Establishing Trajectories of Moving Objects without Identities: the Intricacies of Cell Tracking and a Solution

Mirela T. Cazzolato[a], Agma J. M. Traina[a], Klemens Böhm[b]

[a]*Institute of Mathematics and Computer Science*
University of São Paulo, USP - São Carlos, Brazil
[b]*Institute for Program Structures and Data Organisation*
Karlsruhe Institute of Technology, KIT - Karlsruhe, Germany

Abstract

Storing, querying, predicting, and interpolating trajectories of moving objects is a topic which the database community has studied for decades. We study a new variant of this problem in this article: We deal with a set of moving objects which do not have an identity, i.e., one does not know whether an object is identical to one observed earlier at another position. Our use case is a stream of images of cells of developing embryos. There exist so-called tracking tools. They match cells in such image sequences, to build trajectory vectors. However, these trackers have certain weaknesses, including counter-intuitive parameters and the expectation of users manually correcting trajectories. In this paper, we propose fully automatic tracking algorithms. They rely on space partitioning heuristics to match cells. This gives way to much cheaper data-analysis pipelines, as we will explain. We also propose two algorithms predicting the next positions of cells, given earlier ones. Experiments over 12 datasets show that our new approaches reduce the execution time by up to 7.8 times for tracking and 6.2 times for prediction. Prediction quality increases by up to 5.6% over the best tracker.

Keywords:
Moving objects, trajectory prediction, data stream, image, cell tracking.

Email addresses: `mirelac@usp.br` (Mirela T. Cazzolato), `agma@icmc.usp.br` (Agma J. M. Traina), `klemens.boehm@kit.edu` (Klemens Böhm)

1. Introduction

Problem Statement. Moving objects are a concept that the database community has investigated heavily, be it at the conceptual level, be it at the storage level. Such entities commonly are cars, trucks, airplanes, ships, or mobile phone users [1, 2, 3, 4, 5, 6, 7]. Working with the trajectories of moving objects (by storing, querying, predicting, interpolating them) has been a topic the database community has devoted attention to for a long time [8, 9, 10, 11, 12]. These are elementary problems, and existing solutions have also been tightly integrated into database engines, to serve as a building block for sophisticated data-analysis functionality [1, 2, 13]. Uncertainty of movements has been studied as well [1, 14, 15, 16, 17]. In this work, we study a new variant of the problem: Our objective is to establish the trajectories *of sets of objects* under uncertainty. More specifically, there is a set of objects which do not have identities, there are sequences of object positions, and the respective trajectories are sought. A prominent kind of element of such a sequence is 2D or 3D images of moving objects which one cannot tell apart from each other. Different variants of the problem exist. For instance, it may be the case that the set of objects is fixed, or that new objects may come into existence over time or vanish. In this article, we focus on the later, more complex version of the problem.

The problem is generic and occurs (or is about to occur) in various scientific disciplines, for instance:

- Nanorobotics (where robots are tiny, with very limited capabilities, and cannot identify themselves)
- Materials sciences (think of dislocations and their intricacies, which currently are not well understood)
- Ecology (when tracking animals without having marked them explicitly)
- Biology at a larger scale (when particles are inserted into an organism and then tracked with modern albeit simple methods)
- Cell biology, as we will explain right away.

From among this list, the problem has already been studied explicitly in cell biology to some extent, and benchmark data is available [18, 19]. This instantiation of the problem is of high practical importance, and it will be our use case throughout this paper. Clearly, the problem may have different peculiarities in those various domains, but covering this systematically across domains is a larger effort that would exceed the scope of one publication.

In concrete terms, we aim to identify and match cells in sequences of images that depict developing embryos over time [20, 21]. To illustrate, Figure 1 (a) features a sequence of microscopic images from an embryo. The goal is to construct cell trajectories by matching points corresponding to the same cell in subsequent images, so-called cell-to-cell correspondences, cf. Figure 1 (b). This task is commonly referred to as cell tracking. Algorithms capturing cell movement and splitting clearly are helpful when studying biological processes [22, 23] and understanding the mechanobiology of cells [24, 18]. Cells in sequences of images are a specific type of moving objects. We study how to use their spatial locations and possible movements, captured from the images, to identify cell correspondences over time.

(a) Example of a developing embryo

(b) Example of cells moving

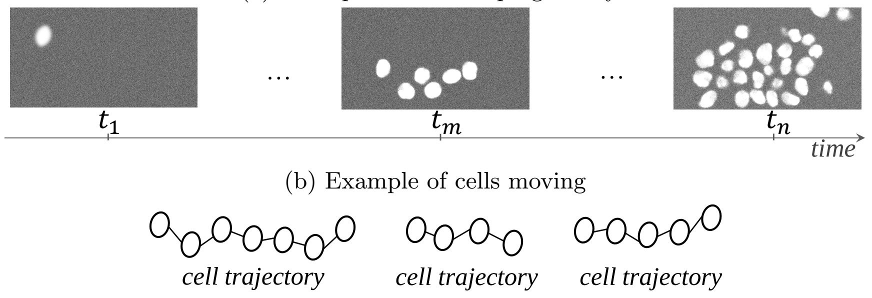

Figure 1: Sequence of images with developing cells.

Difficulties. Figure 2 depicts the conventional cell tracking pipeline [21, 25, 26, 20], the steps of existing methods that process sequences of microscopic images. The pipeline encompasses (i) image acquisition, (ii) data transfer to a computer, (iii) image processing to separate the particles of interest from the background and to detect their positions, and finally (iv) cell tracking. Steps (i) to (iii) are computationally costly, and avoiding them in some iterations can speed up the actual cell tracking. Next, various existing trackers work only semi-automatically, relying on users to manually correct or complete cell-to-cell correspondences [20, 27]. On the other hand, existing fully automated approaches tend to expose/let users configure unintuitive parameters related to algorithm internals [28, 29]. They also tend to lack on precision [21].

We propose the so-called prediction pipeline to track cells from just a few iterations, and we predict cell positions when some steps, including image acquisition, data transfer, and image processing, are omitted. The main difficulties behind our proposal are:

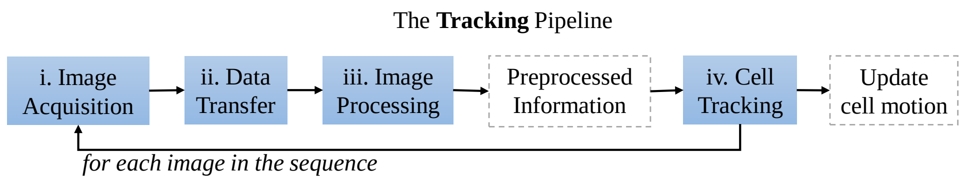

Figure 2: The tracking pipeline to perform cell tracking.

- **Trajectory accuracy:** The cell trajectories computed must be as close as possible to the actual ones.
- **Model updating:** An automated approach must monitor the prediction error while establishing cell trajectories, and the prediction model needs to be updated whenever necessary to keep high precision. When updating the prediction model, discarding the entire current model would be a simple, obvious solution, but this might eliminate more points than necessary. Instead, our approach replaces only some of the points used to build the model to continue predicting.
- **Modularity:** The processing pipeline needs to be flexible. In particular, it uses existing implementations for tasks such as image segmentation and cell detection. We expect new, improved implementations to become available over time. Accordingly, trackers and predictors should be orthogonal to every other component of the pipeline. We target at tracking and prediction pipelines that are modular, i.e., require minimal parametrization.
- **Automation:** The solution must be fully automatic, thus not relying on manual correction of motion vectors constructed with actual and predicted cell points.

Contributions. In this work, we propose two prediction pipelines to establish cell movement. The pipelines replace individual steps of the conventional pipeline in Figure 2. Our first pipeline processes a subsequent batch of images and predicts cell positions for the next timestamps. It monitors the error and renews the prediction model by reading and processing another batch of images. This pipeline requires as parameters the batch size, the maximum error allowed before updating the prediction model, the share of the points to replace in this case, and a distance threshold to decide for a match. The second prediction pipeline reads two non-consecutive intervals of images and estimates the intermediary cell positions. By estimating intermediary cell positions, this pipeline can improve the description of the cell motion. In contrast to the trackers featured in the literature, this pipeline requires only one parameter, the batch size. We argue that this parameter is intuitive to set, since it refers to the number of time steps to be predicted.

In an earlier, preliminary version of this study [30], we already introduced the notion of 'prediction pipeline'. In this current work, we now propose new prediction pipelines and study how and to which extent different movement models lead to better prediction results. We also propose fully automatic tracking algorithms to serve as a baseline for cell tracking. These algorithms rely on space partitioning heuristics to match cells. The experimental analysis over twelve different datasets shows that our approach can predict cell motion with high precision. The best prediction configuration achieves up to 5.6% better results than the best tracking algorithm, and up to 19% better results than the best competitor. Our proposed predictor is also up to 26% faster than the best cell tracking algorithm. All our algorithms are open-source to allow reproducibility and are available in the Git repository `https://github.com/mtcazzolato/tapcells`.

Paper Outline. Section 2 describes preliminary concepts. Section 3 reviews the related work. Section 4 introduces tracking algorithms to be used as reference points. Section 5 proposes the prediction pipelines. Section 6 features the objectives, datasets and methods employed in the experimental analysis. Section 7 covers experiments. Section 8 concludes this work.

2. Terminology and Background

This section covers the terminology and fundamental concepts behind cell motion tracking and prediction.

Moving Objects. In the context of this article, a cell is a moving object with a spatial location in a 2D or 3D space. Each cell moves through time. A cell can split. This generates two other cells. Analyzing and querying moving objects from different domains, combined with their concise representation and intrinsic uncertainty, have been intensively studied by the database community along the years [2, 31, 3, 16, 17]. Moving objects come from different application scenarios. They often are, say, pedestrians, cars, and boats [32, 33, 10, 11, 12]. The motion patterns of moving objects can be used for different tasks, including prediction, evaluation of queries, function modeling, and event classification [34, 35, 4, 36]. Also, in video-based object tracking, existing work employed deep-learning solutions for object tracking, combining object detection with tracking [37]. For instance, in [38] the authors combine appearence and motion features in a multi-scale Siamese atrous CNN for single-object tracking. In this work, we represent cells as a specific kind of moving object, composed of trajectories of points moving under uncertainty. Cells are indistinguishable, suffer from unknown transformations that deform their shape, move, grow and push each other. The related work also assumes such characteristics when establishing cell-to-cell correspondences from frame to frame. Accordingly, we propose generic methods that can accurately construct cell trajectories based on their positions, independently of additional cell property information. We track cell positions along time and take advantage of the movement of cells to predict their future positions.

