Published in final edited form as:
IEEE Trans Image Process. 2012 July ; 21(7): 3150–3156. doi:10.1109/TIP.2012.2188808.

Self-Crossing Detection and Location for Parametric Active Contours

Arie Nakhmani and
Department of Electrical Engineering, Technion, Haifa 32000, Israel. He is now with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02115 USA

Allen Tannenbaum
Department of Electrical and Computer Engineering and Department of Biomedical Engineering, Boston University, Boston, MA 02115 USA, and is also with the Department of Radiology/Wallace Cancer Center, University of Alabama at Birmingham, Birmingham, AL 35294-1150

Arie Nakhmani: anry@bu.edu; Allen Tannenbaum: tannenba@bu.edu

Abstract

Active contours are very popular tools for video tracking and image segmentation. Parameterized contours are used due to their fast evolution and have become the method of choice in the Sobolev context. Unfortunately, these contours are not easily adaptable to topological changes, and they may sometimes develop undesirable loops, resulting in erroneous results. To solve such topological problems, one needs an algorithm for contour self-crossing detection. We propose a simple methodology via simple techniques from differential topology. The detection is accomplished by inspecting the total net change of a given contour's angle, without point sorting and plane sweeping. We discuss the efficient implementation of the algorithm. We also provide algorithms for locating crossings by angle considerations and by plotting the four-connected lines between the discrete contour points. The proposed algorithms can be added to any parametric active-contour model. We show examples of successful tracking in real-world video sequences by Sobolev active contours and the proposed algorithms and provide ideas for further research.

Index Terms

Active contours; image segmentation; self-crossing; snakes; tracking

I. Introduction

Active contours (*snakes*) are widely used in segmentation and tracking. The development of this field began with the seminal paper by Kass *et al.* [1]. Snakes have applications in various fields, from medical imaging to surveillance. The purpose of snakes is to define an object's outline in the tested image, i.e., by minimizing an energy associated with different object properties (e.g., average intensity value), and intrinsic curve properties (e.g., smoothness).

Parametric snakes are given as curves that explicitly provide the (x,y) coordinates of the given points on the given contour (these are called *snaxels*). The advantage of such a parametrization is in the fast evolution of the snake and, in fact, has dominated the tracking literature [2]. The problem that occurs with parametric snakes, particularly when used to

segment noisy images, is the development of false loops [see Fig. 1(c)–(e)]. If the snakes are region based (e.g., in [3] and [4]), then these loops can cause a divergence of the snake since they change the normal direction and thus the *inside* and *outside* of the closed contour. In other cases, the false loops may catch irrelevant features for tracking. This brings us to the importance of the contour self-crossing detection and of the consequent topological changes (loop elimination or snake splitting).

Currently, there are five main approaches to tackle the problem of self-crossing. First, *level sets* are perhaps the most popular approach. The curve is defined by zero level set of the graph of surface, usually starting with the distance function (see [3] and [5] the references therein). With level sets, the topological changes are automatically managed, and the problem of self-crossing does not occur. In general, the snake evolution in the level-set framework is slower than the evolution of parametric snake models. Moreover, automatic topological changes may be less appropriate for tracking, where a single target may be unintentionally split or two distinct targets may have merged.

A second popular class of approaches for detection and location of the crossing points is based on the classical Bentley–Ottmann line segment intersection algorithm in computational geometry. These approaches are reviewed in [6, Ch. 2] and [7]. The general idea of this type of techniques is to sort the snake's points by the x-coordinate in ascending order and to employ the plane sweep algorithm to detect the overlapping x and y projections of the segment pairs. The actual crossing is tested only if both x and y projections overlap. These algorithms are fast and efficient but are not simple to implement, partly due to the need of checking various special cases.

The third approach is grid based [8]–[11]. For easy collision and crossing detection, the motion of snake points can be restricted to the edges of the grid, as in [10], or one can inspect the crossings with a constant rectangular grid [9] or simplicial cell decomposition [8], [11]. Unfortunately, these algorithms are not always able to detect all the crossings because of the discretization of the snaxel locations or the restrictions.

The next approach is perhaps the simplest. The idea is to complete the lines with additional points computed from linear interpolation between the snaxels, i.e., to complete the segments between the endpoints and to make the snake segments connected (continuous) on the given discrete grid [12]. Then, all the points are plotted on a 2-D counter raster (matrix), where the crossing is detected and located if the points are plotted more than once to the same raster location. The obvious problems with this method are explained in [7]. The method cannot provide sub-pixel accuracy in crossing location because of the discretization of snaxel locations. In addition, by the nature of the interpolation algorithm, false positive and false negative detections may frequently occur (see Fig. 4). The proposal to use double-width lines makes the problem of false positives much worse, and the additional computational time for the direct checking of these pseudocrossings virtually eliminates all the advantages of the algorithm.

