Quantitative Evaluation of Multicellular Movements in Drosophila Embryo

Perrine Paul-Gilloteaux, Sébastien Tosi

7
Quantitative Evaluation of Multicellular Movements in *Drosophila* Embryo

Perrine Paul-Gilloteaux[1,2] and Sébastien Tosi[3]

[1]Institut Curie, Centre de Recherche, Paris 75248, France
[2]Cell and Tissue Imaging Facility, PICT-IBiSA, CNRS, UMR 144, Paris 75248, France
[3]Institute for Research in Biomedicine (IRB Barcelona), Advanced Digital Microscopy, Parc Científic de Barcelona, c/Baldiri Reixac 10, 08028 Barcelona, Spain

7.1
Overview

7.1.1
Aim

In this chapter you will learn to track cell movements within the monolayer epithelium of a *Drosophila* embryo. The segmentation of the cells is performed by an ImageJ macro on microscope time lapses (movies) of the embryo presented in the maximum intensity projection (step 1). Another macro can optionally be used to discard weak cell–cell junctions that are likely to be segmentation errors (step 2). The tracking is then performed in Matlab from this binary stack (step 3). Finally, you will learn how to visualize the results of the cell tracking (step 4): the cell area evolution and the cell tracks.

7.1.2
Introduction

Quantitative information on the relative movement of cells and derived properties such as the evolution of cell areas and orientation are crucial to understand the organization of tissue and the fate of different cell subpopulations (Figure 7.1, left). The cell-based approach usually offers much information on the processes during tissue remodeling in comparison to estimating the velocity field performed by particle image velocimetry (PIV) [1,2] alone.

In this chapter, the cell tracking is performed in the apical plane of the tissue, so that the tissue is modeled as a polygon tiling. This is not just a mere simplification, but motivated by the fact that the strongest cell–cell junctions and most

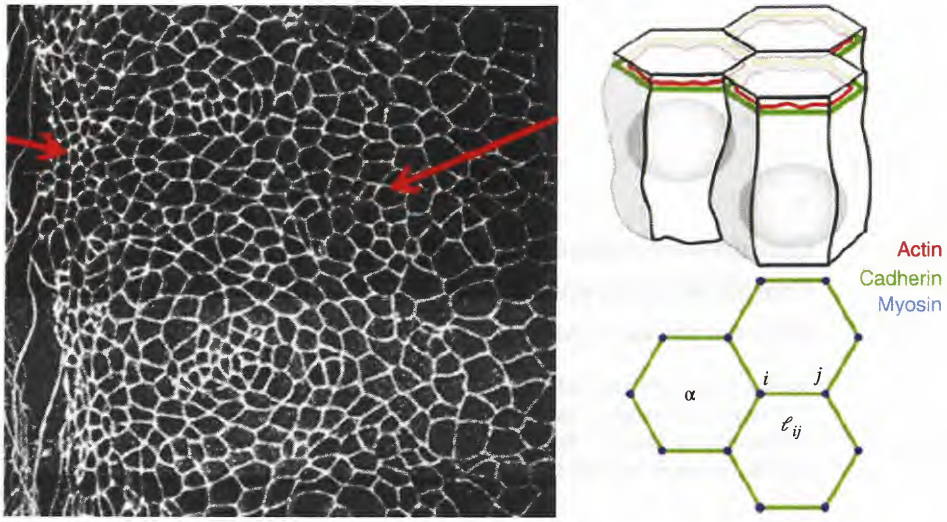

Figure 7.1 *Left:* In the apical plane of the tissue, the cell membrane appears as a polygon tiling. The red arrows point to the ventral (left) and dorsal (right) subregions. *Right:* Vertex model of cell shapes. (Figure taken from Ref. [3].)

of the cell internal shaping forces are believed to be exerted in the apical part of the tissue. For image analyses, the position of the cell vertices is an invaluable input to mechanical models, for example, the vertex model [3]. These models represent the epithelium as a 2D mechanical system with cells modeled as constant volume elastic polyhedrons sitting below the apical plane (Figure 7.1, right).

7.1.3
Data Sets

Data sets show the apical tissue of the embryo, which is imaged by a spinning disk microscope. The cell–cell junctions are labeled by a GFP-E-cadherin construct. Three time lapses of data are provided that show the maximum intensity projection of different tissue morphologies and dynamics. In this chapter we will work only on the *first time lapse*, the other movies can be used in the assignments.

7.2
Step 1: Cell Segmentation

The cell membrane appears brighter than the cytoplasm and, being the natural cell barrier, is highly suitable for cell segmentation. We will segment the cells in each time frame and generate a segmentation mask (binary image) for each

frame. The final result is a binary stack (one image per frame). Here, we focus on a workflow based on the watershed algorithm, another possible solution is provided in the appendix.

7.2.1 Workflow

Get Image Files

Open the time lapse "TissueMovie1.tif" in ImageJ.

Preprocessing

The purpose of the preprocessing is to smooth out the images to facilitate the segmentation. Filter the whole image stack with a Gaussian filter of radius 1.5 pixels: `Process > Filters > Gaussian Blur...`

Regional Minima Detection

The cells are segmented by finding internal starting points (ideally a single point close to the cell center) together with the region covered by the cell utilizing the watershed algorithm. The watershed can be conceptualized by coding pixel intensities as height: The cells then appear as basins separated by higher watershed lines. The starting points are intensity regional minima: a region surrounded by brighter pixels.

To apply the watershed, call `Process > Find Maxima...` with the option "Light Background" ticked (find minima instead of maxima). Choose a good value for "Noise Tolerance" (minimum relative height of surrounding bright barrier) to get rid of spurious minima and obtain exactly one regional minimum in most cells. For this, tick the "Preview" option to tune the noise tolerance.

Once you find a working value, choose the option "Segmented Particles" as output type: This will apply the watershed algorithm from the detected regional minima. This ImageJ command cannot process a whole stack at once, we will have to write a macro to generate the complete segmentation stack slice-by-slice.

Exercise 7.1: Writing a Macro to Segment the Cells in the Complete Stack

Record the sequence of operations that you manually performed in the previous section. Modify the code generated by the macro recorder in order to automatically process all the slices of the time lapse. The result should be a binary stack showing the segmented cells.

Hint 1: You should start by creating an 8-bit empty stack with the same dimensions as the original stack. Each slice of the original should then be processed by the previous sequence of operations.

Hint 2: You can select a slice of the active stack with the macro function setSlice(). A for-loop can be implemented to wander all the slices. The macro function nSlices returns the number of slices of the active stack.

Hint 3: To transfer the resulting segmentation mask to the empty image stack, you can use copy/paste option. At each iteration, ensure to process the correct slice of the original stack and copy it to the corresponding slice of the empty stack. The output stack should look as shown in Figure 7.2.