Nomenclature. A microscopic image depicts one or more *cells* of the *embryo* at a timestamp t. Each cell has different spatial coordinates. This gives way to sequences of images depicting the development of an embryo over time. Cells split over time, resulting in new cells, as the embryo is developing. This manifests itself in the sequence of images. The type, quality, and configuration of the image-acquisition equipment determine the characteristics of the data, including image channels (e.g., RGB color channels), intervals of time between subsequent images, and image resolution. For ease of exposition in this article, we assume that images come in subsequent and equidistant time intervals. Equidistance also means that there is a constant frame rate. Different *cell types* can present distinct movement, shape, number and size, depending on their role in the studied organism [24, 39]. Equipment can generate sequences of 2D or 3D images, referred to as `2D+t` and `3D+t` sequences, respectively. A *segmentation algorithm* separates the visual image content into cells and background [39, 40]. In this work, we use the segmentation approach from [41], which provides cell positions at each image from the sequence, as we focus on the cell tracking task. A cell does not have an individual structure or identity that would make it discernible from other cells. Instead, each cell has the corresponding timestamp t and a spatial location detected by the segmentation algorithm, which is called 'seed point' or simply 'seed'.

Cell Tracking and Prediction. The term 'cell tracking' has been used in the literature to refer to both the cell segmentation in images and cell-to-cell correspondences over time. Cell tracking is also referred to as cell association or particle linkage. In this work, we use the term *cell tracking* for algorithms that find cell-to-cell correspondences along time, carried out to yield trajectories or motion vectors. A *seed-to-cell* correspondence is a *match* of a new seed point with an existing cell. Figure 3 illustrates the matches over time. When a cell split occurs, the resulting cells have to be associated with their corresponding parent from the previous timestamps.

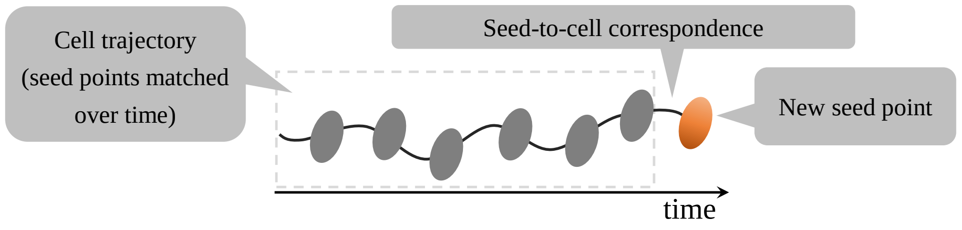
Cell trajectory (seed points matched over time)
Seed-to-cell correspondence
New seed point
time

Figure 3: Example seed points from a cell matched over time.

We call the algorithm that finds seed-to-cell correspondences 'tracker'. Existing trackers tend to find such correspondences with a k-NN search for every new seed point, with $k = 1$. The seed-to-cell correspondences of every cell depicted in the image sequence result in trajectory vectors that describe *cell motion*. In this work, *cell prediction* refers to the estimation of cell positions based on previous observed locations of the same cells. The prediction (or estimation) model relies on observed cell coordinates, obtained from actual images of the sequence. We call the algorithm that performs cell prediction a 'predictor'. The prediction pipeline is introduced in Section 5.

3. Related Work

This section covers related work that implements the Cell Tracking step of the tracking pipeline (see Figure 2). Performing segmentation and tracking of cells on label-free sequences of microscopic images can be very challenging. Manual tracking tools often are in use for this task [21]. Manual approaches come as point-and-click tools. Working with them can be laborious, error-prone, and heavily dependent on a specialist to perform the task [42, 20].

Recent literature reports automatic tracking techniques and the option of manual corrections with so-called semi-automatic tools. See Table 1, which presents features that are relevant for this work. Footnotes provide the URLs of the tools, last accessed in May of 2021. *Tracks* means that the algorithm provides a solution to solve seed-to-cell correspondences over time. *Predicts* means that the algorithm has the option to predict cell positions without processing all images. *Automatic* indicates whether the tool provides a fully-automatic solution, or whether it relies on the intervention of a user to do cell tracking. *Cell Type* tells us whether the approach is restricted to specific types of cells. Finally, the last column describes what the approach *Requires* for cell tracking. – The last row of Table 1 describes our proposal.

Table 1: Summary of cell tracking approaches.

Approach	Year	Tracks	Predicts	Automatic	Cell Type	Requires
[1] *SpotTracker* [28]	2005	✓	✗	✓	Single particles in noisy images.	`2D+t` sequences of fluorescence images.
[2] *Schnitzcell* [42]	2011	✓	✗	✓	E. Coli and B. Subtilis bacteria.	`2D+t` sequences of fluorescence and phase images.
[3] *iTrack4U* [25]	2013	✓	✗	✗	Not restricted	Manual annotation of cell centers in a few images from the sequence.
[4] *Cell-Tracker* [21]	2016	✓	✗	✓	Not restricted	`2D+t` sequences of fluorescence images.
[5] *ManualTracking* [27]	2017	✓	✗	✗	Not restricted	`2D+t` and `3D+t` sequences of images, and the manual annotation of each cell, in every image of the sequence.
[6] *MTrack2* [26]	2017	✓	✗	✓	Not restricted	`2D+t` sequences of black and white (8-bit) images.
[7] *TrackMate* [20]	2017	✓	✗	✗	Not restricted	`2D+t` sequences of grayscale images and the manual annotation of cells depicted in the image sequence.
[8] *Proposal:* ***Cell Prediction***	2021	✓	✓	✓	Not restricted	`2D+t` and `3D+t` sequences of images.

SpotTracker [28] is a Java-based ImageJ [43, 44] plugin optimized to

[1] *SpotTracker*: `http://bigwww.epfl.ch/sage/soft/spottracker/`
[2] *Schnitzcell*: `https://biii.eu/schnitzcells/`
[3] *iTrack4U*: `https://sites.google.com/site/itrack4usoftware/home/`
[4] *Cell-Tracker*: `http://celltracker.website/`
[5] *ManualTracking*: `http://rsb.info.nih.gov/ij/plugins/track/track.html`
[6] *MTrack2*: `https://imagej.net/MTrack2`
[7] *TrackMate*: `https://imagej.net/Manual_tracking_with_TrackMate`
[8] Our proposal is available at: `https://github.com/mtcazzolato/tapcells`

track a single particle over noisy image sequences. *SpotTracker*'s implementation, which is available, copes with sequences of `2D+t` images. It has three main steps: (i) sequence alignment, (ii) spot enhancing filtering and (iii) tracking. Step (i) compensates for possible movements performed during the acquisition in a preprocessing step. Step (ii) is the application of noise reduction algorithms and is important to *SpotTracker* due to the noisy data the tool intends to work with. Step (iii) performs tracking, which is restricted to a single particle.

Schnitzcell [42] is a MATLAB tool to work with *Bacillus* and *Escherichia Coli* (*E.Coli*) cells. *Schnitzcell* can (i) create cell segmentation masks, (ii) track cells and division events by similarity, and (iii) provide result files and facilitate (optional) manual corrections of intermediary results. The tracker matches points in successive pairs of frames. *Schnitzcell* requires both phase and fluorescent images as input, restricting the use of the tool for other types of images.

iTrack4U [25] is a Java-based tool for cell tracking. Its cell segmentation focuses on video microscopy, combines mean-shift and phase-contrast approaches. It is based on the study carried out in [45]. The tracking step requires the manual annotation of each cell center on the last image of the sequence, followed by a backward tracking using an approximation of each cell border as a polygon.

Cell-Tracker [21] is a MATLAB-based tool for the segmentation and cell tracking of sequences of images and videos. *Cell-Tracker* can perform automatic, semi-automatic, and manual tracking of cells. The semi-automatic tracking takes as input manually-marked cells from a few images, with a rectangle as a cell contour. For manual tracking, the user must click on all cells of each image. *Cell-Tracker* merges the image segmentation step with the tracking algorithm proposed in [46]. They track cells detected from the beginning of the sequence, match cells of adjacent images by proximity, but do not consider varying numbers of cells over time due to, say, cell splits.

ManualTracking [27] is a Java-based tool implemented as a plugin for ImageJ. The tool takes as input the center of each cell or particle, manually marked by the users at each image, making the usability laborious and prone to errors. *MTrack2* [26] is a Java-based tool implemented as a plugin for ImageJ. The tool requires the images to be preprocessed and converted in black and white (8-bit images) to work. It automatically detects the positions of the objects in every image and constructs object trajectories by matching them in subsequent images, based on their spatial location. The tool works with generic objects and particles and supports only `2D+t` sequences of images [47].

Finally, *TrackMate* [20] is a Java-based tool initially implemented as a plugin for Fiji. *TrackMate* provides tools for the entire tracking pipeline. It segments cells requiring the input parameters pixel width and height, voxel depth, and time interval. The tool has four tracking options [48]: (i) manual tracking; (ii) a linear motion estimator, which maps a point to its next position; (iii) a two-phase tracking approach, which starts by providing an approximate match between points and then corrects the match by looking for splitting events (cell divisions); and (iv) a k-NN-based search, which looks for the nearest point of a given object position, but ignores particle splitting. Although manual intervention is optional, *TrackMate* usually requires the user to adjust the segmentation and to track accurate results manually.

None of the approaches described in this section performs predictions to establish the trajectories of cells. Instead, they rely on the traditional pipeline of cell tracking. This means that every image from the sequence has to be processed in order to obtain cell motion. We will compare the results of our proposal to *Cell-Tracker* and *MTrack2*. Both perform fully automatic tracking, do not restrict the cell type to a single particle, and do not require additional information, such as phase images required by *Schnitzcell*. Most existing work that tracks cells looks for the closest matching points of subsequent images. In the next section, we describe algorithms for the tracking pipeline that follow the heuristic of matching the closest points. These algorithms will serve as a baseline for our experimental validation.

4. Algorithms for Cell Tracking

In this section, we present tracking algorithms to serve as reference points for the proposed prediction algorithms. The steps listed in this section are roughly the same for the various trackers from related work, with differences such as the option of manual corrections, the segmentation of all images before constructing trajectory vectors, or the heuristic employed to decide whether cells match. Unlike our proposed prediction approaches, which avoid processing all images from the input sequence to establish cell motion, trackers use the cell positions detected in every image from the sequence to construct cell trajectories.

