Review of Cellpose

Xiangyu Xie

I. Concepts

Flow representations
dynamics about the flows
mask
ROI: region of interest
manually annotated cells

II. Track mate

A. Input file

AVI file with 8 bit or 16 bit. These terms refer to the bit depth of the image, which is a measure of the number of bits used to represent each pixel in the image.

8-bit Images:

Each pixel in an 8-bit image is represented by 8 bits (1 byte), which means each pixel can have one of 256 different intensity or gray levels (from 0 to 255).

8-bit images are often used for grayscale images where this range of intensities is sufficient to represent the image accurately.

16-bit Images:

In 16-bit images, each pixel is represented by 16 bits, allowing for 65,536 different intensity levels (from 0 to 65,535). This higher bit depth is beneficial for scientific images, especially when a greater range of intensity data is required.

For the mp4 file, we need to download the FFMPEG plugin and click Import and Movie(FFMPEG) to load the mp4 file. Then it will generate a series figures based on the input video. And we need to transform these figures into 8 bit or 16 bit type since Trackmate does not support JPG file. Just click type and 8 bit or 16 bit.

Then we Just need to click Plugins-Tracking-Trackmate to load the Trackmate Plugins.

This Plugin comes from the paper[1].

B. Detector algorithms

LoG detector:

This detector applies a LoG (Laplacian of Gaussian) filter to the image, with a sigma suited to the blob estimated size. Calculations are made in the Fourier space. The maxima in the filtered image are searched for, and maxima too close from each other are suppressed. A quadratic fitting scheme allows to do sub-pixel localization.

Estimated object diameter

Quality threshold

DoG detector:

This detector is based on an approximation of the LoG operator by differences of Gaussian (DoG). Computations are made in direct space. It is the quickest for small spot sizes (5 pixels). Spots found too close are suppressed. This detector can do sub-pixel localization of spots using a quadratic fitting scheme. It is based on the scale-space framework made by Stephan Preibisch for ImgLib.

Hessian detector:

This detector is based on computing the determinant of the Hessian matrix of the image to detector bright blobs. It can be configured with a different spots size in XY and Z. It can also return a normalized quality value, scaled from 0 to 1 for the spots of each time-point. As discussed in Mikolajczyk et al.(2005), this detector has a better edge response elimination than the LoG detector and is suitable for detect spots in images with many strong edges.

Label Image detector:

This detector creates spots by importing regions from a label image. A label image is an image where the pixel values are integers. Each object in a label image is represented by a single common pixel value (the label) that is unique to the object. This detector reads such an image and create spots from each object. In 2D the contour of a label is imported. In 3D, spherical spots of the same volume that the label are created. The spot quality stores the object area or volume in pixels.

Mask detector:

This detector creates spots from a black and white mask. More precisely, all the pixels in the designated channel that have a value strictly larger than 0 are considered as part of the foreground, and used to build connected regions. In 2D, spots are created with the (possibly simplified) contour of the region. In 3D, a spherical spot is created for each region in its center, with a volume equal to the region volume. The spot quality stores the object area or volume in pixels.

Thresholding detector:

This detector creates spots by thresholding a grayscale image. Pixels in the designated channel that have a value larger than the threshold are considered as part of the foreground, and used to build connected regions. In 2D, spots are created with the (possibly simplified) contour of the region. In 3D, a spherical spot is created for each region in its center, with a volume equal to the region volume. The spot quality stores the object area or volume in pixels.

C. Some definitions

Jonker-Volgenant solver: The Jonker-Volgenant solver is an algorithm used for solving the Linear Assignment Problem (LAP), an optimization problem where the goal is to find the best assignment of a set of agents to a set of tasks with minimum total cost. The Jonker-Volgenant algorithm is known for its efficiency, especially in handling large-scale assignment problems.

This solver is particularly effective because it improves upon the traditional Hungarian algorithm for LAP, offering enhanced performance in terms of computational time and memory usage. It is commonly employed in various applications, including in particle tracking systems like the one you mentioned, where it helps to efficiently match particles across frames based on a cost matrix.

gap-closing (when a particle temporarily disappears), splitting (one track diverging into multiple tracks), and merging (multiple tracks converging into one).

