Efficient and Reliable Estimation of Cell Positions

Mirela T. Cazzolato, Agma J. M. Traina
Institute of Mathematics and Computer Sciences
University of São Paulo, USP
São Carlos, Brazil
mirelac@usp.br,agma@icmc.usp.br

Klemens Böhm
Institute for Program Structures and Data Organisation
Karlsruhe Institute of Technology, KIT
Karlsruhe, Germany
klemens.boehm@kit.edu

ABSTRACT

Sequences of microscopic images feature the dynamics of developing embryos. Automatically tracking the cells from such sequences of images allows understanding the dynamics which a living element demands to know its cells movement, which ideally should take place in real-time. The traditional tracking pipeline starts with image acquisition, data transfer, image segmentation to separate cells from the background, and then the actual tracking step. To speed up this pipeline, we hypothesize that a process capable of predicting the cell motion according to previous observations is useful. The solution must be accurate, fast and lightweight, and be able to iterate between the various components. In this work we propose *CM-Predictor*, which takes advantage of previous positions of cells to estimate their motion. When estimation takes place, we can omit costly acquisition, transfer and process of images, speeding up the tracking pipeline. The designed solution monitors the error of prediction, adapting the model whenever needed. For validation, we use four different datasets with sequences of images with developing embryos. Then we compare the estimated motion vectors of *CM-Predictor* with traditional tracking methods. Experimental results show that *CM-Predictor* is able to accurately estimate the motion vectors. In fact, *CM-Predictor* maintains the prediction quality of other algorithms and performs faster than them.

CCS CONCEPTS

• **Computing methodologies** → **Tracking**; • **Applied computing** → **Life and medical sciences**; **Computational biology**; **Imaging**;

KEYWORDS

Cell tracking, moving objects, microscopic images, developing embryo, predictive analysis

ACM Reference Format:
Mirela T. Cazzolato, Agma J. M. Traina and Klemens Böhm. 2018. Efficient and Reliable Estimation of Cell Positions. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271734

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271734

1 INTRODUCTION

Problem Statement. Tracking cells of developing embryos from sequences of microscopic images is an important task regarding the study of the dynamics of biological processes [3, 6]. Such tracking must take place automatically, to facilitate scalability. Figure 1(a) is the traditional pipeline of cell tracking. It encompasses image acquisition, data transfer, image processing and then the actual tracking. After each iteration, the pipeline is performed again; the thin arrow represents this. These steps must be done in a timely manner, to process the data ideally in real-time. The conventional solution applies a segmentation algorithm to separate cells from the background of the images and to collect their positions (spatial coordinates). However, limitations of image capturing and segmentation introduce artifacts in cell curvatures, making the tracking of cells a difficult problem [4]. Tracking algorithms now establish cell-to-cell correspondences along the sequence of images, by matching cells according to their locations. When tracking cells, conventional algorithms also do build vectors of cell trajectories, but this takes place late in the data-processing chain, so the vectors are not used for tracking. In this paper, we study whether and to which extent establishing an explicit representation of the motion of cells from the images early in the chain gives way to better cell tracking.

If such an explicit representation is given, one can use it to accurately and efficiently predict/estimate the positions of the cells. This in turn would avoid transmitting the full images and storing them before any analysis takes place – in our institution, this currently takes hours for one embryo and is the bottleneck of the entire embryo-breeding workflow. There is a close relationship to moving objects and moving objects databases [2, 13, 15], a field that the database-research community has devoted much attention to over the years, and we will elaborate on the connection. Briefly, while moving objects traditionally are traffic participants (cars, pedestrians etc.), investigating whether ideas from there also help with the tracking of cells of developing embryos is worthwhile.

Figure 1(b) depicts the pipeline proposed in this paper. The steps from the traditional pipeline (boxes inside the gray area) are omitted in iterations when this estimation takes place. All this means that different kinds of improvement should be possible (be it detection quality, be it speed of tracking). However, different metrics to assess the improvements are necessary, and we discuss such measures as well. There are also alternative models of cell motion. For instance, alternatives relate to the number of points used for interpolation, and the distance threshold used for matches. We establish the design space and evaluate the plausible options systematically.

Difficulties. Our solutions must update the representation of the motions as new data arrives. Accordingly, when taking advantage of previous cell positions for prediction, the solution must monitor the prediction error at each iteration, in order to adapt

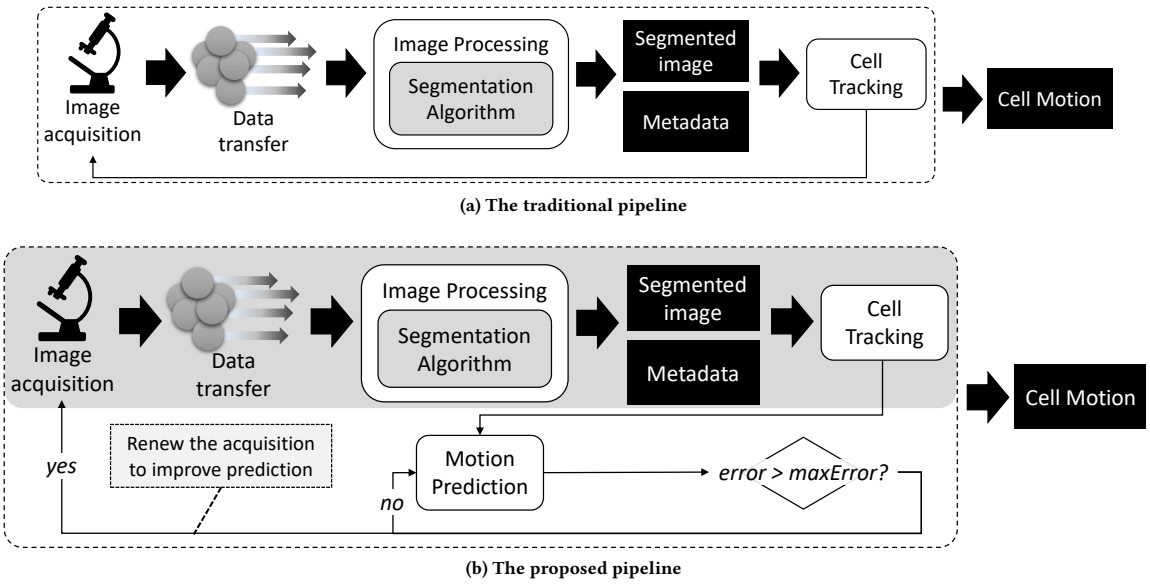

Figure 1: Difference between (a) the traditional pipeline of cell tracking and (b) the proposed one: by predicting cell positions, most steps of the traditional pipeline can be skipped for many iterations, speeding-up the entire process.

the model when necessary. Extreme solutions like just throwing away the previous model or adapting the model only minimally do not appear to be promising. Our proposal also requires reorganizing the interaction between the various components such as image segmentation or tracking. At the same time, the solution must be light-weight. Very importantly, it must work orthogonally to existing tracking tools, i.e., be combinable with existing implementations of the other components of the pipeline. In fact, our objective is not to "replace" existing techniques for image segmentation and tracking, but to use available implementations and focus on the establishment of vectors describing cell motions. Finally, modularity is a related yet different requirement. Dealing with image segmentation and tracking separately makes the pipeline more flexible, by being able to replace individual components.

