Biophysical Journal, Volume *111*

Supplemental Information

Ccoffinn: Automated Wave Tracking in Cultured Cardiac Monolayers

Jakub Tomek, Rebecca A.B. Burton, and Gil Bub

Supplementary materials: Ccoffinn toolkit: Automated wavefront tracking and feature extraction in optical mapping of cardiac cell cultures

J. Tomek[1] R.A.B. Burton [2] G. Bub [3]

[1]Corresponding author. E-mail: jakub.tomek.mff@gmail.com. Department of Physiology, Anatomy, and Genetics, University of Oxford, Oxford, United Kingdom
[2]Department of Pharmacology, University of Oxford, Oxford, United Kingdom
[3]Department of Physiology, McGill university, Montreal QC Canada

S 1 Methodology description

S1.1 Spike-mining

The pseudocode summarising the spike-mining process is given in Pseudocode S.1 with further details below.

Pseudocode S.1 The algorithm for determining which blobs are active in which frames.

```
function blobActivity = getActivity(
            frames, {Sequence of recorded frames}
            mask {Result of blob segmentation}
            smoothingParameter {Kernel width for averaging filter}
            frameRate {Frame rate of the frame sequence}
            activityThreshold {Threshold for finding spike times}
            apdN {Level of APD-N%, on scale 0-1}
            minSpike, maxSpike {Limits of spike duration}
            isDoubleHump {1 if additional filtering is to be applied}
            )
begin

diffThreshold = activityThreshold / frameRate;
{Get pixels occupied by each blob.}
blobPositions = getCP(mask);
{Getting and filtering intensity traces}
bI = getBI(frames, blobPositions); {Blob intensities}
bIfilt = filterBI(bI, frameRate, isDoubleHump);
[upstrokeTimes, zeroTimes] = processDVDT(bIfilt, frameRate,
                                         diffThreshold);
blobActivity = findSpikes(bIfilt, upstrokeTimes, zeroTimes,
                          minSpike, maxSpike, frameRate);
end;
```

First, $diffThreshold$ is computed as $activityThreshold/frameRate$, which scales the *activityThreshold* by an element proportional to the duration between consecutive frames, dt. When we later compare discrete derivative of intensity (dV) to a threshold, this scaling makes sure we are comparing dV/dt to an element proportional to a fixed-value *activityThreshold*, which makes the threshold invariant to sampling frequency used.

Then, a list of pixels belonging to each blob is extracted from the segmentation mask in `getCP`. In the second step (`getBI`), the average brightness of a blob's pixels is measured for all the blobs in all frames and stored in the matrix bI, so that:

$$bI(i,j) = \frac{\sum_{\forall k} b(B_k^i(j))}{||B^i||} \tag{1}$$

where B^i is the list of pixels belonging to the i-th segmented blob and $b(B_k^i(j))$ is the brightness of the k-th pixel belonging to the i-th blob in frame j. $||B^i||$ is the number of pixels belonging to the i-th blob.

In `filterBI`, the intensity of each blob is first scaled to the 0–1 range and its baseline is subtracted (the baseline is estimated as the intensity after median filtering with the filter size representing 2s duration, thus with the filter size depending on the frame rate, with symmetric treatment at the edges of the trace). Then, the intensity is smoothed using a median filter with filter size of *smoothingParameter*, yielding *bIfilt*. Median filtering has the advantage that it does not change the time of the action potential upstroke, unlike filters based on averaging. If *isDoubleHump* is true, zero-phase averaging of the same filter width is further applied to connect the two humps representing an action potential. Traces with two humps per action potential are commonly seen in temporally filtered traces from dye-free imaging experiments, since both excitation and contraction generate an increase in intensity. The added averaging step helps to connect the two humps so that they are segmented as a single spike for the purposes of activation detection.

In `processDVDT`, times of maximum dV/dt are determined first (dV stands for intensity in general, which traditionally represents voltage). For each blob, *dIS* is defined so that *dIS*[1] = 0 and *dIS*[i] = *bIfilt*[i] - *bIfilt*[i-1]. The variable *dIS* is then thresholded using *diffThreshold* and within each block above threshold, the maximum value of *dIS* is found and saved in *upstrokeTimes*. Furthermore, all times when *dIS* is zero or crosses zero are saved in *zeroTimes*.

Ultimately, the times of activity of each blob are determined in `findSpikes`. For each blob and time of its upstroke t_u, let us consider the following variables:

- $tBaseline_{prev}$: The last element of $zeroTimes$ smaller than t_u. This gives a time when the blob is likely to be repolarised before the currently processed action potential.
- $tBaseline_{peak}$. The first element of $zeroTimes$ that is larger than t_u. This is the time of maximum intensity of the given action potential.
- $tBaseline_{2after}$. The second element of $zeroTimes$ that is larger than t_u. This gives the time when the blob is likely to be repolarised after the current action potential.
- $intensity_{prev} = bIfilt[intensity_{prev}]$
- $intensity_{peak} = bIfilt[intensity_{peak}]$
- $intensity_{after2} = bIfilt[intensity_{after2}]$

- $baseline = \frac{intensity_{after} + intensity_{before}}{2}$

Then, we define a threshold for the given spike as follows:

$$spikeThreshold = baseline + (1 - apdN)(intensity_{peak} - baseline)$$

The particular blob is then considered active in the interval of frames represented by the segment of $bIfilt > spikeThreshold$ which contains frame t_u (note that this happens for each upstroke detected in the particular blob and the times of activity are combined, rather than overwriting one another). Also, if the duration of the interval in ms (obtained using the number of frames taken and *frameRate*) does not fit within *minAPD* and *maxAPD*, the interval is discarded as an artifact.

S1.2 Wave segmentation

As mentioned in the main text, the first step of wave segmentation is to split blobs active in a particular frame to separate waves. Then, each wave is divided into border blobs ($\mathcal{B}$ blobs) and inside blobs ($\mathcal{I}$ blobs). The $\mathcal{B}$ blobs are labeled as wavefront blobs ($\mathcal{W}$ blobs) if they are in the phase of an action potential when voltage rises, and the subset of these that were not wavefront blobs in the previous frame are labelled new-wavefront blobs ($\mathcal{NW}$ blobs). The whole process is sketched in Pseudocode S.2; described below in depth for a single frame:

Using the output of `getActiveBlobs`, we first extract the locations of blobs that are active in the given frame. The function `filterActiveBlobs` filters the active blobs according to their centroids to retain as active only the ones that have at least *minDensity* neighbours within *distanceThreshold* pixels, while the rest are rejected (This helps to remove spurious action potentials and helps the later tracking steps focus on the propagation of organised activity.).

Then, a graph *G=(V,E)* is created. The vertices V are the centroids of active blobs in this frame. In `getEdges`, near blobs are linked with an edge: $\forall v_i, v_j \in V : (v_i, v_j) \in E \Leftrightarrow d(v_i, v_j) <= distanceThreshold$, where d is Euclidean distance and *distanceThreshold* a parameter. The graph G is then decomposed into connected components which represent separate waves.

The next step is to determine which blobs are at the border of the waves. To do this, a *wave mask* is created (in `cleanMask`):

$$wave\,mask(i,j) = \begin{cases} 1, & \text{If } \exists k \text{ such that } (i,j) \in P_k \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

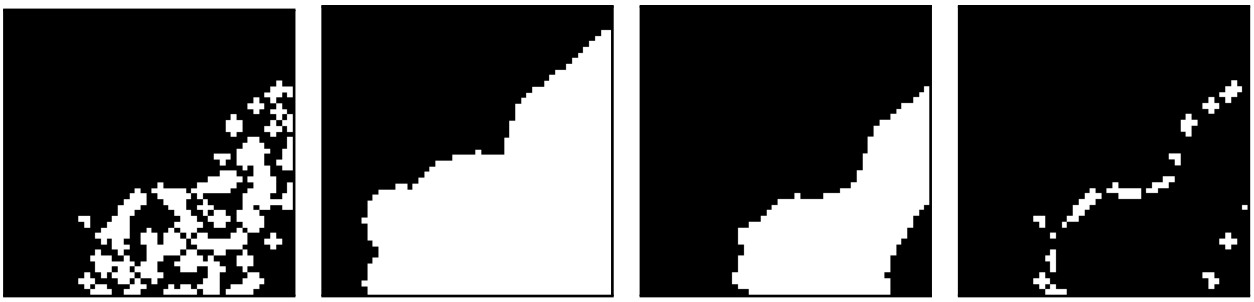
where P_k is the k-th blob active in the current frame.

Pseudocode S.2 The algorithm for segmenting blobs active in a given frame into waves. The blobs are also labelled according to their position in their respective wave.

```
procedure waveSegmentation(
              frames, {Sequence of recorded frames}
              frameRate, {Rate of acquiring frames}
              mask, {a matrix with the same size as a frame,
                     with 1 where blobs are, 0 elsewhere}
              blobActivities, {output of getActivity}
              bISmoothed, {as computed within getActivity}
              minDensity, {Minimum density of blobs}
              distanceThreshold) {The minimum allowed distance of waves}
begin
[centroids, dist] = getInfo(mask) {Get centroids and distances of blobs
                                   in mask}
for frame in frames do
begin
  {Part a) Segmenting blobs into waves}
  activeBlobs = getActiveBlobs(frame, blobActivities);
  activeBlobsFiltered = filterActiveBlobs(centroids[activeBlobs],
                                   distanceThreshold, minDensity);
  V = centroids[activeBlobs];
  E = getEdges(activeBlobsFiltered, distanceThreshold);
  waves = getConnectedComponents(V,E, dist);

  {Part b) Determining which active blobs are are border blobs
   and which are inside blobs}
  waveMask = cleanMask(mask, waves);
  wH = getWrapperHull(waves, waveMask, distanceThreshold);
  [insideBlobs, borderBlobs] = splitIB(activeBlobs, wH);

  {Part c) Further segmenting border blobs}
  [wfBlobs, nonwfBlobs] = splitWavefront(borderBlobs, frameRate,
                                   blobBrightnessBS);
end;
end;
```



(a) (b) (c) (d)

Figure S.1: Four stages of determining the border blobs. In (a), a part of the binary mask of blobs active in a given wave is shown, with the result of its morphological dilation in (b). The wrapper hull, morphological erosion of (b) is shown in (c). Ultimately, (d) shows the pixels that were 1 in (a), but not in (c): these represent the blobs at the border of the given wave.

Then, we compute a *wrapper hull* as a binary mask obtained using the following steps, illustrated in Figure S.1. The name 'wrapper hull' is used due to similarities with a a convex hull, However, since the waves are not always convex objects, a convex hull would not work sufficiently well in this application.

1. Morphologically dilate the wave mask with a circular structural element of radius $\lfloor \frac{distanceThreshold-1}{2} \rfloor$. Note that due to the waves being at least *distanceThreshold* apart by their definition, this operation cannot merge separate waves into a single object.

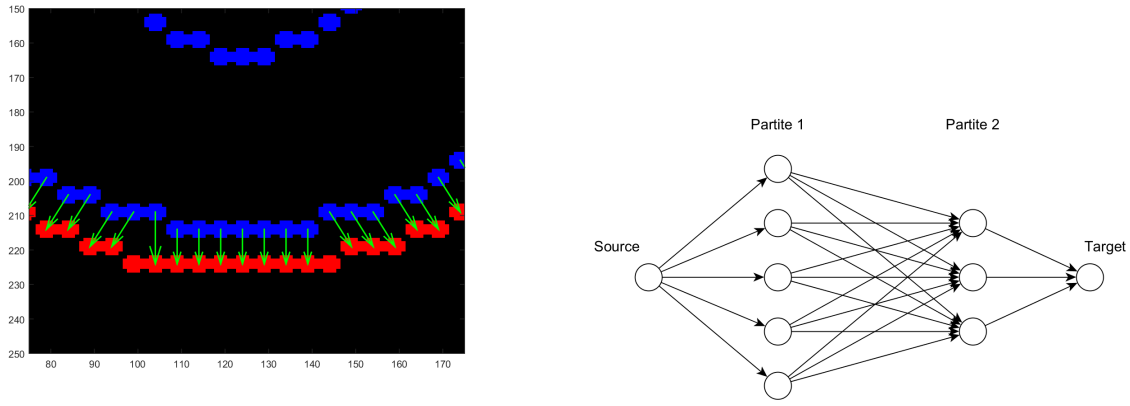
2. Morphologically erode the wave mask with a circular structural element of radius $\lfloor \frac{distanceThreshold-1}{2} \rfloor + 2$.