Gap-Closing: Refers to bridging the periods when a particle temporarily disappears from view (due to factors like moving out of focus or being obscured) and then reappears. The tracking algorithm predicts where the particle might be and resumes tracking when it reappears.

Splitting: Occurs when one track diverges into multiple tracks. This can happen if an object splits into parts or if two closely moving objects are initially detected as one.

Merging: Involves multiple tracks converging into one, typically when separate objects come together and are identified as a single entity by the tracking system.

D. Tracking algorithms

TrackMate ships three classes of particle-linking algorithms. A first class is derived from the LAP framework proposed by Jaqaman and colleagues [2]. Base linking costs are calculated from the square distance between particles, which makes it ideal to tackle Brownian motion. However, costs can be modulated by feature value differences, penalizing the linking of particles that are different in intensity distribution, rough shape, etc. A convenient GUI allows tuning these costs directly in TrackMate.

A second particle-algorithm relies on the Kalman filter[3] to tackle linear motion. Finally, particle linking based on nearest-neighbor search is proposed as the simplest linking algorithm.

LAP tracker:

This tracker is based on the Linear Assignment Problem mathematical framework. Its implementation is adapted from the paper [2].

LAP is formulated as an optimization problem, where the goal is to minimize (or maximize) a cost function. This cost function is based on the distances or differences between elements in the two sets.

In particle tracking, costs are calculated for each possible pair of particles between two consecutive frames. These costs usually represent the spatial distance between particles, but they can also include other features like intensity, shape, etc.

Solving the LAP relies on the Jonker-Volgenant solver, and a sparse cost matrix formulation, allowing it to handle very large problems.

Furthermore, TrackMate comes with a user-friendly graphical user interface (GUI). This GUI allows users to conveniently tune and adjust the linking costs based on the specific requirements of their experiments. This level of customization is essential for researchers working with diverse types of biological data, as it enables them to optimize the tracking algorithm for their specific needs.

Simple LAP tracker:

This tracker is identical to the sparse LAP tracker present in this trackmate, except that it proposes fewer tuning options. Namely, only gap closing is allowed, based solely on a distance and time condition. Track splitting and merging are not allowed, resulting in having non-branching tracks.

In LAP tracker, the track segments are further analyzed for complex events like gap-closing (when a particle temporarily disappears), splitting (one

track diverging into multiple tracks), and merging (multiple tracks converging into one).

Simple LAP tracker focuses solely on closing gaps in particle tracks, which is based on distance and time conditions. This is useful for handling instances where particles disappear and reappear in the imaging sequence.

Kalman tracker:

Kalman Filter: This is a predictive algorithm used to estimate the future position of each tracked object (spot) based on its current and past states. The Kalman filter can adjust its predictions to account for moderate changes in the object's speed and direction.

Use of LAP Framework: The predictions made by the Kalman filter for each object's next position are then matched to actual detected positions in the subsequent frame using the LAP framework. This matching process ensures that each object is consistently tracked across frames.

Gap Bridging: If an object is not detected where the Kalman filter predicts it should be, the tracker doesn't immediately lose the track. Instead, the Kalman filter makes another prediction for the next frame, continuing the search for the object. This allows the tracker to handle situations where objects temporarily disappear from view.

Importance of Initial Frames: The tracker's ability to accurately predict an object's movement is heavily dependent on the data from the first few frames. The initial positions and movements help establish the base state for the Kalman filter's predictions. Spurious detections or inaccuracies in these early frames can lead to incorrect tracking.

Parameters: Max Search Radius: This defines the area around the predicted position within which the tracker will search for the actual spot. Initial Search Radius: This parameter determines how far apart two detected spots can be for the tracker to consider starting a new track.

Advanced Kalman tracker:

different features values

This tracker is an extended version of the Kalman tracker, that adds the possibility to customize linking costs and detect track fusion (segments merging) and track division (segments splitting).