Contributions. In this paper we propose *CM-Predictor*, which takes advantage of cell locations from previous images, to estimate their motion along time. By estimating cell motion, we can speed up the generation of the output. We perform a motion estimation after the acquisition of sufficiently many cell points, taking advantage of Lagrange's polynomials [1]. We show this to be adequate for the problem at hand. *CM-Predictor* relies on existing segmentation algorithms to obtain the position of cells in previous images. We compare the estimated motion vectors of *CM-Predictor* with the results of conventional tracking methods, for four different datasets. *CM-Predictor* can accurately estimate cell-motion vectors, improving the tracking pipeline by presenting better results than the existing tracking approaches. It is able to maintain prediction quality while being faster than other algorithms.

Paper Outline. Section 2 describes relevant background; Section 3 presents related work; Section 4 describes two algorithms for cell tracking; Section 5 introduces the *CM-Predictor*; Section 6 features experiments; Section 7 concludes.

2 BACKGROUND

This section features fundamental concepts related to cell motion. A sequence of microscopic images depicts the development of an embryo: It starts with a single **cell** that splits over time, as the **embryo** is developing. The images are obtained after fixed intervals of time, depending on the quality and configuration of the acquisition equipment. Each cell detected in the image by the segmentation algorithm has a spatial location, referred to as **seed** (or seed point) at time t. A match of a new seed point and an already existing cell is a *seed-to-cell correspondence*. The accurate monitoring of topological changes, e.g., splitting objects over time, relies on the accuracy of cell detection in each time frame [12]. An image-processing algorithm is applied to the images, to separate objects from the background, thus detecting and segmenting cells. One such algorithm is *TWANG* [14]. We have used *TWANG* in this work, as it is fast and segments images with high accuracy. Such an algorithm outputs segmented images, with the detected cells, as well as metadata information, containing the position of each cell in each image. The literature also reports the problem of segmenting overlapping and touching cells [11, 17]. While this is a challenge as well, it is beyond the scope of this current article.

Given a sequence of images, and since a cell changes its position in each image, we can represent the **trajectory** of a cell as a list of seeds. For example, given a Cell c, detected in the images in time interval $[3, n]$, its trajectory is $T = (sc_3, ..., sc_n)$, where sc_i is the seed of c at time i with $3 \leq i \leq n$. The next section reviews algorithms for the tracking of cells from sequences of images.

3 RELATED WORK

This section does not cover all related work; we discuss more when introducing our approach, to enable more direct comparisons.

Tracking microscopic objects. Biologists use the outputs of cell tracking to analyze changes induced by the use of substances, such as contrast agents, observing the evolution in cell motion and morphology [7]. There is a lot of work regarding the tracking of cells. Chakraborty *et al.* [5] propose to represent the cells of an embryo using a graph structure. Their method works with cells that are in close contact with each other, i.e., share an edge. The method constructs a graph structure for the cells every time a new image is processed. Jiuqing *et al.* [9] define six local events (move, divide, appear, disappear, split, merge) to describe linking patterns between pairs of consecutive images. Unlike our proposed pipeline, their proposal consists of a joint detection and tracking method, predicting cell trajectories by solving a linear programming problem and learning the model parameters with a structured SVM.

Hielsenbeck *et al.* [8] present a review of existing tools. Among the tools mentioned is the *TLM-Tracker* [10]. We use this tool as a reference, to compare with our methods, since it attempts to perform a fully automated cell tracking and is available online. *TLM-Tracker* has several steps, which range from the pre-processing of input images, the detection and tracking of cells, to the visualization of the motion vector of the embryo. *TLM-Tracker* applies a segmentation algorithm to the images and searches for seed-to-cell correspondences in two consecutive images, using two different approaches: (i) based on the areas of cells, (ii) based on the center points of cells. The first approach (i) gets the minimum bounding rectangle (MBR) of each cell, from two consecutive images. Then it gets the distance between the cells in time by computing the relative overlap of their MBRs. If there is no overlap, the second approach (ii) computes the Euclidean distance between the center points of cells. *TLM-Tracker* has a joint detection and tracking step to predict the cell positions. This prediction is computed using a fitted polynome: the approaches restrain the search space for the segmentation algorithm, to look for the probable match point of a certain cell in the next image. Although the tool was originally proposed to work with elongated cells, the authors state that *TLM-Tracker* works well with different kinds of cells [10].

The aforementioned state-of-the-art tracking approaches work with cell points detected from sequences of images. However, the traditional pipeline to obtain the motion vector of embryo cells can be computationally expensive. To overcome this issue, in Section 5 we propose *CM-Predictor* to establish the motion vectors of cells, based on cell points from previous images.

Moving objects. The movement of objects and its concise representation, possibly together with uncertainty, have been of interest to the database community. For instance, Tao *et al.* [15] focus on the problem of predicting motion pattern. They argue that individual trajectories may vary significantly, but most motion types show self-similar behavior, in the sense that one can often predict the current location from the ones in the recent past. They propose a framework to index object locations and to process queries, based on estimated position of objects. More recently, AlMuhisen *et al.* [2] have proposed a characterization of the behavior of moving objects, obtained by GPS locations of smartphones, cars etc. They look for correspondences between hidden patterns and trajectories, using frequent pattern mining. They tag city maps to visualize the behavior of different spatio-temporal values. Saltenis *et al.* [13] have modeled the positions of moving objects as functions of time. According to them, modeling the movement of objects not only facilitates predictions, but also solves the problem of frequent updates that would be required to approximate continuous movement in a traditional setting. Borrowing from these ideas, we now target at useful representations of a specific kind of moving object, developing cells in microscopic images.

4 THE TRADITIONAL PIPELINE: TRACKING CELLS

When applied to the images, the segmentation algorithm generates the segmented image and metadata. The metadata contains the position of each detected cell and represents them as seed points. These seed points serve as input to the tracking algorithm, which is responsible to add each newly detected seed point to its corresponding cell (i.e., a previously detected cell). In this section we present two algorithms: *Direct-Tracker* and *Clever-Tracker*. They are basic building blocks which we will employ for the task of tracking cells. The segmentation algorithm used in this work is *TWANG* [14]. However, any segmentation algorithm that outputs the cell positions in a given image can be used here.

4.1 Data Structures

The tracking approaches use a distance-threshold value th to decide if a pair of seeds is a match. This pair consists of seed points from time $t-1$ that have already been added to the description of cells of the embryo and the new seeds, detected at time t. Given this threshold, the approaches do not need to test all possible combinations of seeds to determine a match, speeding up the matching. Thus the tracking algorithms receive as **input**:

- S_{t-1}: a list with all seeds at time $t-1$, sorted by the order of detection. These seeds have already been inserted to the description of cells from an embryo;
- S_t: a list with all seeds detected at time t, sorted by the order of detection;
- th: the distance threshold.

And as **output**, the approaches return:

- *embryo*: an embryo, composed of moving cells, each one represented as a sequence of seed points.