To obtain $\mathcal{B}$ blobs (in `splitIB`), the wave mask is taken and all 1s in it that are also 1s in the wrapper hull are set to 0. Only the blobs that were not fully removed from the wave mask are then labelled as $\mathcal{B}$ blobs: their existence is a consequence of the morphological erosion being performed with a slightly larger structural element than the dilation.

Ultimately (`splitWavefront`), the $\mathcal{B}$ blobs are further separated according to the slope of *cISmoothed* at the given frame; the slope is estimated as the difference between consecutive elements. The border blobs with positive slope at the given frame (i.e., becoming activated) are labelled as $\mathcal{W}$ blobs; the rest of the blobs stays labelled as $\mathcal{B}$ blobs.

S1.3 Wavefront Tracking

The wavefront tracking is performed in a hierarchical way. First, separate waves (connected components of active blobs) in frame f are linked to separate waves in frame *f-1*. For each



(a) (b)

Figure S.2: In (a) is an example of tracking problem in a section of a spiral or target wave. In blue are the $\mathcal{B}$ blobs in the previous frame, in red are the $\mathcal{NW}$ blobs in the current frame. The resulting tracking arrows are shown in green. Due to the fact that there are more $\mathcal{NW}$ blobs than $\mathcal{B}$ blobs, two $\mathcal{NW}$ are not linked to anything via an arrow. The subfigure (b) contains an illustration of a graph used in computation of minimum-cost maximum-flow.

wave w_i in frame *f*, a wave w_j in frame *f-1* is chosen so that the number of active blobs that are present both in w_i and w_j is the maximum possible. If the size of the largest found intersection is 0, no link is formed. These steps ensure that only blobs belonging to the same wave are tracked.

When the high-level links (w_i, w_j) are formed between waves, $\mathcal{W}$ blobs in w_i and $\mathcal{B}$ blobs in w_j are then tracked as described below. An illustration of the tracking output of our software is given in Figure S.2a.

Two groups of tracking arrows are defined:

- The trivial group is "zero" arrows: if a $\mathcal{W}$ blob in frame f is also a $\mathcal{B}$ blob in frame $f-1$, it means that the wave is static there, not propagating further.
- The second group of tracking arrows contains the arrows representing an actual movement of a wavefront. However, rather than linking full wavefronts at frames f and *f-1*, we link all the border blobs in frame *f-1* to the $\mathcal{NW}$ blobs ($\mathcal{W}$ blobs in frame f that were not $\mathcal{W}$ in frame *f-1*) to obtain a smaller integer programming problem later. Also for increased computational performance, we disregard the $\mathcal{B}$ blobs in frame *f-1* with no $\mathcal{NW}$ blob in frame f within *maxSpeed*; these removed border blobs are too far from the wavefront to be linkable to the new wavefront. Finally, tracking arrows longer than *maxSpeed* are removed.

To obtain the nontrivial tracking arrows, we form a complete oriented bipartite graph

with $G = (V_1 \cup V_2, E)$ where vertices in V_1, the first partite, represent centroids of eligible $\mathcal{B}$ blobs in frame $f-1$ and vertices in V_2 represent $\mathcal{NW}$ blobs in frame f, with $(u,v) \in E \Leftrightarrow u \in V_1, v \in V_2$. Furthermore, we set $\forall (u,v) \in E : c(u,v) = 1, a(u,v) = d(u,v)$, where c is capacity, a is cost and d is Euclidean distance of the vertices. We can then formulate the problem of finding the best tracking arrows as finding the maximum-size matching of the two partites in G with minimum total cost. Such a matching may be found, for example, by maximising network flow. Using this approach, the source and sink vertices are added to the graph (also with unitary capacities, but zero costs) and linked to their respective partites as shown in Figure S.2b. Then, we aim to find the cheapest flow f of size $\phi = min(|V_1|, |V_2|)$. In order to make sure that an edge is either fully used in the flow or not at all (i.e., a tracking arrow is present there or not), we formulate the task as an integer programming problem with variables $x_{(u,v)} = f(u,v)$:

$$\text{minimize:} \quad \sum_{u \in V_1, v \in V_2} x_{(u,v)} a(u,v) \tag{3.1}$$

$$\text{subject to:} \quad x_{(u,v)} \leq 1 \qquad \forall (u,v) \in E \tag{3.2}$$

$$x_{(u,v)} \geq 0 \qquad \forall (u,v) \in E \tag{3.3}$$

$$\sum_{(u,v) \in E} x_{(u,v)} - \sum_{(v,w) \in E} x_{(v,w)} = 0 \qquad \forall v \in V_1 \cup V_2 \tag{3.4}$$

$$\sum_{(source,v) \in E} x_{(source,v)} = \phi \qquad \forall v \in V_1 \tag{3.5}$$

$$\sum_{(v,sink) \in E} x_{(v,sink)} = \phi \qquad \forall v \in V_2 \tag{3.6}$$

$$x_{(u,v)} \in \mathbb{Z} \qquad \forall (u,v) \in E \tag{3.7}$$

After the optimum of this system is found via the Branch and Bound method, the arrows are filtered. The arrows longer than *maxSpeed* are discarded, as well as arrows that are estimated to pass outside wrapper hull in the frame containing the blobs at the end of the arrows; this heuristic aims to discard artifactory arrows between unrelated wavefronts which typically pass through zones outside waves. The exact criterion for discarding arrows is as follows: First, for the i-th arrow considered, L_i is the set of pixels under the i-th tracking arrow, determined using Bresenham's line-drawing algorithm between the arrow's start and end points. Then, let $\|wh(L_i)\|$ be the number of pixels of L_i belonging to wrapper hull and $\|L_i\|$ is the total number of pixels belonging to L_i. The i-th arrow is then discarded if $2\|wh(L_i) > L_i\|$ and $\|wh(L_i) > 2\|$. The first condition serves to discard arrows predominantly traversing outside the wrapper hull, the second condition then protects very short arrows in slow conducting zones: due to the nature of wrapper hull and the fact that erosion is larger than dilation by two pixels, each arrow traversing less than four pixels would be

discarded automatically.

The arrows passing the filtering that are not longer than *maxSpeed* are drawn in places of oriented edges $(u, v) : u \in V_1, v \in V_2$ where $x_{(u,v)} = 1$ and the coordinates of their start and end points are saved for future use in feature extraction, along with the "trivial" zero-length arrows representing static wavefronts.

Note that the integer programming formulation above is fairly restrictive, allowing only a single arrow leaving a blob and a single arrow pointing to a blob. This prevents the artifactory behaviour of nearest-neighbour tracking which tends to label several blobs with many outcoming or incoming arrows. The integer programming framework allows relaxing the "at most one in, at most one out" restriction in an elegant way: the capacities of the edges from *source* to V_1 directly correspond to the maximum number of allowed outcoming arrows and the capacities of the edges from V_2 to *sink* directly correspond to the maximum number of allowed incoming arrows. In the following subsection, the required generalisation is described in detail.

For convenience, Ccoffinn contains an implementation of the frequently used method by Bayly et al. (1), although in general case, we recommend using the above-described minimum-cost-maximum-flow approach, as discussed in section S7. For cases when high number of blobs is present and the default approach would be too computationally demanding, Ccoffinn also contains a common and computationally cheap algorithm that can be used in place of the default one: the nearest-neighbour tracking (2). There a wavefront blob in frame *f* is linked to the nearest border blob in frame time *f-1*. According to our experience, however, the method performs poorly in the case of waves that are not very simple (e.g., in case of spiral or wavelet waves).

S1.3.1 Generalisation of the MCMF tracking

The tracking algorithm described previously may suffer from the rule that allows a maximum of one arrow exiting and one arrow entering a blob. An example of a potential problem is a target wave spreading faster in one direction than in others. In such a case, conduction velocity and potentially direction of conduction would be under-tracked; a concrete example of this problem is given in section S7.5.

The integer programming may be modified to accommodate for more arrows entering or leaving a blob. The parameter *maxArrows* determines the maximum allowed number of arrows leaving/entering a blob. It can be set to 'auto', which then uses the smallest integer value that is sufficient to cover all the blobs from the larger partite of the tracking graph; e.g., if movement from 4 to 10 blobs is tracked, *maxArrows*='auto' is considered 3. The exact value is computed separately for each instance of the tracking problem. The integer programming may be then modified as follows:

- The upper bound on flow in the inequality 4.2 is increased to *maxArrows* for all edges from the source to the first partite and from the second partite to the sink. Then, e.g.,

when flow 3 enters a vertex in the first partite, this then forces the tracking to send three distinct arrows from the given vertex (the maximum capacities of edges between the partites are still capped at 1, preventing duplicate arrows).

- ϕ is redefined from $\phi = min(|V_1|, |V_2|)$ to $\phi = min(max(|V_1|, |V_2|), maxArrows \cdot min(|V_1|, |V_2|))$, where V_1, V_2 are the first and second partite respectively. The first component of the minimum makes sure that more tracking arrows are never required than is the size of the larger partite. The second component then makes sure that more arrows than available are never requested of the tracking algorithm. It can be clearly seen that for $maxArrows = 1$, the original algorithm is obtained, while for *maxArrows* = 'auto', the number of tracking arrows is precisely the size of larger partite.

S 2 Feature description

Below are given definitions of features, followed by a brief discussion of their physiological relevance.

S2.1 Feature definitions

S2.1.1 Number of blobs

A scalar value giving the total number of blobs segmented in the dataset.

S2.1.2 Sparseness of blobs

A multiset containing the distances from each blob to their nearest non-self neighbour, providing information on how evenly are blobs spread in the culture.

S2.1.3 Wavefront speed

A multiset of lengths of tracking arrows found throughout the recording, including or excluding the zero-length arrows, based on the value of *useZeroArrows*. The mean value may be used as an indicator of overall conduction velocity, however, note that unlike manual operators who generally focus on the propagation of the fastest wavefronts, mean *wavefront speed* takes into account the speed of wave propagation in all directions where wavefronts are propagating. According to our observations, the 90th or 95th percentile can be used in our datasets to obtain the conduction velocity in the fastest-propagating wavefronts. Using only the longest arrow per dataset is problematic as in complex wave patterns, such as wavelets, long artefactual arrows may appear, linking two parts of the observed wave, that should not be considered representative.

The choice of *useZeroArrows* depends largely on the data available. The motivation for using zero arrows is that in slow-conducting tissue (or very quickly sampled one), a motion may not be captured between consecutive frames, even though it is present. E.g., a straight wave moving one layer of blobs per two frames has a speed 0.5 layer per frame, but the nonzero arrows will be only of length one; these would be captured every other frame, while in the complementary frames, only zero-length arrows are recorded. Thus, averaging all the arrow lengths, one gets the correct answer of 0.5 when considering zero-length arrows. On the other hand, the zero-length arrows may introduce artefactual drop in wavefront speed estimation in, e.g., a well conducting tissue where a dish edge is present - when the wave reaches the dish edge, it will contain zero-length arrows there for several consecutive frames, contributing low CV information to the overall pool of tracking arrows. Thus, the switch *useZeroArrows* should be set to 1 in tissues with low propagation velocity in pixels/frame, while being set to 0 in other cases.

S2.1.4 Interspike duration

For each blob, let us consider its vector of activity (i-th row of *blobActivities*, as obtained in the Section SS1.1). Then, for i-th blob, let IS_i be the multiset of lengths of 2-connected sequences of **0**s that are not leading, nor trailing, that cannot be further extended (i.e., bordering with **1**s on both sides of the interval of **0**s). The leading and trailing intervals of **0**s are excluded as they may not represent a full period between two action potentials. Then,

$$interspike\ duration = \biguplus_{i=1,\ldots,n} IS_i \tag{4}$$

where n is the number of blobs and $\uplus$ stands for multiset union (i.e., union with allowed repeats of elements). Ultimately, all the values are scaled by $1000/frameRate$ to convert the interspike duration in frames to milliseconds.

S2.1.5 APD

This feature is an analogy of interspike duration, except that lengths of non-trailing and non-leading intervals of **1**s are computed. Note, that while feature is a correlate of physiological APD, it may not represent it exactly and the level of APD-N it represents may depend on the nature of the data. This is chiefly due the fact that the APD-N is estimated using the median-filtered signal and sharp peaks in the signal (as seen in the synthetic data) are removed. Thus, estimating, for example, APD70 of the filtered signal may not be equivalent to APD70 in the original signal as the thresholds used for measuring action potential shape depend on the signals peak value.