This tracker is especially well suited to objects that move following a nearly constant velocity vector. The velocity vectors of each object can be completely different from one another. But for the velocity vector of one object need not to change too much from one frame to another.

In the frame-to-frame linking step, the classic Kalman tracker infer most likely spot positions in the target frame from growing tracks and link all extrapolated positions against all spots in the target frame, based on the square distance.

This advanced version of the tracker allows for penalizing links to spots with different features values using the same framework that of the LAP tracker in TrackMate. Also, after the frame-to-frame linking step, track segments are post-processed to detect splitting and merging events, and perform gap-closing. This is again based on the LAP tracker implementation.

Overlap tracker:

This tracker is a simple extension of the Intersection - over - Union (IoU) tracker.

It generates links between spots whose shapes overlap between consecutive frames. When several spots are eligible as a source for a target, the one with the largest IoU is chosen.

The minimal IoU parameter sets a threshold below which links won't be created. The scale factor allows for enlarging (¿1) or shrinking (¡1) the spot shapes before computing their IoU. Two methods can be used to compute IoU: The Fast one approximates the spot shapes by their rectangular bounding-box. The Precise one uses the actual spot polygon.

Careful: this tracker is only suited to 2D images. It treats all the spots as 2D objects. The Z dimension is ignored.

Nearest-neighbor tracker:

This tracker is the most simple one, and is based on nearest neighbor search. The spots in the target frame are searched for the nearest neighbor of each spot in the source frame. If the spots found are closer than the maximal allowed distance, a link between the two is created. The nearest neighbor search relies upon the KD-tree technique implemented in imglib by Johannes Schindelin and friends. This ensure a very efficient tracking and makes this tracker suitable for situation where a huge number of particles are to be tracked over a very large number of frames. However, because of the naiveness of its principles, it can result in pathological tracks. It can only do frame-to-frame linking; there cannot be any track merging

or splitting, and gaps will not be closed. Also, the end results are non-deterministic.

III. Experimental results

I've discovered a plugin named Trackmate-Cellpose for ImageJ. This plugin allows us to implement the Cellpose detector algorithm within the Trackmate Plugin. I've included a figure below for reference.

I set the cell diamater as 20 pixels and detection model cyto 2.

I seems to take a lot of time detecting the cells by Trackmate-cellpose plugin. It may depend on the support of GPU. The tracking algorithms are very fast.

Here, I only attached the tracking results of 10 frames.

The first frame is shown in the following figure.

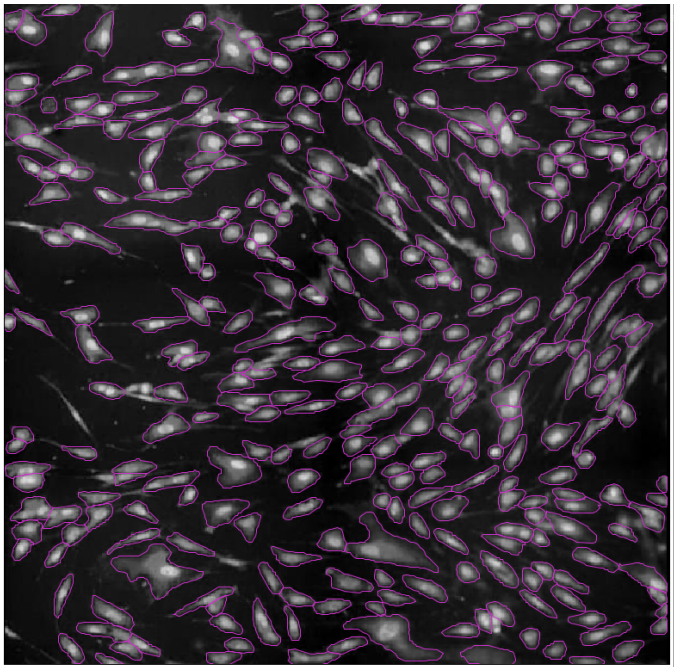

Fig. 1: The first frame

And the tracking results are shown as below:

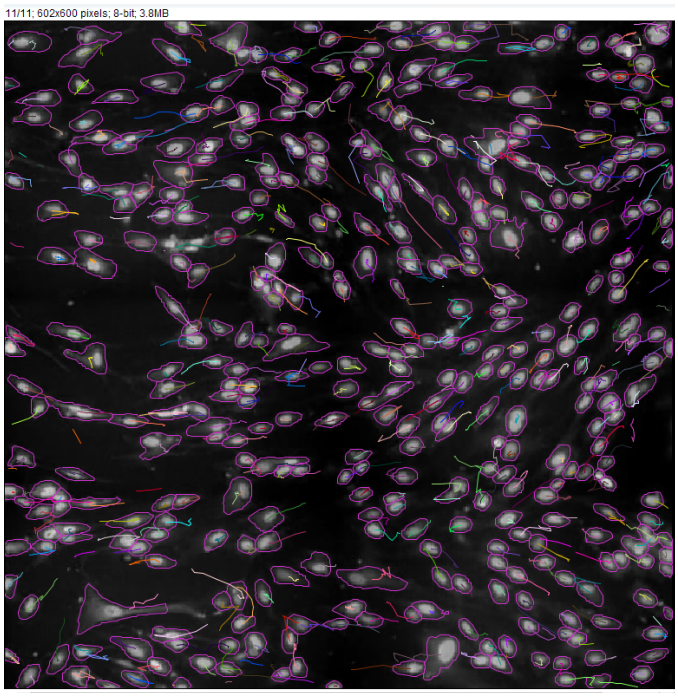

Fig. 2: the tracking results

IV. Robust single-particle tracking in live-cell time-lapse sequences

This paper generated particle tracks in two steps (Fig. 3)

First, we constructed track segments by linking the detected particles between consecutive frames, under the condition that a particle in one frame could link to at most one particle in the previous or the following frame. These track segments started and ended not only because of the true appearance and disappearance of particles, but also because of apparent disappearances owing to noise and limitations in imaging, for example, when a particle temporarily moved out of the plane in focus or when two particles approached each other within a distance smaller than the resolution limit.

The track segments obtained in this step tended to be incomplete, resulting in a systematic underestimation of particle lifetimes. In addition, because of the one-to-one assignment of particles, this step could not capture particle merges and splits, which by definition required one particle in one frame to be assigned to two particles in the previous or subsequent frame, respectively.

Therefore, in a second step, we linked the initial track segments in three ways: (i) end to start, to close gaps resulting from temporary disappearance, (ii) end to middle, to capture merging events, and (iii) start to middle, to capture splitting events.

End to Start: This method connects the end of one track segment to the start of another. It's used to bridge gaps that might occur when the object or entity being tracked temporarily disappears from observation and then reappears, creating disjoint

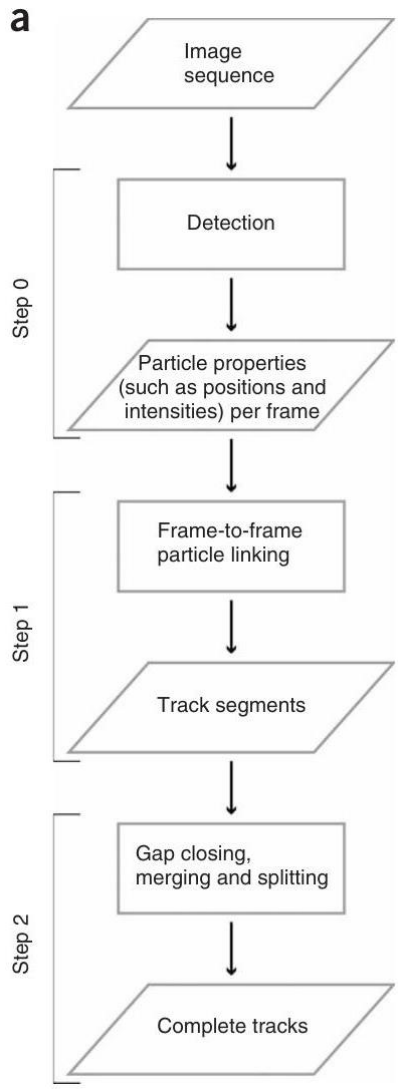

Fig. 3: generated particle tracks in two steps

segments that need to be connected to form a continuous track.