4.2 Adding a Cell Match

Let C be the set of cells of an *embryo*, seen so far in the sequence of images. Recall that each seed in S_{t-1} stands for a cell $c \in C$. To decide whether a pair of points is a match we employ the Euclidean Distance (L_2). The approaches we will propose use the Function *AddMatch* to add a new match: The new seed point $s_2 \in S_2$ is added to a cell C which contains the matching seed point $s_1 \in S_1$ (Lines 1–3). Then both seeds, which are from $t-1$ and t, are removed from the lists S_{t-1} and S_t (Lines 4–5). If an existing cell has more than one new seed point as a match at time t, this means that the cell has been split. Thus, the tracking approach creates a new cell for each match with a new seed.

The data structures and the method for adding another cell to the embryo are the same for both *Direct-Tracker* and *Clever-Tracker*.

Function AddMatch(*embryo*, s_1, s_2, S_1, S_2)

1 Let C be the set of cells of *embryo* seen so far
2 $c \leftarrow$ cell $\in C$ containing seed s_1;
3 c.addSeed(s_2);
4 Remove s_1 from S_1;
5 Remove s_2 from S_2;
6 **return** *embryo*

4.3 Tracking Approaches

Direct-Tracker and *Clever-Tracker* are as follows. They select the pair of seeds to be matched $p(s_{t-1}, s_t)$ based on a specific criterion:

- *Direct-Tracker* works in a greedy manner, matching the seed point of the existing cell to the first seed point within a distance threshold;
- *Clever-Tracker* maps each new seed point to its nearest cell.

Direct-Tracker

Algorithm 1 presents *Direct-Tracker*. For each newly detected seed at time t (s_t), *Direct-Tracker* searches for a matching seed at time $t-1$ (s_{t-1}) (Lines 3–5). Here, s_{t-1} belongs to an existing cell and is less than th of the distance from s_t. Accordingly, if the distance between s_t and s_{t-1} is less than a distance threshold, we consider them a match (Line 7). *Direct-Tracker* works in a greedy manner: It deems the first existing seed point (s_{t-1}), which is close to a new seed point (s_t) according to a distance threshold, a match. Note that this does not ensure that the seed point s_{t-1} is the nearest seed point of s_t. However, *Clever-Tracker* can do that.

Algorithm 1: *Direct-Tracker*

Input : S_{t-1}: a list with all seeds from time $t-1$
S_t: a list with all seeds from time t
th: the distance threshold

Output: The *embryo*, containing seed points of already existing cells and new seeds.

1 **begin**
2 initialization;
3 **foreach** *seed* $s_t \in S_t$ **do**
4 $matched\_t \leftarrow -1$;
5 **foreach** *seed* $s_{t-1} \in S_{t-1}$ **do**
6 $d \leftarrow ComputeDistance(s_t, s_{t-1})$;
7 **if** $(d < th$ *AND* $matched\_t = -1)$ **then**
8 $AddMatch(embryo, s_t, s_{t-1}, S_t, S_{t-1})$;
9 $matched\_t \leftarrow 1$;
10 Create new cells for remaining seeds and add them to *embryo*
11 **return** *embryo*

The result of *Direct-Tracker* depends on the order of seeds. The segmentation algorithm usually searches for cells following the order of pixels: for example, from the pixel in the top left corner of the image to the one at the bottom right. Thus, the cells tend to maintain the same order when listed by the segmentation algorithm, with minor changes due to new cells, resulting from cell splits.

Clever-Tracker

For each new seed point, *Clever-Tracker* uses the Function *CheckNearestSeed* (Algorithm 2) to search for its nearest existing cell that is already in the embryo. Given a seed s_q existing at time t, Function *CheckNearestSeed* checks if there is a seed s_{t-1} in S_{t-1} where $d(s_q, s_{t-1}) < mindist$, i.e., if there is a seed close to s_q (Lines 10–14). The distance between two seed points must be less than a distance threshold to be considered a match.

Algorithm 2 is the pseudo-code of *Clever-Tracker*. For each new seed at time t (Line 3), *Clever-Tracker* searches its closest seed (belonging to an existing cell) at time $t-1$ (Line 5). Function *CheckNearestSeed* (Line 5) searches the nearest seed, and the matches are added by calling Function *AddMatch* (Line 7).

Algorithm 2: *Clever-Tracker*

Input : S_{t-1}: a list with all seeds from time $t-1$
S_t: a list with all seeds from time t
th: the distance threshold

Output: The *embryo*, containing seed points of already existing cells and new seeds.

1 **begin**
2 initialization;
3 **foreach** *seed* $s_t \in S_t$ **do**
4 $mindist \leftarrow +\infty$;
5 $(matched\_t, mindist) \leftarrow CheckNearestSeed(s_t, S_{t-1}, th, mindist, -1)$
6 **if** $(matched\_t > -1)$ **then**
7 $AddMatch(embryo, s_t, s_{matched\_t}, S_t, S_{t-1})$;
8 Create new cells for remaining seeds and add them to *embryo*
9 **return** *embryo*
10 **Function** `CheckNearestSeed`(*sq, S, th, mindist, matched_t*)
11 **foreach** *seed* $s_i \in S$ **do**
12 $d \leftarrow ComputeDistance(sq, s_i)$;
13 **if** $(d < mindist$ *AND* $d < th)$ **then**
14 $mindist \leftarrow d$;
15 $matched\_t \leftarrow i$;
16 **return** *matched_t, mindist*;

The next section describes the estimation of cell positions based on seed points detected previously.

5 THE PROPOSED PIPELINE: ESTABLISHING CELL MOTION ALONG TIME

As just discussed, tracking algorithms use seed points detected in sequences of images to construct cell trajectories, following the traditional pipeline. We propose modifications to this pipeline. It now predicts future positions of cells, taking advantage of previously detected ones, to estimate cell motion. When we have enough cell

points for motion estimation, prediction takes place. We propose the *CM-Predictor* (**C**ell **M**otion **Predictor**) algorithm, described next.

5.1 Motion Estimation

CM-Predictor estimates new positions of points by using Lagrange's polynomials [1], see Equation 1. P(x) is the polynomial of degree $\leq n-1$ passing through n data points. (x_0, y_0), ..., (x_j, y_j), ..., (x_{n-1}, y_{n-1}) are these n points, where no two x_i are equal, and $0 \leq j \leq n$.

$$P(x) = \sum_{j=1}^{n} P_j(x), \tag{1}$$

where

$$P_j(x) = y_j \cdot \prod_{k \in \{1,\ldots,n\}-\{j\}} \frac{x - x_k}{x_j - x_k}.$$

Here, $P_j(x)$ is the polynomial at j. Function Interpolate-Points is the pseudo-code for estimating the points of each cell belonging to an embryo, relying on Equation 1. Note that we estimate the position of a new seed point based on last observed points of a cell. Thus, the algorithm assigns the estimated point to this cell (Line 7), without looking for a matching cell.