S2.1.6 Wavefront roughness

Wavefront roughness serves to describe the lack of smoothness in signal propagation. A tracking arrow from location u to location v in frame f is called *suitable* if there is another tracking arrow in the frame f with the origin closer than *usm* (a parameter) micrometers from u. Informally, suitable arrows are these with an arrow nearby in the same frame. For the i-th suitable arrow α, we define β_i as the set of tracking arrows in the same frame with their origin closer than *usm* to the origin of α (including α). Then, we can define:

$$wavefront\,roughness = \biguplus_{i=1,\ldots,m} \overline{S}(\varphi(\beta_i)) \tag{5}$$

where $\varphi(\beta_i)$ is the set of angular components of polar coordinates (in degrees) of all the arrows in β_i, $\overline{S}$ is the circular standard deviation and m is the number of suitable tracking arrows.

It is expected that when a wavefront is propagating smoothly, such as in straight waves, most of its tracking arrows will point in the same direction, contributing low values to *wavefront roughness*, while chaotic activity with many wavebreaks will display lack of local smoothness (i.e., high $\varphi(\beta)$), contributing high values of *wavefront roughness*.

S2.1.7 Beat frequency

Beat frequency of a recording can be used to estimate the beating rate of blobs in the recording. For a single blob and its brightness trace, we define its beat frequency as the number of spikes per second. The *beat frequency* of the whole recording is then simply a multiset union of the beat frequencies of single blobs.

S2.1.8 Entropy

This feature is a variant of the spatiotemporal entropy described previously (3), using the representation of a wave as a set of blobs, rather than a set of pixels (which seems to be the approach used by Jung). Let us construct a 3D space S where $(x, y, f) \in S \Leftrightarrow \exists$ blob c so that (x, y) is the centroid of blob c and $blobActivitiesFiltered(c, f) = 1$ ($blobActivitiesFiltered$ is obtained in Section SS1.2). Informally, a point is present in the space if it represents a centroid of an active blob in a certain frame.

We then define a distance d_S in the space S:

$$d_S((x_1, x_2, x_3), (y_1, y_2, y_3)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \zeta(x_3 - y_3)^2} \tag{6}$$

where ζ is the mean of *Sparseness of blobs* feature. This distance function is essentially an euclidean distance with weighted distance in the z-axis (this is equivalent to an ordinary euclidean distance in a space where frames are spaced ζ pixels apart).

Let us define a graph $G'(V', E')$, where V' contains vertices representing all points in space S and $(u, v) \in E' \Leftrightarrow d_S(u, v) < distanceThreshold$. A *spatiotemporal wave* is then defined as a connected component of G' with the number of the points in the connected component giving its size.

Using notation from the work by Jung (3), spatiotemporal volume of class of size s is defined as $V_s = s \cdot n_s$, where s is the size of connected component and n_s is the number of such connected components in the data. Then, relative coverage of a size class s is $v_s = \frac{V_s}{\sum_s V_s}$. Spatiotemporal entropy is then defined as follows:

$$S = -\sum_s v_s \ln v_s \tag{7}$$

Even though entropy is a popular measure of order, providing a single number for the whole dataset, it has its limitations. We can observe, for example that a completely random activity of isolated blobs, yielding many connected components of size 1, has the same (zero) entropy as when there is a single large wave present in the recording. Also, a very heterogeneous wave, such as a group of wavelets, may still be a single connected object in the 3D space, depending on the exact shape of the waves and quantity of noise; this feature then becomes fairly unstable, with small changes to the recording causing large difference in entropy.

Efficient computation of entropy in our context, namely the decomposition of G' into connected components is not entirely trivial. The only difficulty in computing the entropy is the decomposition of the graph G into connected components due to large size of the graph itself. With hundreds of active blobs per frame and a video that is thousand frames long (which are both possible in our data), the number of edges present in the graph can be too large to fit into memory of a desktop computer (especially using memory-inefficient representations, such as adjacency matrix).

The first observation that helps is that the whole graph does not have to be represented explicitly at any time for us to obtain the decomposition into connected components. The standard DFS-based algorithm for finding connected components can be adapted to not require the whole graph. Let us recall that an edge is present between vertices in G if the two vertices are closer than *distanceThreshold*. Thus, the step when the search expands into a vertex's neighbours (usually determined directly from the graph representation) can be modified to instead query all the other unlabelled vertices (candidate neighbours), expanding into these that are near enough to the vertex expanded. While this greatly reduces the memory complexity from $O(|V|)^2$ to $O(|V|)$, it also increases the time complexity from $O(|V|)$ to $O(|V|)^2$, making the computation impractically long.

A second improvement to the basic algorithm is restricting the search of candidate neighbours so that not all vertices are searched. We can divide the 3D spatiotemporal space occupied by the vertices of G into cubes with side length of *distanceThreshold* and for each cube, we record the list of vertices present in it. Then, when looking for neighbours of a vertex to

be expanded v, instead of testing the distance to all the vertices in the space, we test only the distance to the same cube and the 26 neigbouring ones. In a standard case of our data, 256x256 pixels and *distanceThreshold* of ca. 16, we search only at most 27 cubes instead of 4096 there are, which is a considerable time-saver. This function has been implemented in C to further improve performance and connected to the main code via MEX interface.

S2.1.9 Perimeter-to-area ratio

In order to counteract the deficiencies of *entropy*, we have designed a different measure of the degree of order of the recorded activity. For each frame, we obtain the mask of active blobs in the frame (after filtering out blobs in sparse regions, i.e., using *blobActivityFiltered*). Morphological closing with circular structural element and radius *distanceThreshold*/2 is performed to fill holes between blobs belonging to a same wave (but not mixing separate waves together). Then, we define the *perimeter-to-area ratio* in the i-th frame as:

$$perimeter - to - area(i) = \frac{\#sidePixels(m_i)}{\#pixels(m_i)} \tag{8}$$

where m_i is the processed mask of the i-th frame, $\#sidePixels(m_i)$ is the number of one-containing pixels in m_i that border with a zero ("perimeter"), and $\#pixels(m_i)$ is the total number of ones in m_i ("area").

The whole feature *perimeter-to-area ratio* is then defined as a multiset union of all the $perimeter - to - area(i)$ such that $\#pixels(m_i) > 0$.

We can clearly see that a wavelet activity will have overall high *perimeter-to-area ratio* due to large perimeter and low area, unlike, e.g., round target waves, which have large areas and small perimeters.

A slightly different approach would be to compute the surface and volume of a 3D voxel, instead of computing perimeter-to-area ratio for each frame separately. However, such approach would be then very sensitive to wave speed: an organised wave that moves very fast would have larger surface than a wave of the same type that travels more slowly due to larger number of facets between two frames. We consider this effect undesirable, as a measure of wavefront structure should not depend on propagation speed, which is why we have adopted the frame-by-frame approach instead.

A limitation of this approach is that waves entering the field of view (FOV) may have a fairly high *perimeter-to-area ratio* at the given frame, as most of the true area of the wave is not seen. However, such cases are highly unlikely to affect the feature value in a major way, as waves tend to be present in FOV for multiple frames. If this limitation affects the results severely for some unexpected reasons, the code for this feature's extraction could be modified to ignore several leading and trailing of activity in each wave when *perimeter-to-area ratio* is determined, thus discarding the frames when a wave enters or leaves the FOV altogether.

S2.2 Physiological relevance of features

Wavefront speed, *APD*, and *firing frequency* are widely reported measures of cardiac wave dynamics that yield information on connectivity, upstroke velocity, automaticity, and ion channel kinetics, e.g. calcium dynamics (1, 4, 5). *Interspike duration* is a measure of cardiac recovery time, which is often used to describe the presence or absence of dynamic changes in action potential shape, e.g., alternans (6). The other measures described in the paper (*wavefront roughness*, *entropy*, and *perimeter to area ratio*) attempt to quantify the relative complexity of propagating waves in tissue. Since complex dynamics can be caused by a wide variety of different causes, it is often impossible to assign a one to one electrophysiological or structural correlation to these measures. However, these measures are directly relevant to the understanding wavefront interactions and arrhythmogenesis in our preparations.

Wavefront roughness is an indirect measure of the connectivity of the tissue as well as the number and size of local structural heterogeneities. Well connected tissue will generate smooth wavefronts whereas locally disconnected tissue, or tissue in the presence of spatially distributed heterogeneities, tend to generate wavefronts that appear fractured (7–11). *Entropy* is a measure of spatiotemporal complexity (3, 12, 13), and can be used to quantify variability in wavefront structure. Tissue experiencing changes in electrophysiology, or connectivity (14) may be expected to display higher entropy values. However, *entropy* may be difficult to interpret as it does not necessarily map onto an intuitive understanding of complex cardiac dynamics: for example, many multiple interacting wavelets (e.g. fibrillation) may have low entropy if the wavefronts have similar sizes. Entropy calculation is also sensitive to noise and the threshold for activation. In contrast, wave *perimeter to area ratio* directly maps to an intuitive measure of spatial complexity in our data sets. Wave *perimeter to area ratio* gives information on the shape of a wave (larger values map to smaller waves and waves with non-circular shapes). As with the *wavefront roughness* descriptor, *perimeter to area ratio* is a function of local heterogeneity, but also a measure of dynamic instability in homogeneous tissue. For example multiple wavelets caused by steep restitution can have a low roughness value and a high perimeter to area value, whereas multiple wavelets caused by local heterogeneity are expected to have high values for both descriptors. Taken together, these measures can be used to differentiate different experimental preparations (e.g. Table 1).

S 3 An overview of parameters

Table S.1 shows a list of constants given by physical conditions of the experiment, Table S.2 contains the free parameters of the methods used in our toolkit, and Table S.3 contains the computed parameters, values of which are based on constants and free parameters. The table S.4 contains an overview of switches that control the behaviour of the software.

Name	**Description**
$frameRate$	The frame rate of the recording in frames per second.
$mimps$	The upper limit of speed of waves in micrometres per ms.
$pixelMicroMetres$	How many micrometres does a side of a pixel measure (based on the resolution of the recording).

Table S.1: An overview of constants that need to be specified as a part of input. These numbers can be different for each video processed in a batch (e.g., all videos of cell cultures created and imaged using a fixed protocol, before and after addition of a drug).

Name	**Description**	Value
hlt, *llt*, *cws*	The parameters of the SeNeCA segmentation algorithm for the case when blob-based segmentation is used: High light threshold, low light threshold , and contrast window size respectively (these are used only if SeNeCA is used for blob segmentation). Fourth parameter described in the article on SeNeCA, *minLight*, is not used, i.e., set to 0.	1.4, 1.3, 8
$binningfactor$, $minimumSNR$	The parameters of the filtered binning segmentation algorithm: the video is binned into squares with side length $binningfactor$, ignoring squares traces of which have SNR lower than $minimumSNR$ (these are used only if filtered binning is used for blob segmentation). $minimumSNR$ is 0 by default, meaning that no filtering is performed.	3, 0
$smoothingFactor$	Used to compute $smoothingParameter$; the larger it is, the shorter the filter. E.g., for the default value and 100fps, smoothing width is 100ms	10
$activityThreshold$	Used to determine which segments of dIS (as obtained in Section S1.1) are labelled as containing an AP upstroke	2
$apdN$	The N determines the APD level measured; scaled to 0-1. I.e., 0.5 represents APD50. The default value used is suited for data with relatively high amount of noise and disorder.	0.3
$minAPD, maxAPD$	The minimum/maximum duration of an action potential in ms. If a candidate action potential does not fit within this range, it is discarded.	25, 800
$distanceThreshold$	If two blobs are closer than this (in pixels), they are considered to belong to the same wave. This parameter may be either given exactly as a number, or as 'auto', being estimated heuristically as 4· median distance of each blob to its nearest non-self neighbours.	'auto'
$minDensity$	If a blob considered active based on $activityThreshold$ does not contain at least this many other blobs considered active within $distanceThreshold$, it is relabelled as not active to keep focus on non-sparse activity.	5
$maxArrows$	The maximum number of arrows starting and ending in each blob; used in MCMF tracking. This may be an integer value or 'auto'.	1
usm	Used in feature extraction, determines the radius (in micrometres) around a start point of a tracking arrow in which other arrows are considered for computing roughness of wavefront.	750

Table S.2: An overview of free parameters that need to be specified as a part of input; these numbers are expected to be the same for all the videos processed in a batch. The third column contains values that we used in experiments described in this text, included in order to improve the reproducibility of our results.

Name	Description	Formula
$uniformitySurroundings$	usm in pixels	$\frac{usm}{pixelMicroMetres}$
$maxSpeed$	Maximum speed of waves in pixels per frame	$\frac{1000 \cdot mimpms}{frameRate \cdot pixelMicroMetres}$
$smoothingParameter$	Size of averaging filter (in frames) used in smoothing intensity traces of blobsA. It set to make a single cardiac action potential smooth, but not to merge two separate ones.	$\frac{frameRate}{smoothingFactor}$

Table S.3: An overview of parameters based on constants and free parameters.