End to Middle: This approach is used for capturing merging events. In such a case, the end of one track segment is connected to a point within another segment, not at its start or end. This would be relevant in situations where the tracked object or entity joins or becomes part of another moving object or entity, effectively merging their paths.

Start to Middle: Conversely, this method is used to capture splitting events. Here, the start of one track segment is connected to a point within another segment. This would apply in scenarios where an object or entity that was part of a larger group or formation breaks away and starts its own separate path.

We formulated both the frame-to-frame particle linking step and the gap closing, merging and splitting step as LAPs ([4],[5]).

In the LAP framework, every potential assignment (particle assignment in the first step, track segment assignment in the second step) was characterized by a cost C (matrix entries in Fig. 4 and Fig. 5). The goal of solving the LAP in each step was then to identify the combination of assignments with the minimal sum of costs:

$$\hat{A}_{\text{arg min}} = \sum_{i=1}^{\substack{\text{Number} \\ \text{of rows}}} \sum_{j=1}^{\substack{\text{Number of} \\ \text{columns}}} A_{ij} C_{ij} \tag{1}$$

such that

$$\sum_{i=1}^{\substack{\text{Number} \\ \text{of rows}}} A_{ij} = 1 \quad \text{and} \quad \sum_{j=1}^{\substack{\text{Number of} \\ \text{columns}}} A_{ij} = 1 \tag{2}$$

How to define mid points? d and b?

In equation 1, A is any assignment matrix with entries 1 (link) and 0 (no link), and $\hat{A}$ is the assignment matrix that minimizes the sum of costs. The conditions on A (and consequently $\hat{A}$) in equation 2 guaranteed that the selected assignments were mutually exclusive, that is, no particle or track segment could be included in more than one assignment.

Thus, when a particle or track segment had multiple potential assignments, the assignments competed with one another. Although assignments with a lower cost tended to win, the requirement for a globally minimized cost could result in the selection of assignments in which the costs were not the lowest.

In the frame-to-frame particle linking step, three types of potential assignments were in competition (Fig. 4).

d and b only works for each particle. Diagonal matrix.

A particle in the source frame t could link to a particle in the target frame $t+1$ (cost function ℓ). Alternatively, a particle in the source frame could link to nothing, leading to a track segment end (cost function d), or a particle in the target frame could get linked by nothing, leading to a track segment start (cost function b). The decision between these possibilities was made globally in space (equations 1 and 2). However, the assignment was temporally

greedy as it was based on particle configuration in the specific source and target frames only.

In the gap closing, merging and splitting step, six types of potential assignments were in competition (Fig. 5). The end of a track segment could link to the start of another track segment, thus closing a gap (cost function g), the end of a track segment could link to a middle point of another track segment, leading to a merge (cost function m), or the start of a track segment could get linked by a middle point of another track segment, leading to a split (cost function s).

Alternatively, the end of a track segment could link to nothing, leading to a track termination (cost function d), the start of a track segment could get linked by nothing, leading to a track initiation (cost function b), or the track segment middle points introduced for merging and splitting could link to nothing, refusing a merge or a split (cost functions d' and b'). In this step, all track segments throughout the entire movie competed with each other. Thus, the LAP solution (equations 1 and 2) was global in both space and time.