If the next cell position predicted by *Interpolate-Points* is at least th of the distance from the last position, we deem the prediction of this point correct; otherwise this is an interpolation error. The interpolation continues as long as the percentage of cells wrongly predicted is less than *maxError*, where $0 \leq maxError \leq 1$. When the error is higher, *CM-Predictor* discards the oldest points of each sequence of points (cell) which are used for interpolation, see Subsection 5.3.

Function Interpolate-Points(*embryo*)

```
1 foreach cell c in embryo do
2     S ← c.getSeeds();
3     for (i = 0 to S.size() − 1) do
4         newSeed.x ← Predict x using Eq. 1;
5         newSeed.y ← Predict y using Eq. 1;
6         newSeed.z ← Predict z using Eq. 1;
7         Add newSeed to cell c of the embryo;
8 return embryo;
```

5.2 Parameters

CM-Predictor receives as **input**:

- S_{t-1}: a list with all seeds detected at time $t-1$;
- S_t: a list with all seeds detected at time t;
- th: a distance threshold;
- w: the window size;
- pw: share of the window discarded after *maxError* is reached;
- *maxError*: maximum error allowed when interpolating points.

We use the current motion vector for prediction until the error exceeds a threshold. Then the oldest points of the current motion vector are discarded, and the window is renewed, i.e., new points are added to the window used for the interpolation. The size of the window w is an exogenous parameter in this current study, and we will study its influence experimentally.

5.3 The Cell Motion Predictor

Algorithm 4 is the pseudocode of *CM-Predictor*. The tracking takes place until all images have been processed. As long as *CM-Predictor* has not processed w images (Line 4), a tracker matches the cells. *CM-Predictor* deploys one of the approaches from Section 4. It does so by calling Function *PerformTracking* (Lines 5 and 6). The window is incremented (Line 7), and the algorithm checks if w images have been processed (Lines 8 and 9). If so, the algorithm uses the last w seeds of each cell for the interpolation. In the next iteration, the algorithm predicts the next points by calling *Interpolate-Points* (Line 11). This function uses the w last seeds added to the embryo to estimate the next points (Lines 16–22).

CM-Predictor estimates the new points using Equation 1, returning the embryo with the predicted seeds in the cells. The algorithm uses a multiple of the distance threshold $c \cdot th$ to bound the distance between the existing cell and its predicted new position. The parameter c forces the estimation of points to be near the previous position of the cell, ensuring that the predicted point is closer to its original cell than th and not totally off. Alternatively, when $c = 1$, the algorithm employs the same threshold th used during the tracking step. If a predicted point is not closer than the distance threshold $c \cdot th$ to its original cell, the algorithm deems the prediction a mistake and increments the number of errors. At each iteration, *CM-Predictor* checks if the number of errors is greater than *maxError* (Line 12). If so, the percentage pw of oldest points in the window is removed, and the algorithm processes the next image (from time $t+1$) (Line 13). The next image is captured and processed by the segmentation algorithm, before *CM-Predictor* continues working. These pw oldest points are at the beginning of each sequence of seed points, which configures the movement of a cell. If the number of errors is not greater than *maxError*, *CM-Predictor* performs the interpolation again, predicting the points at time $t+1$. This process is repeated until all images from the sequence are processed (Line 3).

As mentioned, *CM-Predictor* has the parameters w (size of the window), pw (percentage of the window discarded after reaching the maximum error allowed), and th (the distance threshold). Section 6 will study their effects experimentally.

6 EXPERIMENTAL ANALYSIS

In this section we present the experimental analysis of the various methods. All experiments have been executed in an Intel(R) Core(TM) i7-4770 CPU 3.40GHz 1TB-HD 16GB machine, running the Ubuntu 16.04.3LTS operating system.

6.1 Evaluation Measures

We employ the following measures to validate our method: the total number of cells created by the trackers, the tracking error, and the trajectory of cells (by plotting their distance to the image centroid), which are quality measures; and the average execution time, to assess the performance of the algorithms. These measures are as follows. For each image of the sequence, the tracker matches new seed points with existing cells and creates new cells for seed points that have not been matched. We compute the **total number of cells** created by the tracker and added to the embryo. Ideally, this number must be close to the number of cells reported by the

ground truth and *TWANG*, as depicted in Figure 3. We consider both numbers because the ground truth reports the actual number of cells present in the image, while the output of *TWANG* is the cells that the trackers must work with.

Algorithm 3: *CM-Predictor*

Input : S_{t-1}: a list of seeds from time $t-1$
S_t: a list of seeds from time$t$
th: a distance threshold
w: the window size
pw: percentage of the window to be discarded
$maxError$: the maximum error allowed

Output: The *embryo*, containing seed points of already existing cells and new seeds.

```
1  begin
2      initialization;
3      while (t < sizeOfSequence) do
4          if (window < w) then
5              foreach seed s_t ∈ S_t do
6                  embryo ← PerformTracking();
7              window++;
8              if (window = w) then
9                  Initialize polynomial;
10         else
11             embryo ← Interpolate-Points(embryo);
12             if (error > maxError) then
13                 Remove pw% of the oldest points in the
                    window and process next image
14         t++;
```

A **tracking error** occurs when the algorithm misses a seed-to-cell correspondence or adds a non-existing one. In other words, if the algorithm erroneously creates a new cell, this is a tracking error. If the algorithm adds the new seed point to an existing cell when it should have created a new cell, this is a tracking error as well. With the number of tracking errors we compute the **Tracker Detection Rate (TDR)** [4]. It is $TDR = TP/n$, where True Positives (TP) is the number of images with no tracking errors, and n is the number of images in the sequence. TDR allows to evaluate seed-to-cell correspondences, and TDR values close to 1 represent optimal results. We also evaluate motion vectors of cells estimated by *CM-Predictor* by computing the **distance of all cell points to the centroid of the image**. This distance is important to observe how the cells were matched along time, and to compare the position of cells detected in the image to the ones predicted by *CM-Predictor*.

To evaluate the performance of trackers, we look at the **average execution times** (during the tracking step only). More specifically, we run each algorithm 1,000 times, and we then take the average.

6.2 Dataset and Parameters Setup

We use the datasets provided by the Cell Tracking Challenge[1]. In this benchmark, each dataset has its original images, ground truth annotations and segmented images. The annotations have been done manually. They are available in the following format: *"id of cell"*, *"initial time"* (indicating the first time the cell has been detected), *"final time"* (indicating the last time the cell has been detected), and *"id of parent cell"* (if this cell has been generated from the split of another one). Table 1 lists the datasets used in each experiment, with the original name and the number of images, and the type of images in the dataset ($2D + t$ and $3D + t$). Figure 2 contains examples of such images.

Table 1: Datasets used in each experiment.

Experiment	Dataset	Set	Type	# images
Exp1.1	Fluo-N3DH-SIM+	01	3D+t	150
Exp1.2	Fluo-N3DH-SIM+	02	3D+t	80
Exp2.1	Fluo-N2DH-GOWT1	01	2D+t	92
Exp2.2	Fluo-N2DH-GOWT1	02	2D+t	92

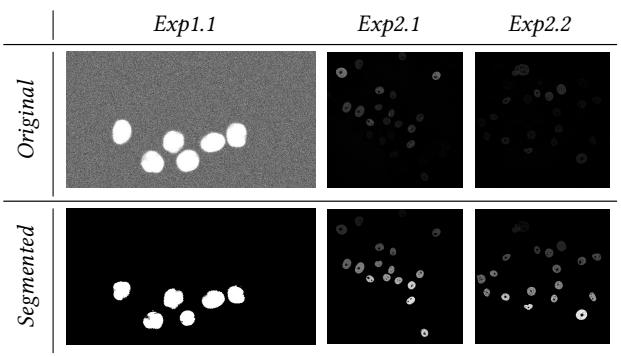

Figure 2: Examples of original and segmented images.