The last approach is based on an idea of Kass *et al.* [1]. Here, one employs a "repulsion force, " which forces the snaxels not to be too close to one another. This force can be incorporated into the energy minimization framework. Ivins and Porrill [4] propose to compute the repulsion force from the tension, stiffness, and reversed pressure. Unfortunately, introducing the new repulsion force will not always prevent crossings; therefore, an additional self-crossing check may be needed. In addition, introducing the repulsive force affects the snake dynamics and convergence properties, which may not be desirable.

It is important to note that while self-crossing does not occur very frequently, nevertheless, its appearance may ruin the tracking of a target in a given scenario. Consequently, our goal is to effectively detect the existence of self-crossing as quickly as possible, and when it exists, to locate the crossing points, and to apply the appropriate topological change to the snake (loop removing or snake splitting). Since the problem of detection is simpler than the simultaneous detection and location of the crossing (as in the approaches described earlier), this problem can be solved more efficiently with regard to space usage and run time. We propose two algorithms: The first is for an efficient self-crossing detection, and the second is for the computation of crossing locations.

The first algorithm should be run for each snake iteration, without computing the crossing segments and the crossing point. We base our idea on the Hopf turning-number topological theorem [13, p.162], which states that the total net angle change of a simple continuous closed curve (without self-crossings) can be 360° for a clockwise (CW) curve and −360° for a counterclockwise (CCW) curve, as explained in Section II. The same is true for simple polygons.

In Section II-B, we show that an explicit angle computation is unnecessary and may be replaced by a more efficient algorithm that counts the number of quadrants that the curve has passed in turn. In Section II-C, we investigate the rare cases when our detection algorithm may fail, and propose a possible solution.

The second algorithm solves the problem of crossing localization. We propose a particular line completion algorithm for that purpose. In Section II-D, we describe our novel interpolation scheme. The proposed algorithm provides four-connected segments (each snake point has two neighbors, from the left, right, top, or down), which totally eliminates the false negatives without a significant increase in false positives. The crossing points are computed only if a self-crossing is detected (Sections II-C and II-D). The self-crossing can be found by checking the appropriate segments for crossing, which can be accomplished by solving certain simultaneous parametric equations [14] or by testing on which side of the given segment lies the endpoints of the second segment, as explained in the Section II-D. In case of self-crossing, the snake can be split by a simple method described in [14] or [15] (see Section II-E).

We have successfully tested our approach on numerous real-world videos. We have used region-based Sobolev snakes [16] with the Chan–Vese energy [3]. Representative results and the discussion are provided in Section III. Section IV summarizes this and proposes possible further research.

II. Self-Crossing Detection

In this section, we propose an easily implementable algorithm for detecting active-contour self-crossing, which is based on the concept of *turning number*.

Let $\alpha : \mathbb{R} \rightarrow \mathbb{R}^2$ be a regular closed curve. The *turning number* of α, i.e., **Turn**(α), is the total signed curvature of α divided by 360° [13, p.156]. It can be shown that, for closed curves, the number **Turn**(α) is an integer, and it is equivalent to the topological rotation index [13, p.160]. Intuitively, **Turn**(α) is the number of full CCW turns until the curve returns to the (arbitrarily) chosen initial point. If the turns are CW, then **Turn**(α) is negative. Similarly, the total curvature is the net change of unwrapped angle $\Delta\phi$ (see Fig. 1).

For convenience, we define the turning direction by

$$\mathbf{dir}(\alpha)=\begin{cases} 1, & \text{if the curve is CCW} \\ -1, & \text{if the curve is CW} \end{cases} \quad (1)$$

We now state the key result due to Hopf [13, p.162].

Theorem 1—The turning number of a simple closed curve α (without self-crossings) equals to **dir**(α).

As we have mentioned earlier, this theorem can be restated in terms of the total net angle change of 360°·**dir**(α). In particular, this theorem is true for any simple closed polygon, as is typically given with the snake representation. In the following subsection, we use this observation to formulate a straightforward algorithm for the detection of crossings.

A. Crossing Detection Algorithm

Suppose the parametric snake is defined by its point coordinates: $\mathbf{p_i} = (x_i, y_i)$ $i = 1,\ldots, n$, where $\mathbf{p_{n+1}} = \mathbf{p_1}$. The corresponding segment between $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$ is denoted by s_i (see Fig. 2).

The unwrapped angle of segment s_i is denoted by ϕ_i as follows:

$$\tan(\phi_i)=\frac{y_{i+1}-y_i}{x_{i+1}-x_i}. \quad (2)$$

The angle is defined up to 360°; therefore, one should note the meaning of the unwrapped angle: If the angle difference between the consecutive segments is larger than 180°, then multiples of +360° or −360° should be added to ϕ. As a result, the angle ϕ can get any real value and is not bounded by ±180°.