Name	Description	Values
isDebug	If 1, traces of all the blobs segmented in the processed recording are stored in tmp subfolder, along with their smoothed version and segmentation of activity.	0/1
isDoubleHump	If 1, additional smoothing is applied in spike-mining to connect "double-humps" resulting from pre-processing of dye-free imaged data.	0/1
segmentationMethod	This switch selects between SeNeCA cell segmentation (for recordings with blobs) and binningSNR segmentation (for continuously activated recordings)	'seneca'/'binningSNR'
toPlot	If 1, the segmentation and tracking progress is visualised on-screen	0/1
trackingMethod	This determines whether the default minimum-cost-maximum-flow approach or Bayly's method is used for tracking wavefronts.	'MCMF'/'Bayly'
useZeroArrows	If 1, zero-length arrows are considered in CV estimation. This is to be used in cases of low pixels-per-frame conduction.	0/1
verbose	If 1, progress of computation is output to the console and progress bars are displayed to track the progress of time-consuming steps.	0/1

Table S.4: An overview of switches used in Ccoffinn.

S 4 Data description

S4.1 Synthetic data

While it is important to demonstrate usefulness of methodology using real data, such an evaluation is limited as the underlying signal is intrinsically unknown and the irregularity of the activity in real systems makes it complex to analyse. Therefore, in addition to evaluating our software on real data, we also performed an evaluation using synthetic datasets, where we can determine the correctness of the evaluated methodology. The main aim is to investigate the robustness of the method to sampling frequency and noise quantity in the data.

The cardiac activity was modelled using a cellular automaton (15), which allowed us to create various types of waves and to measure them at different sampling frequencies. Below are described three experiments: Syn1, Syn2, and Syn3, which aim to test different properties of the process of feature extraction. All created videos have the resolution of 255x255 pixels and include 2601 blobs (a grid of 51-by-51). Examples of frames of the synthetic data can be found in Figure S.3 and Figure S.5.

In experiment Syn1, three 400 frame videos sampled at 50 fps are compared, containing a spiral wave, a planar wave, and a target wave respectively. These three types of waves are the major types of waves observed in cardiac monolayers and our aim was to confirm that they are segmented correctly.

In experiment Syn2, three different types of a wave (planar, spiral, and wavelet) are visualised at three different sampling frequencies (25, 50, and 100 fps; 200, 400, and 800 frames respectively) and we observe the effect on the values of the features extracted. The recording is generated at 100fps and then subsampled to the respective lower sampling frequencies.

In experiment Syn3, the same three types of wave are observed for 400 frames at 50 fps with three levels of Gaussian additive noise with zero mean and variances 0.02, 0.04, 0.06,..., 1 (on a scale of 0-1) and the focus is on the effect of the noise on the features extracted.

S4.2 Real data

Two types of slow-conducting cell cultures imaged using a dye-free approach were used to evaluate the effectiveness of the toolkit: a culture of ventricular myocytes from SD rat pups (n=19) and a co-culture of ventricular myocytes grown along stellate cardiac neurons of litter mates of the rats (n=16). The protocol of culturing is given in (16). A temporal filter has been applied to the raw data, replacing f_i with $abs(f_i - f_{i-5})$, where f_i is the i-th frame. The filtered data can be interpreted as the amount of motion (contraction or relaxation) in a given pixel. A sample script is provided with Ccoffinn, showing how to apply such filtering to a dye-free dataset obtained using phase imaging.

Additionally, several datasets of calcium-imaged cell cultures (with high CV compared to

the dye-free imaged cultures) were used to demonstrate the usefulness of Ccoffinn for such an imaging modality. The protocol used is described in (17).

S 5 Implementation notes

The methods for segmentation and tracking of waves, as well as the feature extraction, were written in Matlab, using C code and MEX interface in performance-critical areas. Pure-Matlab alternatives to the C functions are also provided for convenience and greater platform independence. The GLPK library was used as a fast integer programming solver, connected to Matlab via GLPKMEX interface. The cellular automata used to generate the synthetic datasets is written in Java. The runtime of the segmentation and tracking depends greatly on the quantity and the pattern of activity of the blobs in the data. On the real data analysed in this report, the average number of objects segmented in a video was ≈ 4300 and the average runtime of segmentation and tracking was 0.11 seconds per frame (with the visualisation and console verbosity turned off). The computer used for the computation was a PC with a six-core processor Intel Xeon E5-1650 v3, 16GB of operating memory and a magnetic hard drive.

S 6 Results: Validation using synthetic data

Below are described the results of the three experiments Syn1, Syn2, Syn3, described in Section S4.1.

S6.0.1 Syn1: Results

Figure S.3 shows an output of the segmentation and tracking algorithms on three model waves. The output of the segmentation and tracking algorithms was visually assessed and found to be correct.

S6.0.2 Syn2: Results

The features extracted from the three datasets (varying by their sampling frequency) show that the sampling frequency of the video generally does not seem to drastically affect the features extracted (note that the change in *number of blobs* is minuscule, as can be judged from y-axis). However, we note that while *entropy* does not seem to be overly affected by the sampling frequency, it does not capture the complexity of waves very well. Intuitively, we would expect straight waves to be the most orderly with wavelets most disorganised. However, it is clearly not so and the straight waves have by far the largest entropy. The reason for this is that in the 3D spatiotemporal space, different iterations of a straight wave

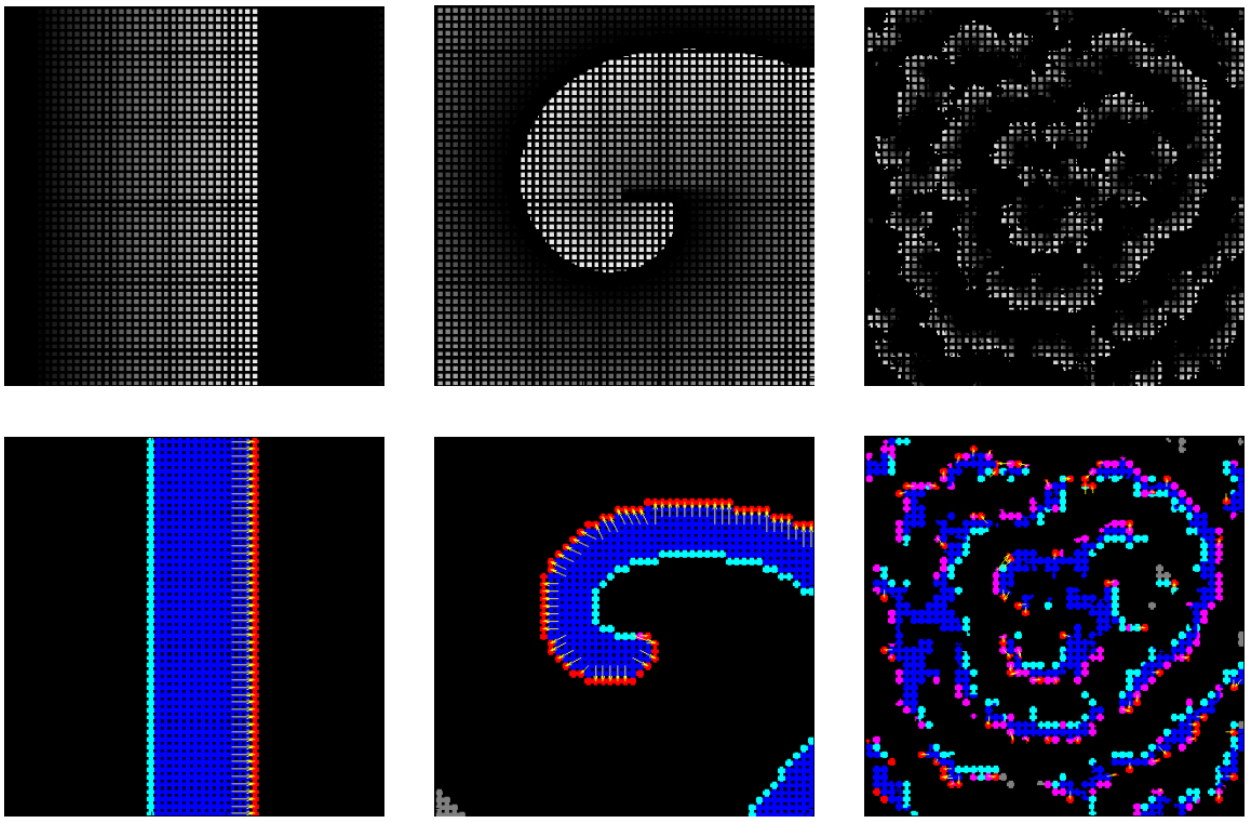


Figure S.3: Three types of synthetic waves, before (top) and after (bottom) segmentation and tracking (we note that the tracking depends on the previous frame). To the left is a straight wave, in the middle is a spiral wave, and to the right is a wavelet wave. The interpretation of colours is identical to Figure 1 in the main text.

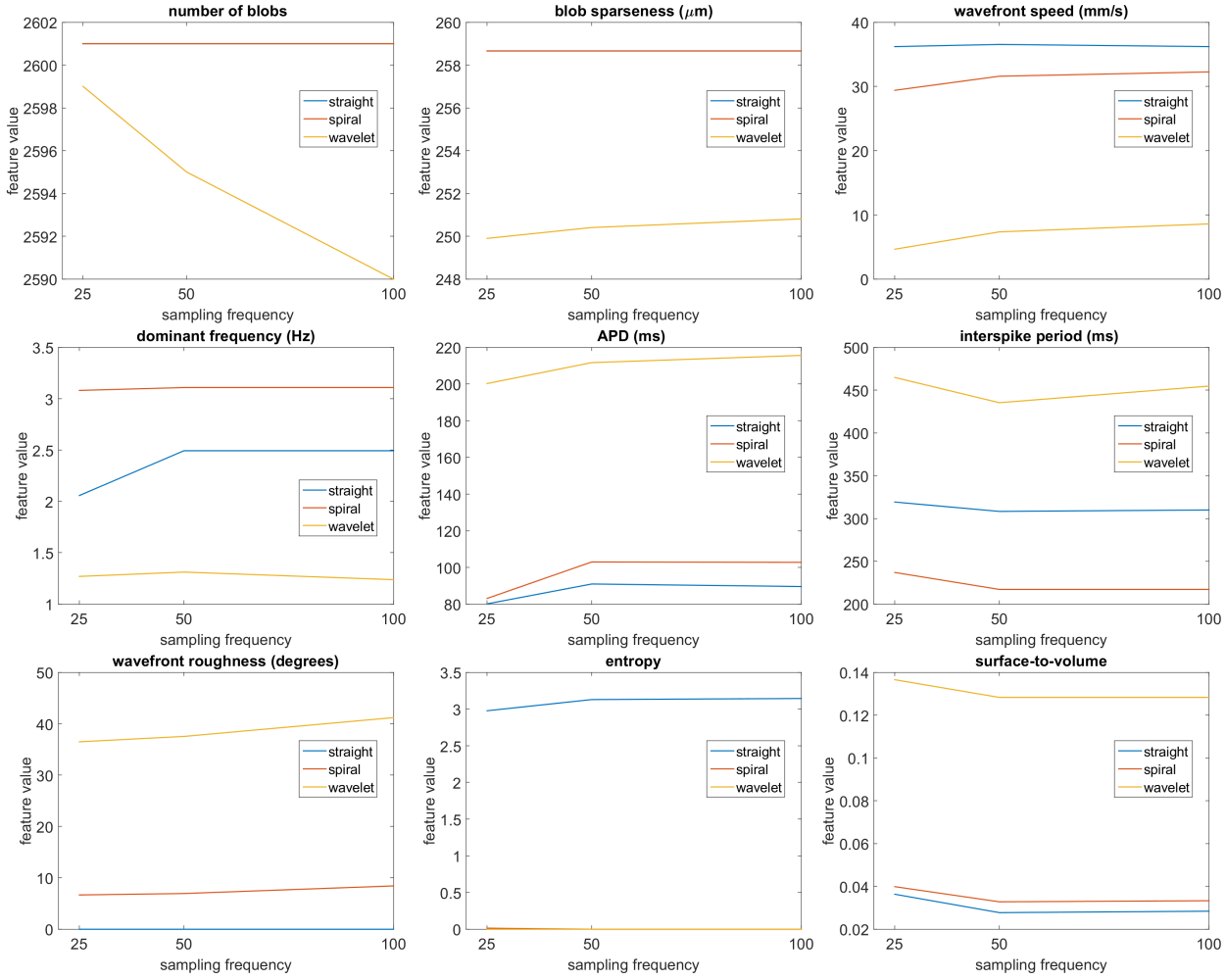


Figure S.4: The effect of sampling frequency on values of features (or means of features in case of non-scalar features). In *number of blobs* and *blob sparseness*, spiral and straight wave have the same values for all sampling frequencies; the same holds for wavelet and spiral wave values in *entropy*. In all the waves, ground truth for *number of blobs* is 2601. For straight waves, ground truth values of *wavefront speed*, *dominant frequency*, and *wavefront roughness* can be estimated; the values are 36.8 mm/s, 2.5 Hz, and 0, respectively.

are completely separate objects of possibly slightly different size. Unlike that, spiral waves (and many wavelet activities) are merely a single connected spatiotemporal object. Thus, even though *entropy* is an elegant and theoretically well-justified method, this flaw is too serious for the feature to be practically usefl and alternative methods of order estimation provided in the toolkit are recommended.