In the segmentation step, *TWANG* has the parameter values $SpacingX = 0.3$, $SpacingY = 0.3$ and $SpacingZ = 2$ for the four used datasets, obtained experimentally. All other parameters have their default values, see [14]. *TLM-Tracker* includes several segmentation algorithms to detect cells from images. We use *TLM-Tracker* with the segmentation algorithm Chan-Vese for Exp2.1 and Watershed for Exp2.2, since they had yielded the best segmentation results. We have not used *TLM-Tracker* in Experiments Exp1.1 and Exp1.2, since it does not support 3D+t images.

6.3 Pre-processing: Detecting Cells from Images

We have applied *TWANG* and *TLM-Tracker* to the images (in the segmentation step) to obtain the seed points of the cells, which the trackers can then use subsequently. Detecting seed points can be seen as a preprocessing step of our proposal, which extracts the cell information from images. Figure 3 compares the total number of cells obtained with *TWANG*, *TLM-Tracker* and the ground truth. Both approaches result in different numbers of cells. The *TLM-Tracker* features more divergence than *TWANG* compared to the ground truth. *TLM-Tracker* did not detect all cells in the images from Exp2.1 and Exp2.2. This is because most cells from Exp2.1 and Exp2.2 are faded, i.e., they are not easy to recognize, see Figure 2.

[1]The Cell Tracking Challenge: www.celltrackingchallenge.net [16] – Last access in August 15th, 2018.

Once we have the positions of the cells, we proceed to the tracking and prediction steps.

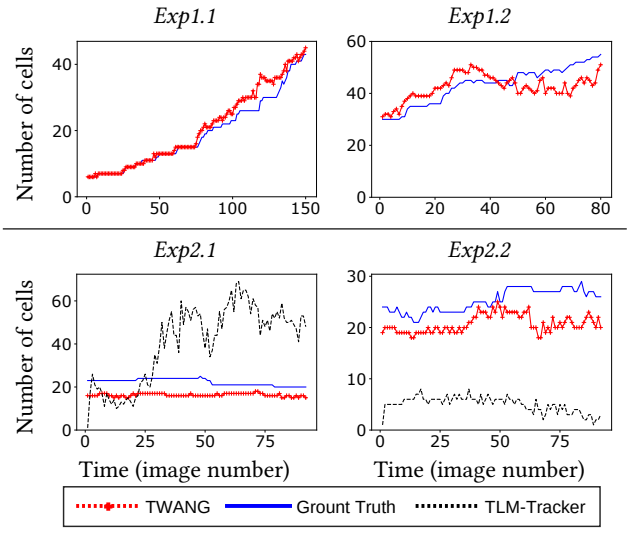

Figure 3: Number of cells detected by *TWANG* and *TLM-Tracker*, compared to the number of the ground truth data.

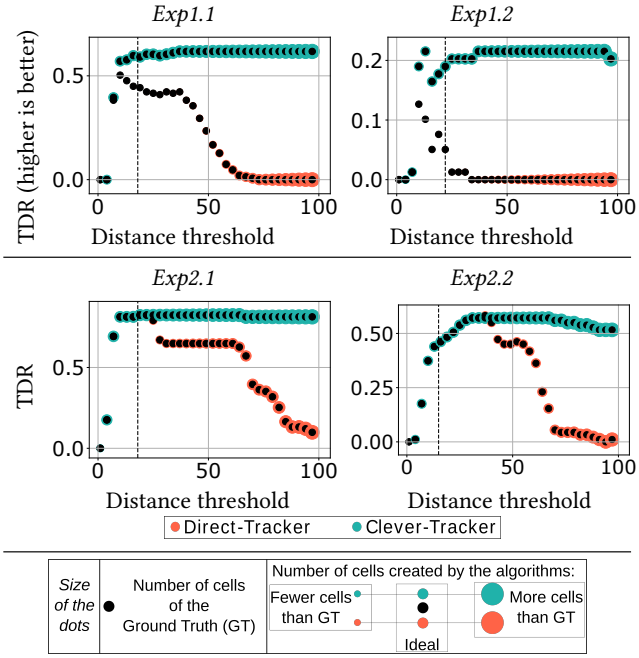

Figure 4: Testing different distance threshold values.

6.4 Tracking and Predicting Trajectories of Cells

We first define the **distance threshold** th to be used in each experiment. Figure 4 shows the TDR results with *Direct-Tracker* and *Clever-Tracker*. Each point represents the TDR result obtained after processing all images of the sequence, using a specific distance-threshold value. The size of the dots represents the number of cells created by the algorithms. We also plot the total number of cells from the ground truth as dark dots in the middle of each point. The best threshold value must yield high TDR values and dot sizes close to the ground truth. The total number of cells created by the algorithm also needs to be close to the ground truth. We can see that, as th increases, the number of cells created is higher, diverging even more from the ground truth. For small values of th, i.e., about less than 10, the TDR results also tend to be low. Accordingly, the best values are those between approximately 10 and 30. The dashed vertical lines represent threshold values th used in the remaining experiments: $th = 18$ for Exp1.1, $th = 22$ for Exp1.2, $th = 18$ for Exp2.1 and $th = 15$ for Exp2.2.

Using these threshold values, we generate all combinations for the following parameter values of *CM-Predictor*: w from 1 to 15 (increment of 1), pw and $maxError$ from 0.1 to 1.0 (increment of 0.1), and threshold from $0.1 \cdot th$ to $1.0 \cdot th$ (increment of 0.1). This results in $15 \times 10 \times 10 \times 10 = 15,000$ combinations, from now on called configurations. Figure 5 shows the TDR results of all configurations, where the size of the dots corresponds to the number of cells created, and the dark dots inform us on the number of cells of the ground truth. The best results are those with small blobs, and they are at the top of the charts. Each point represents the result obtained after processing all images of the sequence, using the specific configuration. We observe that *CM-Predictor* obtains low TDR when creating many more cells than expected. On the other hand, the method obtains the best TDR results when the number of cells created is smaller than the one of the ground truth. As we aim to obtain high TDR values and a number of cells close to the ground truth, we only use a subset of the configurations to evaluate the other parameters. We first select all combinations with $TDR > 0.7$ for Exp1.1, $TDR > 0.4$ for Exp1.2, $TDR > 0.85$ for Exp2.1, and $TDR > 0.65$ for Exp2.2.

