid phototoxicity in long recordings. For similar reasons the signal-to-noise ratio is often quite low. Natural scene data often consist of millions of megabyte-sized images, biomedical data of far fewer images but sometimes each terabyte sized. Finally, the emphasis placed on different evaluation metrics (e.g., speed, accuracy, and ease of use) varies. In certain applications, for example, autonomous driving, an extremely low error rate, is required. Similarly this is the case to be able to answer many biological questions. This can often only be achieved at the cost of slower speed or more tunable parameters and precise ground-truth.

The challenges of building a ground-truth

Having correct and extensive ground-truth data is a strong requirement for deep learning approaches. As a very general rule of thumb, more ground-truth leads to better performance. This is particularly important for more difficult cases, e.g., cell divisions which usually are much rarer, cells deeper within the tissue that often have a lower signal-to-noise ratio, and denser areas where, in the case of stained nuclei, they appear to be touching, due to the limited spatial resolution of the microscope. Moreover, potential slight inaccuracies in the ground truth can lead to mistraining of the algorithm. For example, in strongly anisotropic data misplacing a cell by a slice or two would correspond to a large distance in the metric space. Often a lot of time is necessary to set up these ground-truth datasets that will be used to train the algorithm and one has to be careful when building the ground-truth, especially making sure to be extensive and error-free.

Detection and segmentation using deep learning

A number of deep learning–based approaches specialized to cell tracking have been developed in the last couple of years. As explained above, a major limiting factor is the need for ground-truth. The cost of obtaining enough ground-truth data for large, 3D datasets can be very high. Because of this limitation the existing methods are largely split into two categories: first, the segment-and-track category which relies on segmentation-based ground-truth, and second, the detect-and-track category which relies on detection-based ground-truth. This split into two categories is due to the fact that, as explained earlier, manually building segmentations are significantly more time-consuming than manually building detections, that is because segmentation involves reconstructing the shape of an object and detection involves "just" detecting the position of an object. Detection annotations are also referred to as weak annotations (because they contain less information). For smaller 2D datasets segment-and-track approaches already developed for natural images can be adapted or refined. For example, Chen et al. [11] extends the popular Mask R-CNN [28]

framework to tracking by predicting similarity scores for cells in successive frames. These similarity scores are similar to the similarity measure S between cells discussed in Section "Two-step methods" and are adapted from the Mask R-CNN method. Having a (predicted) segmentation allows for overlap-based linking, which is especially powerful for less complex 2D and 3D data.

The quality of the final tracks depends both on the quality of the segmentation or detection part and on the quality of the linking part. An improvement in either usually results in an improvement of the tracks. And the improvement of segmentation and detection models in the last few years has been significant. Therefore an alternative approach is to replace the classical segmentation or detection part in existing tracking tools with a deep learning–based model while keeping the original linking approach unchanged. The core of most currently used learning algorithms for biomedical data is a U-Net [57]. Examples are StarDist [63,78], mostly for nuclei or nucleus-shaped objects (i.e., ellipse or ellipsoid shapes), and Cellpose [68] or PatchPerPix [30], which also work for more arbitrarily shaped cells. StarDist predicts, for each cell, a star-convex polygon or polyhedra (which limits the types of cells it can be used for to convex cells) and is applicable in 2D and 3D. In Cellpose a neural network is trained to predict a topographically smooth surface in which each valley corresponds to one object. This surface can then be segmented easily with the classical watershed algorithm. For linking, we can employ any modular tracking system. For example, in Ref. [17], StarDist has been combined with TrackMate. Another option, in case of segmentation masks, is to load the results into Ilastik and use their tracking system.

Disadvantages are the usual ones for deep learning–based systems. To train such a model a decent amount of ground truth annotations are required. Moreover, powerful hardware (usually at least a good GPU) and a lot of model training is necessary. The resulting model is often specific to the type of data it has been trained on. Recently a lot of effort has been invested in lowering the bar of entry to use deep learning models in the area of biomedical image analysis; ZeroCostDL4Mic [75] can be used with almost no programming knowledge.

Deep learning–based tracking

However, instead of combining deep learning detection or segmentation with classical tracking systems, other systems extend the deep learning aspects into the linking, too. Many use the detect-and-link approach and share some similarities. In a nutshell, deep neural networks are used to both detect cell locations and to predict links between cells. In a second step some form of optimization (greedy or linear matching, ILP, etc.) is used to connect the links over time and potentially to take certain kinds of errors and inaccuracies in the predictions into account [27,46,52,69].