The following algorithm allows one to check if self-crossings exist, without actual computing the crossing points. The algorithm is based on Theorem 1 for polygons, and it should be run for each snake iteration.

Algorithm for Self-crossing detection

Input : snaxels $\mathbf{p}_i$

Compute the angle ϕ by (2)

if $|\phi(n) - \phi(1) - 360° \cdot \mathrm{dir}| > 180°$ then

report that crossing is detected

Remarks

- Only a single comparison of the first and the last segment angles is needed in our algorithm.
- The first segment is chosen arbitrary.
- No sorting procedure is required.

In most practical situations, the proposed algorithm robustly and efficiently detects the existence of self-crossings. The special cases when this algorithm fails [an even number of crossings with double twist, as in Fig. 3, and an equal number of inward and outward loops, as in Fig. 1(e)] are explored in Section II-C, and a solution for these cases is proposed.

The next subsection describes a more efficient version of the self-crossing detection algorithm.

B. Efficient Detection

The next algorithm is based on the observation that any simple closed curve is topologically equivalent to a square. Accordingly, the turning number may be computed by considering how many quadrants the curve has passed in turn, or how many 90° turns the curve performed along the way. If this number is not 4, then the curve is not simple. The increments in the number of quadrants may be computed by inspecting the signs of $\Delta y_i = y_{i+1} - y_i$ and $\Delta x_i = x_{i+1} - x_i$ as described in the following.

Let (+,+)(i) denote sign(Δx_i) = +1 and sign(Δy_i) = +1, (+,−)(i) denote sign(Δx_i) = +1 and sign(Δy_i) = −1, (−,+)(i) denote sign(Δx_i) = −1 and sign(Δy_i) = +1, and (−, −)(i) denote sign(Δx_i) = −1 and sign(Δy_i) = −1.

The increment in the number of quadrants is summarized in the Table I.

The sign of 2 on the second diagonal (for a very sharp angle change) is computed by inspecting the slopes of the consecutive segments s_i and s_{i+1} as follows:

$$\text{Slope}_i = \left| \frac{\Delta y_i}{\Delta x_i + \varepsilon} \right| \quad (3)$$

where ε is a small number that is added to prevent the division by zero.

All the values in Table I (except the second diagonal) may be computed by the following formula:

$$\Delta q = -0.5 \text{sign}(\Delta y_i \Delta x_{i+1}) \left(|\text{sign}(\Delta x_{i+1}) - \text{sign}(\Delta x_i)| + |\text{sign}(\Delta y_{i+1}) - \text{sign}(\Delta y_i)| \right). \quad (4)$$

Note that sign($\Delta y_i \Delta x_{i+1}$) is mathematically equivalent to a binary XOR operation and can be efficiently implemented.

We can now describe our algorithm as follows.

Efficient Algorithm for Self-Crossing Detection

```
Input : snaxels p_i
q_0 ← 0
For i = 1 to n
   Compute Δ_q by (4)
   If |Δ_q| = 2 then
   q_i ← q_{i−1} + sign(Slope_i − Slope_{i+1})Δ_q
   else
   q_i ← q_{i−1} + Δ_q
   end
end
If q_n ≠ 4 · dir then
   report that crossing is detected
```

Remarks

- The index i of the number of quadrants q and the values of q for different indexes will be used in the next subsection; otherwise, only the last value q of is of interest. Thus, only a single integer variable q can be used, and minimal memory space requirements can be satisfied.
- In the worst case (zigzag curve), our algorithm needs n divisions and $3n$ additions (subtractions), but for smooth enough curves, we need only comparisons, and $3n$ additions and subtractions. Our algorithm uses a natural order of snaxels, without sorting. The algorithms for detection and localization described in [6, Ch.2] need at least $O(n \log n)$ operations, that is less time efficient, but they provide the location of crossing as well. The run time of our algorithm is negligible with regard to the snake's evolution time.

C. Special Cases

As we stated earlier, for an even number of self-crossings, if double twisting occurs (see Figs. 1(e) and 3), our algorithm will not find the crossing. The algorithm for detecting self-crossings in these cases is more computationally demanding but can be still accomplished by exploring the snake segment angles.

The test for self-crossing is based on the following important observations.