S6.0.3 Syn3: Results

An example of data with four selected levels of additive noise are given in Figure S.5. Figure S.6 shows how features are affected by increasing noise. In features based on segmentation, namely *number of blobs* and *blob sparseness* (min distance to nearest neighbour), the only case with a trend effect of noise is the case of straight waves, where at high noise levels, locations of blobs are sometimes estimated incorrectly with the centroid being slightly shifted. We wondered why there is a difference between straight and spiral waves, which are both organised. Examining the data, we found that the straight-waves pass through any given point less frequently than in the case of spiral waves (this is also manifested in the *dominantFrequency*) and that the process of segmenting the averaged frames of the recording thus has an input with poorer signal-to-noise ratio in the case of straight waves (relatively more frames of noise for a given pixel than in the case of spiral wave), which is why straight waves are more damaged by the noise. This reveals an important observation that if Ccoffinn was to be used for recordings with very high levels of noise and very sparse waves, the recording should be summarised in a different way than the average of input frames, as this approach is sensitive to noise (e.g., an approach based on high quantiles and/or maximum might be better in such a context, as pixels containing pure noise are unlikely to manifest as high values as pixels that contain genuine activation).

The feature *wavefront speed* tends to decrease with increasing noise level. The reason behind is the fact that overly long arrows are deleted. Considering a single arrow in a noise-free environment, when noise is added, the arrow can stay the same, shorten, or prolong; however, the prolongations are limited as too long arrows are removed. For this reason, the net effect of noise perturbation is mean arrow shortening.

In *dominant frequency*, we see a decrease as the noise is added, accompanied by prolongation of both *APD* and *interspike period*. This is again due to the asymmetric filtering process: an action potential is much more likely to be shortened below the *minAPD* (or even to be completely removed by noise) than prolonged above *maxAPD*.

The *wavefront roughness* grows with noise in the organised waves, as their usually smooth wavefronts become jagged and heterogeneous, increasing the roughness of propagation. We note, however, that at all levels of noise, the three types of waves are ordered in their most natural order of smoothness of propagation. *Entropy* again fails to discern the degree of order associated with the types of waves, similarly to the result from experiment Syn2. *perimeter-to-area ratio* changes little with increasing noise and maintains the ordering of organised

versus disorganised waves at all noise levels. The separation of spiral and straight waves is relatively small, but stable.

Overall, we believe that while the whole process of data analysis and feature extraction is not immune to the quantity of noise present in the data, it is at least fairly resistant, giving good results even at high noise levels in most features (the only exception is *wavefront roughness*). Especially in the case of noise with standard deviation smaller than 0.2, most features are highly stable.

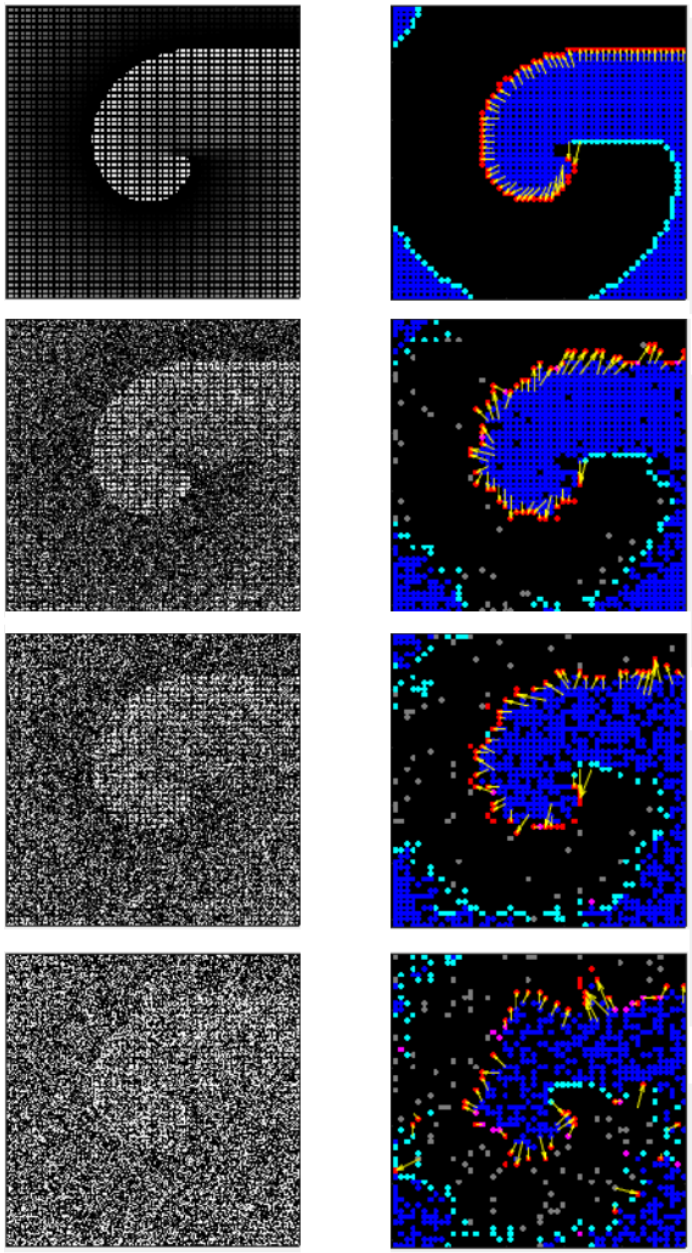


Figure S.5: A spiral wave generated using a cellular automaton with four levels of additive noise (given as the standard deviation of the noise on scale 0-1): 0, 0.2, 0.4, and 1. The interpretation of colours is identical to Figure 1 in the main article.

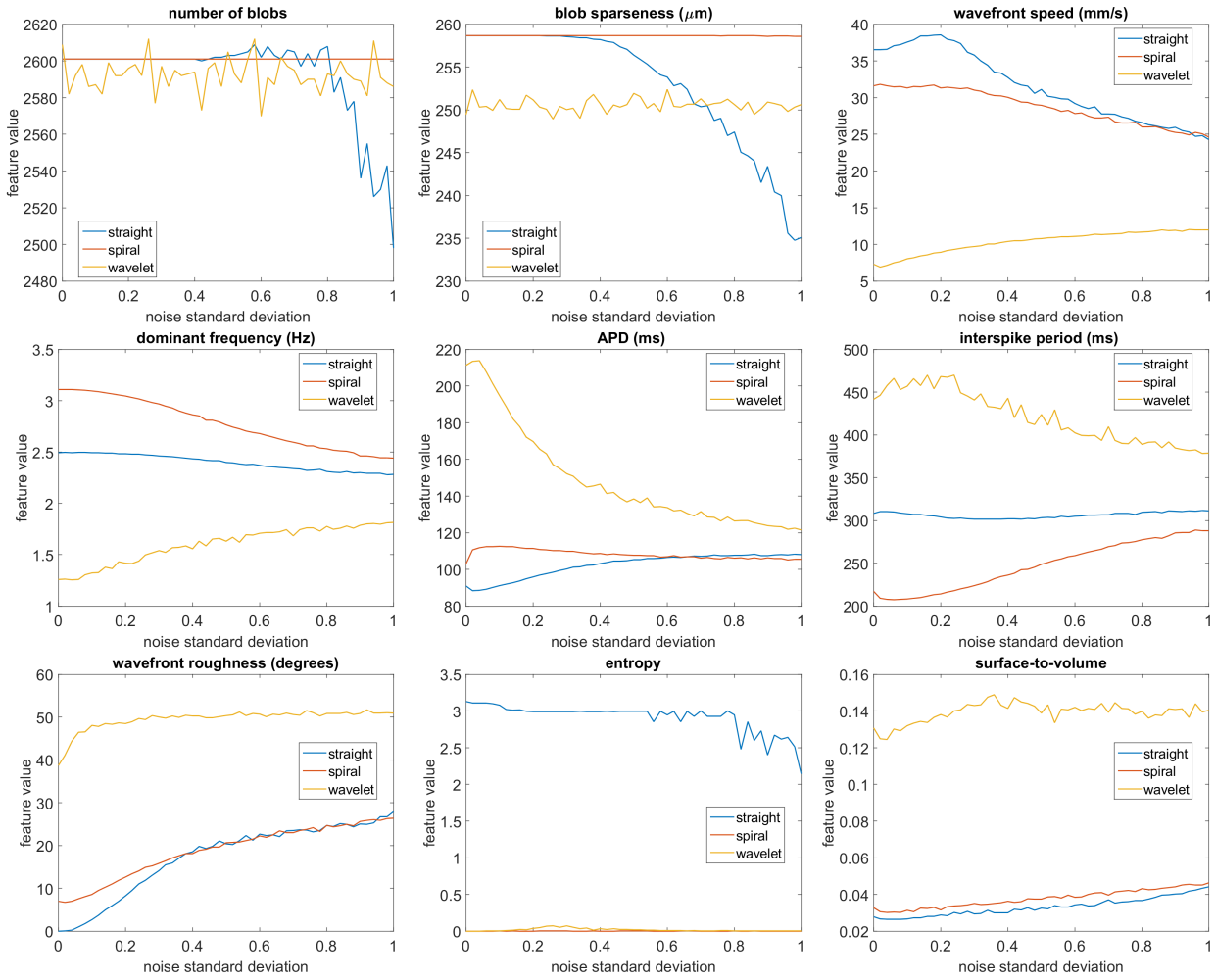


Figure S.6: The effect of increasing noise standard deviation on values of features (or means of features in case of nonscalar features). In all the waves, ground truth for *number of blobs* is 2601. For straight waves, ground truth values of *wavefront speed*, *dominant frequency*, and *wavefront roughness* can be estimated; the values are 36.8 mm/s, 2.5 Hz, and 0, respectively.

S 7 Bayly's wave tracking algorithm and MCMF tracking in high-resolution anisotropic data

In this section are discussed the problems associated with the algorithm for wave tracking and conduction velocity (CV) estimation by Bayly et al. (1) in high-resolution, heterogeneously conductive tissue, as well as discussing potential shortcomings of the MCMF algorithm which is the default approach of Ccoffinn.

Bayly's method considers each "active point" in a wave (e.g., a pixel), finds points in 3D spatiotemporal space that are closer than given thresholds in the x,y,t dimensions, and fits a quadratic surface through these using least squares criterion. Partial derivatives are then used along x,y axes to find a velocity vector for the given active point. A small problem intrinsic to this approach is the fact that the problem of surface fitting may be poorly conditioned, making an estimation of CV impossible in a given point. This problem happened several times in most datasets available to us, however, this error did not occur frequently enough to significantly impact CV mapping or make the results uninformative.

In the relatively high-resolution data available to us, where a wave is represented by up to thousands of active objects and for a reasonably sized spatiotemporal neighbourhood, the basic algorithm is fairly slow, taking almost a minute per frame of data, making high-throughput analysis impossible. However, using Ccoffinn's framework for wave segmentation, an adjustment to the basic algorithm can be made, where for an active point (x, y, f) (in frame f), "near active points" are taken only from wavefront blobs in frames $f-1$, f, and $f+1$, closer than *maxSpeed* in the plane given by first two dimensions of the space. Using only two adjacent frames along the one in which the point (x, y, f) is used, as a quadratic surface does not have enough degrees of freedom to correctly fit all the points in a larger temporal neighbourhood.

Considering only wavefront blobs, rather than all the active blobs, the algorithm becomes much faster, even though it is still an order of magnitude slower than the minimum-cost-maximum-flow (MCMF) approach described previously. We note that this holds for the data available to us at the moment; it would most likely not scale for extremely large wavefronts of hundreds of thousands blobs, where the potential exponentiality of the integer programming used to solve the MCMF problem would overcome the high-multiplicative-constant polynomial complexity of the Bayly's algorithm. However, both algorithms would be extremely slow in such a case and an entirely new approach to tracking would have to be adopted.