In the following we will describe the general concept in more detail and point to the differences and the general difficulties. A model, typically a neural network such as the U-Net [57], is trained to predict the locations of all cells in the data; often the network has access to multiple frames to make use of temporal information. To detect the cells a mixture of Gaussian blobs placed at each point annotations is regressed. The maxima of the prediction corresponds to cell candidates [31]. Then a neural network (the same or a second one, if it is the same this is referred to as multitask learning, there are inconclusive reports on the effectivity) predicts the cells' locations in the previous frame. Typically, and similarly to most tracking algorithms; the links are predicted backwards in time as a cell might have two successors in the case of a cell division but always has only a single predecessor. This will usually not directly result in perfect tracks. The detection network might produce false positives (overdetection) or false negatives (miss a cell/under detection). Large location changes, especially during cell division, might corrupt the output of the linking network (see Section "How to choose the best suited type of method").

Different levels of optimization complexity are employed to alleviate these issues, from locally optimal solvers (e.g., greedy) to globally optimal ones (e.g., ILP) by selecting a subset of detections and links. The final solution consists of the best (for a particular solver) valid selection of proposed candidates and their links. A selection is valid if it adheres to a number of different constraints (see Section "Two-step methods").

Future challenges

Cell tracking is a hard problem that is far from being solved even though there is a large variety of methods existing already (see Table 20.1 for a small subset of the existing methods). The main reason why it is so difficult is that even a very low linking error rate between consecutive frames results in the breaking of most of the tracks after few time points (illustrated in Fig. 20.10; for more details, see Section "How to choose the best suited type of method"). While manual methods are less impacted by this difficulty, the always growing amount of data generated makes these methods inefficient most of the time. Out of necessity automatic methods are therefore becoming the method of choice but the high difficulty of the cell tracking task has side effects. First, the methods are still not perfect and require improvement to reach the level of quality necessary to be used routinely without having to check the quality

TABLE 20.1 Detection, segmentation, and tracking methods mentioned in this chapter.

Name	Method	Type	Organelle	Image type	References
ASTEC	A	S + L	Membrane	Fluo, 3D	[23]
Cellpose	A	S only	Nuclei	Any, 2D/3D	[68]
CellProfiler	SA/A	S + L	Nuclei	Any, 2D	[10]
Icy	SA/A	D + L	Nuc, Mem	Any, 2D/3D	[12]
Ilastik	M/SA/A	S + L	Nuc, Mem	Any, 2D/3D	[5]
Linajea	A	D + L	Nuclei	Fluo, 3D	[46]
MaMuT	M/SA	D + L	Nuclei	Any, 3D	[79]
StarDist	A	S only	Nuclei	Any, 2D/3D	[63,78]
StarryNite	A	D + L	Nuclei	Fluo, 3D	[4]
TGMM	A	S + L	Nuclei	Fluo, 3D	[3]

A, Automatic; *D*, Detection; *L*, Linking; *M*, Manual; *SA*, Semiautomatic; *S*, Segmentation.

of the results. Second, the methods are becoming more and more complex and can be difficult to deploy, especially for nonexpert users. Third, even once deployed, the methods can be difficult to use, especially when it comes to their parameterization.

Improving the quality of the results

One (obvious) way to improve the quality of the results is to improve the quality of the methods in order to decrease the error rate. Many research groups are currently working on improving the detection, the segmentation, and/or the linking processes. One way that seems particularly promising for cell tracking is to combine detection or segmentation together with linking. As mentioned in section "One-step methods" this allows to constrain the detection by the linking and vice versa. An early method that applied this concept to large-scale biomedical data was TGMM [3], a greedy approach: when a link is added between consecutive cell snapshots, it cannot be altered within the scope of the algorithm (postcorrection algorithms can still be run). Linajea [46], a newer method combining detection and tracking, follows a global approach, considering the tracks in almost their entirety and by this significantly improving the results.

Another way to improve the quality of the tracking algorithms is to improve the quality of the images. There are two options: recording higher-quality images or preprocessing the time series data before supplying it to the tracking algorithm. Recording higher-quality time series requires the recording of completely new data, which is not always an option, and without significant progress in the field of microscopy is often not even within the realm of possible options at all. The remaining option is preprocessing, this step was not previously discussed in this chapter as most of the time it is included in the detection or segmentation algorithms. Nevertheless new methods, based on deep learning, are being developed to improve the quality of the images. These methods (e.g., Refs. [40,44,77]) are becoming more and