1. A given segment s_i cannot cross segments s_{i+1} and s_{i-1}.
2. No curve can cross itself without turning more than 180° first (CW or CCW). This is equivalent to passing through (at least) three different quadrants. The necessary condition for the crossing of s_i and s_j is $|q_j - q_i| \geq 3$ (see Section II-B).
3. If the segments s_i and s_j have not reached 180° between them, then any segments between i and j cannot cross.
4. If the segments s_i and s_j cross each other, then they have the overlapping in the x-coordinates ((x_i, x_{i+1}) with (x_j, x_{j+1})), as well as in the y-coordinates ((y_i, y_{i+1}) with (y_j, y_{j+1})).
- 5) Suppose that s_i and s_j are the tested candidates for crossing each other. Then, if we remove these segments and connect $\mathbf{p_i}$ to $\mathbf{p_{j+1}}$ and $\mathbf{p_j}$ to $\mathbf{p_{i+1}}$, we obtain two separated closed curves (see Fig. 3). If there are no self-crossings in the curve and if there are no crossings with the added segments, then the algorithm proposed in Section II-A should return that the crossing is not detected for both curves. If there were an even number of self-crossings in the curve, as in all the special cases, then the algorithm in Section II-A should report that one curve has self-crossings (by removing one crossing, we convert the total number of crossings to odd). All the angles of curves are available from the previous computation; therefore, only the two angles of the added segments should be computed in order to apply the algorithm. Similarly, we can connect $\mathbf{p_{i+1}}$ to $\mathbf{p_j}$ and $\mathbf{p_{j+1}}$ to $\mathbf{p_i}$ to obtain two separate closed curves, but without removing the segments. The result should be the same since, by this snake separation process, we have added one crossing if the segments were crossed before. The second kind of curve splitting is included to verify that there were no crossings with the added segments.

All five tests (or relevant portions of them) can be used in the proposed order to rule out the segments that cannot cross. All the proposed conditions are necessary but not sufficient; thus, the segment pairs that passed all the proposed tests should be directly checked for crossing by solving simultaneous equations [14], or explicitly, for the segments s_i and s_j, if

$$0 \le \frac{\det\begin{pmatrix} x_i - x_j & x_{j+1} - x_j \\ y_i - y_j & y_{j+1} - y_j \end{pmatrix}}{\det\begin{pmatrix} x_{j+1} - x_j & x_{i+1} - x_i \\ y_{j+1} - y_j & y_{i+1} - y_i \end{pmatrix}} \le 1$$
$$0 \le \frac{\det\begin{pmatrix} x_i - x_j & x_{i+1} - x_i \\ y_i - y_j & y_{i+1} - y_i \end{pmatrix}}{\det\begin{pmatrix} x_{j+1} - x_j & x_{i+1} - x_i \\ y_{j+1} - y_j & y_{i+1} - y_i \end{pmatrix}} \le 1 \quad (5)$$

then the segments are crossed.

It is important to note that if one does not halt the algorithm after the first crossing, then all the crossings may be found in this manner. Another possibility for self-crossing localization is to use the algorithm proposed in the following subsection. This algorithm is simple and fast but cannot achieve subpixel accuracy.

D. Computing the Crossing Point

In addition to the known methods for detection and computation of crossing points outlined in Section I and the method described in the previous subsection, we propose another approximate method that allows finding the location of all the crossing points by four-connected linear interpolation of the snake segments. The idea is to connect the sequential but distant points of the given snake with additional points in between, and if the added points will fall on the same grid cell, then a crossing is located. All the points are registered in a 2-D raster accumulator. Algorithms of simple linear interpolation (single- and double-width lines plotted on a raster) are prone to false positives and false negatives, as shown in Fig. 4.

It is clear in Fig. 4 that the reason for the false negative is the two crossing lines that are not four connected (vertically or horizontally connected). Thus, we propose a simple algorithm for four-connected line interpolation between the two given snake points as follows:

Algorithm for four-connected line interpolation

```
Input : rounded to the closest integer segment
   endpoint (x1,y1) and (x2,y2)
if x1 = x2 then draw a vertical line
else
Slope = |(y2 − y1)/(x2 − x1)|
r ← Slope
Repeat until the (x2, y2) points is reached
   make [r] steps in sign(y2 − y1) y direction
    (i.e., move 1 point in signed y direction)
   make 1 step in sign(x2 − x1) x direction
   r ← − [r] + Slope
   end
end
```

Remarks

- The slope has been already defined in (3) and computed via the efficient algorithm for self-crossing detection (Section II-B).
- The $[r]$ denotes the closest to r integer.
- As simple as it seems, the problem of defining a four-connected linear interpolation is not a well-defined problem because the solution is not necessarily unique. Our algorithm provides only one possible solution. In some cases, it adds false positives, but it detects all the crossings without false negatives. The false positives are easily detected and removed [12].

E. Contour Splitting and Reordering

When a self-crossing is detected and located, the snake will generally go through a topological change (splitting into two snakes or removing the loop). Different methods for topological changes have been proposed earlier [8]–[12], [15]. In this paper, we use the simplest splitting proposed in [14]. If the segments $s_i : \mathbf{p_i} \rightarrow \mathbf{p_{i+1}}$ and $s_j : \mathbf{p_j} \rightarrow \mathbf{p_{j+1}}$ are crossed, then we remove these segments and add the segments between $\mathbf{p_i} \rightarrow \mathbf{p_{j+1}}$ and $\mathbf{p_j} \rightarrow \mathbf{p_{i+1}}$ (see Fig. 3). If some loops are removed, e.g., for single-target tracking, the shortest loops are eliminated, and the removed points are distributed in the most sparse regions of the remaining loop.