Use the macro you just wrote to process the first movie (a possible solution is provided in "Step1SampleCodeLoop.ijm"). To check the results of the segmentation, you can use `Image > Color > Merge Channels` to overlay the original stack (in gray) and the cell boundaries (e.g., in green) as shown in Figure 7.3.

Warning: To merge two images, they should have the same bit depth. If necessary, convert the original image to 8 bit.

As you can observe, it is quite likely that you do not get a perfect segmentation; some "false" cell junctions are sometimes created when several regional minima are detected in the same cell (inhomogeneous intensity inside the cell). This is often referred to as "oversegmentation" and can be mitigated by a proper preprocessing (filtering, background subtraction, etc.). In the next exercise we will implement a simple manual correction. Step 2 implements an algorithm to automatically discard weak junctions by measuring their mean intensity.

Note: Undersegmentation is the opposite phenomenon, it takes place when a single minimum is detected inside two neighbor cells (e.g., breach or weak cell junction). This is usually trickier to correct for; however, if cells can be assumed convex, then splitting lines can be drawn between concavities, for instance, by applying `Process > Binary > Watershed`.

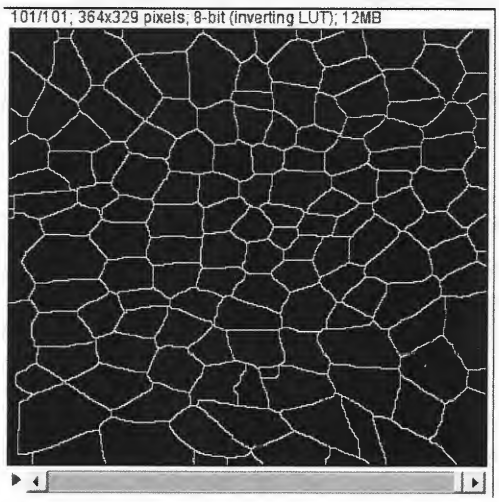

Figure 7.2 Output of the macro.

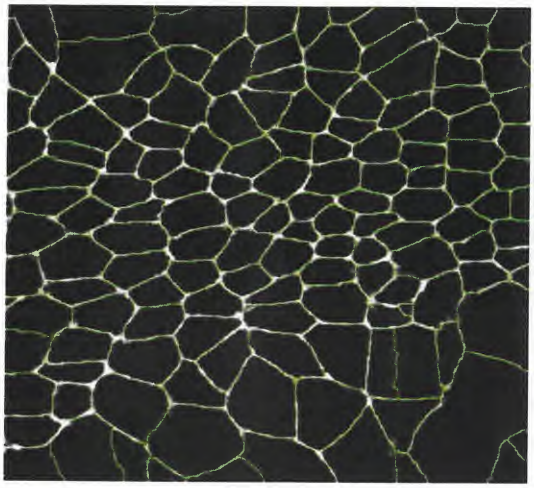

Figure 7.3 Channel merging of original stack (grayscale) and segmentation (green). Some "false" cell junctions have been created, due to the presence of several regional minima in the same cell.

Exercise 7.2: Understanding an Existing Macro

The ImageJ macro "Step1CellSegment.ijm" performs all the steps previously described and also enables the manual correction of the segmentation mask. Try it out and read the code!

The last section of the macro implements the manual correction. Try to understand how this user interaction is written and how the cell merging is implemented. Refer to the macro language documentation if you do not understand some macro functions.

Error detection and manual inspection/correction are fundamental steps of any automatic analysis. Perform a thorough manual correction of the *first frame* of the movie and save the binary stack to file as it will be used in steps 2 and 3.

Note: ImageJ segmentation masks by default are LUT inverted images so that the objects appear as black over a white background. When writing a macro, it is always a good idea to force this default behavior at initialization from `Process > Binary > Options...`. The options should appear as shown in Figure 7.4.

7.2.2 Summary of Tools Used

- `selectImage(ID)`: Activates the image with the specified ID (a negative number). If ID is greater than zero, it activates the IDth image listed in the Window menu. The ID can also be an image title (a string).

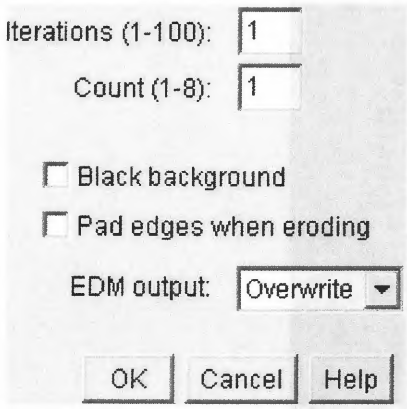

Figure 7.4 IJ default binary options.

- *Gaussian Blur* `[Process > Filters > Gaussian Blur]`: Smoothes out an image by performing convolution with a 2D Gaussian kernel of user-defined sigma. This helps reducing noise and disparities in the image, but smears out object edges and details.
- *Find Maxima* `[Process > Find Maxima]`: The command Find Maxima identifies regional maxima: A regional maximum is the highest intensity pixel that is surrounded by a closed valley of lower intensity pixels. The noise tolerance defines the minimal intensity difference between the regional maximum and the highest intensity of its surrounding valley.
- *Copy/Paste* `[Edit > Copy]`, `[Edit > Paste]`: The default behavior is to copy all the intensity values of the pixels inside a ROI to another (active) image. The command `Edit > Paste control` allows you to change the way the image is copied (e.g., the background can be transparently superimposed).
- *Merge Channels* `[Image > Color > Merge Channels...]`: Merging different channels allows you to simultaneously visualize them in the same hyperstack. It is possible to keep the original LUT of each channel when merging them.

The following are the useful macro functions:

- nSlices: Returns the number of slices/frames in the active stack.
- setSlice(): Change the position of active stack slice slider.

7.3 Step 2: Removal of Weak Segments (Optional)

We observed that oversegmentation can occur when several regional minima are detected in a nonhomogeneous cell. This situation can be revised since the

intensity along a "false" cell junction is likely to be weaker than for a "valid" cell junction. We are now going to measure the mean intensity along each cell junction in the original image and remove all cell junctions with mean intensity below an empirical value.

7.3.1 Workflow

The first task consists in detecting the cell junctions. To do so, we will first use the ImageJ command `Plugins > Skeleton > Skeletonize (2D/3D)` to enforce that the cell junctions are represented by a single pixel wide line.

Note: Ensure to invert the segmentation mask before skeletonization since now the objects of interest are not the cells but the cell junctions! Also note that skeletonize (2D/3D) interprets a stack as a 3D image: You should again process slice-by-slice (for loop).