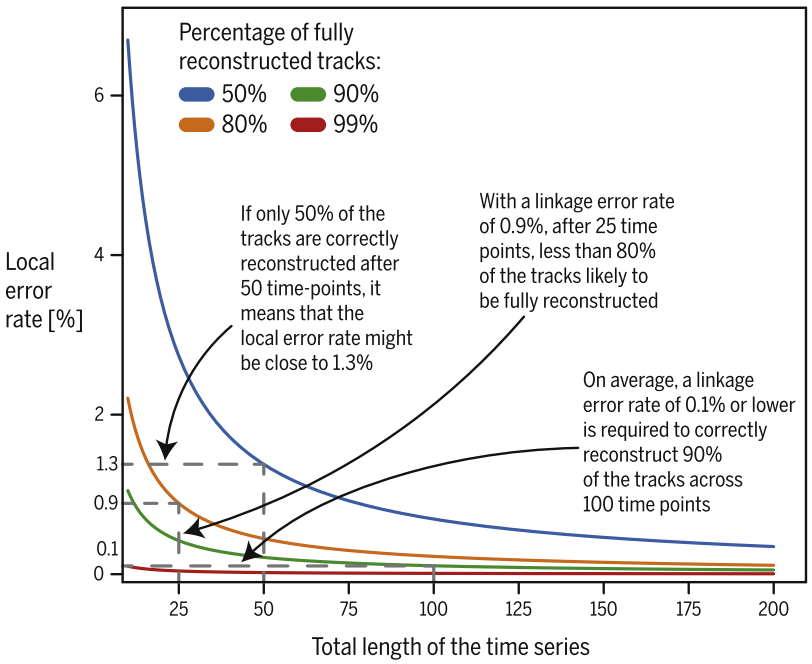


FIGURE 20.10 **Impact of local errors to the global tracking.** The figure shows the average maximum local error rate (or linkage error rate) that is necessary to obtain a given percentage of fully reconstructed tracks, function of number of time points in a time series. The different curves show different global error targets. The arrows give examples on ways to read the figure.

more common and greatly help to improve the quality of the images, in turn helping cell tracking algorithms.

While developing ever better performing algorithms is necessary, the state-of-the-art methods have reached a level of quality where improving upon becomes challenging. New improvements often involve large amount of work but are also often small steps as testified by the small improvements observed in the results of the cell tracking challenge over the past few years (see Ref. [73] and http://celltrackingchallenge.net/).

A very different approach is to accept that the tracks are not perfect and to take advantage of the fact that the tracking results are still usually of extremely high quality locally. Recent methods such as the Statistical Vector Flow (SVF) method [49] make use of this property to reconstruct the probabilistic cell flows across consecutive frames, reconstructing cell movements over long time series. The resulting flow reconstructed in SVF comes at the cost of losing cell identity information at cell divisions which can be prohibitive for a lot of applications.

Accessibility to the methods

Automatic methods can be quite difficult to install and often require expert knowledge, especially for methods using Graphical Processing Units (GPUs) which is the case for most deep learning methods. A lot of effort has been put for some years now into making image analysis methods more accessible through easy-to-install and easy-to-use softwares such as ImageJ/Fiji [62], Icy [12] or CellProfiler [10] but similar solution did not exist for deep learning methods until recently. For example, CLIJ [26] and DeepImageJ [21] provide easy access to GPU accelerated image processing tools paving the path toward more accessible GPU-based tracking methods. Another solution, proposed within the framework of ZeroCostDL4Mic [75], for example, is to take advantage of the free computational resources offered in Google Colab to provide deep learning–based algorithms for biomedical image analysis (including but not limited to detection, segmentation, and tracking algorithms).

Facilitating the training

One other current drawback of deep learning methods that was previously mentioned is the fact that comprehensive ground truth is necessary. In cases where the datasets are complex it sometimes means that a large amount of ground truth is necessary. Therefore, one way toward facilitating the training is to facilitate the manual creation of annotations. As mentioned previously, manual tracking is hard, especially in 3D, because of the fact that the datasets are projected onto a computer screen, in 2D. In order to alleviate this problem, new methods which allow to visualize datasets and track their cells in 3D using virtual reality have been recently developed and are getting more and more popular. The software [20], for example, not only allows to visualize the datasets in virtual reality but it also allows to detect and track cells by using the direction the user is looking toward. Reducing the amount of ground truth is also a direction that has been studied recently and that is promising. Linajea [46], for example, is designed in such a way that only partial annotation is necessary.

Another way to reduce the amount of annotation necessary is the use of transfer learning (or domain adaptation) methods. In transfer learning a model that has been trained on one task or type of data, is used to improve the learning process on a different task or dataset. For example, a model that has been trained on one type of nuclei is used to initialize a second model that is to be trained on a different type of nuclei. As objects often share visual features the two tasks are related and the knowledge incorporated in the first model can be a useful foundation for the new model. Consequently fewer annotated examples (ground truth) are required for good performance. This is a very active field of research at the moment; Roels et al. [56] apply it to two different biomedical image datasets; Hsu et al. [32] even transfer from natural images to biomedical images.

Where do you go from here?