Additional cues about the loops that should be removed can also be extracted from the angle information. If we suppose that the direction $\mathbf{dir}(\alpha)$ does not change through tracking or segmentation, we can remove the loops with the wrong direction.

III. Results and Discussion

In this section, we show how the proposed algorithms improved the tracking with parametric Sobolev snakes [16] with the Chan–Vese energy functional [3]. The snake was used to track a single target in video sequences with a constant-velocity dynamic model (current frame initialization contour is a translated version of the previous frame result). The algorithms have been implemented in Matlab.

We have chosen two representative videos. The first is the well-known “Walking in Finland” sequence from the University of Oulu. This sequence consists of 22 frames and is frequently used to test visual trackers. We run the algorithm of the Sobolev snake with the same parameters ($\lambda = 1$, step size $dt = 1$, six iterations per frame), with and without the loop detection and elimination algorithms. The target is manually selected in the first frame. For the following frames, the predicted in the previous frame result is used to initialize the snake. The results are shown in Fig. 5.

Without the proposed algorithm, in frame 3, the snake develops a loop, which prevents further convergence of the snake around the target. When the developing loop was detected via our self-crossing detection algorithm, it was removed, and the tracking was successful.

The second sequence, i.e., “The Hand, ” is filmed by infrared (IR) camera, and has 1050 frames. Again, only with the self-crossing detection algorithm, the Sobolev snake managed to track the hand in the entire sequence. The results of tracking are shown in Fig. 6.

The additional time to detect and remove the loops for all the tested sequences was less than 2%, compared with the running time of the Sobolev snake algorithm.

Some more experiments illustrating our methodology may be found on the website: http://www.youtube.com/playlist?list=PL0E921BB9455C23B3. The failure scenarios for these videos (without our algorithm) are shown in the top row in Fig. 7, and the success scenarios (with the self-crossing detection and elimination) are show at the bottom row of this figure.

IV. Conclusion and Further Research

We have proposed a fast and simple approach to snake self-crossing detection and localization. The algorithms can be adapted to any type of parametric snakes, which is defined by snaxel coordinates, and do not restrict the snake's motion and evolution in any manner.

The proposed crossing-point computation algorithm uses rounded integer values for snaxels and is appropriate in cases where no subpixel accuracy is needed. Its implementation is fast and efficient, with a low number of false positives (that can be easily rejected).

We have shown that the proposed algorithm can be used for tracking real-world scenarios and helps to prevent the divergence of the active contour from the target of interest. We believe that future research of angle dependencies for an even number of self-crossings will simplify the algorithm for the special cases discussed in this paper. The idea of Perrin *et al.* [17] of using the assumption that, initially, the contour is simple and that each moved snaxel should be tested only if the simplicity condition is violated should be applicable to our framework by testing the changes in the angles of each moved point. Additional improvements may be done by considering the work of Whitney and Graustein [13, p.164] and the connection of the right-hand and left-hand crossings with the total angular change.

Acknowledgments

This work was supported in part by grants from AFOSR and ARO. This project was also supported by the National Center for Research Resources under Grant P41-RR-013218 and the National Institute of Biomedical Imaging and Bioengineering under Grant P41-EB-015902 of the National Institutes of Health. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research under Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Gang Hua.

References

1. Kass M, Witkin A, Terzopoulos D. Snakes: Active contour models. Int J Comput Vis. Jan; 1988 1(4):321–331.
2. Blake, A.; Isard, M. Active Contours: The Application of Techniques From Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion. New York: Springer-Verlag; 1998.
3. Chan T, Vese L. Active contours without edges. IEEE Trans Image Process. Feb; 2001 10(2):266–277. [PubMed: 18249617]
4. Ivins J, Porrill J. Active region models for segmenting textures and colours. Image Vis Comput. 1995; 13(5):431–438.
5. Osher, S.; Fedkiw, R. Level Set Methods and Dynamic Implicit Surfaces. New York: Springer-Verlag; 2003.
6. De Berg, M.; Cheong, O.; Van Kreveld, M.; Overmars, M. Computational Geometry: Algorithms and Applications. 3. New York: Springer-Verlag; 2008.
7. Smith C, Schaub H. Efficient polygonal intersection determination with applications to robotics and vision. Proc IEEE/RSJ Int Conf Intell Robots Syst. 2005:3890–3895.
8. McInerney T, Terzopoulos D. T-snakes: Topology adaptive snakes. Med Image Anal. Jun; 2000 4(2):73–91. [PubMed: 10972323]