Next call `Plugins > Skeleton > Analyze Skeleton (2D/3D)` on each skeleton image to code cell junctions and vertices (Figure 7.5) with a different intensity. The cell junctions are now easy to segment from the output of `Analyze Skeleton (2D/3D)`: They are connected particles with a gray value equal to 127.

In the last step of the macro, the detected cell junctions will be wandered (loop) for each image slice and classified as "valid" or "weak" based on their mean intensity in the original image.

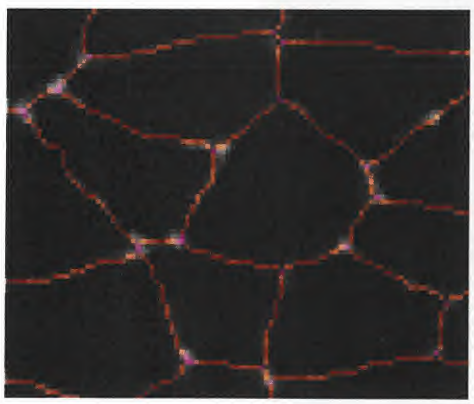

Figure 7.5 Overlay of the original image and the analyzed skeleton. The cell junctions appear in orange (gray level 127), the cell vertices in purple (gray level 70), and the end points in blue (gray level 30, invisible in this image).

Exercise 7.3: Modifying an Existing Macro

For this exercise, we provide a fully functional macro:

"Step2WeakJunctionRemoval.ijm": The original movie and the segmentation mask generated in step 1 both must be opened and respectively named "TissueMovie1.tif" and "ParticlesStack.tif" before launching the macro.

All the operations are by default performed in batch mode (no apparent windows) so that the intermediary steps are not shown. To better understand the sequence of operations, perform the following tasks:

- Comment the lines that are enabling and disabling batch mode: setBatchMode(true) and setBatchMode("exit and display")
- The segmented cell junctions are first added to the ROI manager before their intensity is measured. Add a pause (waitForUser) in the loop over the junctions, just after the ROI selection.
- Change the value of the variable JunctionThr and observe the effect (you can, for instance, use very low and very high values), try to optimize this value.

To compare the results before and after correction, you can again use `Image > Color > Merge Channels` to create a hyperstack with three channels: original image and masks before and after correction (Figure 7.6).

Save the corrected binary mask to file.

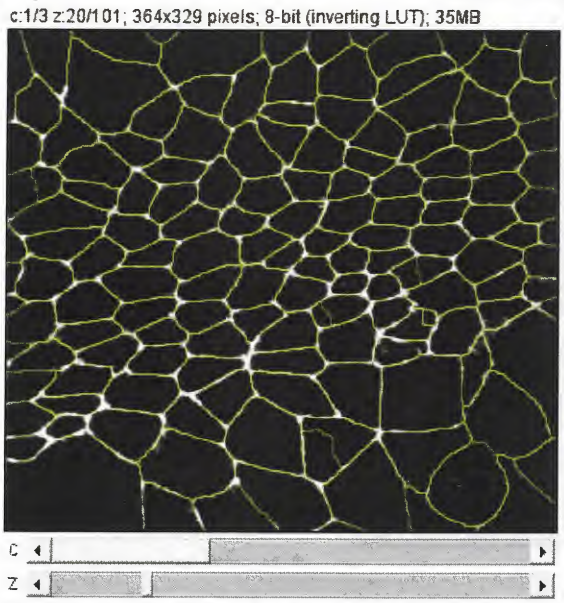

Figure 7.6 An example slice after cell segmentation correction. Removed junctions appear in green and kept junctions in yellow (red + green).

Exercise 7.4: Making the Macro More Generic

It is not very convenient to have the image names and weak segment threshold hard-coded in the macro. Display a dialog box at the beginning of the macro with three UI elements: two drop-down menus (one for the original movie and another for the binary mask) and one numerical field (junction threshold). Drop-down menu can be added with the command Dialog.addChoice and numerical field with the command Dialog.addNumber.

To fill the names of all opened images in the drop-down menus, you will have to first write a small loop over all the opened images to retrieve their titles and store them to an array of strings that will be passed as argument to Dialog.addChoice.

Hint: Even if this is usually not advised, it is possible to select an image by passing positive numerical values to the function selectImage, these indexes run from 0 to nImages-1 and correspond to the order of creation/opening of the images.

A solution to this exercise is the macro:
"Step2WeakJunctionRemoval_dialog.ijm."

Exercise 7.5: Advanced – Automated Threshold Selection

Try to automate the selection of the threshold used to discard the weak cell junctions (the solution is not provided).

Hint: The average intensities of the "valid" junctions can be assumed quite uniform. So a good strategy is to select the threshold as a fraction of the overall median of junction intensities (as long as "valid" junctions are in majority).

7.3.2 Summary of Tools Used

- *addNumber(label,default):* Adds a numeric field to the dialog, using the specified label and default value.
- *addChoice(label,item):* Adds a pop-up menu to the dialog, where items is a string array containing the menu items.
- *Skeletonize* – `Plugins > Skeleton > Skeletonize (2D/3D)`: It extracts the medial axis of each connected particle of a binary mask, this medial axis is also known as skeleton (points lying at exactly the same distance from two edges of a connected particle in the segmentation mask). Skeletonization is a morphological operation based on iterative erosions (see section on *Morphology* in Module 1). The command also provides an extensive report of the skeleton branch statistics and can optionally prune the shortest branches.
- *AnalyzeSkeleton* – `Plugins > Skeleton > AnalyzeSkeleton (2D/3D)`: This command introduced in Ref. [4] is part of the Fiji distribution. From a skeleton (graph-like binary image), it identifies the vertices (crossing of junctions) based

on the number of neighbors of each pixel. The junctions are coded with gray level 127, the vertices with gray level 70, and the end points with gray level 30.

- *waitForUser:* The command waitForUser("Message") interrupts the macro until the user presses the "OK" button. Interaction with ImageJ such as calls to internal commands or image selection/edition is possible. This behavior is different from the result as you invoke showMessage("message"); in this case, no interaction with ImageJ is allowed: ImageJ is "frozen" until the user presses "OK" (pressing "Cancel" interrupts the macro).

7.4 Step 3: Cell Tracking

In this section, we are going to track the cells in Matlab. The input of the script is the binary stack from step 1 (or step 2) and the output is a *label* mask with each cell filled by a unique gray level throughout the whole stack.

7.4.1 Introduction to Tracking

Most particle trackers such as ImageJ Particle Tracker 2D/3D and TrackMate are built for spot-like particles: The particle linking is performed over a list of detected spots. The linking can be straightforward, for example, linking a spot to the closest spot in the next frame, or more advanced as described in Refs [5,6].