The cost functions, in contrast, must be tailored to the specific tracking application. In our implementation, we technically solved this by treating the cost functions themselves as input variables together with their parameters. Here we demonstrate the definition of cost functions for tracking isotropic random motion, such as pure or confined Brownian motion, owing to its prevalence in cell biological applications. In this case, the costs for linking (ℓ), gap closing (g), merging (m) and splitting (s) were functions of distance and intensity:

$$\ell_{ij} = \delta_{ij}^2 \tag{3}$$

$$g_{IJ} = \delta_{IJ}^2 \tag{4}$$

$$m_{IJ}, s_{IJ} = \begin{cases} \delta_{IJ}^2 \times \rho_{IJ}, & \rho_{IJ} > 1 \\ \delta_{IJ}^2 \times \rho_{IJ}^{-2}, & \rho_{IJ} < 1 \end{cases}. \tag{5}$$

In equation 3,δ_{ij} is the distance between particles i and j. In equation 4, δ_{IJ} is the distance between the end of track segment I and the start of track segment J. In equation 5, δ_{IJ} is the distance between the end or start of track segment I and the middle point of track segment J, and ρ_{IJ} is the ratio of the intensities A_I (of track segment I) and A_J (of track segment J) before and after merging or splitting:

$$\rho_{IJ}(\text{ merge in frame } t) = \frac{A_J(t)}{A_I(t-1) + A_J(t-1)},$$
$$\rho_{IJ}(\text{ split in frame } t) = \frac{A_J(t-1)}{A_I(t) + A_J(t)}. \tag{6}$$

The intensity factor increased the cost when the intensity after merging or before splitting was different from the sum of intensities before merging or after splitting, with a higher penalty when the intensity was smaller.

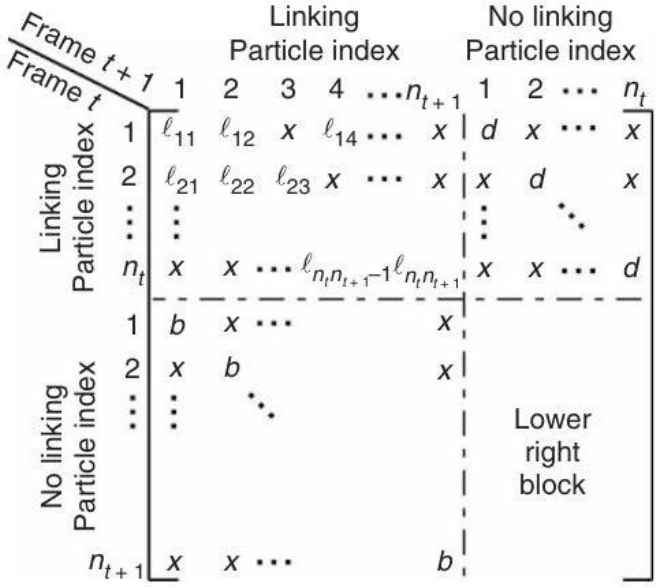

Fig. 4: matrix entries

V. Add Shape

A. Introduction to Hu Moments

The central moments for an image function $f(x, y)$ (which, in the context of image processing, is typically the pixel intensity at coordinates (x, y)) are defined as:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y)$$

where $\bar{x}$ and $\bar{y}$ are the coordinates of the centroid of the image (or shape), and p and q are the orders of the moment. The centroid coordinates are given by:

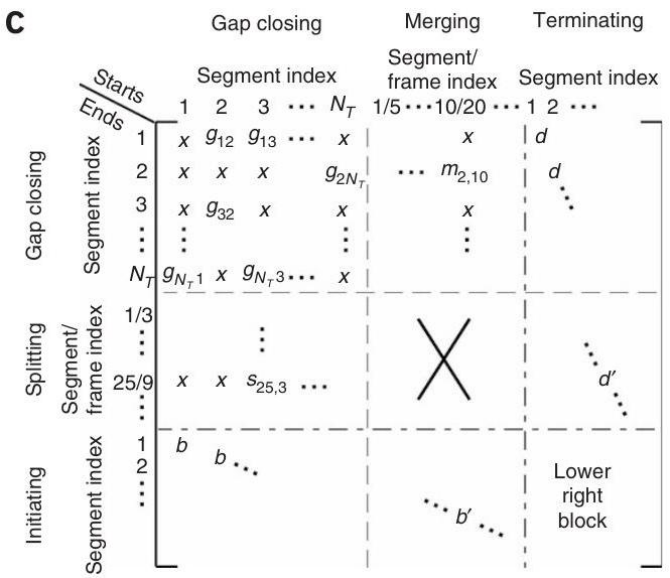

Fig. 5: matrix entries

$$\bar{x} = \frac{m_{10}}{m_{00}}$$
$$\bar{y} = \frac{m_{01}}{m_{00}} \tag{7}$$

Here, m_{pq} are the raw moments defined as:

