Detecting Curves with Unknown Endpoints and Arbitrary Topology Using Minimal Paths

Vivek Kaul, Anthony Yezzi, *Senior Member, IEEE* and Yichang (James) Tsai

Abstract—Existing state of the art minimal path techniques work well to extract simple open curves in images when both endpoints of the curve are given as user input or when one input is given and the total length of the curve is known in advance. Curves which branch require even further prior input from the user, namely each branch endpoint. In this work, we present a novel minimal path based algorithm which works on much more general curve topologies with far fewer demands on the user for initial input compared to prior minimal path based algorithms. The two key novelties and benefits of this new approach are that (1) it may be used to detect both open and closed curves, including more complex topologies containing both multiple branch points and multiple closed cycles without requiring a priori knowledge about which of these types is to be extracted and (2) it requires only a single input point which, in contrast to existing methods, is no longer constrained to be an end point of the desired curve but may in fact by ANY point along the desired curve (even an internal point). We perform quantitative evaluation of the algorithm on 48 images (44 pavement crack images, 1 catheter tube image, and 3 retinal images) against human supplied ground truth. The results demonstrate that the algorithm is indeed able to extract curve-like objects accurately from images with far less prior knowledge and less user interaction compared to existing state of the art minimal path based image processing algorithms. In future, the algorithm can be applied to other 2D curve like objects and it can be extended to detect 3D curves.

Index Terms—minimal paths, keypoints, multiple branches, closed cycles, arbitrary topology,curve detection, cracks.

1 Introduction

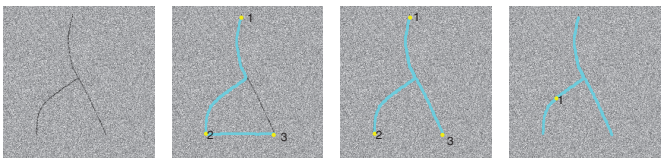
Minimal path techniques are connected with variational energy minimization principles that were first introduced into computer vision by Terzopoulos et al. [14]. Terzopoulos et al. [14] proposed an active contour model(snakes) that evolves an initially drawn curve in an image to minimize an energy functional consisting of two terms: an external energy which is an image-based term to guide the evolving curve towards desired features (*e.g.* edges) and an internal energy term that keeps the curve smooth as it evolves. Later, a more geometric version of this model was introduced in [5], [16] that combined the internal and external energies into a single term which no longer depended upon the abitrary parameterization of the evolving curve. However, the results still strongly depended on the local placement of the initial contour since the contour evolution was merely a gradient descent flow which would stop at the first local minimum encountered. Cohen and Kimmel [7] introduced a global minimal cost path approach to tackle this problem. Their approach allowed the user to simply specify two endpoints of a desired curve, rather than an entire initial curve, and an Eikonal equation is then solved to directly determine the curve that minimizes the same contour dependent energy (as [5], [16]) between these two endpoints. Researchers in different areas like geometric optics, computer vision, robotics and wire routing have previously solved related minimum-cost path problems using graph search and dynamic programming principles. However, the graph based search method for a digitized image with a Cartesian sampling pattern finds only minimal paths with segments that are vertical, horizontal or diagonal. Thus, graph search methods like Dijkstra's algorithm often suffer from metrication errors as discussed in [7]. The minimal path method based on the Eikonal equation on the other hand, given its connection with active contours, formulates the problem in the continuous domain and is less susceptible to metrication errors.

The original minimal path technique can be extended to 3D tree-structured object extraction [10] but not for general 3D surface extraction. A topology-based saddle search routine is needed to extend the minimal path technique for closed curve extraction. Ardon *et. al.* [2] proposed a more general scheme for 3D surface extraction between user supplied end-curves. Minimal path techniques have also been successfully used for contour completion [6], tubular surface extraction [10], [13], skeletonization [8], [12], and motion tracking [4].

Almost all prior approaches require precise knowledge of the desired curve's endpoints. A notable exception are the methods in [8] and [12] to extract skeletons of solid objects. However, the focus of this paper is on the extraction of thin contour-like features (particularly pavement cracks) in noisy backgrounds which present a challenge to skeletonization algorithms. In [3], a variant of the minimal path approach called Minimal Path Method

• *V.Kaul and A. Yezzi are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: vkaul1@yahoo.com, ayezzi@ece.gatech.edu.*
• *Y. Tsai is with the Department of Civil and Environmental Engineering, Georgia Institute of Technology, Savannah, GA, 31407. E-mail: james.tsai@ce.gatech.edu.*

Digital Object Indentifier 10.1109/TPAMI.2011.267



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

Fig. 1: (a) Desired Curve (b) Results from traditional minimal path (c) Results from MPWKD (d) Results from proposed algorithm.

With KeyPoint Detection (MPWKD) was devised to find curves under less restrictive prior information by providing just one endpoint on the desired curve that will lead to detection of representative keypoints along the curve via front propagation. For closed curves, a stopping criterion was derived that requires no further prior information from the user. For open curves, however, the desired length of the curve is required a-priori in order to terminate the key point propagation process.

The main goal of the current work is to be able to detect the same types of contour-like image structures currently extracted by state of the art minimal path techniques, but to be able to do so with far less prior knowledge about both the topology as well as the endpoint locations of the desired curve. Figure 1 contains a branched curve to illustrate this goal. The traditional minimal path approach [7] requires prior knowledge of all three endpoints (marked 1,2 and 3). First, the minimal path between point 1 and point 2 is found. Subsequently, the minimal path algorithm is run again between points 2 and 3. The results are not accurate because the minimal path is not able to travel along the elongated desired curve between 2 and 3 as shown in Figure 1b. The more recent key-point detection algorithm in [3] solves this problem as shown in Figure 1c, but it still requires prior knowledge of the three endpoints. In contrast, the proposed algorithm uses only one arbitrary initial point 1 (does not even have to be an endpoint) to find the complete desired curve shown in Figure 1d. Moreover, while other state of the art methods require prior knowledge of the topology in addition to the endpoints for detecting complex curves with closed cycles and multiple branches, the proposed algorithm detects the same complex curves still with only the prior knowledge of just one arbitrary point.