These trackers are not adapted to nonspot-like (or ellipsoid) objects such as the cells of a tissue. A possible solution is to create a binary stack made of detected object centroids from a segmentation mask and to process this stack with the tracker. This is left as an exercise in the assignments.

Another strategy is to make use of the overlap between the particles in two consecutive frames. This will be illustrated in the following.

Exercise 7.6: Analyzing Connected Particles with Matlab

First we review a simple Matlab script importing a binary image stack, analyzing 3D connected particles, and exporting the resulting label mask stack to file.

Open the Matlab script "Step3ConnectedParticles3D.m." Before launching the script, set the variable BaseFolder to the folder where you saved the stack "DummyStack.tif." The script exports the label mask stack to a file "LabeledDummyStack.tif" to the same folder. After launching the script, open the original stack and the exported stack in ImageJ and go through the code to understand how it works.

Hint: The function bwconncomp allows analyzing connected particles in a *n*-dimensional image (here a 3D image). See Figure 7.7 for an illustration in two dimension.

Figure 7.7 Illustration of 2D connected particles in a 2D mask; in 3D the same principle holds but each voxel has 26 neighbors instead of 8.

Exercise 7.7: Tracking Cells with Matlab

The trick we will use is to interpret the binary stack we generated in step 1 as a 3D stack with time being the third dimension: The cells overlapping from frame to frame are now connected in three dimension (see Figure 7.8) !

In the remainder of the text, the word *particle label* refers to the label of a 2D connected particle (cell) in a given time frame (can be different for the same cell in two different frames). The word *object label* refers to the unique label of the cell along time (considered as a 3D object).

Exercise 7.7.1: Understanding the Matlab Script

Open the Matlab script "Step3TissueCellTrackv10_simple_incomplete.m." For now, do not launch the script but read the code and the functional block diagram provided in Figure 7.9. Try to associate each block with a section of the program.

Notes:

The loop that runs the code for each object is not included yet, do not try to find the associated code. Also, an operation to run the code for each frame is missing at this point (next exercise).

Image stack importation and exportation are very similar to the previous example.

The particles touching the edges are discarded to avoid tracking incomplete cells.

The separation between the cells has to be thick enough to avoid any merging of neighboring cells (in 3D), the cell junctions are enlarged by binary dilation (disk element of radius BoundDilate).

To close the top and bottom of the 3D particles, an empty image is added at the beginning and at the end of the movie: The binary stack passed to bwconncomp holds two slices more than the original stack.

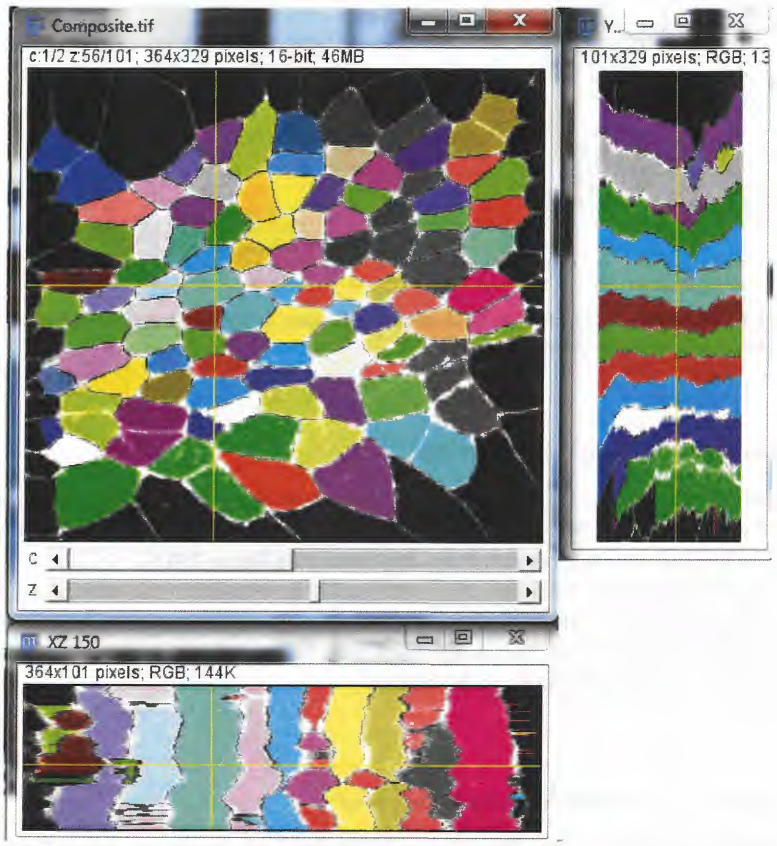

Figure 7.8 The cell overlap in time is apparent from orthogonal views when the movie is viewed as a 3D stack (`Image > Stacks > Orthogonal Views`).

The 3D object label mask (ObjLbl) is created by writing the labels slice-by-slice (second loop). It can also be created by the Matlab function labelmatrix as in the previous script. This is just to illustrate an alternative method.

The cell junctions are eroded back to their initial width at the end.

Exercise 7.7.2: Finding the Missing Operation

Before launching the script, set Basefolder to the folder where the input file (binary mask) is located. The variable fname holds the name of segmentation mask saved in step 1 (or step 2). After launching the script, inspect the exported results Tracking-obj1-10.tif in ImageJ. You will notice that the script is not working since an operation is missing: Identify it and add it to the code.

Hint: The missing operation can be performed by a single line of code. A solution to the exercise is given in

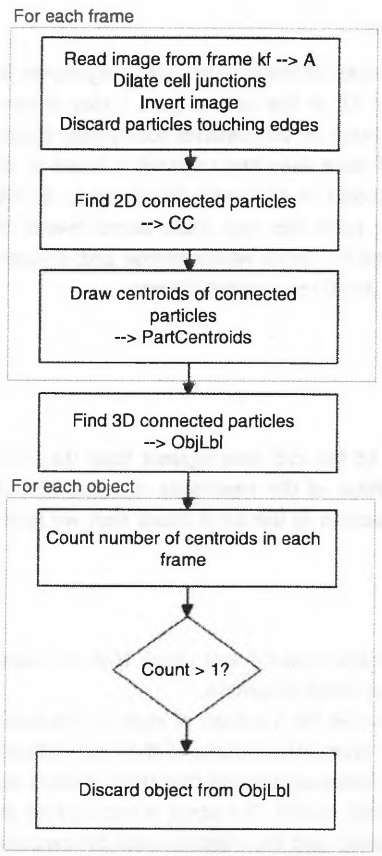

Figure 7.9 Workflow of the cell tracker (Algorithm 1 – TissueCellTrackv10).

"Step3TissueCellTrackv10_simple.m."