In line with the tracking pipeline from Section 2, trackers compare pairs of seed points detected in every image, decide for matches and output the cells as sequences of seed points matched over time. As input, the algorithms receive:

- I: the sequence of images to process, and
- P: the set of specific parameters used by the tracker.

Algorithm 1 lists the basic steps for cell tracking, constructing a trajectory vector for every cell. The tracker creates a new cell for each detected cell in the first image from the sequence (Lines 2 and 3) and increments the timestamp t. For each remaining image of the sequence (Line 5), the algorithm segments the images to get the cell positions (Line 6). Function *MatchSeedsToCell* matches every new seed point with an existing cell (Line 7). Next, we propose two algorithms (*Direct-Tracker* and *Clever-Tracker*) that implement the function *MatchSeedsToCell*.

Algorithm 1: Cell Tracking

```
Input  : $I$: Sequence of images to process
         $P$ : Set of parameters used for tracking
Output: cells: cells with seed points seen up to time $t$
1 begin
      // Initialize vector with the initial cells
2     $S_1 \leftarrow$ Segment first image in $I$ and get detected seed points;
3     Create a new cell for each seed point in $S_1$ and add them to cells;
4     $t \leftarrow 2$;
5     foreach remaining image $i$ in the sequence $I$ do
6         $S_t \leftarrow$ Segment image $i \in I$ and get detected seed points;
7         cells $\leftarrow$ MatchSeedsToCell(cells, $S_{t-1}$, $S_t$, $P$);
8         Increment timestamp $t$;
9     return cells;
```

4.1. Direct-Tracker and Clever-Tracker

Both *Direct-Tracker* and *Clever-Tracker* find the seed-to-cell correspondences of points from $t-1$ and t with the *MatchSeedsToCell* function, returning the cells containing the new seed points. The approaches receive as input:

- S_{t-1}: list of seed points from time $t-1$
- S_t: list of seed points from time t
- *cells*: the set of cells observed so far, and
- *th*: distance threshold.

The set *cells* represents every cell as a sequence of seed points that were matched along the sequence of images. Observe that all seed points in S_{t-1}

already belong to existing cells. Let c^i be the cell numbered with i, and let s^i_{t-1} be its last seed point. By matching s^i_{t-1} and s_t, we add s_t to c^i, and s_t becomes the last seed point of the cell. To avoid the comparison of *each* possible pair of seed points s_{t-1} and s_t to find the best match, *Direct-Tracker* and *Clever-Tracker* use a distance-threshold value th. This parameter restricts the search space in order to speed up cell matching. The algorithms compare s_t with only seed points within the search space defined by th.

Algorithm 2 is *Direct-Tracker*, which greedily searches for the matching point of each new seed point at time t (Lines 2–4). If the distance between a pair of points is less than the distance threshold ($\delta(s_{t-1}s_t) \leq th$), *Direct-Tracker* decides for a match (Lines 5 and 6). We use the Euclidean Distance (L_2) as the metric δ. If there is more than one match for a cell, the interpretation of the algorithm is that a split occurred. To keep track of this, *Direct-Tracker* sums the number of matches of each existing cell (Line 8) and adds the pair of seed points to a set of matching pairs (Lines 7–10). If s_t does not have a matching point from time $t-1$, the algorithm creates a new cell (Lines 11 and 12). *Direct-Tracker* adds the new seed points to the corresponding cells (Lines 13 and 14). If a cell c has more than one matching seed point, the approach understands that it has split and creates a new cell for every matching seed point, marking cell c as the parent of the new cells. *Direct-Tracker* matches the query cell position with the first seed point that is inside the given threshold distance. This matching procedure is random only in exceptional cases. Namely, *Direct-Tracker* depends on the order of the seed points, and the detected cell positions appear more or less in the same order as they appear in the previous image, depending on the segmentation algorithm employed in the tracking pipeline. Observe that *Direct-Tracker* does not ensure that s_{t-1} is the nearest seed point of s_t. *Clever-Tracker* in turn, which we describe next, features this.

Algorithm 3 is *Clever-Tracker*. For each new seed point, *Clever-Tracker* calls function *FindNearestSeed* to find its matching seed point from time $t-1$ (Lines 1–3). *FindNearestSeed* searches for the new seed point closest to s_{t-1}, such that $\delta(s_{t-1}, s_t) \leq th$ (Lines 13–20). If there is a match for s_t, *Clever-Tracker* adds it to the list of matching pairs and sums the number of matches of the corresponding cell (Lines 4-7). Otherwise, the algorithm creates a new cell for s_t (Lines 8 and 9). Finally, *Clever-Tracker* adds the matching correspondences to the existing cells (Lines 10 and 11). Like *Direct-Tracker*, if a cell c has more than one matching seed point, *Clever-Tracker* understands that it has split and creates a new cell for every matching seed point, marking cell c as the parent of the new cell.

Algorithm 2: *Direct-Tracker* implements *MatchSeedsToCell*

```
1  Function MatchSeedsToCell(cells, $S_{t-1}$, $S_t$, th):
2      foreach seed $s_t \in S_t$ do
3          hasMatch $\leftarrow$ 0;
4          foreach seed $s_{t-1} \in S_{t-1}$ do
5              distance $\leftarrow \delta(s_{t-1}, s_t)$;
6              if ((distance $\leq$ th) AND (hasMatch = 0)) then
7                  Let c be the cell in cells containing $s_{t-1}$;
8                  c.numberOfMatches++;
9                  hasMatch $\leftarrow$ 1;
                   // Add the pair of seeds to a list of matches
10                 matchingPairs.add($s_{t-1}$, $s_t$);
11         if (hasMatch = 0) then
12             Create a new cell for $s_t$ in cells;
13     foreach pair $(s_{t-1}, s_t)$ in matchingPairs do
14         cells $\leftarrow$ AddMatch(cells, $s_{t-1}$, $s_t$);
15     return cells
```

The task of finding the nearest existing seed point to a new seed point is a k-NN search, with $k = 1$. For this task, *Direct-Tracker* and *Clever-Tracker* rely on the distance threshold th. Parameter th restricts the search space and speeds up cell matching since it avoids comparing the new seed point to every existing cell. However, it is *not trivial to fix the value for th* without performing a round of experiments for parameter calibration. Indexing structures can help to find the nearest seeds in a straightforward manner, as described next.

We also present the index version of *Direct-Tracker* and *Clever-Tracker*, which takes advantage of the spatial location of cell points to match cells. The tracker outputs the cells as sequences of seed points. The *input parameters* of the index-based tracker are the same as *Direct-Tracker* and *Clever-Tracker*, except for th, which is no longer required.

Algorithm 4 gives the pseudocode of the matching seeds process. In this context, *tree* can be any hierarchical indexing structure. We use both KDTree and BallTree indexes, yielding the so-called *KDT-Tracker* and *BLT-Tracker* algorithms, respectively. For each new seed point from time t, the algorithm searches for the nearest seed point from time $t - 1$ using a tree structure (Lines 3 and 4). The algorithm adds the new seed point to the matched cell (Line 4), and it adds the new seed points from time t to the index (Line 5). As a result, the algorithm returns the updated cells, with

Algorithm 3: *Clever-Tracker* implements *MatchSeedsToCell*

```
1  Function MatchSeedsToCell($cells$, $S_{t-1}$, $S_t$, $th$):
2      foreach seed $s_t \in S_t$ do
3          $s_{t-1} \leftarrow FindNearestSeed(s_t, S_{t-1}, th, 0)$;
4          if ($s_{t-1}$ is not null) then
5              Let $c$ be the cell in $cells$ containing $s_{t-1}$;
6              c.numberOfMatches++;
               // Add the pair of seeds to a list of matches
7              matchingPairs.add($s_{t-1}$, $s_t$);
8          else
9              Create a new cell for $s_t$ in $cells$;
10     foreach pair $(s_{t-1}, s_t)$ in matchingPairs do
11         $cells \leftarrow AddMatch$(cells, $s_{t-1}$, $s_t$);
12     return $cells$
13 Function FindNearestSeed($s_q$, $S$, $th$, $hasMatch$):
14     mindist $\leftarrow +\infty$; // Initialize minimum distance
15     foreach seed $s_i \in S$ do
16         distance $\leftarrow \delta(s_q, s_i)$;
17         if (distance < mindist AND distance $\leq th$) then
18             mindist $\leftarrow$ distance;
19             match = $s_i$;
20     return match
```

all seed points seen up to time t (Line 6).

Direct-Tracker and *Clever-Tracker* rely on the distance threshold parameter th to limit the search space when looking for matching cells. Otherwise, the algorithms would have to test every possible new cell matching, which would increase the execution time of the approaches. However, the approaches relying on th may not find the matching cell within the established region, making the results differ from the other implementations (*KDT-Tracker* and *BLT-Tracker*). Accordingly, when *Clever-Tracker* has $th = +\infty$, the tracking results are the same as the index-based approaches.

The algorithms presented so far follow the tracking pipeline. Next, we propose the prediction pipeline, which avoids processing all images from the sequence to establish cell motion.

5. Proposed Methods for Cell Prediction

In this section, we discuss how to establish cell motion without processing every image from the sequence. Accordingly, we propose two pipelines

Algorithm 4: *MatchSeedsToCell* using an index

```
1 Function MatchSeedsToCell($cells$, $S_{t-1}$, $S_t$, $tree$):
2     Update $tree$ with seed points from time $t-1$;
3     foreach $seed\ s_t \in S_t$ do
          // tree can be a KDTree or a BallTree structure
4         match $\leftarrow$ $tree.getNearestPoint(s_t)$;
5         $cells \leftarrow AddMatch(cells, s_t, \text{match})$;
6     return $cells$
```

for cell prediction: predicting future positions of cells and estimating cell positions between different moments in time. We discuss each pipeline and the design space of prediction algorithms in the following subsections.

5.1. The Cell Prediction Pipeline

Figure 4 presents the pipeline to predict future cell positions. The pipeline receives a sequence of images as input, and it outputs cell points matched and predicted over time. Step (i) includes acquiring, transferring, and processing a subset of images before cell prediction occurs. This step repeats (ii) until there are enough points to make a prediction. The pipeline uses the spatial positions of cells detected from images (iii) to predict their next positions (iv). When predicting, the pipeline omits steps of image acquisition, transfer, and segmentation. When prediction quality goes down, the pipeline returns to image acquisition (Step (i) of the pipeline). Step (v) updates the cell motion, adding the predicted points to the corresponding cells. The entire process outputs cell motion from detected and predicted cell points. A specific design of the prediction pipeline relies on options to define, for instance, different metrics to establish the detection quality, cell motion models, and the distance threshold used for matching. In the following, we describe one instantiation of the prediction pipeline: the *CM-Predictor* algorithm.