We evaluate the impact of each parameter of *CM-Predictor* by plotting the outcomes of the configurations selected in the previous step. Figures 6, 7, 8 are the representative charts for Exp1.1, Exp2.1 and Exp2.2 respectively. To save space, we omit the charts regarding Exp1.2, which are comparable to the ones for Exp1.1, Exp2.1 and Exp2.2. In Exp1.1, $w < 4$, $pw < 0.4$, $maxError$ between 0.4 and 0.8 with the distance threshold of $0.5 \cdot th$ has led to the best results. In Exp1.2, the highest number of cells has been created with $w = 2$, the distance threshold $0.9 \cdot th$, $pw < 0.6$ and independently of the value of $maxError$. In fact, with this configuration the whole window is discarded once *CM-Predictor* removes $\lceil w \times pw \rceil$ points from the window when the error becomes too large. With a high th value, points that are not very close from each other do not match, yielding more errors and forcing the window to be updated. Next, with only two points in the window, *CM-Predictor* generates poor estimates of the position of the next cells. This results in a large distance between points and more cells being created. The best results are obtained with $w = 3$, with high TDR values using w between 4 and 7. In both Exp2.1 and Exp2.2, small values of w (< 4) yield TDR results with the number of cells close to the ground truth. The highest concentration of points occurs with distance threshold of $c > 0.8$. This means that, when *CM-Predictor* allows for a relatively high distance between points to still be a match, the

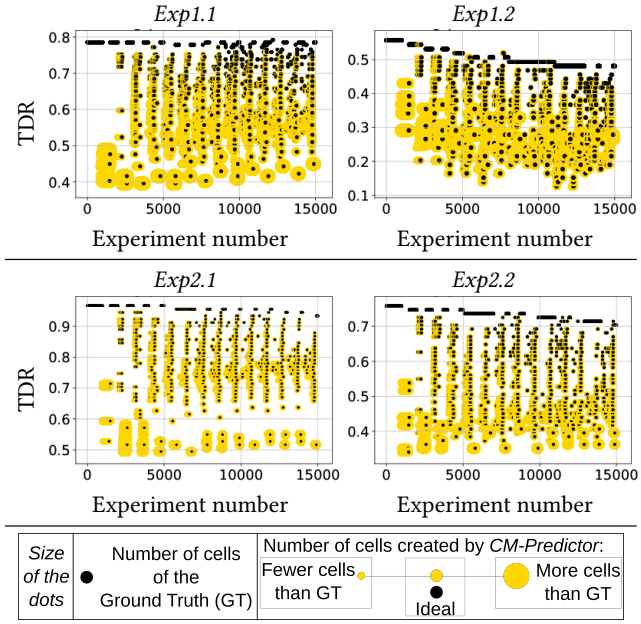

Figure 5: Parameter combinations of *CM-Predictor*.

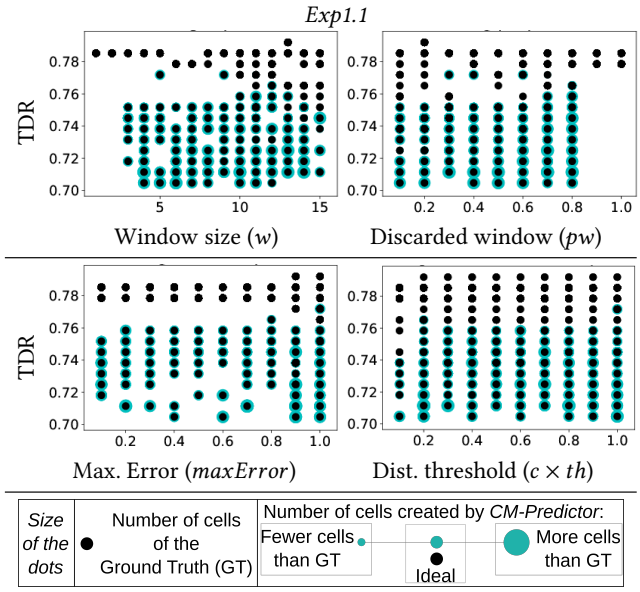

Figure 6: Parameter evaluation of *CM-Predictor* for Exp1.1.

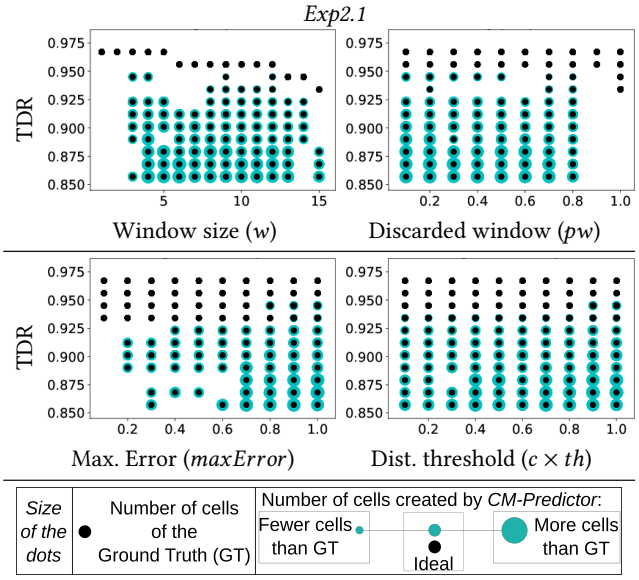

Figure 7: Parameter evaluation of *CM-Predictor* for Exp2.1.

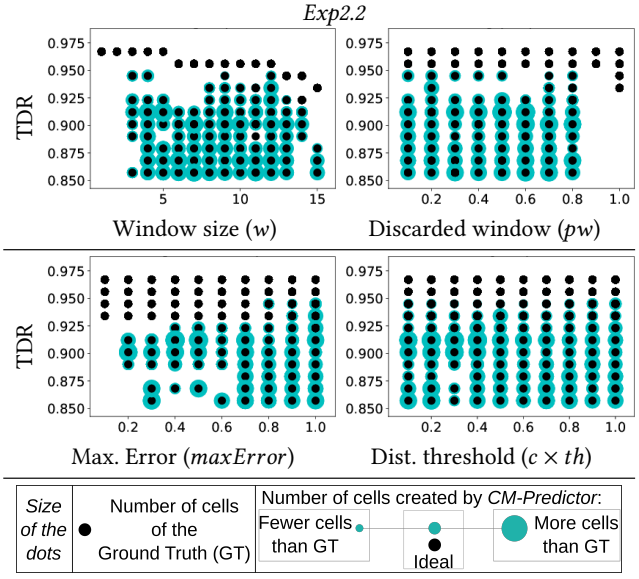

Figure 8: Parameter evaluation of *CM-Predictor* for Exp2.2.

results are the best. Overall, for all datasets *CM-Predictor* is best with parameter values $w = 3$, $pw = 0.3$ and $maxError = 0.9$.