To better visualize the exported stack in ImageJ, you can apply the LUT Random.lut provided in the code folder. To do this, you first have to copy the LUT into the subfolder *luts* of ImageJ installation folder and call `Help > Refresh Menus`.

Note: The script also exports a stack holding the centroids of the 2D particles found in each frame. It will be used during the assignments to track the cell centroids.

This simple algorithm suffers from a severe limitation: Dividing or transiently merging cells (undetected cell junctions) are assigned the same label. In the next section, we will implement a procedure to detect these situations and automatically discard the problematic cells.

Exercise 7.7.3: Discarding Erroneous Objects

You can now open the script "Step3TissueCellTrackv10.m." This algorithm is described by the block diagram in Figure 7.9. In the second part, a loop is performed over each 3D object and the number of 2D particles (centroids) inside the object is computed for each frame. If more than one centroid is found in *at least one frame*, then the object is discarded in all frames (label set to 0). Of course, this is not ideal, but following the same idea one could detect events of merge and division and split the 3D object in frames where merge and division occur (set label to 0) before recomputing the 3D connected objects.

7.5
Step 4: Feature Extraction

In this section, we will plot the evolution of the cell area against time (i.e., the recorded frame) as well as track the position of the centroids of some of the cells. The input of all the scripts of this section is the label mask that we have computed (and saved to file) in step 3.

Exercise 7.8: Plotting Cell Area Evolution

Open the Matlab script "Step4PlotCellAreaExercise.m" and check that the base folder and file name are correctly set in the script preamble.

The script only displays the area evolution for a subset of cells of the label mask. The variable displayLbl is a vector taking the indices of the user-defined cells from which the information is to be extracted (ensure that these indices do not exceed the maximum value of the label mask!). The script is incomplete: In the loop over all the time frames, you should add the missing code to compute the areas of the cells of interest and store them in the variable Area(cellindex, frame).

Hint: Inside the loop over the time frames, you should write a loop over all the labels stored in displayLbl, compute the areas of each cell, and store this area to the matrix Area(for the correct cell, for the correct frame index). A way to compute the area of a cell at a given frame is to count how many pixels are set to the label of this cell in that particular frame.

The solution to the exercise is provided in "Step4PlotCellArea.m."

Exercise 7.9: Plotting Cell Centroid Tracks

Try now to retrieve the centroids of the cells in each frame with regionprops (Obj,'Centroid'). Store theses positions to two arrays: X(frame) and Y(frame) and plot the tracks to a 2D map. The solution to the exercise can be found in "Step4PlotCellAreaTrack.m."

7.5.1
Complete Track Plotting

A program plotting all the cell tracks (with start and end points labeled) and overlaying them to the first frame of the label mask is provided in "Step4Cell-TrackPlotter.m." The workflow of the program is described below and a possible output is illustrated in Figure 7.10 (without the label mask overlay).

First we define a minimal track length MinimalLengthofTrajectory to discard short tracks that are often erroneous.

In order to get the total number of objects, we find the last label index nbmax-cell (pixel with maximum value in the object label stack).

Then we create a vector of Matlab structures called *cells*. The vector is indexed by the object label. Each element of this structure array gathers two features of a given cell: the list of time frames where it is present and the list of its centroid positions in these frames. The structure is filled by the results returned by the Matlab function regionprops. Other features can be easily added to the structure array.

Finally, we plot the cell centroids for each time frame by sequential calls to *plot*. Only the tracks of length above MinimalLengthofTrajectory are plotted and the graphics are directed to the same canvas by calling *hold on*. The tracks are additionally decorated to mark their start and end points.

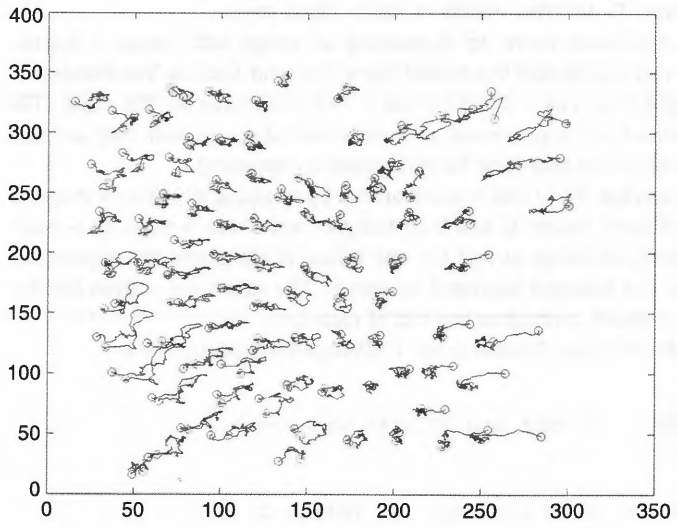

Figure 7.10 Example of cell tracks plotted for movie 1. Each blue line represents a cell trajectory, the first point of the trajectory is a red circle and the last point a green circle.

7.6 Assignments

If you intend to use the other two movies, you will notice that they suffer from a strong lateral drift; it is hence important to first register them, for instance, by applying `Plugins > Registration > StackReg` before the segmentation. This will allow a more efficient tracking in step 3.

- *Tracking of cell centroids:* In step 3 we generated a stack holding the cell centroids extracted from the segmentation mask. We can use this centroids stack as input to a spot-like particle tracking algorithm. In light microscopy, an infinitely small particle would never appear as a single isolated point but rather as a Gaussian-like shape (the point spread function of the microscope). This can easily be simulated by performing a 2D Gaussian blurring (radius around 1 pixel) of the centroid stack.

  After this filtering process, the centroid stack with `Plugins > Mosaic > Particle Tracker` or `Plugins > Tracking > TrackMate`. You should set the particle size to the minimum setting and prefer DoG (Difference of Gaussian) to LoG (Laplacian of Gaussian) detector in `TrackMate`.
- *PIV analysis*: To complement the analysis, we will now apply PIV to the same movies. The PIV is a procedure to estimate the velocity field of a time lapse at discrete positions and for each time frame. The PIV will be computed by ImageJ with `Analyze > Optic flow > PIV Analysis`. This function generates two images: The image U codes the X component of the velocity field, while the image V codes its Y component. We will export these two images to file, import them in Matlab, and visualize the velocity field (vector field) with the function *quiver*. To test this workflow, follow these steps:
  1) Create a two-frame movie by duplicating an image with `Image > Duplicate...` and slightly shift the second frame. Save this stack as "twoframes.tif."
  2) Call ImageJ `Analyze > Optic Flow > PIV analysis` on the stack. The displacements are represented as a color-coded image and they are also provided as two images (one for each speed component).
  3) Spatially average the U and V components by applying `Process > Smooth`.
  4) Save the filtered images U and V *as text files* with `File > Save As > Text`. When saving an image as text file, the values of the pixels are sequentially written as text (strings) separated by spaces. The values are written line-by-line with a return carriage at the end of each line.
  5) Launch the following Matlab script ("XAssignVisPivMatlab.m"):

```
1 % Folder where "U.txt" and "V.txt" are saved
2 Basefolder = '...\';
3
4 % Load the text files and copy the values to the
    matrices u and v
5 u = load([Basefolder,'U.txt']);
```

```
6  v = load([Basefolder,'V.txt']);
7
8  % Load first frame of the time-lapse
9  imagetissue = imread([Basefolder,'twoframes.tif'],1);
10
11 % Display the vector field overlaid on the first
       frame of the time-lapse
12 imshow(imagetissue,[]);
13 hold on;
14 quiver(v,u);
```