We have the tracks, what now? To quantify and model cell movement dynamics from time series, as it was done in the studies from the introduction, for example, and for further biomedical studies, cell tracking is a necessary but not sufficient step. Once a satisfying set of cell tracks has been reconstructed, one has to extract the quantitative information from it to then model the observations. Before any quantification, it is crucial to visualize the resulting tracks, it allows to make sure that nothing went completely wrong, that there are no glaring errors in the tracks and to get a first sense of the behavior of the cells. This first visualization of the track will develop intuitions on the dataset and will help to drive the future analyses. Visualizing tracks can be done with some of the tools mentioned before, e.g., CellProfiler, ImageJ/FIJI, and TrackMate. But, if the dataset is more complex, for example, large 3D time series, these tools might come short. In that case more specific programs might be considered such as the previously mentioned MaMuT [79] or software that is dedicated to visualization such as Morphonet [43], linus [76], or Napari [65]. To start the quantification and analysis of the cells and track their properties, some of the tools discussed in this chapter offer built-in

methods (CellProfiler, TrackMate, and ImageJ/FIJI, Icy). Moreover, more effort has been put recently into formalizing the quantification of cell dynamics from cell tracks. For example, in Refs. [24] and [51] or more recently in Ref. [67]. Still, it is often necessary that the user has to develop the methods to precisely quantify the specific phenomenon that they want to observe themself. By our experience, this quantification part can be equally difficult as the tracking itself.

Glossary

cell snapshot An instance of a cell in a particular frame of a time series.

dataset complexity In the context of the chapter, the complexity of a dataset is tied to multiple factors such as its dimensionality, quality, temporal and spatial resolution, and its physical properties. The more complex a dataset, the longer it will take a user to reconstruct its tracks.

detection When tracking objects, detection is the action or task of finding the position of the object(s) to track.

false negative In the context of tracking algorithm, false-negative errors are when the method fails to detect or segment a cell or a link between two consecutive cells.

false positive In the context of tracking algorithm, false-positive errors are when the method detects or segments a cell or a link between two consecutive cells where it should not have.

frame rate Number of images recorded per unit of time (usually seconds for classical video recordings).

Graphical Processing Unit (GPU) Specialized hardware, originally developed for computer graphics, now routinely used for general purpose computing, powerful for specific types of computation, for instance, matrix multiplication, a core operation of machine learning algorithms.

ground-truth Manual annotations, for example, per image in the form of labels, or pixelwise in the form of detection or segmentation for some dataset.

image modality The image modality refers to the method with which an image was acquired, specifically the type of microscope and the labeling method.

intensity profile The intensity profile of an object is a list of intensities taken along one or multiple lines. It informs about how the intensity changes along one or several directions. For example, when imaged with fluorescence microscopy, nuclei usually have an intensity profile which increases from the border of the nuclei to their centers and decreases afterward. Depending on the labeling method and the microscopy modality, the intensity profile may vary.

lineage A cell lineage refers to the complete developmental history of a cell, tissue, or fertilized embryo.

linking When tracking objects, linking is the action or task of connecting two objects (cell snapshots of the same cell) across two time frames.

model In the context of machine learning model refers to both a specific system or set of parameters that is to be trained (e.g., a neural network, decision trees, etc.) and a trained instance of such a system.

overdetection For detection methods, same as false-positive error.

oversegmentation For segmentation methods, same as false-positive error.

segmentation When tracking objects, segmentation is the action or task of finding the object(s) to track and reconstructing their shape.

signal-to-noise ratio Measure that compares the level of a desired signal to the level of noise in a given image or dataset. If the value is larger than 1, the signal is higher than the noise. The higher the measure is, the better the signal is compared to the noise of the image.

spatial resolution Quantification of the number of measurements (e.g., pixels or voxels) per spatial unit. The spatial resolution informs about the minimum distance two objects have to be apart to be resolved. A spatial resolution that is too low would not allow for the separation of two different cells, for example.

temporal resolution Quantification of the number of measurements (e.g., images) per temporal unit. A temporal resolution that is too low would not allow to correctly infer where a cell comes from.

time series Time series refers to a sequence of consecutively recorded images of a set of cells.

track The sequence of cell snapshots of a single cell along its complete lifetime.

tracking Tracking is the task of locating objects in a time series and linking them across time.

underdetection For detection methods, same as false-negative error.

undersegmentation For segmentation methods, same as false-negative error.

user time The time a user has to personally invest in order to get the tracking for a given dataset.

References

[1] Achterberg T. SCIP: solving constraint integer programs. Math Program Comput 2009;1(1):1–41.

[2] Al-Kofahi O, Radke RJ, Goderie SK, Shen Q, Temple S, Roysam B. Automated cell lineage construction: a rapid method to analyze clonal development established with murine neural progenitor cells. Cell Cycle 2006;5(3):327–35.

[3] Amat F, Lemon W, Mossing DP, McDole K, Wan Y, Branson K, Myers EW, Keller PJ. Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data. Nat Methods 2014;11(9):951–8.