$$m_{pq} = \sum_x \sum_y x^p y^q f(x,y)$$

Derived from these central moments, Hu introduced seven moments (in his 1962 paper) which are invariant to image transformations:

$$\begin{aligned}
H_1 &= \eta_{20} + \eta_{02} \\
H_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
H_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
H_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
H_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] \\
H_6 &= (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
H_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\
&\quad (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]
\end{aligned} \tag{8}$$

In these formulas, η_{pq} are the normalized central moments, defined as:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\left(1+\frac{p+q}{2}\right)}} \tag{9}$$

These seven Hu Moments provide a compact but effective representation of a shape, and are widely used in pattern recognition and image analysis due to their invariance properties.

B. Shape similarity based on Hu moments

The 'matchShapes' function in OpenCV compares two shapes S_1 and S_2 by computing their Hu Moments H_1 and H_2 respectively. The similarity between these shapes is then calculated using one of the following methods:

Method I1 (default):

$$D(S_1, S_2) = \sum_{i=1}^{7} \left| \frac{1}{\text{sign}(h_{1i}) \log |h_{1i}|} - \frac{1}{\text{sign}(h_{2i}) \log |h_{2i}|} \right| \tag{10}$$

Method I2 (alternative method):

$$D(S_1, S_2) = \sum_{i=1}^{7} |\log |h_{1i}| - \log |h_{2i}|| \tag{11}$$

Method I3 (another alternative):

$$D(S_1, S_2) = \sum_{i=1}^{7} \frac{|h_{1i} - h_{2i}|}{|h_{1i}|} \tag{12}$$

Here, h_{1i} and h_{2i} are the ith Hu Moments of shapes S_1 and S_2 respectively.

A lower value of D indicates higher similarity.

C. Simulation results

We generate second image by adding random noise.

The result is as the following figure:

D. Experiential results

We test the first six frames

The following figure is the stacked figure of two nearby frames.

The following figure is the tracking performance of two near frames.

The following figure is the tracking performance of two near frames when considering shape similarity.