code/XAssignVisPivMatlab.m

The ImageJ macro "XAssignPIVCompute.ijm" performs the previous workflow. The script "XAssignVisuOutputPIV.m" is an advanced version of the previous Matlab script to visualize the vector field in Matlab. The output should look as in Figure 7.11.

- *Overlap-based cell tracking:* The tracking algorithm based on 3D connectivity is simple, but the results are not fully satisfactory as dividing and merging cells are discarded from the label mask. The algorithm proposed in this section will correctly track most of the cells up to a division/merging.

  Here the cell linking is performed iteratively and by processing pairs of consecutive time frames. The tracking starts from the first frame and proceeds through the time lapse, it is *extremely important* to check that the segmentation mask is valid in the *first frame.*

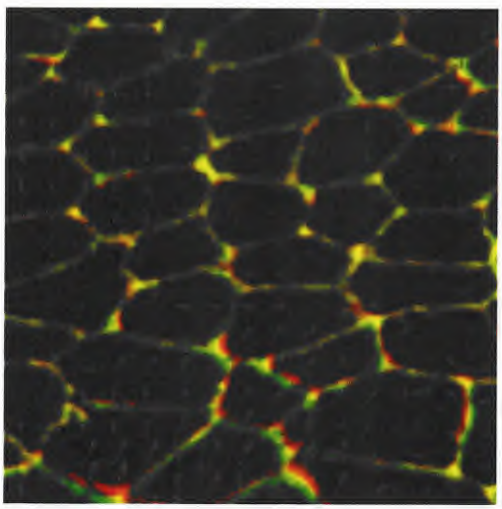

Figure 7.11 Example result of the PIV analysis. The first frame is green and the second frame is red. The displacements between first and second frames are indicated by green arrows.

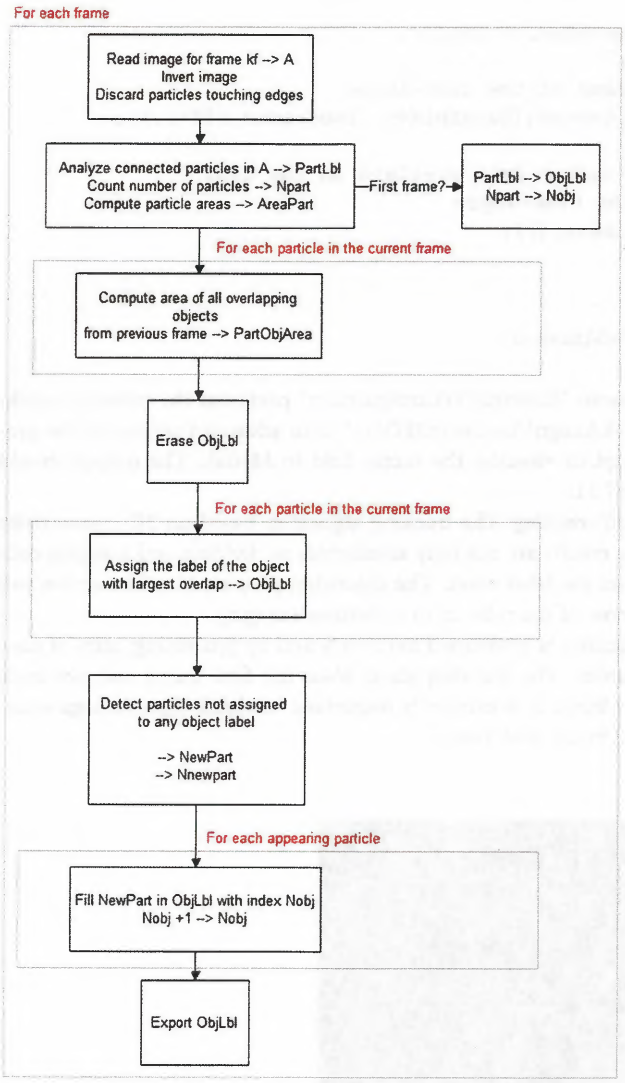

Figure 7.12 Workflow of the cell tracker (Algorithm 2 – TissueCellTrackv20).

Open the script "XAssignTissueCellTrackv20.m" and launch it after setting BaseFolder. A block diagram of the script is provided in Figure 7.12.

The object label mask (ObjLbl) is now built recursively: In the first frame, the 2D connected particles (PartLbl) are copied to the object label (ObjLbl) of the first frame. Particles here means connected objects in one frame, i.e candidate cells to be assigned to an object (i.e tracked cell). The 2D particles are then

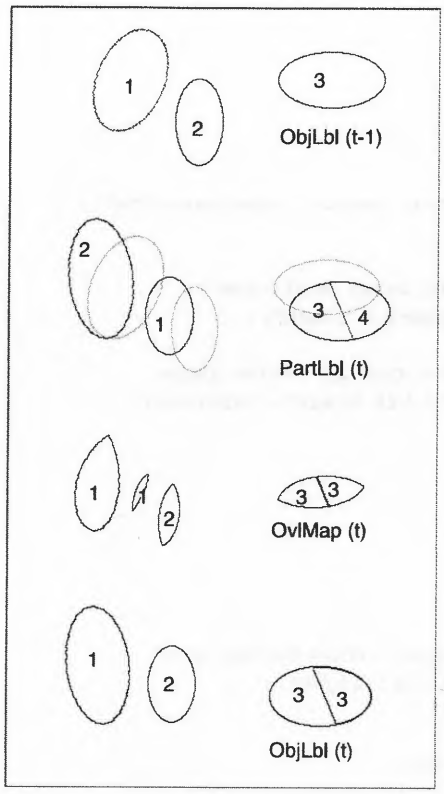

Figure 7.13 Steps involved to build the object label mask ObjLbl at time t from the object label mask at time t-1 and the particle label mask at time t. OvlMap(t) represents the overlap between the particles at time t and the objects at time t-1. The numbers represent the labels of the objects.

analyzed in the next frame t (PartLbl t) and their overlap with the object(s) in frame t-1 (ObjLbl t-1) are computed. Each particle is then assigned the most likely object label from the previous frame (maximum overlap), this label is written to the object label mask of the current frame (ObjLbl t). The process is then iterated until the last frame of the movie.