[4] Bao Z, Murray JI, Boyle T, Ooi SL, Sandel MJ, Waterston RH. Automated cell lineage tracing in caenorhabditis elegans. Proc Natl Acad Sci USA 2006;103(8):2707–12.

[5] Berg S, Kutra D, Kroeger T, Straehle CN, Kausler BX, Haubold C, et al. Ilastik: interactive machine learning for (bio) image analysis. Nat Methods 2019;16:1226–32.

[6] Bertet C, Sulak L, Lecuit T. Myosin-dependent junction remodelling controls planar cell intercalation and axis elongation. Nature 2004;429(6992):667–71.

[7] Blankenship JT, Backovic ST, Sanny JS, Weitz O, Zallen JA. Multicellular rosette formation links planar cell polarity to tissue morphogenesis. Dev Cell 2006;11(4):459–70.

[8] Brown KE, Keller PJ, Ramialison M, Rembold M, Stelzer EH, Loosli F, Wittbrodt J. Nlcam modulates midline convergence during anterior neural plate morphogenesis. Dev Biol 2010;339(1):14–25.

[9] Cao J, Guan G, Ho VWS, Wong M-K, Chan L-Y, Tang C, Zhao Z, Yan H. Establishment of a morphological atlas of the caenorhabditis elegans embryo using deep-learning-based 4D segmentation. Nat Commun 2020;11(1):1–14.

[10] Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, Guertin DA, Chang JH, Lindquist RA, Moffat J, Golland P, Sabatini DM. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biol 2006;7(10):R100.

[11] Chen Y, Song Y, Zhang C, Zhang F, O'Donnell L, Chrzanowski W, et al. CellTrack R-CNN: a novel end-to-end deep neural network for cell segmentation and tracking in microscopy images. 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI) 2021:779–82.