In Section 2, the theory of minimal path techniques is introduced. Section 3 describes a novel algorithm for detecting open curves with just one arbitrary input point (not necessarily an endpoint) even for curves with multiple branches. Section 4 further extends the algorithm to curves of more general topology that may contain closed cycles as well as open segments. In Section 5, we present two modifications to increase the robustness to noise and sharp corners at branch points. Section 6 presents experimental results of the algorithm, and conclusions and future directions are discussed in Section 7.

2 Theory of Minimal Path Techniques

In the minimal path technique, an energy E of the form

$$E(\gamma) = \int_{\gamma} \Phi(\gamma(s))\mathrm{d}s, \tag{1}$$

is computed along the curve γ within the image domain Ω. A potential $\Phi : \Omega \rightarrow \mathbb{R}^+$ is built from a given 2D or 3D image $I : \Omega \rightarrow \mathbb{R}^+$, where $\Phi > 0$, that takes lower values near the desired features of the image I. The choice of the potential Φ depends on the application. For example, the potential function Φ for cracks can be taken to be a function of intensity value at each pixel because cracks are darker than the background. In some other applications, edge based potential functions can be used. The goal is to extract a curve that minimizes the energy functional $E : \mathcal{A}_{p_1,p_2} \rightarrow \mathbb{R}^+$ between two user defined points p_1 and p_2

$$E(\gamma) = \min_{\gamma \in \mathcal{A}_{p_1,p_2}} \int_{\gamma} \Phi(\gamma(s))\mathrm{d}s, \tag{2}$$

where $\mathcal{A}_{p_1,p_2}$ is the set of all paths connecting p_1 to p_2 and s is the arc length parameter. The curve connecting p_1 to p_2 that globally minimizes the energy $E(\gamma)$ is called the *minimal path* between p_1 and p_2 or the *geodesic*. The geodesic curve is denoted by C_{p_1,p_2}. The solution of this minimization problem is obtained through the computation of the minimal action map $U_1 : \Omega \rightarrow \mathbb{R}^+$ associated with p_1. The minimal action map is defined as the minimal energy integrated along a path between p_1 and any point x of the domain Ω:

$$\forall x \in \Omega, U_1(x) = \min_{\gamma \in \mathcal{A}_{p_1,x}} \int_{\gamma} \Phi(\gamma(s))\mathrm{d}s. \tag{3}$$

In this document, the minimal action map will also be called the *geodesic distance map*. All equations here are derived for isotropic grids. The values of U_1 are the arrival times of a front propagating from the source p_1 with velocity $(1/\Phi)$ and $U_1(p_1) = 0$. U_1 satisfies the Eikonal equation:

$$||\nabla U_1(x)|| = \Phi(x) \quad \text{for} \quad x \in \Omega. \tag{4}$$

The minimal action map calculation can be generalized to the case of multiple sources. The minimal action map or the geodesic distance map associated with the potential $\Phi : \Omega \rightarrow \mathbb{R}^+$ with a set of n sources $S = \{p_1, \ldots, p_n\}$ is the function $U : \Omega \rightarrow \mathbb{R}^+$, where

$$\forall x \in \Omega, U(x) = \min_{1 \leq j \leq n} \{U_j(x)\}, \tag{5}$$

and

$$U_j(x) = \min_{\gamma \in \mathcal{A}_{p_j,x}} \int_{\gamma} \Phi(\gamma(s))\mathrm{d}s. \tag{6}$$

$U_j(x)$ is given by (3) where p_1 is replaced by p_j. For all $p_j \in S$, $U(p_j) = 0$ and U satisfies the Eikonal equation given by (4). Another important quantity important for the current work is the *Euclidean distance map*. It is the function $L : \Omega \rightarrow \mathbb{R}^+$ that assigns the Euclidean length

of the minimal geodesic between x and the source S to every point x of the domain Ω.

$$\forall x \in \Omega, L(x) = \int_{C_{p_j,x}} \mathrm{d}s, \tag{7}$$

where

$$C_{p_j,x} = \min_{1 \le i \le n} C_{p_i,x} \qquad \text{and} \qquad j = \operatorname*{argmin}_{1 \le i \le n} U_i$$

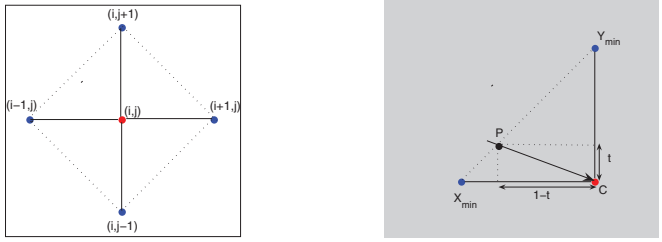
such that p_j is the closest source point to x. $C_{p_i,x}$ is calculated using the potential Φ. In general if $\Phi \neq 1$ for all $x \in \Omega$, then the geodesic distance map U is always different from the Euclidean distance map L.