The label masks are illustrated for a simple case in Figure 7.13. It can be clearly understood that if there is no overlap between the same cell in two consecutive frames (large drift or fast movement), the linking will fail. Two other problems arise if the largest overlap does not take place between the correct pair of cells or if two cells get transiently merged.

Try to modify the script "XAssignTissueCellTrackv20.m" so that a particle is assigned the label of the object with largest overlap *only* if this overlap is larger than a user-defined area threshold (e.g., 70% of the area of the particle). The solution to the exercise is found in "XAssignTissueCellTrackv20_solution.m."

Solutions to the Exercises

Exercise 7.1

```
// Initialization
NoiseTol = 5;
run("Options...", "iterations=1 count=1 edm=Overwrite");

// Filter input image
run("Duplicate...", "title=Filtered duplicate");
run("Gaussian Blur...", "sigma=1.5 stack");

// Create empty stack of same size as active image
newImage("ParticlesStack", "8-bit Black", getWidth(),
     getHeight(), nSlices);
run("Invert LUT");

// Main loop
for(i=1;i<=nSlices;i++)
{
  selectImage("Filtered");
  setSlice(i);
  run("Find Maxima...", "noise="+d2s(NoiseTol,2)+"
     output=[Segmented Particles] light");
  rename("Particles");
  run("Copy");
  selectImage("ParticlesStack");
  setSlice(i);
  run("Paste");
  selectImage("Particles");
  close();
}
run("Select None");
```

"code/Step1SampleCodeLoop.ijm"

Exercise 7.2.3: The Code Should Be Opened from the Provided Material ("Step1CellSegment.ijm" and "Step2WeakJuctionRemoval.ijm")

Exercise 7.4

```
//Parameters
if(nImages<2)exit("At least two images should be opned!");
ImageNames = newArray(nImages);
for (i=0; i < nImages; i++)
{
  selectImage(i+1);
  ImageNames[i] = getTitle();
```

```
}
Dialog.create("Select Images");
Dialog.addChoice("Original Movie", ImageNames, ImageNames[0]);
Dialog.addChoice("Binary Mask", ImageNames, ImageNames[1]);
Dialog.addNumber("Threshold", 50);
Dialog.show();
OriginalMovie = Dialog.getChoice();
SegmentedCells = Dialog.getChoice();
JunctionThr = Dialog.getNumber();

// Initialization
run("Options...", "iterations=1 count=1 edm=Overwrite
   do=Nothing");
run("Set Measurements...", "  mean redirect="+OriginalMovie+"
   decimal=2");
setBatchMode(true);
newImage("CorrectedMask", "8-bit Black", getWidth(),
   getHeight(), nSlices);

// Main loop over the slices
for(s=1;s<=nSlices;s++)

{

  // Set current slice in all stacks
  selectImage("CorrectedMask");
  setSlice(s);
  selectImage(OriginalMovie);
  setSlice(s);
  selectImage(SegmentedCells);
  setSlice(s);

  // Skeletonize and identify junctions / vertices
  run("Duplicate...", "title=Copy");
  run("Invert", "slice");
  CopyID = getImageID();
  run("Skeletonize (2D/3D)");

  run("Analyze Skeleton (2D/3D)", "prune=none prune");
  AnalyzedSkeletonID = getImageID();

  // Add all junctions to ROI manager and measure mean
      intensity in original stack
  setThreshold(100, 255);
 // Threshold junctions: vertices value: 70, junctions value: 127
  run("Analyze Particles...", "size=0-Infinity circularity=0.00-
     1.00 show=Nothing display clear add");
  roiManager("Show None");
```

```
51
52    // Select non null pixels in the skeleton --> mask
53    selectImage(AnalyzedSkeletonID);
54    setThreshold(1, 255);
55    run("Convert to Mask", "method=Default background=Dark
         black");
56    run("Invert LUT");
57    resetThreshold();
58
59    // Erase the weak junctions
60    N = roiManager("count");
61    for (i=0;i<N;i++)
62    {
63      if(getResult("Mean",i)<JunctionThr)
64      {
65        roiManager("Select", i);
66        run("Set...", "value=0 slice");
67      }
68    }
69
70    // Copy to CorrectedMask
71    run("Select All");
72    run("Copy");
73    selectImage("CorrectedMask");
74    run("Paste");
75
76    // Cleanup
77    selectImage(AnalyzedSkeletonID);
78    run("Close");
79    selectImage(CopyID);
80    run("Close");
81  }
82
83  // Exit
84  selectImage("CorrectedMask");
85  run("Invert", "stack");
86  run("Invert LUT");
87  run("Select None");
88  setBatchMode("exit & display");
89  run("Set Measurements...", "  mean redirect=None decimal=2");
```

"code/Step2WeakJunctionRemoval_dialog.ijm"

Exercise 7.6

```
1  BaseFolder = '...\';
2  fname = strcat(BaseFolder,'DummyStack.tif');
3  expfname = strcat(BaseFolder,'LabeledDummyStack.tif');
```

```
4
5  % Gather ParticleStack image info
6  info = imfinfo(fname);
7  num_images = numel(info);
8  Height = info(1).Height;
9  Width = info(1).Width;
10
11 % Initialize buffer image
12 DummyStack = zeros(Height,Width,num_images);
13
14 % Read images (loop over frames)
15 for kf = 1:num_images
16     DummyStack(:,:,kf) = imread(fname, kf, 'Info', info);
17 end
18
19 % Find connected particles (3D)
20 CC = bwconncomp(DummyStack);
21
22 % Generate label mask
23 L = labelmatrix(CC);
24
25 % Erase output file (if exists)
26 if exist(expfname, 'file')
27     delete(expfname);
28 end
29
30 for kf = 1:num_images
31     imwrite(uint16(L(:,:,kf)), expfname, 'WriteMode', 'append',
          'Compression','none');
32 end
```

"code/Step3ConnectedParticles3D.m"