To provide further evidence on the effectiveness of prediction, we compare the results of *CM-Predictor* with the ones of *Direct-Tracker*, *Clever-Tracker*, and *TLM-Tracker*. The parameter values of *CM-Predictor* in these experiments are $maxError = 0.3$, distance threshold $th \cdot 0.4$ and $w = 4$. Table 2 lists the total TDR values with the different algorithms and the total numbers of cells created. First and foremost, our newly proposed methods yield the best results for all experiments, compared to *TLM-Tracker*. In Exp1.1 and Exp1.2, *Direct-Tracker* is the method which has created the highest number of cells. This indicates that its matching has not been correct and has led to the low TDR values. Next, the high number of false positives and false negatives in the cells detected by *TWANG* has bogged down the matching precision of *Direct-Tracker*. *Clever-Tracker* has yielded similar results. When *CM-Predictor* performs predictions, it does not create any cell or delete existing ones. This is one reason why the algorithm has created fewer cells than the other ones. Its overall TDR values are high. When the number of cells from one image to the next one does not change much, TDR tends to perform better, since it does not create false positives. On the other hand, the matching between predicted and detected points can be damaged

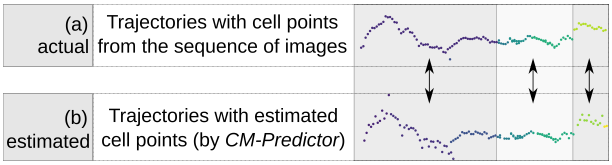

Figure 9: Cell trajectories with (a) actual and (b) estimated points.

when *CM-Predictor* stops predicting (because the error was high) and reads points detected from new images. This is because *CM-Predictor* creates false positive cells.

Table 2: Comparison of the approaches.

Algorithm		Exp1.1	Exp1.2	Exp2.1	Exp2.2
Direct-Tracker	TDR	0.45	0.05	0.82	0.46
	Tot. cells	440	584	31	81
Clever-Tracker	TDR	0.73	0.24	0.62	0.68
	Tot. cells	144	259	31	81
CM-Predictor	TDR	0.58	0.19	0.82	0.46
	Tot. cells	154	538	185	100
TLM-Tracker	TDR	–	–	0.0	0.29
	Tot. cells	–	–	956	142

We also plot the trajectories of the cells for each algorithm, according to their distance to the image centroid. Figure 9 presents examples of how we visually compare cells trajectories from the actual points (detected in the images) and the estimated ones. Figures 10, 11 and 12 show the trajectories of cells for Exp1.1, Exp2.1 and Exp2.2, respectively. *Direct-Tracker* and *Clever-Tracker* yield very similar results, since they use the same cell points, provided by the segmentation algorithm. We observe that *CM-Predictor* is able to predict cell points that are similar to the actual ones, as expected. Although *CM-Predictor* has predicted most of the cells, the estimated motion vectors are similar to the ones detected by the segmentation algorithm and tracked by *Direct-Tracker* and *Clever-Tracker*. This shows that predicting cells positions based on recently observed cell points is a good approach. *TLM-Tracker* did not detect many cells, and thus its trajectories differ from the ones given by the other algorithms: It has created many more cells in Exp2.1 and did not detect all cells in Exp2.2.

Overall, the configurations of *CM-Predictor* show results comparable with the ones with tracking algorithms that work with actual cell positions. This shows the superiority of our proposal:

- The tracking detection rate of *CM-Predictor* is close to the ones reported by the tracking algorithms, that have actual spatial information of the cells;
- The predicted trajectories provided by *CM-Predictor* are similar to the ones composed of actual cell points.

This means that we do improve the tracking pipeline by including predictions of cell motions. Next, we show that *CM-Predictor* also is faster than those tracking approaches.

6.5 Performance Analysis

Table 3 lists the mean execution times (in seconds) of all approaches for the various experiments. *TLM-Tracker* has the higher execution times, but it has been implemented in MATLAB. The other algorithms are available in C++. *Direct-Tracker* is the fastest tracking approach, since it works in a greedy manner, matching the cells by simply checking if they are within a distance threshold. *Clever-Tracker* performs more comparisons between new seeds and existing cells. In iterations where *CM-Predictor* performs predictions, no comparisons between existing cells and new seeds take place. This makes the predictor much faster than the two tracker approaches. Finally, *TLM-Tracker* works according to the same principle as *Clever-Tracker*. But it has the additional cost of predicting the overlap of cell areas if this option is selected to improve results of the tracking step.

Table 3: Mean execution time (in seconds).

Algorithm	Exp1.1	Exp1.2	Exp2.1	Exp2.2
Direct-Tracker	0.12	0.16	0.42	0.06
Clever-Tracker	0.18	0.27	0.06	0.09
TLM-Tracker	–	–	0.348	0.416
CM-Predictor	0.06	0.20	0.06	0.04

6.6 Discussion

The estimation choice of values for the parameters w, pw, $maxError$ and th, used by *CM-Predictor* and discussed in the last subsections, heavily depend on the images being analyzed. However, we only need to estimate the values once for a specific type of embryo. This is because the shape and spatial movement of cells are similar between sequences of images. Consequently, once the optimal values are known, one can also use them in subsequent analyses.

7 CONCLUSIONS

This work has studied the tracking and prediction of trajectories of cells of developing embryos in sequences of microscopic images. After having proposed two tracking algorithms (*Direct-Tracker* and *Clever-Tracker*), our core contribution is a new predictor algorithm (*CM-Predictor*). We have focused on the prediction of cell positions based on previous ones. The new algorithm *CM-Predictor* speeds up the tracking, while maintaining high accuracy. *CM-Predictor* was up to three times faster than *Clever-Tracker* and six times faster than *Direct-Tracker* when tracking/predicting cells. Our experiments show that *CM-Predictor* performs well with both 2D+t and 3D+t. It has been able to accurately estimate the motion of cells of a developing embryo over time. Regarding TDR results, *CM-Predictor* was up to 49% better than *Direct-Tracker* (in Exp1.2), up 31% better than *Clever-Tracker* (in Exp2.1) and up to 66% better than *TLM-Tracker* (in Exp2.2), using parameters $w = 3$, $pw = 0.3$ and $maxError = 0.9$. We conclude that *CM-Predictor* is an enhancement for cell tracking, serving its purpose of improving and speeding up the detection of the motions of cells.

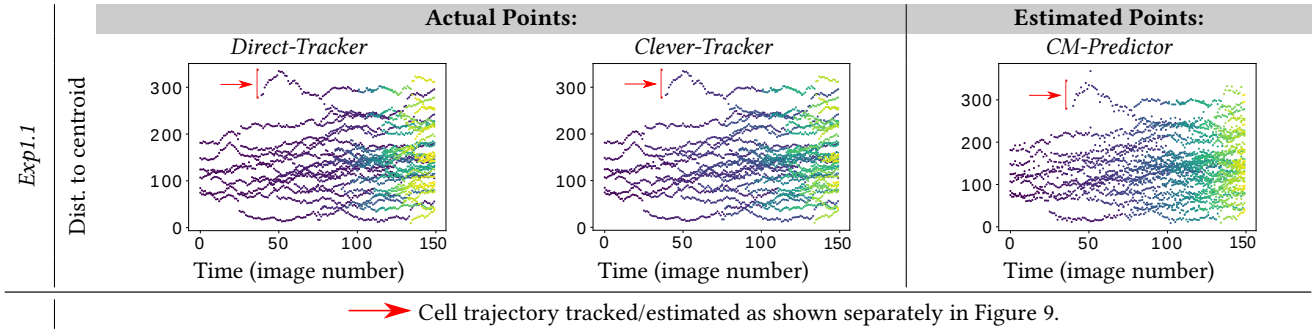

Figure 10: Cell trajectories for experiments Exp1.1.