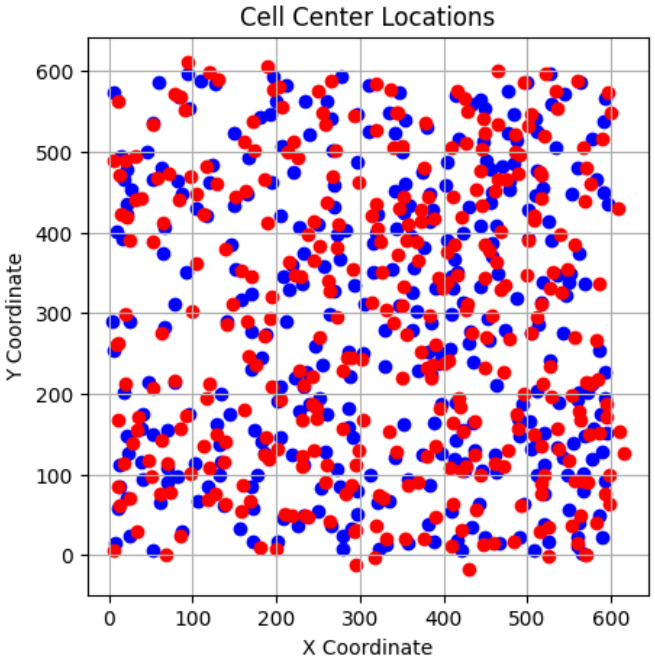

Fig. 6: image by adding random noise

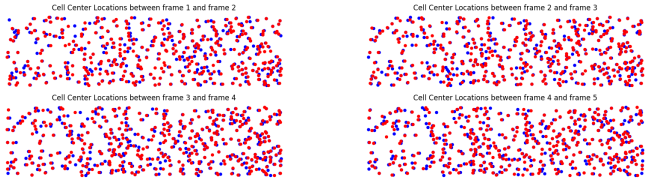

Fig. 7: Comparison image

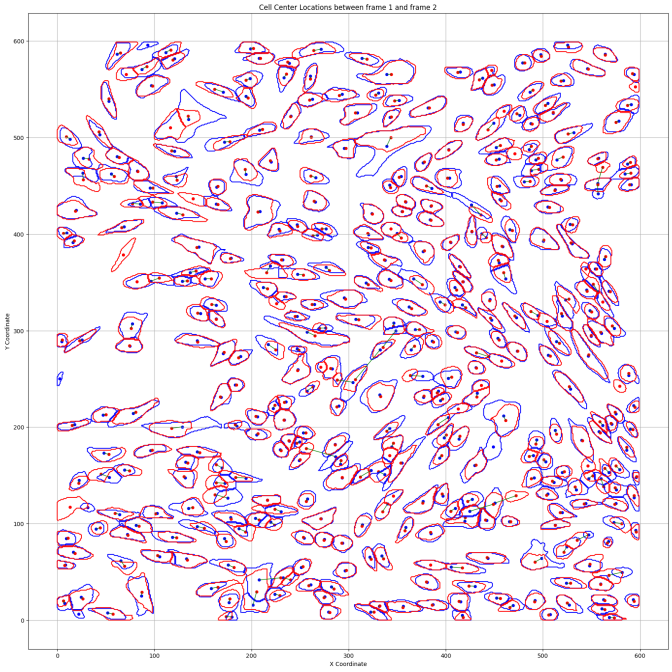

Fig. 8: tracking performance of two near frames

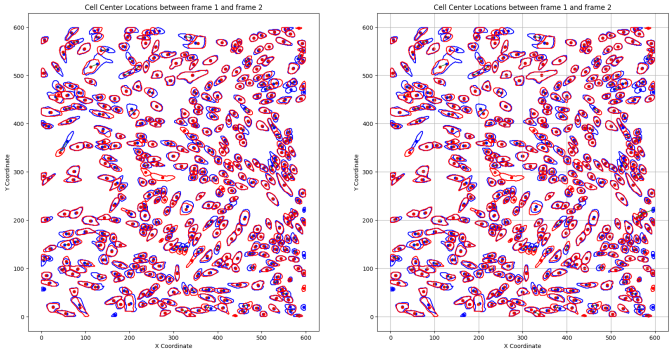

Fig. 9: the tracking performance of two near frames when considering shape similarity

REFERENCES

[1] D. Ershov, M.-S. Phan, J. W. Pylvänäinen, S. U. Rigaud, L. Le Blanc, A. Charles-Orszag, J. R. Conway, R. F. Laine, N. H. Roy, D. Bonazzi *et al.*, "Trackmate 7: integrating state-of-the-art segmentation algorithms into tracking pipelines," *Nature Methods*, vol. 19, no. 7, pp. 829–832, 2022.

[2] K. Jaqaman, D. Loerke, M. Mettlen, H. Kuwata, S. Grinstein, S. L. Schmid, and G. Danuser, "Robust single-particle tracking in live-cell time-lapse sequences," *Nature methods*, vol. 5, no. 8, pp. 695–702, 2008.

[3] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[4] D. Du and P. M. Pardalos, *Handbook of combinatorial optimization*. Springer Science & Business Media, 1998, vol. 4.

[5] R. Jonker and T. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," in *DGOR/NSOR: Papers of the 16th Annual Meeting of DGOR in Cooperation with NSOR/Vorträge der 16. Jahrestagung der DGOR zusammen mit der NSOR*. Springer, 1988, pp. 622–622.