9. Delingette H, Montagnat J. New algorithms for controlling active contours shape and topology. Proc ECCV. 2000:381–395.
10. Bischoff S, Kobbelt L. Parameterization-free active contour models with topology control. Vis Comput. Jun; 2004 20(4):217–228.
11. Oliveira A, Ribeiro S, Farias R, Esperanca C. Loop snakes: Snakes with enhanced topology control. Proc 17th Brazilian Symp Comput Graph Image Process. 2004:364–371.
12. Ji L, Yan H. Loop-free snakes for highly irregular object shapes. Pattern Recognit Lett. Mar; 2002 23(5):579–591.
13. Gray, A.; Abbena, E.; Salamon, S. Modern Differential Geometry of Curves and Surfaces With Mathematica. London, U.K: Chapman & Hall; 2006.
14. Araki S, Yokoya N, Iwasa H, Takemura H. Splitting of active contour models based on crossing detection for extraction of multiple objects. Syst Comput Jpn. Oct; 1997 28(11):34–42.
15. Stoeter S, Papanikolopoulos N. Closed dynamic contour models that split and merge. Proc IEEE Int Conf Robot Autom. 2004:3883–3888.
16. Sundaramoorthi G, Yezzi A, Mennucci A. Coarse-to-fine segmentation and tracking with sobolev active contours. IEEE Trans Pattern Anal Mach Intell. May; 2008 30(5):851–864. [PubMed: 18369254]
17. Perrin D, Ladd A, Kavraki L, Howe R, Cannon J. Fast intersection checking for parametric deformable models. Proc SPIE. 2005; 5747:1468.

Biographies

Arie Nakhmani received the Ph.D. degree in electrical engineering from Technion–Israel Institute of Technology, Haifa, Israel, in 2011.

He is currently a Postdoctoral Researcher and a Lecturer with the Department of Electrical and Computer Engineering, Boston University, Boston, MA. His research interests include visual tracking, image segmentation, estimation, and robust control.

Allen Tannenbaum He is currently a faculty member with the Department of Electrical and Computer Engineering and Department of Biomedical Engineering, Boston University, Boston, MA, and the Department of Radiology/Wallace Cancer Center, University of Alabama at Birmingham, Birmingham. His research interests include medical imaging, control, image processing, and computer vision.

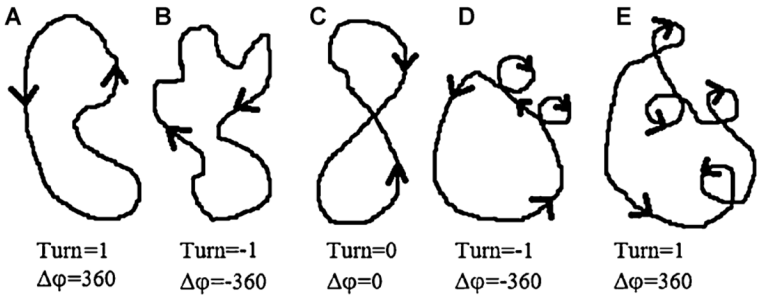

Fig. 1.
Closed curves with the turning number and the total net change of the angle.