The main issue associated with Bayly's algorithm is that in heterogeneously conductive tissue (such as poorly coupled cell cultures, cultures rich in fibroblasts, or intact myocardial tissue with myocardial infarction), where the heterogeneity captured by sufficiently high resolution imaging method is genuine, the surface fitting approach can consistently yield counterintuitive and incorrect results due to its dependence on smoothing. Below are several

examples of wavefront topologies constructed to illustrate certain scenarios, where Bayly's algorithm fails to provide correct answer. We note that the errors demonstrated below are not not a principal flaw of Bayly's algorithm, as it was designed for low-resolution electrophysiological data where the problems tend not to occur, but rather an illustration of the algorithm's struggles with heterogeneously conducting tissue. According to our experience, even in high-resolution data, Bayly's method performs well as long as the tisue is isotropic/homogeneously conducting. Furthermore, in section S7.5 are also discussed potential issues associated with the non-generalised and generalised MCMF algorithm used in Ccoffinn, showing how both MCMF and Bayly's method may struggle in certain context.

We visualise the tracking information using 2D plots containing points in the plane, representing the wavefront, where the colour further codes for the time slice in which the points are considered active; the warmer the colour, the later in the "recording". Estimated vectors are shown using arrows of the given direction and length. When an arrow is missing, it was either a case of a poorly conditioned problem, or the arrow was rejected as too long (longer than neighbourhood radius, which is essentially a representation of maximum conduction velocity). When a particular point is discussed, it is encircled in red and the spatial neighbourhood is shown using a dashed magenta circle (however, only points that are at most 1 frame away from the encircled point may be considered, even if they were close enough in the spatial plane). Furthermore, in selected cases, we show the points in neighbourhood in 3D with the quadratic surface fitted to them.

S7.1 Slow band in a straight wave

Figures S.7 and S7.1 show a case of a straight wave travelling from bottom to top (with central point of the wavefront slowing down over time, limping behind the rest of the wavefront), with three different radii of neighbourhood shown.

In the panel S.7a are shown velocity vectors for the case when neighbourhood radius is 30. The main systematic error is present at the centre of the wave, where tracking arrows tend to aim towards the centre, rather than purely to the top of the frame. The reason is that in frames 6 and 7, only two points are present within the magenta circle, while in frame 8, three points are present; two a part of "normal" wavefront, one a part of the slowed band. The fitted surface (shown in S.7b) passes exactly through the seven points in the neighbourhood, but is practically wrong, both in the direction, and the conduction velocity estimate. A secondary type of error is present at the edges of the frame, where the optimal fit is not unique and the fitting procedure in Matlab returns unstable results. In practical settings, these cases would be best detected automatically and discarded.

At the bottom of panel S.7c is an identical scenario, but with neighbourhood radius of 60. The problem with unstable borders is solved at the sides (where for each point at the side, 9 points are in its neighbourhood rather than 6 in the previous case, making the optimal solution unique). However, the problem with incorrect estimation of direction in the vicinity

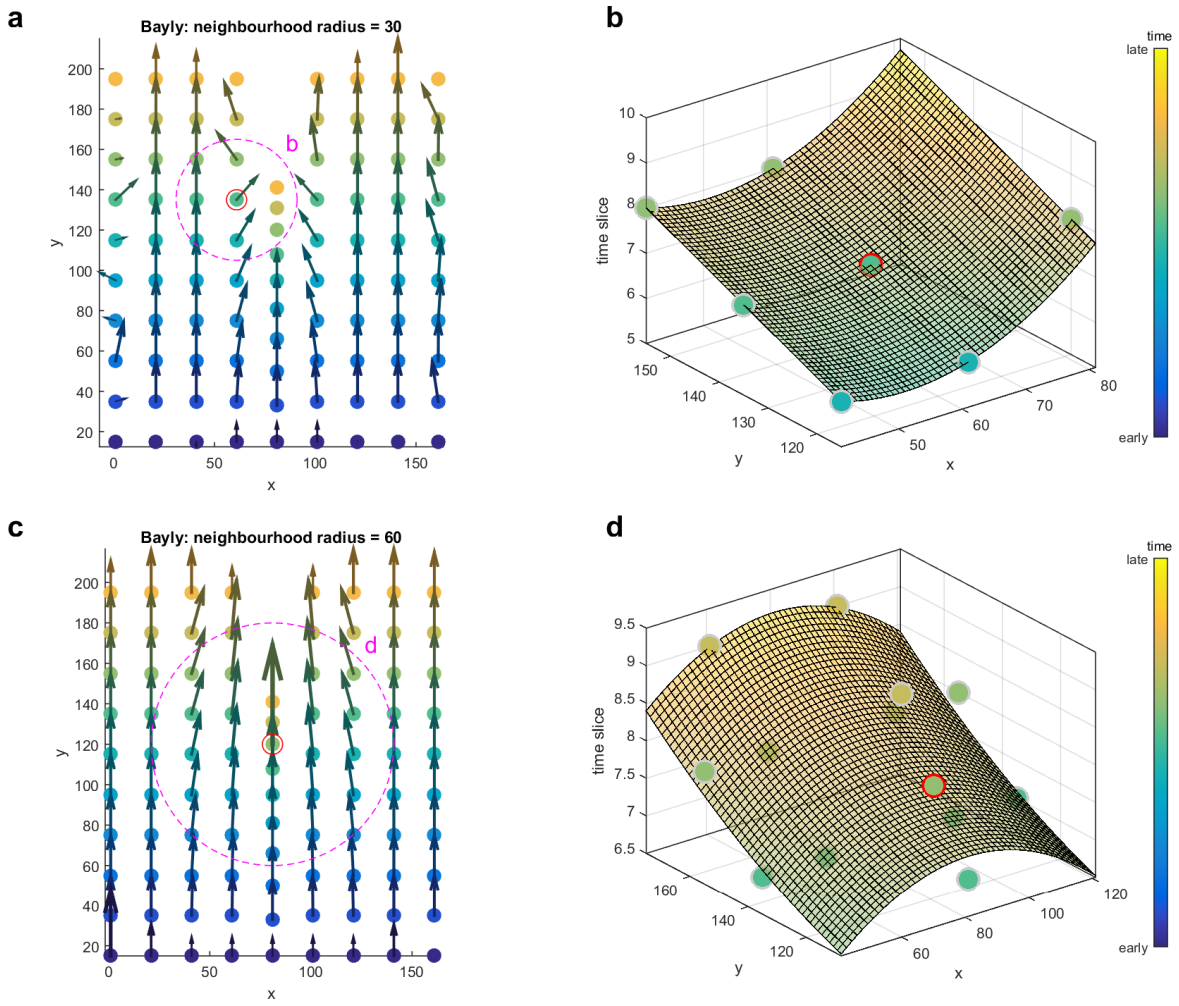


Figure S.7: In the left column are diagrams of wavefront of a straight wave travelling from bottom to top with a slowing-down segment. The colours code for time. Velocity vectors obtained using Bayly's algorithm are superimposed on the locations of "wavefront blobs". In a) is the tracking information for neighbourhood radius 30, in b) for neighbourhood radius 60. In the right column are shown quadratic surfaces fitted through the respective points and their surroundings shown in the left column.

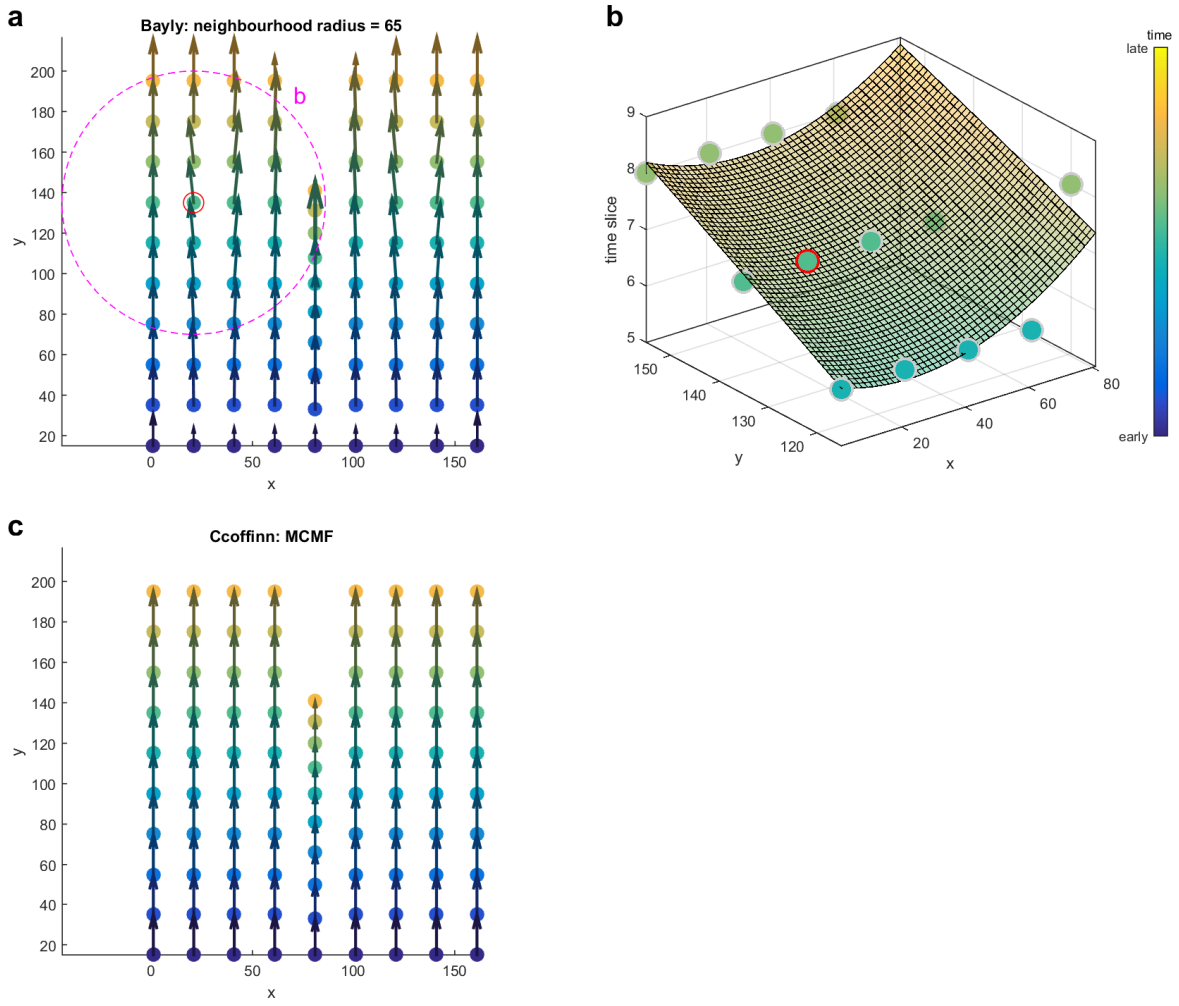


Figure S.8: a) and b) are an analogy of the previous figure, with neighbourhood radius 65. In c) is given the tracking information obtained using Ccoffinn's MCMF tracking algorithm.

of the slowing down band is aggravated overall. Furthermore, CV becomes overestimated severely in the slowing down band, as demonstrated by the case of the encircled point (producing the longest arrow shown in the plot) and the respective fitted surface. This is due to the fact that when a surface is fitted around the encircled point, the algorithm also considers the points of the fast part of wavefront that are some distance away, which results in an incorrect high CV estimate. Indeed, this error increases as the distance between the fast and slow wavefront increases making the estimated CV excessively large.

Increasing the neighbourhood size to 65 yields the plot in S7.1a, revealing yet another type of error, e.g., in the point circled in red, where the velocity vector points away from the slowing down band. When fitting the respective surface (shown in S7.1b), the surface has to be lifted at its right part to pass near the point belonging to the slow part of the wavefront; however, as the fitted model is a quadratic surface, this forces the left part of the surface to be lifted up, making the partial derivative such that the velocity vector then aims away from the centre of both the fitted surface, and the centre of the velocity plot.

The panel c contains the wave tracking information by Ccoffinn's MCMF method, providing correct estimation of both CV and conduction direction.