[12] de Chaumont F, Dallongeville S, Chenouard N, Hervé N, Pop S, Provoost T, Meas-Yedid V, Pankajakshan P, Lecomte T, Le Montagner Y, Lagache T, Dufour A, Olivo-Marin J-C. Icy: an open bioimage informatics platform for extended reproducible research. Nat Methods 2012;9(7):690−6.
[13] Dufour A, Thibeaux R, Labruyere E, Guillen N, Olivo-Marin J-C. 3-D active meshes: fast discrete deformable models for cell tracking in 3-D time-lapse microscopy. IEEE Trans Image Process 2010;20(7):1925−37.
[14] Dye NA, Popović M, Iyer KV, Fuhrmann JF, Piscitello-Gómez R, Eaton S, Jülicher F. Self-organized patterning of cell morphology via mechanosensitive feedback. Elife 2021;10:e57964.
[15] Etournay R, Popović M, Merkel M, Nandi A, Blasse C, Aigouy B, Brandl H, Myers G, Salbreux G, Jülicher F, et al. Interplay of cell dynamics and epithelial tension during morphogenesis of the drosophila pupal wing. Elife 2015;4:e07090.
[16] Faure E, Savy T, Rizzi B, Melani C, Stašová O, Fabrèges D, Špir R, Hammons M, Čúnderlík R, Recher G, et al. A workflow to process 3D+ time microscopy images of developing organisms and reconstruct their cell lineage. Nat Commun 2016;7(1):1−10.
[17] Fazeli E, Roy N, Follain G, Laine R, von Chamier L, Hänninen P, Eriksson J, Tinevez J, Jacquemet G. Automated cell tracking using StarDist and TrackMate. F1000Research 2020;9(1279).
[18] Fernandez R, Das P, Mirabet V, Moscardi E, Traas J, Verdeil J-L, Malandain G, Godin C. Imaging plant growth in 4D: robust tissue reconstruction and lineaging at cell resolution. Nat Methods 2010; 7(7):547−53.
[19] Glickman NS, Kimmel CB, Jones MA, Adams RJ. Shaping the zebrafish notochord. Development 2003;130(5):873−87.
[20] GÃnther U, Pietzsch T, Gupta A, Harrington KI, Tomancak P, Gumhold S, Sbalzarini IF. scenery: flexible virtual reality visualization on the java vm. In: 2019 IEEE visualization conference (VIS); 2019. p. 1−5.
[21] Gómez-de Mariscal E, García-López-de Haro C, Ouyang W, Donati L, Lundberg E, Unser M, Muñoz-Barrutia A, Sage D. DeepImageJ: a user-friendly environment to run deep learning models in ImageJ. bioRxiv 2021:799270.
[22] Grushnikov A, Niwayama R, Kanade T, Yagi Y. 3D level set method for blastomere segmentation of preimplantation embryos in fluorescence microscopy images. Mach Vis Appl 2018;29(1): 125−34.
[23] Guignard L, Fiúza U-M, Leggio B, Laussu J, Faure E, Michelin G, Biasuz K, Hufnagel L, Malandain G, Godin C, Lemaire P. Contact area−dependent cell communication and the morphological invariance of ascidian embryogenesis. Science 2020;369(6500).
[24] Guirao B, Rigaud SU, Bosveld F, Bailles A, López-Gay J, Ishihara S, Sugimura K, Graner F, Bellaïche Y. Unified quantitative characterization of epithelial tissue development. Elife 2015;4: e08519.
[25] Gurobi Optimization. Gurobi optimizer reference manual. 2021.
[26] Haase R, Royer LA, Steinbach P, Schmidt D, Dibrov A, Schmidt U, Weigert M, Maghelli N, Tomancak P, Jug F, Myers EW. CLIJ: GPU-accelerated image processing for everyone. Nat Methods 2020; 17(1):5−6.
[27] Hayashida J, Nishimura K, Bise R. MPM: joint representation of motion and position map for cell tracking. 2020.
[28] He K, Gkioxari G, Dollár P, Girshick R. Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision; 2017. p. 2961−9.
[29] Hirsch P, Kainmueller D. An auxiliary task for learning nuclei segmentation in 3d microscopy images. In: International Conference on medical Imaging with deep learning, MIDL 2020, 6−8 July 2020, Montréal, QC, Canada, volume 121 of Proceedings of machine learning research. PMLR; 2020. p. 304−21.
[30] Hirsch P, Mais L, Kainmueller D. PatchPerPix for instance segmentation. CoRR; 2020.