Centered finite difference schemes for solving the Eikonal equation are unstable. There are three algorithms described in [7] to compute the map U, and they are all consistent with the continuous energy formulation implemented in a rectangular grid. These three algorithms utilize level set methods, shape from shading methods, and Fast Marching methods. Fast Marching methods were favored because of their lower complexity compared to the other methods. Sethian [1], [17] proposed the Fast Marching method to solve the Eikonal equation that relies on a one-sided derivative that looks in the up-wind direction of the moving front. An alternate Fast Marching approach to solve Eikonal equation was provided by Tsitsiklis [19] in the context of isotropic optimal trajectory problem in control theory. The fundamental difference between the control theory and the front expansion formulation is that the former interprets the starting point p_j as the exit point and tries to find the fastest way to reach p_j from x while the latter one looks for the time it will take to advance the front originating from p_j to pass point x. Both Fast Marching method methods compute the values of U in increasing order consistent with Huygens principle of wavefront propagation. These wavefronts propagate from the source set S with a velocity $(1/\Phi)$. The structure of the algorithm is almost identical to Dijkstra's algorithm for computing shortest paths on graphs. In the course of the algorithm, each grid point is tagged as either Solved (a point for which U has been computed and frozen), Active (a point for which U has been estimated but not frozen) or Unvisited (a point for which U is unknown). The set of Active points form an interface between the set of grid points for which U has been frozen (the Solved points) and the set of other grid points (the Unsolved points). This interface may be regarded as a set of fronts expanding from each source until every grid point has been reached. Given the source points, the Fast Marching algorithm helps to calculate geodesic distance map U and the Euclidean distance map L for every grid point. The Fast Marching algorithm is computationally efficient because a priority queue can be used to quickly find the Solved points at each step. The Solved points are points with the smallest U (geodesic distance map) value among the current Active points . If Active points are ordered in a minimum heap data structure, the computational complexity of the Fast Marching method is $O(NlogN)$, where N is the total number of grid points. Benmansour and Cohen [3] compute the geodesic distance map U and the Euclidean distance map L, according to the discretization scheme for Fast Marching method given in [17] . A number of different modifications have been suggested to improve the accuracy of the Fast Marching method [9], [11]. For example, an 8-neighbor scheme was proposed by Danielsson and Lin in [9]. These modifications adopt the control theoretic formulation of Tsitsiklis [19]. The accurate computation of L is essential for the accuracy of our proposed algorithm to detect curves. Therefore, we compute the maps U and L following the Tsitsiklis's control theoretic formulation. The control theoretic framework also provides us options to use modifications suggested in [9], [11] to improve the accuracy of both U and L. In the next section, the calculation of the maps U and L for 4-neighbors is given.

Algorithm *FastMarchComputation*
Input: Image Im, potential function Φ and initial source point set S.
Output: Geodesic distance map U and Euclidean distance map L.
1. **for** all source points $p_1, \ldots, p_n \in S$
2. **do** Set $U(p_j) = 0, L(p_j) = 0$ and Status(p_j) = ACTIVE.
3. **for** all other points $x \in \Omega$
4. **do** Set $U(x) = \infty, L(p_j) = \infty$ and Status(x) = UNVISITED.
5. **while** Number of ACTIVE Points > 0
6. **do** Take out point w_{min} with minimum U.
7. Set Status(p_j) = SOLVED.
8. Update neighbors (STATUS $\neq$ SOLVED) of w_{min} with procedure *UpdateNeighbors*.
9. **repeat**

2.1 Computation of maps U and L

In the Fast Marching algorithm, we first start with the L and U values at the source points in S to be zero and the status of the source points is set to Active. At all other points, U and L values are set to ∞ (a very high value) and their status is set to Unvisited. All the Active points are put in a minimum heap data structure according to the value of U. For each Fast Marching iteration, the grid point with the lowest U value, which is called w_{min}, in the minimum heap structure is removed from the heap and its status is changed to Solved. Next, the L and U values at the neighbors of the point are updated. This corresponds to 4-neighbors in the 2D case and 6-neighbors in 3D case. The algorithm stops once all points are labeled Solved. The Fast Marching algorithm is listed in *FastMarchComputation*. The *UpdateNeighbor* method is used to update the neighbor values at each iteration. Only neighbors that are Unvisited or Active need to be updated. The *UpdateNeighbor* method for the 2D case will be given in this section. It can easily be extended to the 3D case. We show the *UpdateNeighbor* computation for one arbitrary neighbor of the point w_{min}, which we call c. This procedure is carried out similarly for the other three neighbors of w_{min}. Consider point c located at (i, j) and the four triangles formed by the neighbors of c as illustrated in Figure 2a. The Fast Marching boundary can propagate from any of the four directions given by



(a) (b)

Fig. 2: (a) neighbors of current point, (b) triangle of interest

the triangles. However, out of the four virtual triangles formed by the four neighbors of w_{min}, only one will give the minimum value at (i, j). The triangle that should be used for computation of $U(i, j)$ is the one formed by x_{min} and y_{min}. x_{min} is the minimum x-neighbor that corresponds to the x-neighbor ($(i+1, j)$ or $(i-1, j)$)) that takes a lower U value. Similarly, y_{min} is the minimum y-neighbor that corresponds to the y-neighbor ($(i, j-1)$ or $(i, j+1)$) that takes a lower U value. The triangle identified by this method is $x_{min}y_{min}c$ where c is the current point (i, j). This triangle is shown in Figure 2b. The steps in the *UpdateNeighbor* method are as follows.

1) If x_{min} is Unvisited and y_{min} is Active or Solved,

$$U(c) = U(y_{min}) + \Phi(c). \tag{8}$$

$$L(c) = L(y_{min}) + 1. \tag{9}$$

2) If y_{min} is Unvisited and x_{min} is Active or Solved,

$$U(c) = U(x_{min}) + \Phi(c). \tag{10}$$

$$L(c) = L(x_{min}) + 1. \tag{11}$$

3) If x_{min} and y_{min} are not Unvisited, we compute point P, which is given by

$$P = (1-t)x_{min} + ty_{min} \tag{12}$$

where t is the parameter for linear interpolation that lies between 0 and 1. $U(c)$ is given by the solution of the following minimization problem in t:

$$U(c) = \min_{0 \le t \le 1} (1-t)U(x_{min}) + tU(y_{min}) + \sqrt{t^2 + (1-t)^2}\Phi(c) \tag{13}$$

The parameter t that minimizes the expression in (13) is called t_{min}. The minimum t_{min} occurs at the boundary points where t is 0 or 1, or it occurs at the local extrema t^*, which is obtained by differentiating the expression in (13).

$$t^* = 1/2 + \big(U(x_{min}) - U(y_{min})\big)/\sqrt{2D_1} \tag{14}$$

where

$$D_1 = 2\Phi^2(c) - (U(x_{min}) - U(y_{min}))^2. \tag{15}$$

The two scenarios that arise are described below.