The three examples of various mistakes made by Bayly's algorithm on the particular data also highlight the sensitivity of the method to the neighbourhood radius, as the tracking outputs in the vicinity of the slowing down segment differ between the three examples rather considerably. At the same time, there is no "correct" neighbourhood size in general. In a heterogeneously conducting excitable medium, the example given above may be only a small part of the overall activation pattern; "good" values of neighbourhood size may then differ substantially between distinct areas of the tissue. As the algorithm allows only a single parametrisation of neighbourhood size, it then follows that at least some areas will contain tracking artifacts. This issue could be probably ameliorated by extending the Bayly's algorithm to somehow dynamically determine an appropriate neighbourhood size.

S7.2 Near wavefronts in a single wave

A second type of problem associated with the surface-fitting approach of Bayly is that different wavefronts of a single wave can affect one another; one example is shown in the panel S.9a. The panel shows a wave spreading towards the top and bottom of the plot; it also steers towards left in its top-travelling segment. The steering then affects the direction and length of several arrows associated with the bottom-travelling segment of the wave, which is incorrect. The tracking by MCMF shown in the panel S.9b is correct.

S7.3 Wave shape affects CV and direction estimation

The panel S.10a shows two waves of identical CV (10 arbitrary units), travelling from bottom to top; a straight wave to the left and a pointed "arrow" wave to the right. While the CV

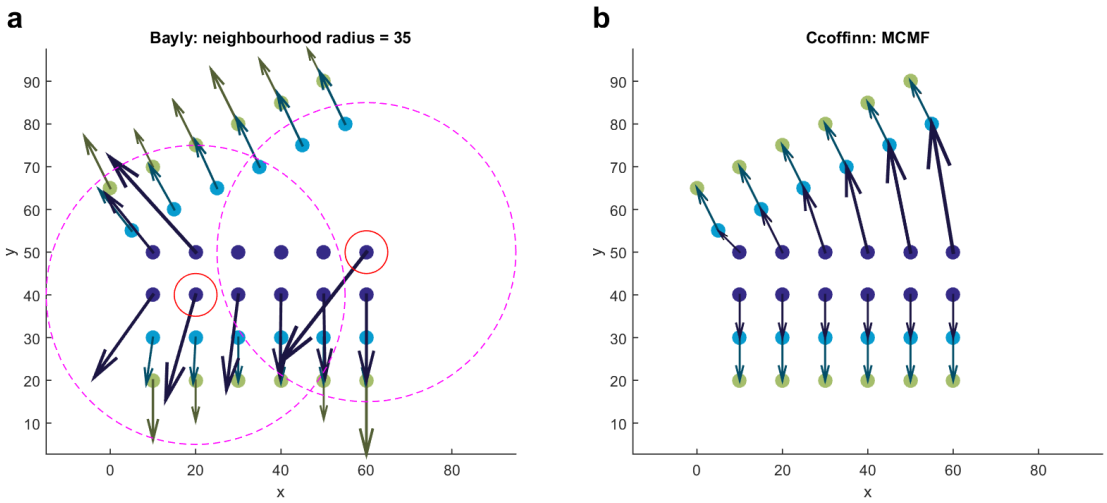


Figure S.9: In a) is shown the activation diagram of a simulated wave spreading simultaneously towards the top and the bottom of the plot; the interpretation of colours and arrows obtained using Bayly's algorithm is identical to the previous figure. In b) is given the tracking information obtained using Ccoffinn's MCMF algorithm.

estimation is correct in the straight wave, when the first and last temporal slices are ignored, both local CVs and propagation directions are almost all incorrect in the arrow-shaped wave, suggesting at a major sensitivity of the Bayly's algorithm to the wave shape, rather than purely to the actual conduction velocity. The panel S.10b gives the surface fitted to the left encircled point's surroundings, showing why CV is mildly overestimated at the point. The panel S.10c then shows the surface fitted to the right encircled point, showing why the direction of propagation is incorrectly estimated. The panel S.10d contains the correct tracking outcome obtained using MCMF algorithm.

S7.4 Wavefront jaggedness yields overestimation of CV

In Figure S.11 is shown the tracking output of Bayly's algorithm on a straight wave with jagged wavefront that propagates from bottom to top. Even though the true CV is 10 units per time step, the estimated CV is clearly and systematically estimated to be higher. In certain cases (e.g., the encircled one), the direction is also estimated poorly, due to the asymmetrical positioning of neighbouring active points in the neighbourhood.

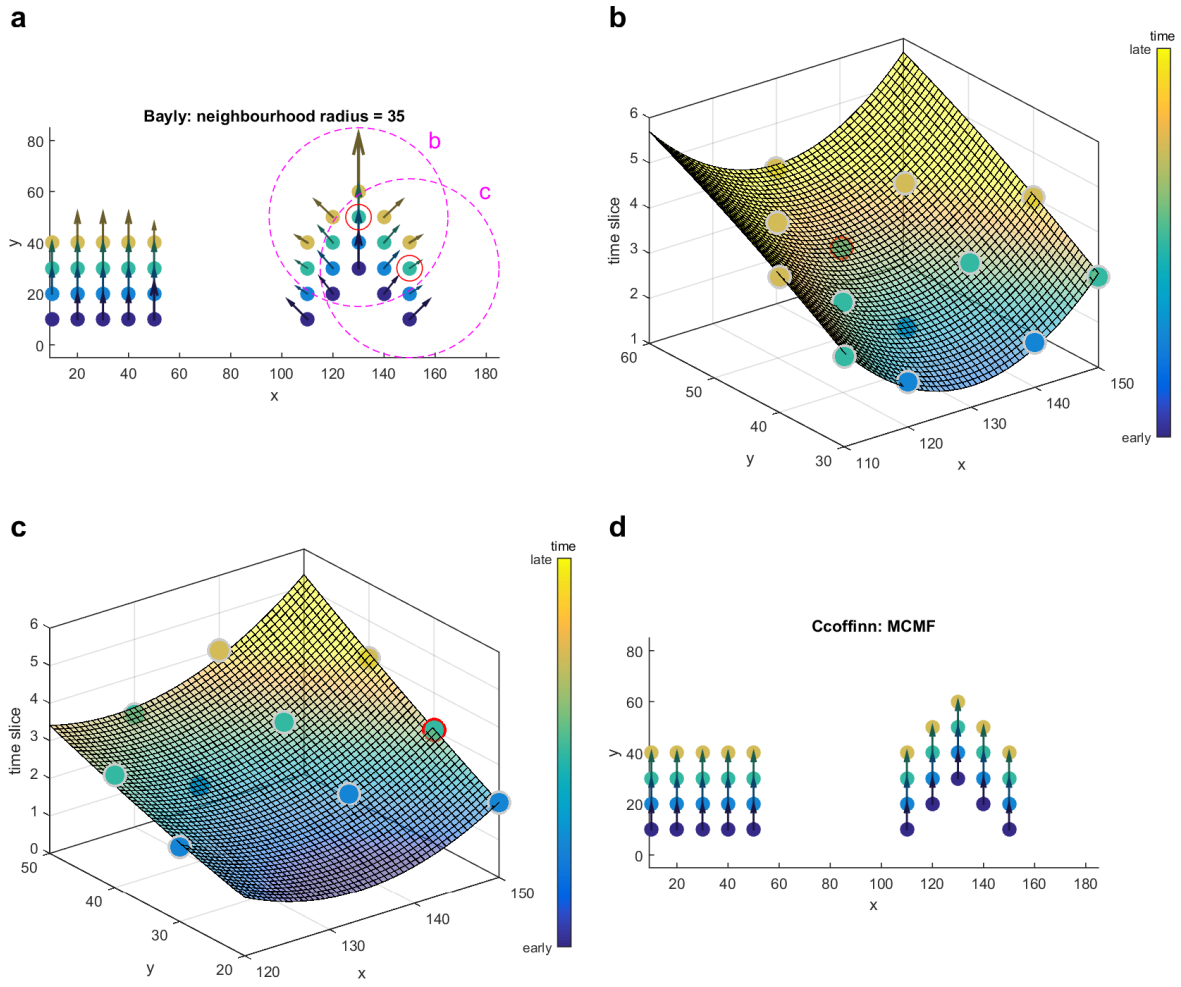


Figure S.10: In a) is shown the activation diagram of two simulated waves of identical conduction velocity: a straight wave to the left, and a arrow-shaped wave to the right; the interpretation of colours and arrows obtained using Bayly's algorithm is identical to the previous figure. In b) and c) are the surfaces fitted to the respective points in a) and their surroundings. In d) is given the tracking information obtained using Ccoffinn's MCMF algorithm.

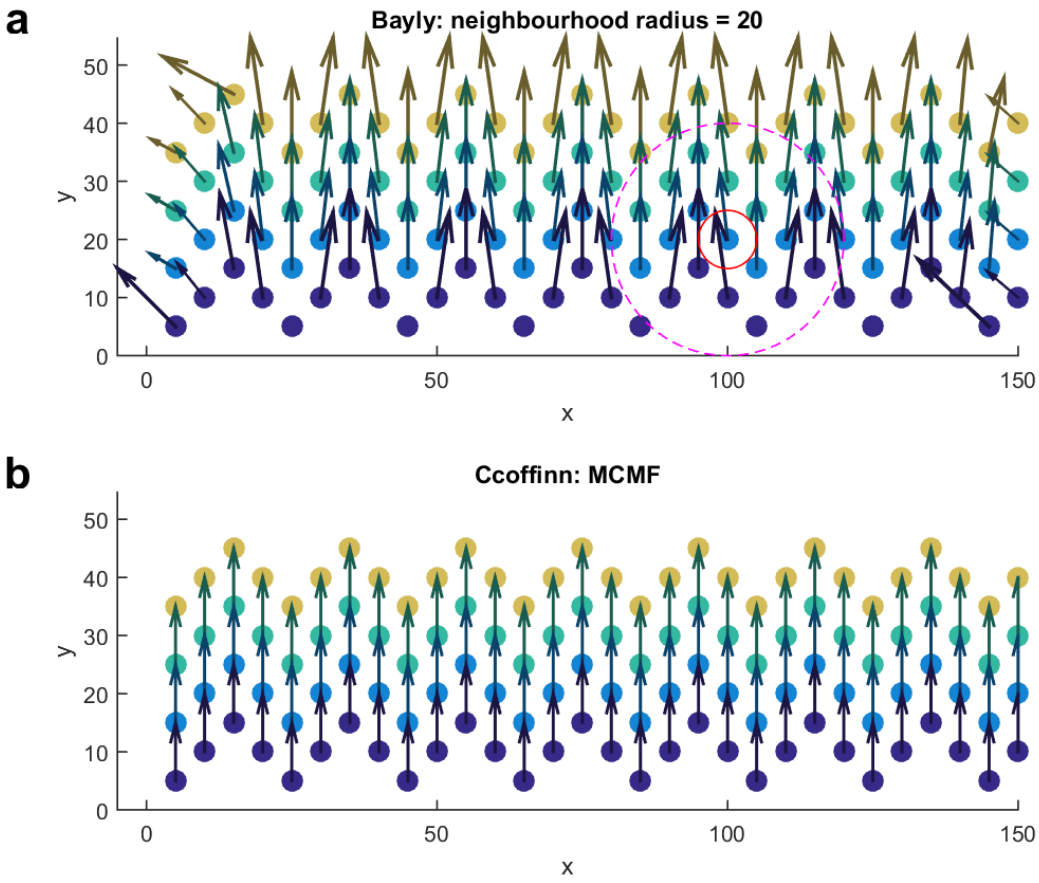


Figure S.11: In a) is shown the activation diagram of a simulated straight wave with jagged wavefront travelling from bottom to top; the interpretation of colours and arrows obtained using Bayly's algorithm is identical to the previous figure. In b) is given the tracking information obtained using Ccoffinn's MCMF algorithm.

S7.5 Heterogeneously spreading target waves and changing wavefront size

A case where the basic version of MCMF algorithm (allowing at most one arrow entering and leaving a blob) may struggle primarily is the case when wavefront size changes considerably (such as in origination of a target wave), possibly under-tracking the propagation. An example of such under-tracking is shown in Figure S.12a. The issue is the most problematic between frames 1 and 2, where the basic MCMF algorithm generates one arrow where there should be four. The CV is estimated correctly in the frame, but the information on direction of propagation is insufficient. In the last two frames it can be clearly seen how the algorithm chooses the shortest arrows possible, underrepresenting the faster-propagating parts of the wave, thus underestimating CV.