Exercise 7.7

```
1  %% Initialization
2  clear all;
3  close all;
4  BaseFolder = '...\';
5  fname = strcat(BaseFolder,'ParticlesStack.tif'); %
       Segmentation mask
6  expfnameobj = strcat(BaseFolder,'Tracking-obj1-10.tif'); %
       Object labels
7  expfnamepartcentroids = strcat(BaseFolder,'Tracking-
       Centroids-v-10.tif'); % Centroid stack
8
9  %% Parameters
10 startframe = 1;
```

```
BoundDilate = 5;

%% Gather ParticleStack image info
info = imfinfo(fname);
num_images = numel(info);
Height = info(1).Height;
Width = info(1).Width;

%% Create image buffers with same size of the original image
     (+ 2slices)
ImageCopy = zeros(Height,Width,num_images+2);
ObjLbl = zeros(Height,Width,num_images+2);
PartCentroids = uint8(zeros(Height,Width,num_images+2));

%% Loop over image frames (2D processing)
for kf = 1:num_images
    disp(kf);

    % Import mask for this frame
  A = imread(fname, kf, 'Info', info);

    % Dilate cell junctions
  A = imdilate(A, strel('disk',BoundDilate));

  % Process mask and analyze connected particles (2D)
  A = 255-A; % Mask coming from ImageJ (inverted LUT)
    A = imclearborder(A); % Remove objects touching the image
       borders

    % Store current frame to ObjLbl
    ImageCopy(:,:,kf+1) = A;

    % Find connected particles in current frame and compute
       their centroids
    CC = bwconncomp(A);
  centroids = regionprops(CC,'centroid');

  % Draw centroids of connected particles
  for l = 1:length(centroids)
    centroidl = centroids(l).Centroid;
        PartCentroids(round(centroidl(2)),round(centroidl(1)),
          kf+1) = 1;
    end

end

%% Analyze connected particles (3D) to identify objects and
     fill label mask
```

```
objList = bwconncomp(ImageCopy);
Nobj = objList.NumObjects;
for label = 1:Nobj
   ObjLbl(objList.PixelIdxList{label}) = label;
end

%% Images post-processing and exportation

% Erase output file (if exists)
if exist(expfnameobj, 'file')
    delete(expfnameobj);
end

% Erode the cell junctions and export the object label mask
for frame=2:num_images+1
     ObjLbl2D = imdilate(ObjLbl(:,:,frame),strel('disk',
         BoundDilate));
     imwrite(uint16(ObjLbl2D), expfnameobj, 'WriteMode',
         'append',  'Compression','none');
end

% Erase centroid stack file (if exists)
if exist(expfnamepartcentroids, 'file')
    delete(expfnamepartcentroids);
end

% Export the centroid stack
for frame = 2:num_images+1
  imwrite(uint8(PartCentroids(:,:,frame)),
    expfnamepartcentroids, 'WriteMode', 'append',
    'Compression','none');
end
```

"code/Step3TissueCellTrackv10_simple.m"

Exercise 7.8

```
clear all;
close all;
startframe = 1;
displayLbl = [54 57];
BaseFolder = '...\';
fname = strcat(BaseFolder,'Tracking-obj1-10.tif'); %
     Exported stack (object)

%%%% Read the object label mask (same label = same
     cell over time)%%%%%%%
info = imfinfo(fname);
```

```
num_images = numel(info);
endframe = num_images;

% get information from last frame
A = imread(fname, endframe, 'Info', info);

% Find highest label: total number of cells
nbmaxcell = max(max(A));

% Initialize arrays
Area = nan(nbmaxcell,endframe);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for kf = startframe:endframe
   A = imread(fname, kf, 'Info', info);
   for Lbl=displayLbl
       Obj = (A==Lbl);
       Area(Lbl,kf) = sum(sum(Obj));
   end
end
plot(startframe:endframe,Area(displayLbl,:));
```

"code/Step4PlotCellArea.m"

Exercise 7.9

```
clear all;
close all;
startframe = 1;
displayLbl = [54 57];
BaseFolder = '...\';
fname = strcat(BaseFolder,'Tracking-obj1-10.tif');  %
     Exported stack (object)

%%%% Read the object label mask (same label = same
     cell over time)%%%%%%%
info = imfinfo(fname);
num_images = numel(info);
endframe = num_images;

% get information from last frame
A = imread(fname, endframe, 'Info', info);

% Find highest label: total number of cells
nbmaxcell = max(max(A));

% Initialize arrays
```

```
Area = nan(nbmaxcell,endframe);
X = nan(nbmaxcell,endframe);
Y = nan(nbmaxcell,endframe);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for kf = startframe:endframe
   A = imread(fname, kf, 'Info', info);
   for Lbl=displayLbl
       Obj = (A==Lbl);
       Area(Lbl,kf) = sum(sum(Obj));
       stats = regionprops(Obj,'Centroid');
       stats = cat(1,stats.Centroid);
       if(~isempty(stats))
        X(Lbl,kf) = stats(1,1);
        Y(Lbl,kf) = stats(1,2);
       end
   end
end
plot(startframe:endframe,Area(displayLbl,:));
figure;
plot(X(displayLbl,:).',Y(displayLbl,:).');
```

"code/Step4PlotCellAreaTrack.m"

7.7 Appendix

In the alternative workflow "XAssignMacroSegmentationalternative.ijm," the cells are segmented by video inverting the fluorescence signal so that they appear as bright on a dark background. A step of background subtraction facilitates thresholding and then a binary watershed is applied with an intent to split the merged cells.

Acknowledgment

We thank Annalisa Letiza (IBMB-CSIC) for sharing the spinning disk movies. Sébastien Tosi collaborated with her to design the automated tissue analysis tools that she used in her own research.

References

1 Desprat, N., Supatto, W., Pouille, P.A., Beaurepaire, E., and Farge, E. (2008) Tissue deformation modulates twist expression to determine anterior midgut differentiation in *Drosophila* embryos. *Dev. Cell*, **15** (3), 470–477.

2 Petitjean, L., Reffay, M., Grasland-Mongrain, E., Poujade, M., Ladoux, B.,

Buguin, A., and Siberzan, P. (2010) Velocity fields in a collectively migrating epithelium. *Biophys. J.*, **98** (9), 1790–1800.
3 Farhadifar, R., Röper, J.C., Aiuluy, B., Eaton, S., and Jüllicher, F. (2007) The influence of cell mechanics, cell–cell interactions, and proliferation on epithelial packing. *Curr. Biol.*, **17** (24), 2095–2104.
4 Arganda-Carreras, I., Fernandez-Gonzalez, R., Munoz-Barrutia, A., and Ortiz-De-Solorzano., C. (2010) 3D reconstruction of histological sections: Application to mammary gland tissue. *Microsc. Res. Tech.*, **73** (11), 1019–1029.
5 Jaqaman, K., Loerke, D., Mettlen, M., Kuwata, H., Grinstein, S., Schmid, S.L., and Danuser, G. (2008) Robust single-particle tracking in live-cell time-lapse sequences. *Nat. Methods*, **5** (8), 695–702.
6 Sbalzarini, I.F. and Koumoutsakos, P. (2005) Feature point tracking and trajectory analysis for video imaging in cell biology. *J. Struct. Biol.*, **151** (2), 182–195.