Case 1 : If the discriminant $D_1 > 0$ and $0 \le t^* \le 1$, the parameter t_{min} is computed first. To identify t_{min}, the values $U_{t^*}(c)$, $U_0(c)$ and $U_1(c)$ are computed, which correspond to the values of the expression in (13) at $t = t^*$, $t = 0$ and $t = 1$ respectively. $U(c)$ is the minimum of the values $U_{t^*}(c)$, $U_0(c)$ and $U_1(c)$ and t_{min} is the parameter t that gives the value of $U(c)$. The point that corresponds to the parameter value t_{min} in (12) is called P_{min}. P_{min} is the point from which the Fast Marching front travels to the current point c. We can use the value t_{min} at the point P_{min} to compute the Euclidean distance L(c) value by replacing $\Phi(c)$ with 1 in (13).

$$L(c) = (1 - t_{min})L(x_{min}) + t_{min}L(y_{min}) + \sqrt{t_{min}^2 + (1 - t_{min})^2}. \tag{16}$$

Case 2: If $D_1 \le 0$ or $t^* \notin [0, 1]$, then we use the minimum of $U(x_{min})$ and $U(y_{min})$ to compute $U(c)$.

$$U(c) = \begin{cases} U(x_{min}) + \Phi(c) & \text{if } U(y_{min}) > U(x_{min}), \\ U(y_{min}) + \Phi(c) & \text{otherwise.} \end{cases} \tag{17}$$

$L(c)$ is computed using the same neighbors as $U(c)$

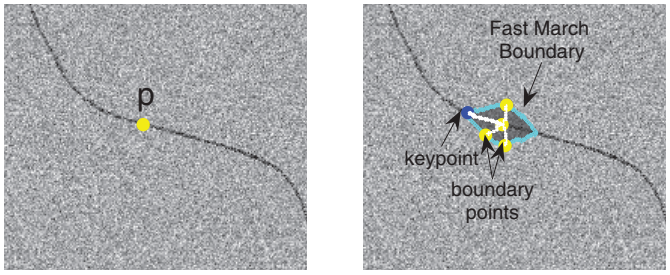
$$L(c) = \begin{cases} L(x_{min}) + 1 & \text{if } U(y_{min}) > U(x_{min}), \\ L(y_{min}) + 1 & \text{otherwise.} \end{cases} \tag{18}$$

2.2 Back Propagation

For a source point set S with a single point p_1, the minimum of U is at the front propagation starting point p_1 where $U(p_1) = 0$. The gradient of U is orthogonal to the propagating fronts since these are its level sets. Therefore, the minimal path between any point q and the starting point is found by sliding back along the gradient of the map U until arriving at $U(p_1)$. This back propagation procedure is a simple steepest gradient descent. It is possible to make a straight forward implementation on a rectangular grid: given a point q, the next point in the chain connecting q to p_1 is selected to be the neighbor p' of q for which $U(p')$ is minimum. The tracking can be made more precise by using the Runge-Kutta method based gradient descent. The back propagation approach can be easily extended to the multiple source points $(p_1, p_2, \ldots, p_n)$ scenario. In the case of multiple source points, the back propagation procedure gives the minimal path between point q and the set of source points $S = \{p_1, p_2, \ldots, p_n\}$. In addition, we can also find the *origin point* $s^* \in S$ from which the minimal path to q originates. The minimal path back propagation procedure will be important for our algorithm and we will denote it by $MinimalPathbk(S, q)$, where S is the set of source points and q is the destination point.

2.3 Minimal Path with KeyPoint Detection

The minimal path procedure described above requires the knowledge of two endpoints. In addition, it also assumes that the potential function is not very noisy and provides enough contrast that can enable the minimal path to track even convoluted, long curves along desired



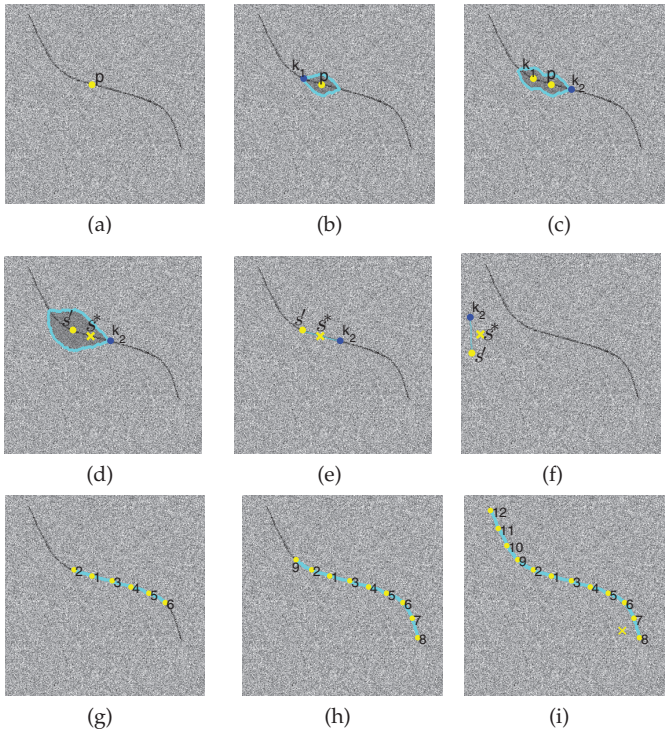
(a) (b)

Fig. 3: (a) Image with a curve of interest and point p. (b) Minimal paths from point p to boundary points (including keypoint).