In Figure S.12b is shown the performance of the generalised MCMF algorithm with *maxArrows* = 'auto', showing improved tracking of the previously underrepresented parts of the wave. The perpendicular arrow pairs in certain regions is an artifact caused by the use of synthetic data in this example which rarely observed in real data; weighted spatial averaging of the arrows might be performed to obtain more smooth velocity field. Figure S.12c,d shows the output of Baylys method on the same data (differing in neighbourhood size of 35 and 60 respectively). It is noteworthy that Baylys algorithm also does not work perfectly in this case as arrows leaving blobs in frames 1 and 2 are either not present due to poor conditioning or they are too long and the CV is overestimated in some other regions as well (e.g., the arrow from the blob at the location of x=90, y = 100).

Unfortunately, the generalisation of MCMF with *maxArrows*> 1 does not always perform better than the basic version of the algorithm. The first example is a wave leaving field of view in a corner, as shown in Figure S.13, where the generalised algorithm considers the wave leaving the field of view to be genuinely narrowing, while the basic algorithm and Bayly's method maintain its relative straightness. At the same time, it is important realise that the same wave morphology might happen within field of view due to conduction heterogeneity making a wave narrow down; in such a case, the generalised MCMF with *maxArrows*> 1 would provide the best tracking information.

There are two cases where MCMF with *maxArrows*> 1 may generate incorrect tracking information. First, in the case of steep increase in wavefront size, the method becomes more vulnerable to noise causing a non-spiking blob to be randomly detected as spiking in the vicinity of the main wave; in such a case, the larger value *maxArrows* is allowed, the more artefactual arrows will enter or leave these "fake-spiking" blobs, polluting the tracking information. A second problematic case happens when a slow wave is imaged at high sampling frequency: even if the wavefront maintains constant size, only a fairly small proportion of the wavefront will be tracked due to the fact that only the became-wavefront blobs are candidates for incoming arrows. At the same time, the set of blobs allowing outcoming arrows (previous-border blobs) will be comparatively large. As a consequence, multiple arrows

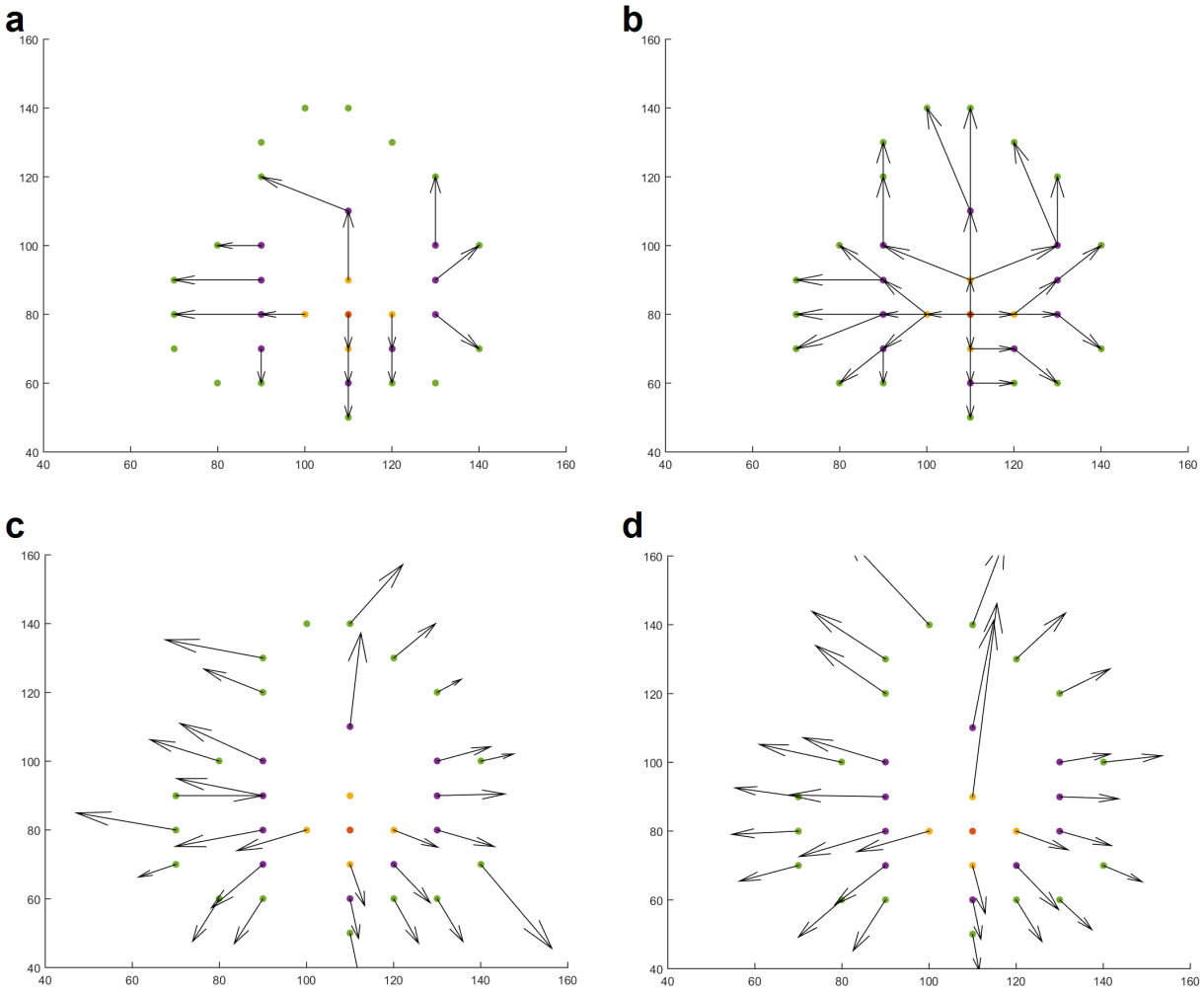


Figure S.12: In a) is shown the tracking of an uneven target wave using the basic MCMF approach. In b) is the same scenario using generalised MCMF tracking with the parameter *maxArrows* set to 'auto'. In c) and d) are shown the outputs of Bayly's algorithm for neighbourhood size 35 and 60 respectively.

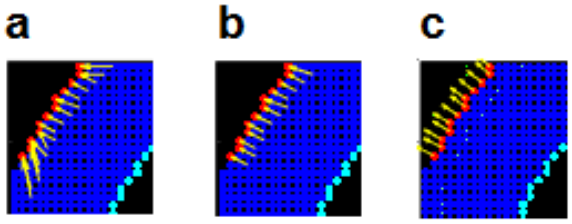


Figure S.13: A segment of a spiral wave leaving the field of view, as tracked by MCMF with *maxArrows* = 'auto' (a), *maxArrows*= 1 (b), and Bayly's method (c).

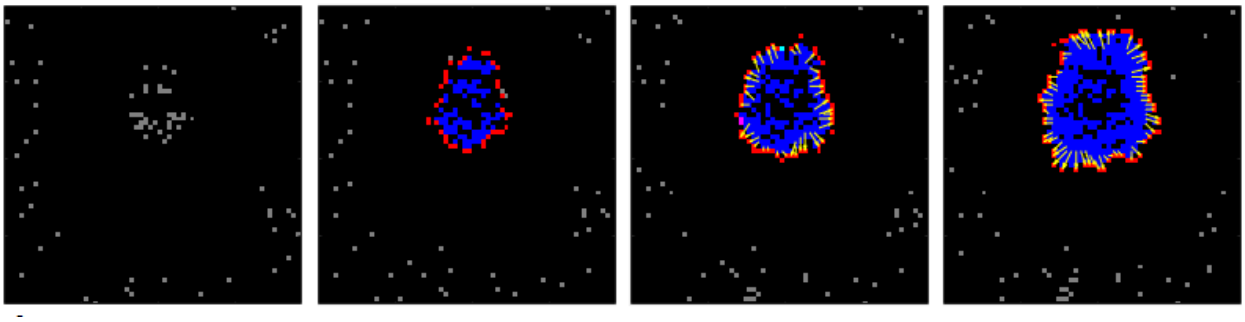


Figure S.14: Four frames of an originating target wave tracked by MCMF with *maxArrows*= 1.

will be pointing to most became-wavefront blobs, most of these being necessarily wrong, overestimating conduction velocity and roughness of propagation.

We note that in real use, an obvious underrepresentation of propagation with MCMF and *maxArrows*= 1 happened only fairly rarely in the data available to us and the cases with the largest potential for steep difference (such as the first two frames capturing the origin of a target wave) are implicitly treated by the fact that Ccoffinn requires waves to have a sufficient size/density to be tracked in the first place. In Figure S.14 is shown a sequence of four frames showing origin of a target wave (sample dataset **dataCa2_target**), as tracked using MCMF with *maxArrows*= 1, showing no major systematic flaw of tracking. The problem of under-tracking is somewhat present in the top left part of the wave between frames 3,4, but the bulk of the wavefront is tracked correctly, even the bottom part which is also heterogeneously quickly conducting.

For the reasons stated above, we believe that too high values of *maxArrows* are expected to bring more problems than benefit and values of 1 or 2 are likely to work best, unless a particular dataset requires otherwise.

References

1. Bayly, P. V., B. H. KenKnight, J. M. Rogers, R. E. Hillsley, R. E. Ideker, and W. M. Smith, 1998. Estimation of conduction velocity vector fields from epicardial mapping data. *IEEE Transactions on Biomedical Engineering* 45:563–571.

2. Konstantinova, P., A. Udvarev, and T. Semerdjiev, 2003. A study of a target tracking algorithm using global nearest neighbor approach. *Proceedings of the 4th International Conference on Computer Systems and Technologies E-Learning - CompSysTech '03* 290–295. `http://portal.acm.org/citation.cfm?doid=973620.973668`.

3. Jung, P., J. Wang, R. Wackerbauer, and K. Showalter, 2000. Coherent structure analysis of spatiotemporal chaos. *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics* 61:2095–8. `http://www.ncbi.nlm.nih.gov/pubmed/11046503`.

4. Miragoli, M., G. Gaudesius, and S. Rohr, 2006. Electrotonic modulation of cardiac impulse conduction by myofibroblasts. *Circulation Research* 98:801–810.

5. Campbell, K., C. J. Calvo, S. Mironov, T. Herron, O. Berenfeld, and J. Jalife, 2012. Spatial gradients in action potential duration created by regional magnetofection of hERG are a substrate for wavebreak and turbulent propagation in cardiomyocyte monolayers. *The Journal of physiology* 590:6363–79. `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3533198{&}tool=pmcentrez{&}rendertype=abstract`.

6. Edwards, J., and L. Blatter, 2014. No TitleCardiac alternans and intracellular calcium cycling. *Clinical and Experimental Pharmacology and Physiology* 41:524–532.

7. Bub, G., A. Shrier, and L. Glass, 2002. Spiral wave generation in heterogeneous excitable media. *Physical review letters* 88:058101.

8. Steinberg, B. E., L. Glass, A. Shrier, and G. Bub, 2006. The role of heterogeneities and intercellular coupling in wave propagation in cardiac tissue. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 364:1299–1311.

9. Rohr, S., 2004. Role of gap junctions in the propagation of the cardiac action potential.

10. Fast, V. G., B. J. Darrow, J. E. Saffitz, and a. G. Kléber, 1996. Anisotropic activation spread in heart cell monolayers assessed by high-resolution optical mapping. Role of tissue discontinuities. *Circulation research* 79:115–127.

11. Spach, M. S., and J. M. Kootsey, 1983. The nature of electrical propagation in cardiac muscle. *The American journal of physiology* 244:H3–H22.

12. Schlemmer, A., S. Berg, T. K. Shajahan, S. Luther, and U. Parlitz, 2015. Quantifying spatiotemporal complexity of cardiac dynamics using ordinal patterns. *In* Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS. volume 2015-November, 4049–4052.

13. Schlemmer, A., S. Berg, T. Shajahan, S. Luther, and U. Parlitz, 2015. Entropy Rate Maps of Complex Excitable Dynamics in Cardiac Monolayers. *Entropy* 17:950–967. `http://www.mdpi.com/1099-4300/17/3/950`.

14. Bub, G., A. Shrier, and L. Glass, 2005. Global organization of dynamics in oscillatory heterogeneous excitable media. *Physical Review Letters* 94.

15. Bub, G., K. Tateno, A. Shrier, and L. Glass, 2003. Spontaneous initiation and termination of complex rhythms in cardiac cell culture. *Journal of cardiovascular electrophysiology* 14:S229–S236.

16. Burton, R. A. B., A. Klimas, C. M. Ambrosi, J. Tomek, A. Corbett, E. Entcheva, and G. Bub, 2015. Optical control of excitation waves in cardiac tissue. *Nature Photonics* 9:813–816.

17. Ambrosi, C., and E. Entcheva, 2014. Optogenetic control of cardiomyocytes via viral delivery. *Methods Mol Biol* 215–228.