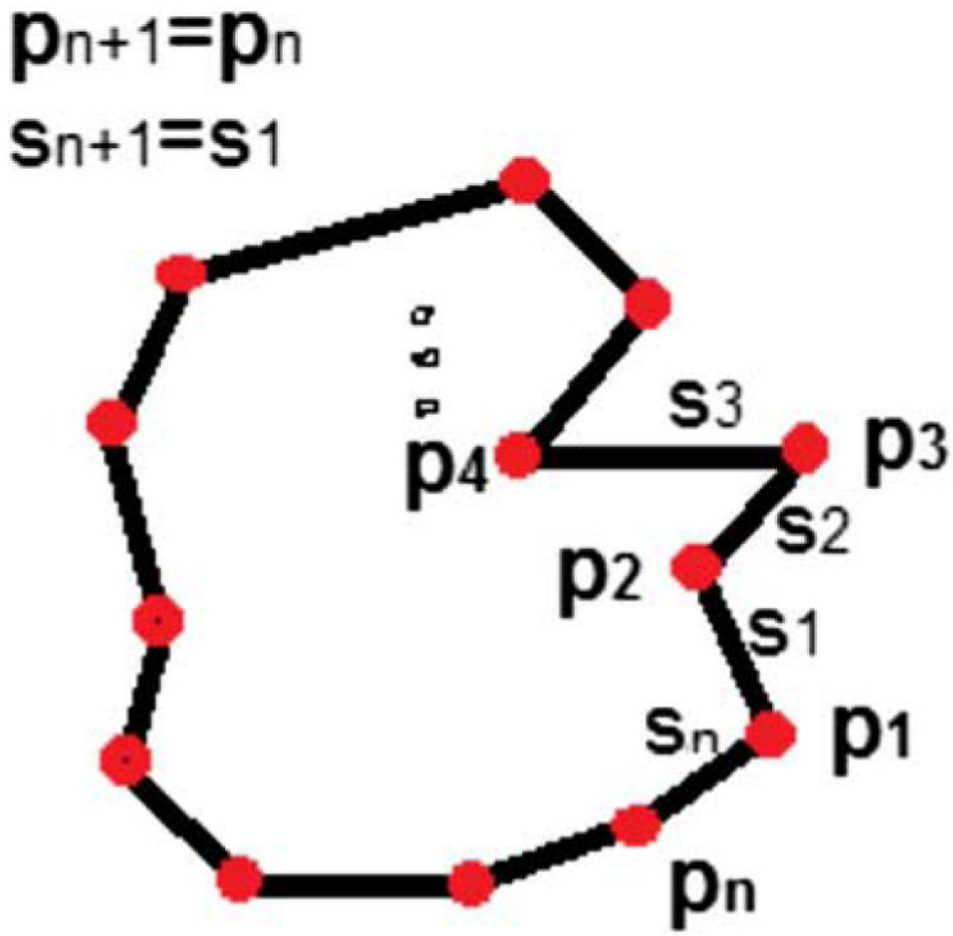

Fig. 2.
Snake coordinates and segments definition.

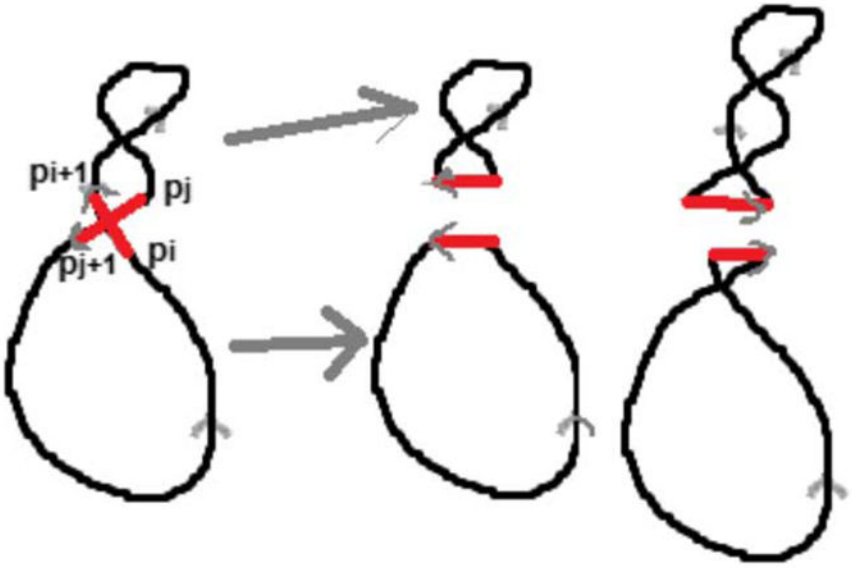

Fig. 3.
Special case: The curve with double twisting.

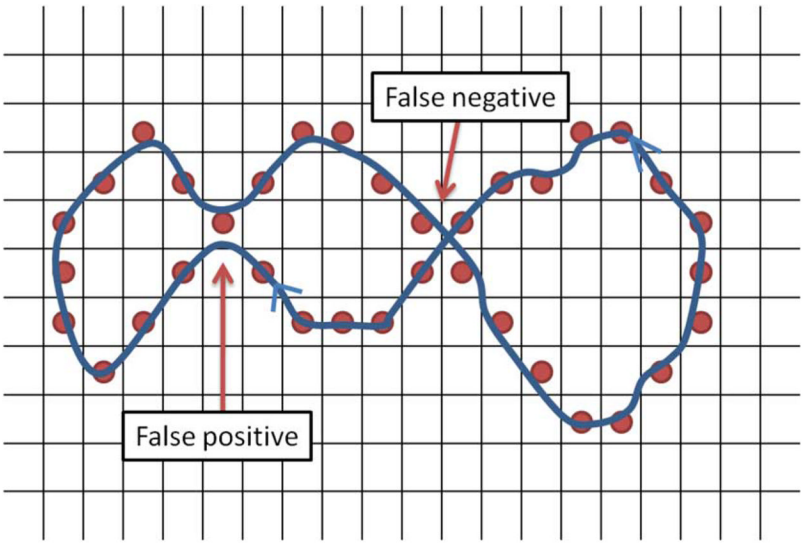

Fig. 4.
Possibility of false positive (detected crossing where it does not exist) and false negative (actual crossing is not detected) with the grid-based algorithms.