features. In many applications, these conditions are not met and the desired feature points have a lower potential only in a local neighborhood region. To overcome the limitations of this approach, Benmansour and Cohen [3] introduced the Minimal Path Method With KeyPoint Detection (MPWKD) approach. This algorithm is based on the idea that among all points on the Fast Marching boundary that have equal geodesic distance U values, the points near the desired features will have the maximum Euclidean distance L. In Figure 3a, the desired curve with an initial point p on the curve are displayed. When the Fast Marching boundary propagates with the potential Φ from a source point set S ($S = p$ initially), the first point for which Euclidean distance L exceeds λ is identified. This point is referred to as a *keypoint*. Figure 3b shows the minimal path from several Fast Marching boundary points (including the keypoint) to the initial point p. Among all the minimal paths from the boundary points to the initial point, the minimal path from the keypoint has the largest length. The reason is that the potential Φ at the feature points is lower than the neighborhood points. Therefore the Fast Marching boundary propagates with high speed $1/\Phi$ and travels the furthest Euclidean distance along the feature points. Keypoints are recursively detected until a known endpoint is reached or until the detected curve exceeds a given Euclidean length. At each iteration, the source point set S is augmented by including the detected keypoint. The MPWKD requires prior knowledge of both endpoints or one endpoint plus the total length of the curve. If the curve has multiple branches then the user needs to know all endpoints of the curve. To relax these prior assumptions, we developed a novel algorithm to detect open curves (even with multiple branches) with the knowledge of just an arbitrary known point. This algorithm is described next.

3 Open Curve Detection

The *keypoints* detected in the MPWKD are approximately equally spaced along the curve. We use this fact to find a good stopping criterion for curve endpoint detection. Starting from the initial point p, fronts are propagated with potential Φ and the first point that crosses a Euclidean distance of λ is detected. This point is labeled



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Fig. 4: (a) Synthetic image with initial point p. (b) Detection of first keypoint k_1. (c) Detection of second keypoint k_2. (d) Fast March propagation from set s'. (e) Minimal path on the curve. (f) Minimal path in background region. (g) Curve and keypoints detected after 5 iterations. (h) Curve and keypoints detected after 8 iterations. (i) Final Image with ordered keypoints and terminating point marked by 'x'.

as the first keypoint k_1. The Fast Marching procedure to find the first keypoint k_1 is referred as $FMM(S, \lambda)$ in this document where S is the source point set. The source point set S includes only the point p initially. We use a synthetic image that has a curve with lower mean intensity value than the random background for illustrating the algorithm. A λ value of 30 was used for this image and the potential function was chosen to be the intensity value. Figure 4a shows the image with initial source point p on the curve. Figure 4b illustrates the use of $FMM(S, \lambda)$ to find k_1. The points inside the Fast Marching boundary are scaled to a lower intensity value for better illustration. The source point set S is augmented by including the new keypoint k_1 and the set S now becomes $p \cup k_1$. An associative map $m : S \mapsto S$ is generated that defines an *origin point* for every keypoint in S. For a new keypoint, the origin point is the point closest in the set S to the keypoint. Hence, for k_1, the origin point is the initial point p. Though the initial point p does not have an origin point explicitly because it is not a keypoint, we assign k_1 as the origin point of p. The reason is that the minimal path for two points is symmetric and can be found by propagating the Fast Marching algorithm from either point (in this case p or k_1). Therefore, if started with k_1 as the initial point in source set S, p would be the keypoint determined by

the procedure $FMM(S, \lambda)$. Hence

$$m(p) = k_1, \tag{19}$$

$$m(k_1) = p. \tag{20}$$

Now, the Fast Marching propagation is carried out from the updated set S and the next keypoint k_2 is detected as shown in Figure 4c. Using the minimal path back propagation algorithm $MinimalPathbk(S, k_2)$, the point s^* in S that is a part of the minimal path from k_2 to S is identified. s^* is the origin point of k_2 and $m(k_2)$ equals s^*. In our example, origin point s^* is the initial point p . Next, we find the *prior origin point* s', which is the origin point of s^*, from the map m (s' is the point k_1 for our example). If there exists a continuous path of desired feature points with low potential values between the points s' and k_2, then the minimal path between s' and k_2 should also pass through the vicinity of s^*. Figure 4e shows the minimal path from s' and k_2 in the case where the minimal path overlaps with the curve. When there is no portion of a curve on this minimal path, the path does not pass through the neighborhood of s^* as illustrated in Figure 4f. We denote the Euclidean distance from a source point set s' to a point k_2 as $L(s', k_2)$. The Fast Marching propagation required to determine $L(s', k_2)$ is denoted by $FMM(s', k_2)$. Figure 4d depicts this Fast March propagation. From Figure 4e, we conclude that if the minimal path between s' and k_2 lies on the curve, then

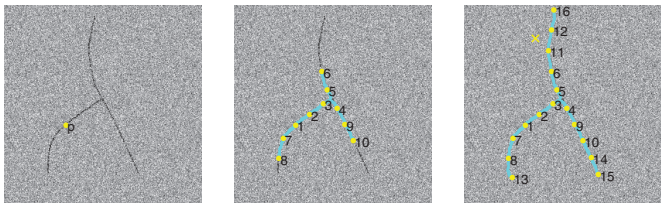
$$L(s', k_2) \approx L(s', s^*) + L(s^*, k_2). \tag{21}$$

$$L(s', k) \approx L(s', s) + L(s, k). \tag{22}$$

As all keypoints are detected sequentially with a fixed Euclidean distance parameter λ, the Euclidean distance L between neighboring keypoints is close to λ. The point s^* is the closest point to the keypoint k_2 in the set S. Hence, by definition of the procedure $FMM(S, \lambda)$, the distance between s^* and k_2 is close to λ . Similarly, the point s' should be at a distance of λ from s^* because s' is the origin point of s^*. This fact is clear from Figure 4e. According to (21), the Euclidean distance $L(S', k_2)$ should then be approximately 2λ if the minimal path from S' to k_2 passes through s^*. If (21) does not hold (like in Figure 4f), then the curve detection procedure can be terminated and the algorithm outputs that there is no curve detected. Otherwise, this procedure of finding keypoints is recursively carried out and the subsequent keypoints are identified as k_2 in the *OpenCurveDetection* algorithm that is given below. The point s^* determined at each iteration is either the initial source point p or a keypoint. A *tolerance error* value ϵ is specified for the termination. We used an ϵ value of 0.2λ or 10% of the total length 2λ for the algorithm. In Section 6.2, a sensitive analysis of the effect of ϵ and λ on curve detection is presented. The intermediate results of the algorithm are shown in Figure 4g and 4h. The complete curve with keypoints (including the initial point) is shown in Figure 4i. The keypoints are labeled according to the order of their appearance. The last keypoint for which (21) does not hold and the algorithm terminates is called the *terminating point* (represented by 'x' in Figure 4i). For a simple curve with no branches, the estimated length of the detected curve should lie within λ distance of the actual length. The above algorithm can also be used on more complex curves. Figure 5a illustrates a synthetic image that has a curve with branches embedded in a noisy random background of higher mean intensity. An arbitrary source point p on the curve is provided as input to the algorithm. Figure 5b demonstrates the intermediate result of the algorithm on the image in which the curve and the keypoints detected are labeled. Figure 5c shows the final result with the complete curve, all ordered keypoints and the terminating point. The next section explores the algorithm for more general curves that have cycles.