The CM-Predictor Algorithm

Algorithm *CM-Predictor* implements the cell prediction pipeline (see Figure 4). It has the following *input parameters*:

- I: the sequence of images to process
- th: a distance threshold
- w: the size of the window
- *maxError*: maximum error tolerated for predictions

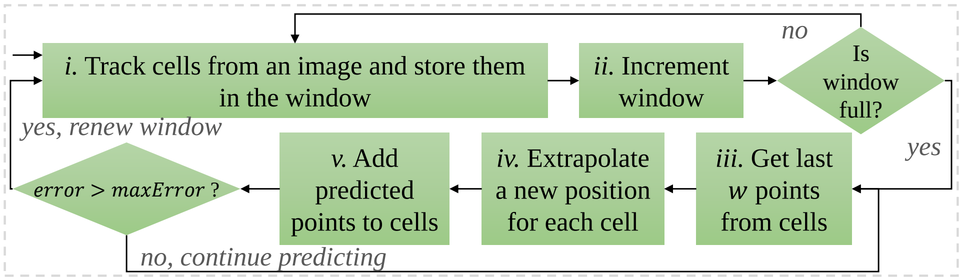
i. Track cells from an image and store them in the window
ii. Increment window
Is window full?
no
yes
iii. Get last w points from cells
iv. Extrapolate a new position for each cell
v. Add predicted points to cells
error > maxError ?
yes, renew window
no, continue predicting

Figure 4: The prediction pipeline.

- pw: portion of the window to discard when error is too large.

Algorithm 5 details *CM-Predictor*. It starts by initializing the variables *windowSize* and *cells*, and performing cell tracking and building the window of points (Lines 2–6). Any tracker can be used in this step, such as *Direct-Tracker*, *Clever-Tracker*, *BLT-Tracker*, or *KDT-Tracker*. When the window is full ($windowSize \geq w$), we assume that there are enough points to extrapolate, and motion prediction starts (Lines 7–8). For each cell in the vector *cells*, *CM-Predictor* gets its w last seed points (Lines 10–12). The algorithm extrapolates points, obtaining the predicted coordinates of the next position of that cell as a new seed point (Line 13). *CM-Predictor* adds the new seed point to the corresponding cell (Line 14). This approach also checks the error in each iteration step (Line 15). Let $\mu\Delta_c$ and $\sigma\Delta_c$ be respectively the average and standard deviation of the spatial displacement of cell c. For each cell, if the predicted point is *not* within $\mu\Delta_c + \sigma\Delta_c$ of distance to the last cell point, *CM-Predictor* increments the *error* variable. Additionally to the spatial displacement, we can check the change of direction in the movement. For this, we can compute $\mu\theta_c$ and $\sigma\theta_c$ as respectively the average and standard deviation of the angle difference of cell c. We evaluate the impact of the alternative error measuring experimentally. If *error* is not acceptable, i.e., exceeds the *maxError* threshold, the predictor removes the oldest $pw\%$ points in the window (Line 16). The algorithm then goes to the main cycle, performs tracking for the next images (from time $t+1$) until the window has w images again. If the error is acceptable, *CM-Predictor* continues extrapolating points at time $t+1$.

CM-Predictor predicts the position of a new seed point based on the w last observed points of each cell. For this, we perform a Polynomial Regression over the points of each cell to obtain the best model for *motion prediction*. We evaluate the impact of different polynomial degrees in cell prediction experimentally. We will also study the impact of weighting cell

Algorithm 5: *CM-Predictor*

```
Input  : I : Sequence of images to process
         th: distance threshold
         w: window size
         pw: portion of the window discarded
         maxError: maximum error allowed
Output: cells: cells with seed points seen up to time t
1  begin
2      initialization;
3      foreach instant $t_i$ in the sequence do
4          if (windowSize < w ) then
               // Perform cell tracking using Algorithm 1
5              Process image and track new seed points;
6              Increment windowSize;
7          else
8              cells, windowSize ← PredictCellPositions(cells, th, windowSize, pw,
                 maxError);
9      return cells;
10 Function PredictCellPositions(cells, th, windowSize, pw, maxError)
11     foreach cell c in cells do
12         S ← c.getSeeds(); // Get last w seed points from cell
           // Estimate coordinates of the next point
13         newSeed.xyz ← Extrapolate(S.x, S.y, S.z);
14         Add newSeed to cell c of cells;
15     if (error > maxError) then
16         Remove pw % of the oldest points in the window, update windowSize
             and process next image;
17     return cells, windowSize
```

points used to find the best polynomial, i.e., giving more importance to the most recent positions.

If the predicted cell position is within a distance of *th* from the last position, the prediction is correct; otherwise, this is a prediction error. When the error is higher than *maxError*, with $0 < maxError \leq 1$, *CM-Predictor* discards the oldest seed points of each cell that are in the window and are used for extrapolation. It then renews the prediction window with the new points collected from images. *CM-Predictor* tracks cells from images while the window of points is not full, and any tracking algorithm can be employed. We call the combinations of *CM-Predictor* with *KDT-Tracker* and *BLT-Tracker* *KDT-Predictor* and *BLT-Predictor*, respectively.

5.2. The Interleaved Tracking and Prediction Pipeline

In this subsection, we propose a new pipeline that tracks non-consecutive overlapping windows of points and interpolates cell positions. Figure 5 presents the pipeline to process non-consecutive windows of actual points and to interpolate intermediary points. The process repeats for each timestamp. The pipeline tracks cells from the first w images, skips w images, and tracks the cells from w more images (Steps (i–iii)). Step (iv) matches one cell from the first window with one from the other one, and Step (v) estimates the w points between the windows, for each cell. Next, the w last points become the first window of points, and the pipeline returns to Step (ii). The pipeline design includes different options to define, for instance, the matching criteria between cells of non-consecutive windows, and cell motion models to estimate cell positions. In the following, we describe one instantiation of the interleaved tracking and prediction pipeline: the *Int-Predictor* algorithm.

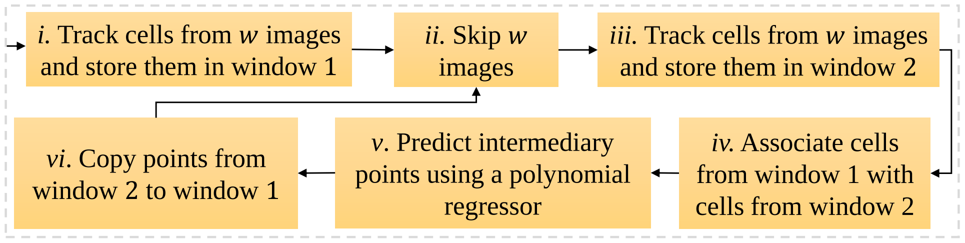
i. Track cells from *w* images and store them in window 1
ii. Skip *w* images
iii. Track cells from *w* images and store them in window 2
iv. Associate cells from window 1 with cells from window 2
v. Predict intermediary points using a polynomial regressor
vi. Copy points from window 2 to window 1

Figure 5: The Interleaved tracking and prediction pipeline.

The Int-Predictor Algorithm

Figure 6 illustrates the working of the *Int-Predictor* algorithm to implement the interleaved tracking and prediction pipeline. In the example, the window size is $w = 3$, and there are two cells at each time stamp. The predictor processes images and tracks cells of windows a and c, skipping window b. *Int-Predictor* matches cells from windows a and c. Then *Int-Predictor* estimates cell positions from the intermediary window b. In the next iteration, *Int-Predictor* processes images from window e and interpolates points in window d, using cells from c and e, and so on. The size of the window w can be 1, so the algorithm estimates every other point, or larger.

Algorithm 6 implements *Int-Predictor* and outputs motion vectors of cells. The *input parameters* of the algorithm are:

- I: the sequence of images to process

- w: the size of the window.

The algorithm starts by processing the first w points using a tracking algorithm (Lines 2–4). It then skips the images corresponding to the next window (Line 5), which will be estimated. The algorithm enters a loop for the remaining images in the sequence of images given as input (Line 6). While the second window is not full, *Int-Predictor* processes the images and tracks cells (Lines 7–9). After filling the second window, the predictor matches cells in the first window with the ones in the second window (Line 11). *Int-Predictor* then interpolates the intermediary points of each cell (Line 12). The employed interpolation function can vary, such as Lagrange's polynomials or linear interpolation. The predictor adds the cells to the motion vectors, and the second window becomes the first one (Lines 13–14). Finally, *Int-Predictor* skips w images and reads w points to fill the second window again (Line 15). As a result, the predictor returns the embryo with actual and estimated (interpolated) points.

Algorithm 6: *Int-Predictor*

```
Input  : $I$ : Sequence of images to process
         $w$: window size
Output: cells: cells with seed points seen up to time $t$
1  begin
2      initialization;
3      Read and process first $w$ images from $I$;
4      Track points and initialize $wind_1$ ;        // fill first window of points
5      $t \leftarrow w + 1$ ;          // increment timestamp to jump an empty window
6      foreach remaining image img in $I$ do
7          if ( $(t \% w)! = 0$ and $wind_2$ is not full ) then
8              Process img, track seed points and add them to $wind_2$;
9              Increment timestamp $t$;
10         else
11             Match cells in $wind_1$ to cells in $wind_2$;
12             Interpolate $w$ intermediary points of cells;
13             Add cells to cells;
14             Copy points of $wind_2$ to $wind_1$ and empty $wind_2$;
15             $t \leftarrow w + 1$ ;   // increment timestamp to jump an empty window
16     return cells
```

When setting the size of the window w, one should consider how much the cells move and how many cells are being tracked. If cells move considerably over time and/or the number (density) of cells is large, small windows should lead to better results than large ones.

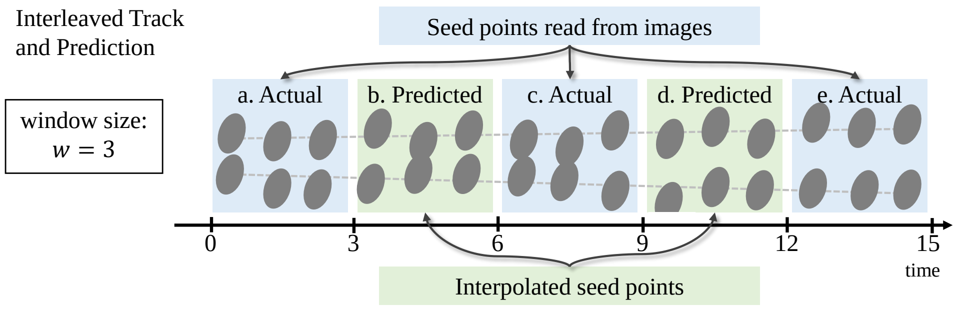

Figure 6: *Int-Predictor* reads two windows of points, matches the cells and interpolates the positions from the intermediary window.