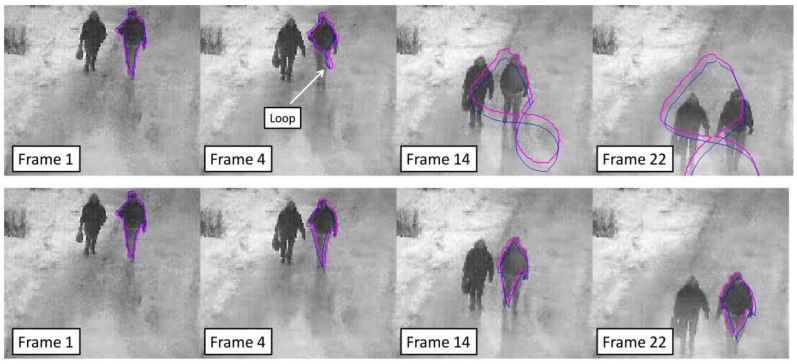

Fig. 5.
"Walking in Finland" sequence (top row) without and (bottom row) with the algorithm for self-crossing detection and loop elimination. The (blue) next frame prediction contour and the (magenta) current final contour are shown.

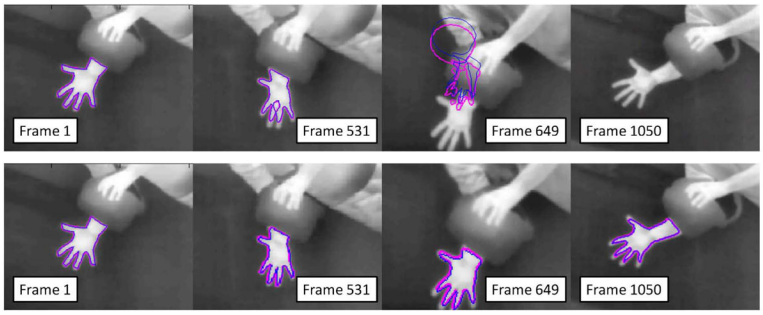

Fig. 6.
"The Hand" sequence (top row) without and (bottom row) with the algorithm for self-crossing detection and loop elimination. The (blue) next frame prediction contour and the (magenta) current final contour are shown.

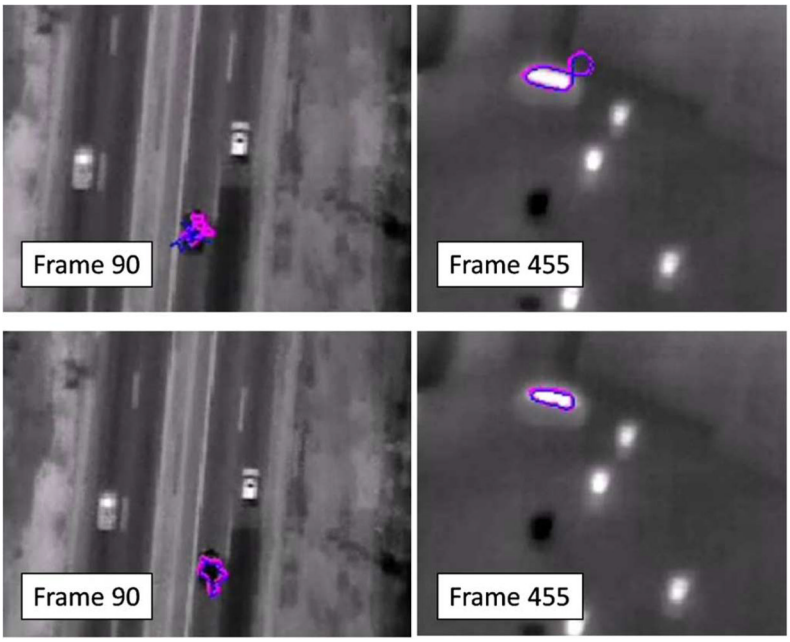

Fig. 7.
"The Car" and "The Robot" sequences. (top row) Frames after the loop's creation. (bottom row) Loops are detected and eliminated by the proposed algorithm.

TABLE I

Increment in the Number of Quadrants

	(+,+)(*i* + 1)	**(+,−)(*i* + 1)**	**(−, +)(*i* + 1)**	**(−, −)(*i* + 1)**
(+,+)(*i*)	0	−1	1	2, if $Slope_i > Slope_{i+1}$ −2, otherwise
(+,−)(*i*)	1	0	2, if $Slope_i < Slope_{i+1}$ −2, otherwise	−1
(−,+)(*i*)	−1	2, if $Slope_i < Slope_{i+1}$ −2, otherwise	0	1
(−, −)(*i*)	2, if $Slope_i > Slope_{i+1}$ −2, otherwise	1	−1	0