Algorithm *OpenCurveDetection*
Input: Image Im, potential function Φ and initial source point set S.
Output: Detected Curve C

1. Start with initial source point set S containing an arbitrary point p on the curve.
2. Set $StopDetection = FALSE$.
3. Use $FMM(S, \lambda)$ to find keypoint k_1.
4. Run $MinimalPathbk(S, k_1)$ and initialize curve C to contain the minimal path between S and k_1.
5. Set $S \leftarrow S \cup k_1$.
6. Set $m(k_1) = p$ and $m(p) = k_1$.
7. **while** $StopDetection = FALSE$
8. **do** Run $FMM(S, \lambda)$ to find new keypoint k_2.
9. Run $MinimalPathbk(S, k_2)$ and find the origin point s^* in set S.
10. Set curve C' to contain the minimal path between S and k_2.
11. Get point $s' = m(s^*)$.
12. Use $FMM(s', k_2)$ to calculate $L(s', k_2)$.
13. **if** $|L(s', k_2) - 2\lambda| < \epsilon$
14. **then** $C \leftarrow C \cup C', S \leftarrow S \cup k_2$ and $m(k_2) = s^*$.
15. **else** $StopDetection = TRUE$.
16. **repeat**

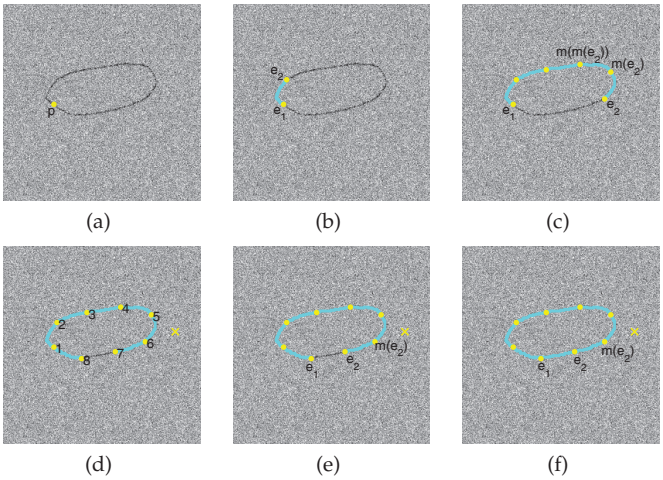


(a) (b) (c)

Fig. 5: (a) Curve with branches and initial point p. (b) Intermediate result from the algorithm after 9 iterations. (c) Final curve detection with keypoints and final terminating point labeled by 'x'.

4 General Curve Detection

The self-terminating minimal path algorithm described in Section 3 can also be extended to curves of general topology that have cycles. For that we need to keep track of keypoints that are *endpoints*. Endpoints are defined as points that have only one neighbor. Two points are neighbors of each other if one point is the origin point of the other. For every point in the source point set S,



(a) (b) (c)

(d) (e) (f)

Fig. 6: (a) Image with a closed curve of interest and initial point p. (b) Endpoints e_1 and e_2 after first keypoint detection. (c) Endpoints e_1 and e_2 and the origin point $m(e_2)$ after 5 iterations. (d) Detected curve from *OpenCurveDetection* with ordered keypoints and terminating point labeled by 'x' . (e) Endpoints e_1 and e_2 and the origin point $m(e_2)$. (f) Complete curve with closure between e_1 and e_2 .

which is augmented each time a new keypoint is added, the origin point is given by the map m as explained in Section 3. The map m is a function, so every point in S has an origin point. Hence, every point in S has at least one neighbor. For any general point q, $m(q)$ and q are neighbors. We illustrate the algorithm on a synthetic image with a closed curve and a starting point p as shown in Figure 6a. When the first keypoint k_1 is detected starting from initial point p, both points are designated as endpoints and added to the endpoint set E as shown in Figure 6b. These endpoints are labeled as e_1 (which is same as p) and e_2 (which is same as k_1) in Figure 6b. For all subsequently detected keypoints, it is easy to update the endpoint set E. Whenever a new keypoint (identified as k_2 in *OpenCurveDetection* algorithm and e_2 in Figure 6c) is detected from the current source point set S and the keypoint satisfies (21) of the *OpenCurveDetection* algorithm, the keypoint is also added to the endpoints E. This is because every new keypoint e_2 has only one neighbor: the origin point of e_2 given by $m(e_2)$. When a new keypoint originating from the point $m(e_2)$ contained in S is added to the set S, the point $m(e_2)$ has a new neighbor e_2. All points in the source point set S have at least one neighbor, which is given by the map m. Hence, the point $m(e_2)$ has at least two neighbors now given by $m(m(e_2))$ and e_2 as shown in Figure 6c. If point $m(e_2)$ is in the endpoint set E (that is it had only one neighbor $m(m(e_2))$ previously), $m(e_2)$ should be removed from the set E. For example, in Figure 6c, $m(e_2)$ was a part of endpoint set E previously and it has to removed after the detection of the new keypoint e_2. Apart from this additional step of keeping track of endpoints, we follow the *OpenCurveDetection* algorithm exactly as before. When the algorithm *OpenCurveDetection* terminates, we examine all pairs of endpoints in the set E. Figure 6d illustrates the results of the *OpenCurveDetection* algorithm with the keypoints and initial point marked in order. Let e_1 and e_2 be any two points contained in the endpoint set E, and $m(e_2)$ be the origin point of e_2 as shown in Figure 6e. A continuous curve between e_1 and e_2 will exist only if the Euclidean distance $L(e_1, e_2)$ is less than 2λ. This is because all keypoints are at a minimum distance of λ from each other and a keypoint cannot be detected on the curve between e_1 and e_2 with the total separation between e_1 and e_2 less than 2λ. If the distance $L(e_1, e_2)$ is more than 2λ, an additional keypoint that lies on the curve between e_1 and e_2 should be detected. For the curve to exist between e_1 and e_2, an additional condition similar to (21) should be satisfied. The condition is given by the following equation:

$$L(e_1, m(e_2)) \approx L(e_1, e_2) + L(e_2, m(e_2)). \qquad (23)$$

By the definition of the origin point, $L(e_2, m(e_2))$ is close to λ, so only the other two quantities need to be computed in the above equation. If (23) is satisfied within a certain tolerance range, the detected curve is closed between endpoints e_1 and e_2 as shown in Figure 6f. This process is repeated for all pairs of points in E (there are only two endpoints in the set E for the image in Figure 6f). The same tolerance value ϵ is used for both Equations (21) and (23) in our algorithm. If the user has prior knowledge that the curve is closed, the keypoint verification given by (21) can be dropped and unconstrained keypoint detection can continue until the two endpoints (there are only two endpoints for a simple closed curve) become closer than 2λ. However, in our algorithm we do not assume this prior knowledge.

Algorithm *GeneralCurveDetection*
Input: Image Im, potential function Φ and initial source point set S.
Output: Detected Curve C

1. Start with initial source point set S containing an arbitrary point p on the curve.
2. Set $StopDetection = FALSE$.
3. Use $FMM(S, \lambda)$ to find keypoint k_1.
4. Run $MinimalPathbk(S, k_1)$ and initialize curve C to contain the minimal path between S and k_1.
5. Set $S \leftarrow S \cup k_1$.
6. Set $m(k_1) = p$ and $m(p) = k_1$.
7. Set $E \leftarrow p \cup k_1$.
8. **while** $StopDetection = FALSE$
9. **do** Run $FMM(S, \lambda)$ to find new keypoint k_2.
10. Run $MinimalPathbk(S, k_2)$ and find the origin point s^* in set S.
11. Set curve C' to contain the minimal path between S and k_2.
12. Get point $s' = m(s^*)$.
13. Use $FMM(s', k_2)$ to calculate $L(s', k_2)$.
14. **if** $|L(s', k_2) - 2\lambda| < \epsilon$
15. **then** $C \leftarrow C \cup C', S \leftarrow S \cup k_2$ and $m(k_2) = s^*$.
16. Set $E \leftarrow E \cup k_2$ and $E \leftarrow E - m(k_2)$.
17. **else** $StopDetection = TRUE$.
18. **for** all pairs of points e_i, e_j in E,
19. **if** $L(e_i, e_j) < 2\lambda$
20. Run $MinimalPathbk(e_i, e_j)$ and initialize C' to contain the minimal path between e_i and e_j.
21. **if** $|L(e_i, m(e_j)) - L(e_i, e_j) - \lambda| < \epsilon$
22. **then** $C \leftarrow C \cup C'$
23. **repeat**

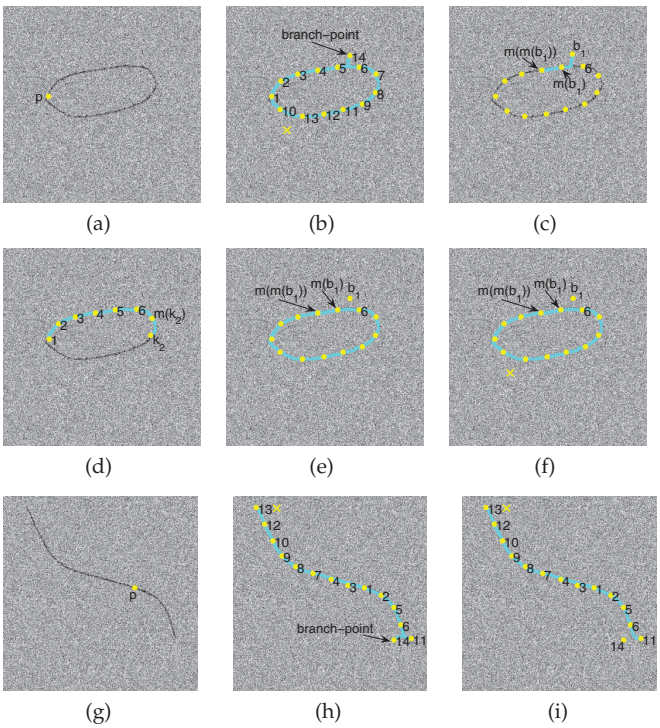
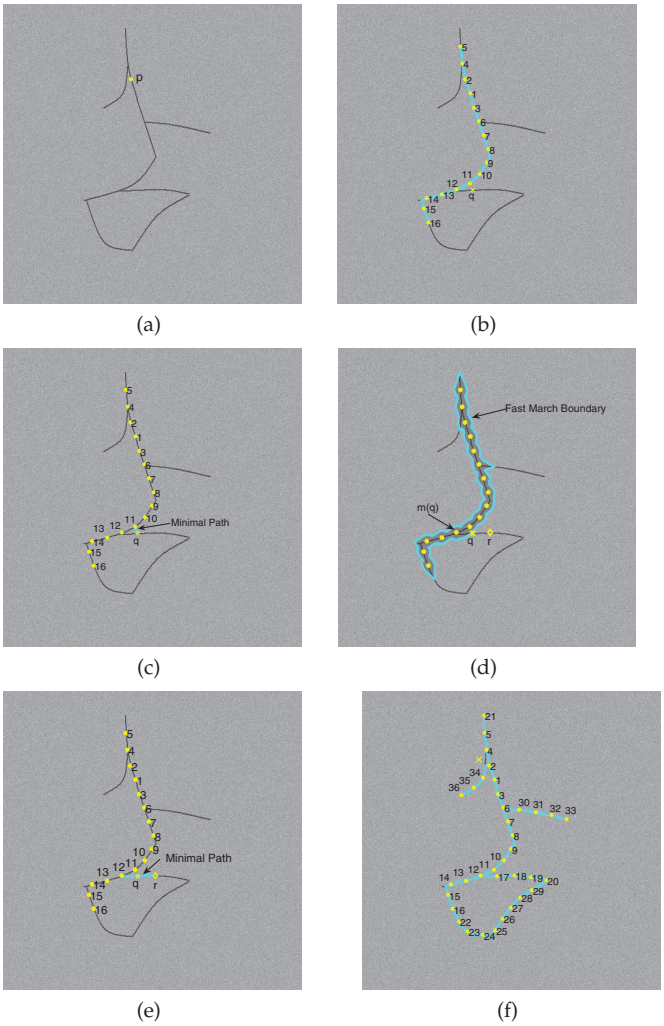