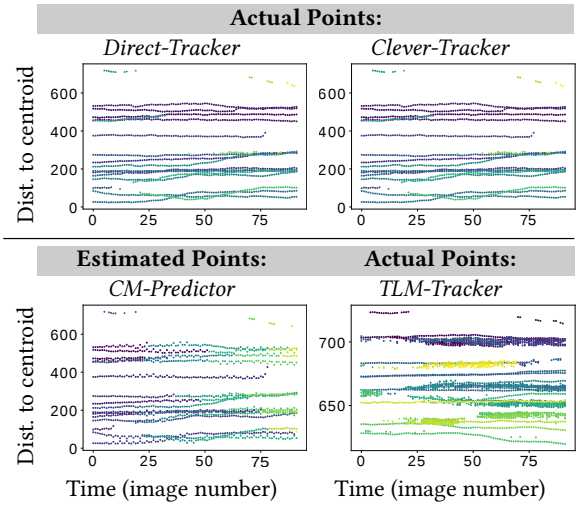

Figure 11: Cell trajectories for experiment Exp2.1.

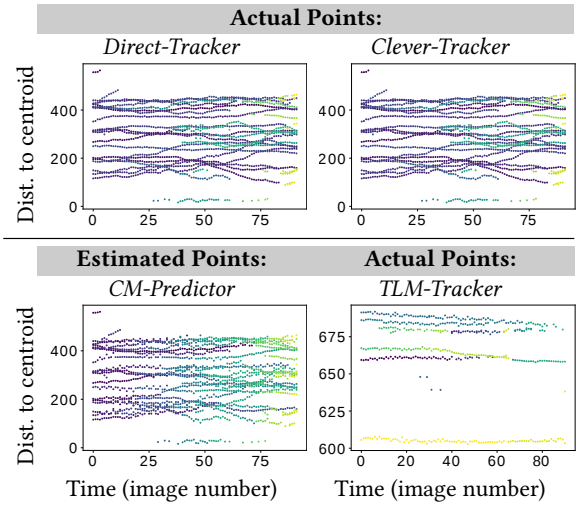

Figure 12: Cell trajectories for experiment Exp2.2.

ACKNOWLEDGMENT

This work was supported by grants #2016/17078-0 from the São Paulo Research Foundation (FAPESP), #88881.13068/2016-01 from Capes-PDSE, Capes and CNPq.

REFERENCES

[1] M. Abdel-Akher, A. Selim, and M. M. Aly. 2015. Initialnum-flow analysis based on Lagrange polynomial approximation for efficient quasi-static time-series simulation. *IET Generation, Transmission Distribution* 9, 16 (2015), 2768–2774. https://doi.org/10.1049/iet-gtd.2015.0866

[2] F. AlMuhisen, N. Durand, and M. Quafafou. 2018. Detecting behavior types of moving object trajectories. *International Journal of Data Science and Analytics* 5, 2 (Mar 2018), 169–187. https://doi.org/10.1007/s41060-017-0076-8

[3] A. D. Balomenos et al. 2017. Image analysis driven single-cell analytics for systems microbiology. *BMC Systems Biology* 11, 1 (2017), 43:1–43:21. https://doi.org/10.1186/s12918-017-0399-z

[4] A. D. Balomenos, P. Tsakanikas, and E. S. Manolakos. 2015. Tracking single-cells in overcrowded bacterial colonies. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 6473–6476. https://doi.org/10.1109/EMBC.2015.7319875

[5] A. Chakraborty and A. K. Roy-Chowdhury. 2015. Context aware spatio-temporal cell tracking in densely packed multilayer tissues. *Medical Image Analysis* 19, 1 (2015), 149 – 163. https://doi.org/10.1016/j.media.2014.09.008

[6] A. Elfwing, Y. LeMarc, J. Baranyi, and A. Ballagi. 2004. Observing Growth and Division of Large Numbers of Individual Bacteria by Image Analysis. *Applied and Environmental Microbiology* 70, 2 (2004), 675–678. https://doi.org/10.1128/AEM.70.2.675-678.2004

[7] T. He, H. Mao, J. Guo, and Z. Yi. 2017. Cell tracking using deep neural networks with multi-task learning. *Image and Vision Computing* 60 (2017), 142–153. https://doi.org/10.1016/j.imavis.2016.11.010

[8] O. Hilsenbeck et al. 2016. Software tools for single-cell tracking and quantification of cellular and molecular properties. *Nature Biotechnology* 34 (07 2016), 703–706.

[9] W. Jiuqing, C. Xu, and Z. Xianhang. 2017. Cell tracking via Structured Prediction and Learning. *Machine Vision and Applications* 28, 8 (Nov 2017), 859–874. https://doi.org/10.1007/s00138-017-0872-0

[10] J. Klein, S. Leupold, I. Biegler, R. Biedendieck, R. Münch, and D. Jahn. 2012. TLM-Tracker: software for cell segmentation, tracking and lineage analysis in time-lapse microscopy movies. *Bioinformatics* 28, 17 (2012), 2276–2277. https://doi.org/10.1093/bioinformatics/bts424

[11] Can Fahrettin Koyuncu, Ece Akhan, Tulin Ersahin, Rengul Cetin-Atalay, and Cigdem Gunduz-Demir. [n. d.]. Iterative h-minima-based marker-controlled watershed for cell nucleus segmentation. *Cytometry Part A* 89, 4 ([n. d.]), 338–349. https://doi.org/10.1002/cyto.a.22824

[12] Y. Li, F. Rose, F. di Pietro, X. Morin, and A. Genovesio. 2016. Detection and tracking of overlapping cell nuclei for large scale mitosis analyses. *BMC Bioinformatics* 17 (2016), 183. https://doi.org/10.1186/s12859-016-1030-9

[13] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. López. 2000. Indexing the Positions of Continuously Moving Objects. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*. 331–342. https://doi.org/10.1145/342009.335427

[14] J. Stegmaier, J. C. Otte, A. Kobitski, A. Bartschat, A. Garcia, G. U. Nienhaus, U. Strähle, and R. Mikut. 2014. Fast Segmentation of Stained Nuclei in Terabyte-Scale, Time Resolved 3D Microscopy Image Stacks. *PLOS ONE* 9, 2 (02 2014), 1–11. https://doi.org/10.1371/journal.pone.0090036

[15] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. 2004. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 611–622. https://doi.org/10.1145/1007568.1007637

[16] V. Ulman et al. 2017. An objective comparison of cell-tracking algorithms. *Nature Methods* 14 (2017), 1141–1152. https://doi.org/10.1038/nmeth.4473

[17] H. Xu, C. Lu, and M. Mandal. 2014. An Efficient Technique for Nuclei Segmentation Based on Ellipse Descriptor Analysis and Improved Seed Detection Algorithm. *IEEE Journal of Biomedical and Health Informatics* 18, 5 (Sept 2014), 1729–1741. https://doi.org/10.1109/JBHI.2013.2297030