In this section, we introduced cell prediction algorithms to obtain cell motion. In contrast to pure cell-tracking approaches, the proposed methods have the advantage of not processing all images from a sequence. In the next sections, we present the validation of the proposed algorithms. We show that the predictors can improve (or maintain) the cell motion quality, with a performance that is comparable.

6. Experimental Setup

In this section, we describe the objectives, datasets, methods, and settings of the experimental analysis.

6.1. Experimental Objectives

In the experimental analysis, we aim to show that the proposed predictors can accurately establish cell motion while processing only a few images from the input sequence. At the same time, the predictors must be able to output cell motion vectors that are very similar to the actual cell motion observed in the sequence of images. Accordingly, we present qualitative and quantitative results to confirm how good the proposed approaches are.

6.2. Datasets

We use benchmark datasets provided by the Cell Tracking Challenge [18, 19] for validation. Each set consists of the original images, ground truth annotations generated by specialists, and segmented images. Table 2 summarizes the datasets used in each experiment, with their original name, set number (each dataset has two sets of data), image type (which can be `2D+t` or `3D+t`), number of images, and number of image slices (for `3D+t` datasets). The last column shows the number of cells from the ground truth.

Table 2: Summary of the datasets used in this work.

ID	Dataset Name	Set	Type	#Images	#Slices	#CellsGT
Ds1	Fluo-N2DH-SIM+	01	2D+t	65	1	81
Ds2	Fluo-N2DH-SIM+	02	2D+t	150	1	117
Ds3	Fluo-N2DH-GOWT1	01	2D+t	92	1	27
Ds4	Fluo-N2DH-GOWT1	02	2D+t	92	1	50
Ds5	Fluo-N2DL-HeLa	01	2D+t	92	1	265
Ds6	Fluo-N2DL-HeLa	02	2D+t	92	1	675
Ds7	Fluo-N3DH-CHO	01	3D+t	92	5	27
Ds8	Fluo-N3DH-CHO	02	3D+t	92	5	24
Ds9	Fluo-N3DH-CE	01	3D+t	250	35	720
Ds10	Fluo-N3DH-CE	02	3D+t	250	31	724
Ds11	Fluo-N3DH-SIM+	01	3D+t	150	59	95
Ds12	Fluo-N3DH-SIM+	02	3D+t	80	59	107

6.3. Evaluated Algorithms, Implementation and Setup

Table 3 lists the proposed algorithms evaluated in the experimental analysis. We compare the results of our proposed algorithms with the existing tracking tools *Cell-Tracker* [21] and *MTrack2* [26], as explained in Section 3. Both of them can process the `2D+t` datasets from Table 2 without manual intervention. For the cell prediction, we performed experiments using Lagrange's Polynomials [49] and Polynomial Regression to find the best model to describe cell motion. We vary polynomial degrees as well as the parameters of the predictors, to obtain the best setting.

Table 3: Evaluated algorithms, acronyms, search approach used in cell matching, type (tracking or prediction), and input parameters.

Algorithm	Acr.	Search	Type	Parameters
Direct-Tracker	*DT*	Brute Force	Tracking	*th*
Clever-Tracker	*CT*	Brute Force	Tracking	*th*
CM-Predictor	*CMP*	Brute Force	Prediction	*th*, *maxError*, *pw*, *w*
KDT-Tracker	*KdT*	KDTree	Tracking	None
BLT-Tracker	*BlT*	BallTree	Tracking	None
KDT-Predictor	*KdP*	KDTree	Prediction	*th**, *maxError*, *pw*, *w*
BLT-Predictor	*BlP*	BallTree	Prediction	*th**, *maxError*, *pw*, *w*
Int-Predictor	*IntP*	KDTree	Prediction	*w*

The proposed algorithms were implemented in Python 3.7.1. We used the KDTree and BallTree implementations from the Scikit-learn[9] Python library. The experiments were performed in an Intel Core i7-4770 3.40GHz x8 CPU, 16GB RAM machine, running Fedora 29 OS. Algorithms are open-sourced to allow reproducibility and are available for download at the Git repository. We also make available the cell positions from every dataset used to evaluate the algorithms.

To obtain the cell positions, we employed the image segmentation algorithm *TWANG* [41] with its parameters set as *SpacingX*$=0.3$, *SpacingY*$=0.3$ and *SpacingZ*$=1$ for all experiments, obtained experimentally. All other settings were used as default.

The parameter setting was performed experimentally. We selected the parameters that yield high TDR results in all datasets (the average cases) for *CM-Predictor*. We have set the parameter values for the distance threshold and window size to the ones that have given way to the best results. For *Int-Predictor*, w affects the matching of the last cell position of the first window and the first cell position of the second window, after "jumping" w timestamps. In this case, the polynomial degree can be high, as long as *Int-Predictor* is able to establish the correspondence of cells after "jumping" w images. In contrast, for *KDT-Predictor* the polynomials are fitted over w points, but with a given degree. In this case, since the approach "extrapolates" the cell positions, low degrees tend to work better than high degrees. The parameter values of *CM-Predictor* are $w = 4$, $pw = 0.3$, $maxError = 0.3$, which were experimentally obtained. Table 4 lists the threshold values used for *Direct-Tracker*, *Clever-Tracker* and *CM-Predictor*.

Table 4: Threshold parameters used for *CM-Predictor*, *Direct-Tracker* and *Clever-Tracker*

Algorithm	**Threshold *th* per dataset**											
	1	2	3	4	5	6	7	8	9	10	11	12
Direct-Tracker	20	22	16	35	11	9	16	20	26	37	8	10
Clever-Tracker	65	22	16	29	54	9	26	20	47	40	34	12
CM-Predictor	20	22	16	35	11	9	16	20	26	37	8	10

For *Int-Predictor* (with Langrange's Polynomials), the best setting corresponds to window size $w = 5$, for all datasets. Table 5 lists the w parameter values and the polynomial degrees employed for *Int-Predictor* and *KDT-Predictor* with Polynomial Regression. Overall, low degrees yield the best results. For the weighted version of the predictors using Polynomial Regression, we employed linear weights, giving higher importance to the last cell

[9]Scikit-learn Python library for machine learning: https://scikit-learn.org/stable/

points in the prediction window.

Table 5: Polynomial degree and w parameter employed for *Int-Predictor* and *KDT-Predictor*.

Algorithm	**Parameter**	**Parameters per dataset**											
		1	2	3	4	5	6	7	8	9	10	11	12
Int-Predictor	w	8	4	9	5	9	5	9	7	10	10	10	8
KDT-Predictor	w	9	6	8	7	3	5	5	9	9	4	9	3
	Degree	2	2	2	2	2	2	2	3	2	2	3	2

6.4. Evaluation Measures

Measuring how much the cell trajectories "mix cells" is of significant importance, but we cannot measure such occurrences, due to the lack of labeled data (ground truth). However, the main point regarding the "cell mixing issue" is that, due to the nature of the problem, cells are indistinguishable and do not have a unique identifier for every cell in every image of the sequence, which would allow the estimation of "cell mixing" errors. Accordingly, we employ measures considering the available information (number of cells at every image and the total number of cells from the sequence). From these values, we compute two different quantitative metrics to evaluate the methods: *Total Cells Relative Error* and *Tracking Detection Rate* [50]. We also measure the *Runtime* of every approach.

6.4.1. Tracking Detection Rate (TDR)

A *tracking error* occurs if the algorithm erroneously creates a new cell instead of adding the corresponding seed point to an existing cell, or if the algorithm adds the new seed point to an existing cell when it should have created a new cell. Let n be the number of images in the sequence, and *True Positive* (TP) be the number of images with no tracking error, with $0 \leq TP \leq n$. Equation 1 introduces the Tracking Detection Rate [50].

$$TDR = \frac{TP}{n} \tag{1}$$

6.4.2. Total Cells Relative Error (TCRE)

Let TC_{tr} be the number of cells created by the tracker tr, and let TC_{gt} be the number of cells according to the ground truth gt. Equation 2 introduces the Total Cells Relative Error ($TCRE$), which is the relative error of the tracker regarding the total number of cells.

$$TCRE = \left| 1 - \frac{TC_{tr}}{TC_{gt}} \right| \tag{2}$$

6.4.3. Runtime (RT)

We ran each proposed approach 100 times and took the average execution time of each algorithm for tracking and predicting trajectories.

6.4.4. Statistical Significance Test

A statistical significance test over *TDR* results evaluates the significance of the tracking quality. We aim to reject the null hypothesis `H0` that the results of a pair of algorithms come from the same distribution. The non-parametric Kolmogorov-Smirnov test rejects this null hypothesis. Then the non-parametric Wilcoxon test statistically validates the results. By rejecting `H0`, the alternative hypothesis `H1` is that the proposed methods obtained better or worse results, with statistically significant differences.

7. Experiments

In this section, we evaluate how well the proposed predictors and trackers establish cell motion.

7.1. TDR and TCRE

Table 6 presents the *TDR* of the tracking and prediction approaches. Table 7 shows the corresponding *TCRE* values. *KDT-Tracker* and *BLT-Tracker* achieved *TDR* results comparable to trackers *Direct-Tracker* and *Clever-Tracker*. The competitor *MTrack2* has its best *TDR* results for the first two datasets, but for datasets three to six, the results are poor, as observed in the *TCRE* values. This is because *MTrack2* created too many false-positive cells.

Tables 8 and 9 show the p-values of the statistical significance using a paired Wilcoxon test to check *if the results of an algorithm are greater than the ones of the other, with a certain confidence.* Hypothesis `H0` is that the median of the differences between each pair of algorithms is zero. The alternative hypothesis `H1` is that the median of the differences is positive, so *the result of the algorithm in the column is statistically better than the one of the algorithms in the row.* The p-values rejecting `H0` are organized according to the confidence levels $\alpha = 0.01$ (1%), $\alpha = 0.05$ (5%), and

Table 6: Tracking Detection Rate (*TDR*) results per dataset.