[31] Höfener H, Homeyer A, Weiss N, Molin J, Lundström CF, Hahn HK. Deep learning nuclei detection: a simple approach can deliver state-of-the-art results. Comput Med Imag Graph 2018;70:43−52.
[32] Hsu J, Chiu W, Yeung S. DARCNN: domain adaptive region-based convolutional neural network for unsupervised instance segmentation in biomedical images. CoRR 2021:01325. abs/2104.
[33] Irvine KD, Wieschaus E. Cell intercalation during Drosophila germband extension and its regulation by pair-rule segmentation genes. Development 1994;120(4):827−41.
[34] Jaccard P. The distribution of the flora in the alpine zone. 1. New Phytol 1912;11(2):37−50.
[35] Kabra M, Robie AA, Rivera-Alba M, Branson S, Branson K. JAABA: interactive machine learning for automatic annotation of animal behavior. Nat Methods 2013;10(1):64−7.
[36] Kausler BX, Schiegg M, Andres B, Lindner M, Koethe U, Leitte H, Wittbrodt J, Hufnagel L, Hamprecht FA. A discrete chain graph model for 3d+ t cell tracking with high misdetection robustness. In: European conference on computer vision. Springer; 2012. p. 144−57.
[37] Keller PJ, Schmidt AD, Santella A, Khairy K, Bao Z, Wittbrodt J, Stelzer EH. Fast, high-contrast imaging of animal development with scanned light sheet−based structured-illumination microscopy. Nat Methods 2010;7(8):637−42.
[38] Keller PJ, Schmidt AD, Wittbrodt J, Stelzer EHK. Reconstruction of zebrafish early embryonic development by scanned light sheet microscopy. Science 2008;322(5904):1065−9.
[39] Knoth P, Robotka V, Zdrahal Z. Connecting repositories in the open access domain using text mining and semantic data. In: Research and advanced technology for digital libraries, vol. 6966; 2011, ISBN 978-3-642-24468-1. p. 483−7. pages 483−487. Published in Lecture Notes in Computer Science, Volume 6966/2011.
[40] Krull A, Buchholz T-O, Jug F. Noise2void-learning denoising from single noisy images. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition; 2019. p. 2129−37.
[41] Krzic U, Gunther S, Saunders TE, Streichan SJ, Hufnagel L. Multiview light-sheet microscope for rapid in toto imaging. Nat Methods 2012;9(7):730−3.
[42] Kuhn HW. The Hungarian method for the assignment problem. Nav Res Logist Q 1955;2(1−2):83−97.
[43] Leggio B, Laussu J, Carlier A, Godin C, Lemaire P, Faure E. MorphoNet: an interactive online morphological browser to explore complex multi-scale data. Nat Commun 2019;10(1):1−8.
[44] Lehtinen J, Munkberg J, Hasselgren J, Laine S, Karras T, Aittala M, Aila T. Noise2noise: learning image restoration without clean data. 2018. arXiv preprint arXiv:1803.04189.
[45] Long F, Peng H, Liu X, Kim SK, Myers E. A 3D digital atlas of C. elegans and its application to single-cell analyses. Nat Methods 2009;6(9):667−72.
[46] Malin-Mayor C, Hirsch P, Guignard L, McDole K, Wan Y, Lemon WC, et al. Automated reconstruction of whole-embryo cell lineages by learning from sparse annotations. bioRxiv 2021. https://doi.org/10.1101/2021.07.28.454016.
[47] Maška M, Ulman V, Svoboda D, Matula P, Matula P, Ederra C, Urbiola A, España T, Venkatesan S, Balak DM, Karas P, Bolcková T, Štreitová M, Carthel C, Coraluppi S, Harder N, Rohr K, Magnusson KEG, Jaldén J, Blau HM, Dzyubachyk O, Křížek P, Hagen GM, Pastor-Escuredo D, Jimenez-Carretero D, Ledesma-Carbayo MJ, Muñoz-Barrutia A, Meijering E, Kozubek M, Ortiz-de Solorzano C. A benchmark for comparison of cell tracking algorithms. Bioinformatics 2014;30(11):1609−17.
[48] Matula P, Maška M, Sorokin DV, Matula P, Ortiz-de Solórzano C, Kozubek M. Cell tracking accuracy measurement based on comparison of acyclic oriented graphs. PLoS One 2015;10(12): e0144959.