TDR **– Datasets:** `2D+t`							
Approach	**Function**	`Ds1`	`Ds2`	`Ds3`	`Ds4`	`Ds5`	`Ds6`
Direct-Tr	–	0.385	0.500	0.815	0.576	0.141	**0.120**
Clever-Tr	–	0.462	0.500	0.815	0.565	0.217	0.109
KDT-Tr	–	0.431	0.460	0.804	0.478	0.207	0.065
BLT-Tr	–	0.431	0.460	0.804	0.478	0.207	0.065
Cell-Tr	–	0.323	0.240	0.717	0.370	0.152	0.054
MTrack2	–	0.508	0.540	0.000	0.000	0.022	0.109
CM-Pr	Lagrange's	0.508	0.480	0.609	0.457	0.087	0.054
KDT-Pr	Lagrange's	0.523	**0.627**	0.717	0.576	0.098	0.043
BLT-Pr	Lagrange's	0.523	**0.627**	0.717	0.576	0.098	0.043
Int-Pr	Lagrange's	0.446	0.567	0.837	**0.609**	0.239	0.087
KDT-Pr	W* Poly Regression	0.477	0.513	0.804	0.587	0.228	0.076
KDT-Pr	Poly Regression	0.477	0.507	0.837	0.500	0.228	0.076
Int-Pr	Poly Regression	**0.554**	0.600	**0.880**	**0.609**	**0.272**	0.087

TDR **– Datasets:** `3D+t`							
Approach	**Function**	`Ds7`	`Ds8`	`Ds9`	`Ds10`	`Ds11`	`Ds12`
Direct-Tr	–	0.337	0.554	0.088	0.080	0.513	0.163
Clever-Tr	–	0.370	0.543	0.272	0.240	0.613	0.212
KDT-Tr	–	0.370	0.489	0.272	0.232	0.613	0.200
BLT-Tr	–	0.370	0.489	0.272	0.232	0.613	0.200
Cell-Tr	–	–	–	–	–	–	–
MTrack2	–	–	–	–	–	–	–
CM-Pr	Lagrange's	0.413	0.554	0.256	0.440	0.400	0.200
KDT-Pr	Lagrange's	0.500	0.522	**0.504**	**0.476**	0.447	0.212
BLT-Pr	Lagrange's	0.500	0.522	**0.504**	**0.476**	0.447	0.212
Int-Pr	Lagrange's	0.565	0.630	0.416	0.360	0.653	**0.362**
KDT-Pr	W* Poly Regression	0.435	0.500	0.324	0.268	0.593	0.275
KDT-Pr	Poly Regression	0.435	0.500	0.324	0.268	0.593	0.275
Int-Pr	Poly Regression	**0.598**	**0.641**	0.432	0.404	**0.687**	**0.362**

* *Weighted polynomial regression*

$\alpha = 0.1$ (10%). Thus, p-values higher than the confidence level of 10% are not statistically significant and do not reject `H0`. For example, if we consider *Clever-Tracker* (third column of Table 8), the corresponding p-values show statistically better results than *KDT-Tracker*, *Cell-Tracker* and *MTrack2* at the confidence level of 1%, than *Direct-Tracker* at the confidence level of 5%, and than *CM-Predictor* using Lagrange's at the confidence level of 10%.

The proposed trackers *Direct-Tracker*, *Clever-Tracker*, *KDT-Tracker* and *BLT-Tracker* (Table 8) performed better than the competitors *Cell-Tracker* and *MTrack2*. Overall, Table 9 shows that the predictors were

Table 7: Total Cells Relative Error (*TCRE*) results per dataset (the best results are in bold).

TCRE **– Datasets:** `2D+t`							
Approach	**Function**	`Ds1`	`Ds2`	`Ds3`	`Ds4`	`Ds5`	`Ds6`
Direct-Tr	–	0.126	**0.009**	0.222	0.180	0.072	0.197
Clever-Tr	–	**0.095**	**0.009**	0.222	**0.160**	0.075	0.193
KDT-Tr	–	0.295	0.579	0.296	0.500	0.079	0.064
BLT-Tr	–	0.232	0.542	0.296	0.500	0.049	**0.040**
Cell-Tr	–	0.116	0.804	**0.111**	0.220	0.166	0.628
MTrack2	–	0.663	0.692	205.889	181.240	22.106	0.142
CM-Pr	Lagrange's	0.253	2.019	5.630	3.240	1.581	1.201
KDT-Pr	Lagrange's	0.726	1.234	4.370	1.880	2.008	1.692
BLT-Pr	Lagrange's	0.726	1.196	4.370	1.880	2.008	1.692
Int-Pr	Lagrange's	0.263	0.785	0.593	0.480	**0.026**	0.056
KDT-Pr	W* Poly Regression	1.926	2.841	1.259	1.560	2.268	1.781
KDT-Pr	Poly Regression	1.926	2.757	0.741	2.160	2.208	1.601
Int-Pr	Poly Regression	0.147	0.626	0.296	0.880	0.075	0.335
TCRE **– Datasets:** `3D+t`							
Approach	**Function**	`Ds7`	`Ds8`	`Ds9`	`Ds10`	`Ds11`	`Ds12`
Direct-Tr	–	**1.963**	**1.042**	0.918	**0.077**	**0.395**	0.444
Clever-Tr	–	**1.963**	**1.042**	1.711	1.738	0.494	**0.436**
KDT-Tr	–	3.000	2.083	1.756	1.794	0.531	0.761
BLT-Tr	–	2.926	2.083	1.714	1.764	0.506	0.675
Cell-Tr	–	–	–	–	–	–	–
MTrack2	–	–	–	–	–	–	–
CM-Pr	Lagrange's	7.333	3.542	3.346	0.530	6.160	4.256
KDT-Pr	Lagrange's	6.630	4.417	**0.501**	0.141	5.716	4.333
BLT-Pr	Lagrange's	6.630	4.417	**0.501**	0.141	5.630	4.333
Int-Pr	Lagrange's	2.778	1.708	1.622	1.733	0.704	0.744
KDT-Pr	W* Poly Regression	9.481	6.125	5.542	8.026	2.235	3.795
KDT-Pr	Poly Regression	9.370	5.917	5.597	8.267	2.173	3.838
Int-Pr	Poly Regression	2.889	2.917	2.331	2.529	0.741	1.462

**Weighted polynomial regression*

significantly better than the trackers. In particular, *Int-Predictor* combined with the Polynomial Regression outperformed all trackers and predictors.

Table 8: Trackers' statistical significance test over *TDR* results.

	Direct-T	Clever-T	KDT-T	Cell-T	MTrack2
Direct-T	–	0.014	0.194	0.999	0.992
Clever-T	0.986	–	0.996	0.999	0.996
KDT-T	0.806	0.004	–	0.999	0.995
Cell-T	0.001	0.001	0.001	–	0.700
MTrack2	0.008	0.004	0.005	0.300	–
CM-P-Lag	0.571	0.091	0.429	0.995	0.996
KDT-P-Lag	0.909	0.733	0.909	0.996	0.998
Int-P-Lag	0.998	0.997	0.999	0.999	0.997
KDT-PR-W	0.986	0.880	0.996	0.999	0.995
KDT-PR	0.983	0.735	0.998	0.999	0.995
Int-P-Reg	0.998	0.999	0.999	0.999	0.999

Confidence levels (α): 1% 5% 10%

Table 9: Predictors' statistical significance test over *TDR* results.

	CM-P-Lag	KDT-P-Lag	Int-P-Lag	KDT-PR-W	KDT-PR	Int-P-Reg
Direct-T	0.429	0.091	0.002	0.014	0.017	0.002
Clever-T	0.909	0.267	0.003	0.120	0.265	0.001
KDT-T	0.571	0.091	0.001	0.004	0.002	0.001
Cell-T	0.005	0.004	0.001	0.001	0.001	0.001
MTrack2	0.004	0.002	0.003	0.005	0.005	0.001
CM-P-Lag	–	0.005	0.005	0.050	0.068	0.002
KDT-P-Lag	0.995	–	0.091	0.623	0.709	0.017
Int-P-Lag	0.995	0.909	–	0.997	0.997	0.004
KDT-PR-W	0.950	0.377	0.003	–	0.704	0.001
KDT-PR	0.932	0.291	0.003	0.296	–	0.001
Int-P-Reg	0.998	0.983	0.996	0.999	0.999	–

Confidence levels (α): 1% 5% 10%

Figure 7 depicts the average *TDR* (the higher the better) and *TCRE* (the lower the better) results of the evaluated trackers. For `2D+t` datasets, the proposed approaches achieved similar *TDR* results, and the competitors *Cell-Tracker* and *MTrack2* had the lowest *TDR* values, with *MTrack2* having the highest *TCRE*. For `3D+t` datasets, *Direct-Tracker* yielded the lowest *TDR*, with higher *TCRE*. This indicates that the greedy approach to match cells leads to many tracking errors. *MTrack2* results indicate that the fully-automatic processing of cells did not accurately identified the existing cells, producing many false-positives. Accordingly, the experiments indicate that a manual adjustment should refine *MTrack2* results in order to improve the cell tracking quality.

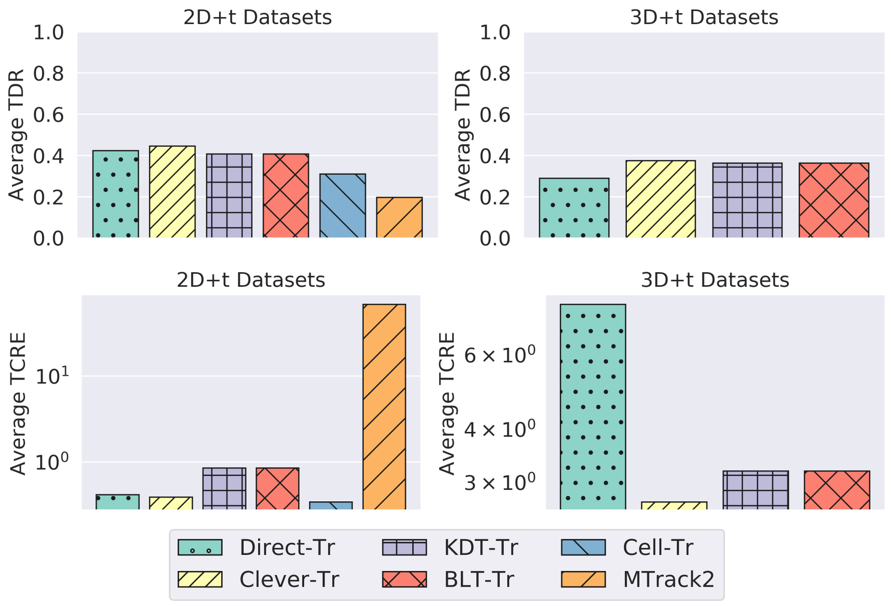

Figure 7: Average *TDR* and *TCRE* of trackers.

All approaches achieved poor *TDR* results for datasets `Ds5`, `Ds6`, `Ds9`, and `Ds10`. This is because the sequences of images from these datasets present a high number of cells (ref. to Table 2). The cells are too close to each other, and the matches identified by trackers and predictors were erroneous in many time stamps.

Overall, predictors achieved better *TDR* results than the trackers, but in contrast, had the highest *TCRE* results. Among predictors, the algorithms with the Polynomial Regression function to predict cell positions achieved better results than the ones with Lagrange's Polynomials. The weighted Polynomial Regression achieved very similar results to the regular regression results. With both prediction functions, the *Int-Predictor* approach achieved the best *TDR* results among all approaches, with the lowest *TCRE* results among predictors.

Figure 8 shows the average *TDR* and *TCRE* results of the evaluated predictors. *Int-Predictor* achieved the best results among all approaches, with the highest *TDR* and lowest *TCRE*. Plus, predictors using a Polynomial Regression function outperformed the ones using Lagrange's Polynomials.

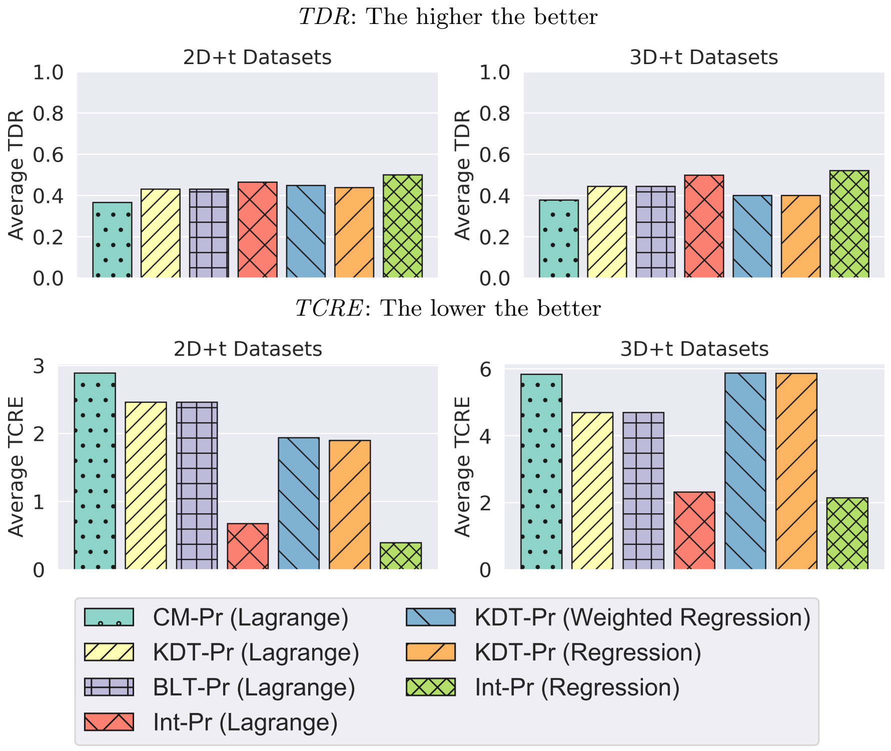

Figure 8: Average *TDR* and *TCRE* of predictors.

Prediction Error with Spatial Displacement and Angle Variation. Here we evaluate the different configurations of *KDT-Predictor* implemented with the error function considering both the spatial displacement and angle variation of the predicted cell positions. Table 10 shows the prediction results. Column *TDR Gain* shows how much the *TDR* results with angle and spatial checking improved the results of the corresponding algorithm with only spatial error checking. In the same line, column *TCRE Difference* informs how much the error changed according to the corresponding algorithm. In general, the results do not show significant improvements considering the direction of the movement in the prediction checking. The lack of improvement occurs because cells move under uncertainty in different directions, which is inherent to the general nature of the problem tackled in this work.

Error testing considering the angle variation versus the original approaches					
	Dataset	***TDR***	***TDR* Gain**	***TCRE***	***TCRE* Difference**
KDT-Pr with Lagrange's	1	0.369	-0.077	2.547	**-3.169**
	2	0.587	**0.374**	1.178	**-3.156**
	3	0.620	-0.098	6.185	1.815
	4	0.424	-0.152	4.280	2.400
	5	0.163	**0.065**	1.868	**-0.140**
	6	0.043	0.000	1.560	**-0.132**
	7	0.424	-0.076	7.556	0.926
	8	0.467	-0.054	6.000	1.583
	9	0.340	-0.164	3.510	3.008
	10	0.304	-0.172	4.159	4.018
	11	0.633	**0.110**	2.926	2.200
	12	0.338	-0.289	3.103	1.869
KDT-Pr with Weighted Polynomial Regression	1	0.354	-0.239	3.432	1.197
	2	0.467	**0.192**	2.486	**-1.309**
	3	0.750	-0.054	2.519	1.259
	4	0.707	**0.120**	0.440	**-1.120**
	5	0.098	-0.130	1.947	**-0.321**
	6	0.043	-0.033	1.519	**-0.262**
	7	0.359	-0.076	6.704	**-2.778**
	8	0.446	-0.054	7.667	1.542
	9	0.308	-0.016	4.992	**-0.550**
	10	0.260	-0.008	5.269	**-2.757**
	11	0.593	**0.116**	3.815	1.888
	12	0.213	-0.301	3.957	1.116
KDT-Pr with Polynomial Regression	1	0.354	-0.239	4.189	2.017
	2	0.540	**0.265**	2.972	**-0.866**
	3	0.750	-0.087	3.148	2.407
	4	0.500	0.000	3.440	1.280
	5	0.098	-0.130	2.815	0.608
	6	0.043	-0.033	2.228	0.627
	7	0.467	**0.033**	9.259	**-0.111**
	8	0.446	-0.054	9.458	3.542
	9	0.268	-0.056	7.842	2.244
	10	0.380	**0.112**	3.519	**-4.747**
	11	0.520	**0.043**	6.395	4.469
	12	0.213	-0.294	5.402	2.645

Table 10: *TDR* and *TCRE* results considering the agle variation to verify prediction errors. Values in **bold** indicate quality improvement compared to the original approaches.

7.2. Runtime

Table 11 presents the runtime of the proposed approaches. As expected, the indexing structures speed up the matching step of the trackers and the predictors. Predictors using Polynomial Regression for cell prediction performed better than the ones using Lagrange's. The weighted versions of the Polynomial Regression were slightly slower than the unweighted ones. *Int-Predictor* was the fastest cell prediction algorithm, followed by *KDT-Predictor*.

Table 11: Runtime of the proposed approaches.

***Runtime* (in seconds)**				
Approach	**Function**	2D+t	3D+t	**All Datasets**
Direct-Tr	–	71.876	186.222	129.049
Clever-Tr	–	121.334	396.763	259.048
KDT-Tr	–	17.267	49.504	33.386
BLT-Tr	–	17.262	49.449	33.355
CM-Pr	Lagrange's	135.467	275.123	205.295
BLT-Pr	Lagrange's	77.441	154.081	115.761
KDT-Pr	Lagrange's	77.949	155.782	116.866
KDT-Pr	W* Poly Regression	47.492	102.082	74.787
KDT-Pr	Poly Regression	45.829	99.032	72.430
Int-Pr	Lagrange's	19.586	46.668	33.127
Int-Pr	W* Poly Regression	16.801	41.128	28.965
Int-Pr	Poly Regression	**15.085**	**36.649**	**25.867**

* *Weighted polynomial regression*

The best cell prediction algorithm *Int-Predictor* was up to 26% faster than the best cell tracking algorithm *BLT-Tracker*. Besides, beyond the runtime reduction, prediction algorithms also reduce the number of images processed.

8. Conclusions

Establishing trajectories of moving objects continues to be an important problem. In this paper, we have studied a new variant of the problem: In contrast to other, more 'conventional' settings, we have looked at the case where objects do not have an identity, i.e., it is not obvious how to establish the connection between objects observed earlier and the ones observed later on. While this problem occurs in different scientific domains, our use case has been cell motion, according to streams of respective digital images. In cell biology, the so-called tracking algorithms already exist, and we have used them as a baseline for comparison. We have proposed two prediction pipelines to estimate cell positions without the need to process every image from the input sequence. The first one reads a window of cell points obtained from images and extrapolates cell points to predict future cell positions. The second prediction pipeline processes non-consecutive windows of cell points acquired from images and estimates intermediary cell positions. Experiments showed that prediction approaches could maintain (or even increase) tracking quality, being up to 26% faster than the fastest tracking approach. All in all, the proposed prediction pipelines lead to better and quicker cell motion establishment than algorithms that implement the conventional tracking pipeline.

The methods proposed in this study focus on cell biology, but we hypothesize them to be applicable in other domains as well, including nanorobotics, materials sciences, ecology, and biology at a larger scale. However, this is difficult as long as respective gold standards are not available. One could do further experiments with 'conventional' data, where the identity of the moving objects is known but is hidden on purpose to evaluate our algorithms, variants of it, or completely new ones. This should be interesting, as we do not expect the settings and hence the necessary solutions to be identical. In cell biology, one could tightly integrate cell predictors with image segmentation algorithms, in order to reduce the search space for cells to segment and speed up the identification of seed points in subsequent images.

Acknowledgments

This research was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, by the São Paulo Research Foundation – FAPESP (grants number 2020/11258-2, 2018/24414-2, 2020/07200-9, 2016/17078-0), by Capes-PDSE (grant number 88881.13068/2016-01), and the National Council for Scientific and Technological Development (CNPq).

References

[1] Y. Huang, S. Liao, C. Lee, Evaluating continuous k-nearest neighbor query on moving objects with uncertainty, Information Systems 34 (2009) 415–437. URL: `https://doi.org/10.1016/j.is.2009.01.001`. doi:`10.1016/j.is.2009.01.001`.

[2] A. Behrend, G. Schüller, M. Wieneke, Efficient tracking of moving objects using a relational database, Information Systems 38 (2013) 1269–1284. URL: `https://doi.org/10.1016/j.is.2012.01.001`. doi:`10.1016/j.is.2012.01.001`.

[3] B. Shen, Y. Zhao, G. Li, W. Zheng, Y. Qin, B. Yuan, Y. Rao, V-tree: Efficient knn search on moving objects with road-network constraints, in: 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017, IEEE Computer Society, 2017, pp. 609–620. URL: https://doi.org/10.1109/ICDE.2017.115. doi:10.1109/ICDE.2017.115.