[49] McDole K, Guignard L, Amat F, Berger A, Malandain G, Royer LA, Turaga SC, Branson K, Keller PJ. In toto imaging and reconstruction of post-implantation mouse development at the single-cell level. Cell 2018;175(3):859–76. e33.
[50] McQuin C, Goodman A, Chernyshev V, Kamentsky L, Cimini BA, Karhohs KW, Doan M, Ding L, Rafelski SM, Thirstrup D, Wiegraebe W, Singh S, Becker T, Caicedo JC, Carpenter AE. Cellprofiler 3.0: next-generation image processing for biology. PLoS Biol 2018;16(7):1–17.
[51] Merkel M, Etournay R, Popović M, Salbreux G, Eaton S, Jülicher F. Triangles bridge the scales: quantifying cellular contributions to tissue deformation. Phys Rev E 2017;95(3):032401.
[52] Moen E, Borba E, Miller G, Schwartz M, Bannon D, Koe N, et al. Accurate cell tracking and lineage construction in live-cell imaging experiments with deep learning. bioRxiv 2019. https://doi.org/10.1101/803205.
[53] Munro EM, Odell G. Morphogenetic pattern formation during ascidian notochord formation is regulative and highly robust. Development 2002;129(1):1–12.
[54] Olivier N, Luengo-Oroz MA, Duloquin L, Faure E, Savy T, Veilleux I, Solinas X, Débarre D, Bourgine P, Santos A, et al. Cell lineage reconstruction of early zebrafish embryos using label-free nonlinear microscopy. Science 2010;329(5994):967–71.
[55] Pietzsch T, Saalfeld S, Preibisch S, Tomancak P. BigDataViewer: visualization and processing for large image data sets. Nat Methods 2015;12(6):481–3.
[56] Roels J, Hennies J, Saeys Y, Philips W, Kreshuk A. Domain adaptive segmentation in volume electron microscopy imaging. In: 2019 IEEE 16th international symposium on biomedical imaging (ISBI 2019); 2019. p. 1519–22.
[57] Ronneberger O, Fischer P, Brox T. U-net: convolutional networks for biomedical image segmentation. CoRR 2015:234–41. abs/1505.04597.
[58] Santella A, Du Z, Nowotschin S, Hadjantonakis A-K, Bao Z. A hybrid blob-slice model for accurate and efficient detection of fluorescence labeled nuclei in 3D. BMC Bioinform 2010;11(1):1–13.
[59] Schiegg M, Hanslovsky P, Haubold C, Koethe U, Hufnagel L, Hamprecht FA. Graphical model for joint segmentation and tracking of multiple dividing cells. Bioinformatics 2015;31(6):948–56.
[60] Schiegg M, Hanslovsky P, Kausler BX, Hufnagel L, Hamprecht FA. Conservation tracking. In: Proceedings of the IEEE international conference on computer vision; 2013. p. 2928–35.
[61] Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, Pietzsch T, Preibisch S, Rueden C, Saalfeld S, Schmid B, Tinevez J-Y, White DJ, Hartenstein V, Eliceiri K, Tomancak P, Cardona A. Fiji: an open-source platform for biological-image analysis. Nat Methods 2012;9(7):676–82.
[62] Schindelin J, Rueden CT, Hiner MC, Eliceiri KW. The imagej ecosystem: an open platform for biomedical image analysis. Mol Reprod Dev 2015;82(7–8):518–29.
[63] Schmidt U, Weigert M, Broaddus C, Myers G. Cell detection with star-convex polygons. Lect Notes Comput Sci 2018;11071.
[64] Shah G, Thierbach K, Schmid B, Waschke J, Reade A, Hlawitschka M, Roeder I, Scherf N, Huisken J. Multi-scale imaging and analysis identify pan-embryo cell dynamics of germlayer formation in zebrafish. Nat Commun 2019;10(1):1–12.
[65] Sofroniew N, Lambert T, Evans K, Nunez-Iglesias J, Bokota G, Winston P, et al. napari/napari: 0.4.11. 2021.
[66] Stegmaier J, Amat F, Lemon WC, McDole K, Wan Y, Teodoro G, Mikut R, Keller PJ. Real-time three-dimensional cell segmentation in large-scale microscopy data of developing embryos. Dev Cell 2016;36(2):225–40.
[67] Stern T, Shvartsman SY, Wieschaus EF. Template-based mapping of dynamic motifs in tissue morphogenesis. PLoS Comput Biol 2020;16(8):e1008049.
[68] Stringer C, Wang T, Michaelos M, Pachitariu M. Cellpose: a generalist algorithm for cellular segmentation. Nat Methods 2021;18(1):100–6.
[69] Sugawara K, Cevrim C, Averof M. Tracking cell lineages in 3D by incremental deep learning. eLife 2022;11:e69380.
[70] Sulston JE, Schierenberg E, White JG, Thomson JN. The embryonic cell lineage of the nematode caenorhabditis elegans. Dev Biol 1983;100(1):64–119.
[71] Tinevez J-Y, Perry N, Schindelin J, Hoopes GM, Reynolds GD, Laplantine E, Bednarek SY, Shorte SL, Eliceiri KW. Trackmate: an open and extensible platform for single-particle tracking. Methods 2017;115:80–90. Image Processing for Biologists.
[72] Ulicna K, Vallardi G, Charras G, Lowe AR. Automated deep lineage tree analysis using a bayesian single cell tracking approach. Front Comput Sci 2021;3:92.
[73] Ulman V, Maška M, Magnusson KEG, Ronneberger O, Haubold C, Harder N, Matula P, Matula P, Svoboda D, Radojevic M, Smal I, Rohr K, Jaldén J, Blau HM, Dzyubachyk O, Lelieveldt B, Xiao P, Li Y, Cho S-Y, Dufour AC, Olivo-Marin J-C, Reyes-Aldasoro CC, Solis-Lemus JA, Bensch R, Brox T, Stegmaier J, Mikut R, Wolf S, Hamprecht FA, Esteves T, Quelhas P, Demirel Ö, Malmström L, Jug F, Tomancak P, Meijering E, Muñoz-Barrutia A, Kozubek M, Ortiz-de Solorzano C. An objective comparison of cell-tracking algorithms. Nat Methods 2017;14(12):1141–52.
[74] Villoutreix P, Delile J, Rizzi B, Duloquin L, Savy T, Bourgine P, Doursat R, Peyriéras N. An integrated modelling framework from cells to organism based on a cohort of digital embryos. Sci Rep 2016;6(1):1–11.
[75] von Chamier L, Laine RF, Jukkala J, Spahn C, Krentzel D, Nehme E, Lerche M, Hernández-Pérez S, Mattila PK, Karinou E, et al. Democratising deep learning for microscopy with zerocostdl4mic. Nat Commun 2021;12(1):1–18.
[76] Waschke J, Hlawitschka M, Anlas K, Trivedi V, Roeder I, Huisken J, et al. linus: Conveniently explore, share, and present large-scale biological trajectory data in a web browser. PLoS Comput Biol 2021;17(11):e1009503.
[77] Weigert M, Schmidt U, Boothe T, Müller A, Dibrov A, Jain A, Wilhelm B, Schmidt D, Broaddus C, Culley S, et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. Nat Methods 2018;15(12):1090–7.
[78] Weigert M, Schmidt U, Haase R, Sugawara K, Myers G. Star-convex polyhedra for 3d object detection and segmentation in microscopy. IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2020:3666–73.
[79] Wolff C, Tinevez J-Y, Pietzsch T, Stamataki E, Harich B, Guignard L, Preibisch S, Shorte S, Keller PJ, Tomancak P, Pavlopoulos A. Multi-view light-sheet imaging and tracking with the MaMuT software reveals the cell lineage of a direct developing arthropod limb. Elife 2018;7:e34410.
[80] Xiong F, Ma W, Hiscock TW, Mosaliganti KR, Tentner AR, Brakke KA, Rannou N, Gelas A, Souhait L, Swinburne IA, et al. Interplay of cell shape and division orientation promotes robust morphogenesis of developing epithelia. Cell 2014;159(2):415–27.
[81] Yao K, Rochman ND, Sun SX. Cell type classification and unsupervised morphological phenotyping from low-resolution images using deep learning. Sci Rep 2019;9(1):1–13.
[82] Yao K, Rochman ND, Sun SX. Ctrl–a label-free artificial intelligence method for dynamic measurement of single-cell volume. J Cell Sci 2020;133(7):jcs245050.