[4] R. H. Güting, Moving Objects Databases and Tracking, Springer New York, New York, NY, 2018, pp. 2307–2314. URL: https://doi.org/10.1007/978-1-4614-8265-9_223. doi:10.1007/978-1-4614-8265-9_223.

[5] Z. Lin, Q. Zeng, H. Duan, C. Liu, F. Lu, A semantic user distance metric using gps trajectory data, IEEE Access 7 (2019) 30185–30196. doi:10.1109/ACCESS.2019.2896577.

[6] Y. Zhou, G. Li, L. Wang, S. Li, W. Zong, Smartphone-based pedestrian localization algorithm using phone camera and location coded targets, in: Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS), 2018, pp. 1–7. doi:10.1109/UPINLBS.2018.8559868.

[7] H. Huang, C. Hsieh, K. Liu, H. Cheng, S. J. Hsu, C. Chan, Multi-sensor fusion approach for improving map-based indoor pedestrian localization, Sensors 19 (2019) 3786. URL: https://doi.org/10.3390/s19173786. doi:10.3390/s19173786.

[8] D. Stojanovic, A. N. Papadopoulos, B. Predic, S. Djordjevic-Kajan, A. Nanopoulos, Continuous range monitoring of mobile objects in road networks, Data Knowledge Engineering 64 (2008) 77–100. URL: https://doi.org/10.1016/j.datak.2007.06.021. doi:10.1016/j.datak.2007.06.021.

[9] R. Zhang, H. V. Jagadish, B. T. Dai, K. Ramamohanarao, Optimized algorithms for predictive range and KNN queries on moving objects, Information Systems 35 (2010) 911–932. URL: https://doi.org/10.1016/j.is.2010.05.004. doi:10.1016/j.is.2010.05.004.

[10] P. Fan, G. Li, L. Yuan, Y. Li, Vague continuous k-nearest neighbor queries over moving objects with uncertain velocity in road networks, Information Systems 37 (2012) 13–32. URL: https://doi.org/10.1016/j.is.2011.08.002. doi:10.1016/j.is.2011.08.002.

[11] X. Li, C. Yu, L. Ju, J. Qin, Y. Zhang, L. Dou, Y. Sun, Position prediction system based on spatio-temporal regularity of object mobility, Information Systems 75 (2018) 43–55. URL: https://doi.org/10.1016/j.is.2018.02.004. doi:10.1016/j.is.2018.02.004.

[12] Z. Chen, Q. Yuan, W. Liu, Monitoring best region in spatial data streams in road networks, Data Knowledge Engineering 120 (2019) 100–118. URL: https://doi.org/10.1016/j.datak.2019.03.002. doi:10.1016/j.datak.2019.03.002.

[13] T. Nguyen, Z. He, R. Zhang, P. G. D. Ward, Exploiting velocity distribution skew to speed up moving object indexing, Information Systems 51 (2015) 72–104. URL: https://doi.org/10.1016/j.is.2015.03.001. doi:10.1016/j.is.2015.03.001.

[14] P. Schmiegelt, A. Behrend, B. Seeger, W. Koch, A concurrently updatable index structure for predicted paths of moving objects, Data Knowledge Engineering 93 (2014) 80–96. URL: https://doi.org/10.1016/j.datak.2014.07.007. doi:10.1016/j.datak.2014.07.007.

[15] J. Ge, Y. Xia, J. Wang, C. H. Nadungodage, S. Prabhakar, Sequential pattern mining in databases with temporal uncertainty, Knowl. Inf. Syst. 51 (2017) 821–850. URL: https://doi.org/10.1007/s10115-016-0977-1. doi:10.1007/s10115-016-0977-1.

[16] S. Prabhakar, R. Cheng, Data uncertainty management in sensor networks, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, Second Edition, Springer, 2018, pp. 647–651. URL: https://doi.org/10.1007/978-1-4614-8265-9_115. doi:10.1007/978-1-4614-8265-9_115.

[17] S. Prabhakar, R. Cheng, Indexing uncertain data, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, Second Edition, Springer, 2018, pp. 1900–1907. URL: https://doi.org/10.1007/978-1-4614-8265-9_80740. doi:10.1007/978-1-4614-8265-9_80740.

[18] M. Maska, et al., A benchmark for comparison of cell tracking algorithms, Bioinformatics 30 (2014) 1609–1617. doi:10.1093/bioinformatics/btu080.

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

[20] J.-Y. Tinevez, N. Perry, J. Schindelin, G. M. Hoopes, G. D. Reynolds, E. Laplantine, S. Y. Bednarek, S. L. Shorte, K. W. Eliceiri, Trackmate: An open and extensible platform for single-particle tracking, Methods 115 (2017) 80 – 90. doi:10.1016/j.ymeth.2016.09.016, image Processing for Biologists.

[21] F. Piccinini, A. Kiss, P. Horváth, Celltracker (not only) for dummies, Bioinformatics 32 (2016) 955–957. doi:10.1093/bioinformatics/btv686.

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

[23] A. Elfwing, Y. LeMarc, J. Baranyi, A. Ballagi, Observing growth and division of large numbers of individual bacteria by image analysis, Applied and Environmental Microbiology 70 (2004) 675–678. doi:10.1128/AEM.70.2.675-678.2004.

[24] P. Friedl, S. Alexander, Cancer invasion and the microenvironment: Plasticity and reciprocity, Cell 147 (2011) 992 – 1009. doi:10.1016/j.cell.2011.11.016.

[25] F. P. Cordelières, V. Petit, M. Kumasaka, O. Debeir, V. Letort, S. J. Gallagher, L. Larue, Automated cell tracking and analysis in phase-contrast videos (itrack4u): Development of java software based on combined mean-shift processes, PLOS ONE 8 (2013). doi:10.1371/journal.pone.0081266.

[26] N. Stuurman, J. Schindelin, C. Elliott, M. Hiner, Mtrack2, 2017. URL: https://imagej.net/MTrack2.

[27] F. Cordelieres, J. Schindelin, Manual tracking, 2017. URL: https://imagej.net/Manual_Tracking.

[28] D. Sage, F. R. Neumann, F. Hediger, S. M. Gasser, M. Unser, Automatic tracking of individual fluorescence particles: application to the study of chromosome dynamics, IEEE Transactions on Image Processing 14 (2005) 1372–1383. doi:10.1109/TIP.2005.852787.

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

[30] M. T. Cazzolato, A. J. M. Traina, K. Böhm, Efficient and reliable estimation of cell positions, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM, Torino, Italy, October 22-26, 2018, pp. 1043–1052. doi:10.1145/3269206.3271734.

[31] G. Zhu, F. Porikli, H. Li, Beyond local search: Tracking objects everywhere with instance-specific proposals, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 943–951. doi:10.1109/CVPR.2016.108.

[32] Y. Tao, C. Faloutsos, D. Papadias, B. Liu, 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, 2004, pp. 611–622. doi:10.1145/1007568.1007637.

[33] S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. López, 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., 2000, pp. 331–342. doi:10.1145/342009.335427.

[34] M. Gong, Y. Shu, Real-time detection and motion recognition of human moving objects based on deep learning and multi-scale feature fusion in video, IEEE Access 8 (2020) 25811–25822. doi:10.1109/ACCESS.2020.2971283.

[35] H. Song, H. Shin, Classification and spectral mapping of stationary and moving objects in road environments using FMCW radar, IEEE Access 8 (2020) 22955–22963. doi:10.1109/ACCESS.2020.2970440.

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

[37] G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, F. Herrera, Deep learning in video multi-object tracking: A survey, Neurocomputing 381 (2020) 61–88. URL: https://www.sciencedirect.com/science/article/pii/S0925231219315966. doi:10.1016/j.neucom.2019.11.023.

[38] X. Jiang, P. Li, X. Zhen, X. Cao, Model-free tracking with deep appearance and motion features integration, in: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), 2019, pp. 101–110. doi:10.1109/WACV.2019.00018.

[39] Z. Wang, L. Yin, Z. Wang, A new approach for cell detection and tracking, IEEE Access 7 (2019) 99889–99899. doi:10.1109/ACCESS.2019.2930539.

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

[41] J. Stegmaier, J. C. Otte, A. Kobitski, A. Bartschat, A. Garcia, G. U. Nienhaus, U. Strähle, R. Mikut, Fast segmentation of stained nuclei in terabyte-scale, time resolved 3d microscopy image stacks, PLOS ONE 9 (2014) 1–11. doi:10.1371/journal.pone.0090036.

[42] J. W. Young, J. C. W. Locke, A. Altinok, N. Rosenfeld, T. Bacarian, P. S. Swain, E. Mjolsness, M. B. Elowitz, Measuring single-cell gene expression dynamics in bacteria using fluorescence time-lapse microscopy, Nature Protocols 7 (2011) 80–88. doi:10.1038/nprot.2011.432.

[43] C. A. Schneider, W. S. Rasband, K. W. Eliceiri, Nih image to imagej: 25 years of image analysis, Nature Methods 9 (2012) 671–675. doi:10.1038/nmeth.2089.

[44] C. T. Rueden, J. E. Schindelin, M. C. Hiner, B. E. DeZonia, A. E. Walter, E. T. Arena, K. W. Eliceiri, Imagej2: Imagej for the next generation of scientific image data, BMC Bioinformatics 18 (2017) 529:1–529:26. doi:10.1186/s12859-017-1934-z.

[45] O. Debeir, P. V. Ham, R. Kiss, C. Decaestecker, Tracking of migrating cells under phase-contrast video microscopy with combined mean-shift processes, IEEE Trans. Med. Imaging 24 (2005) 697–711. doi:10.1109/TMI.2005.846851.

[46] J. C. Crocker, D. G. Grier, Methods of digital video microscopy for colloidal studies, Journal of Colloid and Interface Science 179 (1996) 298 – 310. doi:10.1006/jcis.1996.0217.

[47] E. Meijering, O. Dzyubachyk, I. Smal, Chapter nine - methods for cell and particle tracking, in: P. M. conn (Ed.), Imaging and Spectroscopic Analysis of Living Cells, volume 504 of *Methods in Enzymology*, Academic Press, 2012, pp. 183 – 200. doi:10.1016/B978-0-12-391857-4.00009-4.

[48] J. S. Jean-Yves Tinevez, Nick Perry, TrackMate, 2017 (accessed September, 2020). URL: https://imagej.net/TrackMate.

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

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