Fig. 7: (a) Image with a closed curve of interest and initial point p. (b) Curve, ordered keypoints and terminating point (marked by 'x') detected by *GeneralCurveDetection*. (c) Minimal path between branch-point b_1 and $m(m(b_1)$. (d) Keypoint k_2 for which the origin point $m(k_2)$ has only 2 neighbors. (e) Branch-point b_1 for which the origin point $m(b_1)$ has 3 neighbors branch-point b_1, point '6' and the origin point $m(m(b_1))$. (f) Complete curve detected by modified algorithm with disconnected branch-point. (g) Image with open curve of interest and initial point p. (h) Curve, ordered keypoints and terminating point (marked by 'x') detected by *GeneralCurveDetection*. (i) Complete curve detected by modified algorithm with disconnected branch-point.

5 Robust Algorithm

In our experimental tests, we found that some modifications of the algorithm described in Section 4 can be made to improve the robustness of curve detection. During the *GeneralCurveDetection* algorithm, there is a possibility of generating additional keypoints that are not on the curve after all the keypoints on the curve have been identified. Figure 7a shows a synthetic image with a closed curve and Figure 7b depicts the results of *GeneralCurveDetection* with parameter $\lambda = 30$. Point '14' in Figure 7b is the additional keypoint that is generated and it produces spurious curve portions. We observed that these additional points that lead to spurious curve detection have an origin point that has at least 3 neighbors. Such points are called *branch-points*. In Figure 7c, the branch-point is labeled b_1, which has origin point $m(b_1)$. The point $m(b_1)$ has 3 neighbors that includes its origin point $m(m(b_1))$, the branch-point b_1 and the point '6', which is the sixth keypoint detected by the *GeneralCurveDetection* algorithm. For the branch-point b_1, Condition (21) is satisfied and *GeneralCurveDetection* algorithm is not terminated. This is because b_1 is located in a neighborhood of the curve section between $m(b_1)$ and point '6'. Hence, a major portion of the minimal path between b_1 and $m(m(b_1))$ lies on the actual curve and passes through the vicinity of $m(b_1)$ as shown in Figure 7c . This ensures that (21) is satisfied for the point b_1. Here points b, $m(b)$ and $m(m(b))$ correspond to k_2, s^* and s' in (21). To overcome the problem of detection of spurious branch-points, we keep track of the spurious branch-points using a set B in the modified algorithm. A map $m_1 : S \mapsto Z$ is generated for all the points in the source point set S. This map gives the number of neighbors for each point in S. Whenever a new keypoint k_2 is generated using the procedure $FMM(S, \lambda)$, the neighbor count given by $m_1(k_2)$ is set to 1. In addition, the neighbor count for the origin point of the new keypoint is incremented by 1. For a keypoint k_2 that is not a branch-point (its origin point $m(k_2)$ has only two neighbors) like in Figure 7d, the earlier algorithm procedure given in *GeneralCurveDetection* is followed without changes. However, if the neighbor count of the origin point $m(k_2)$ is greater than 2 , the new keypoint is identified as a branch-point (which is b_1 in Figure 7e) and added to the branch-point set B. It is also added to the source point set S, but not to the endpoint set E. The minimal path from the branch-point b_1 and its origin point $m(b_1)$ is not added to the complete curve so that spurious curve sections are not detected. A map $n : B \mapsto \mathcal{A}_I$ is generated for all branch-points in B where n stores the minimal path between a branch-point b and its origin point $m(b)$ for all $b \in B$. $\mathcal{A}_I$ is the space of continuous curves in the grid generated by the input image I. We also wanted to ensure that if a branch-point b in B corresponds to a genuine branch of the curve, we add the minimal path given by $n(b)$. Therefore, whenever a new keypoint k_2 has an origin point b in set B, both the minimal path from k_2 to b and the minimal path from $m(b)$ to $m(m(b))$ are added to the detected curve. In addition, the origin point b is removed from the set B. Figure 7f shows the complete curve detected using the modified algorithm where the branch-point curve section is not detected. Figure 7g illustrates another image for which the *GeneralCurveDetection* algorithm detects spurious curve portions. Figure 7h shows the results of *GeneralCurveDetection* algorithm and Figure 7i shows the corrected curve detection results using the modified algorithm. Apart from the modification described above, we identified another area of algorithm improvement. We found that for complex topological curves that have sharp corners at branches the *GeneralCurveDetection* algorithm terminates before the detection of the complete curve. This is illustrated by Figure 8a that shows a synthetic image with a complex topological curve with cycles and branches. Figure 8b depicts the curve detection result of *GeneralCurveDetection* algorithm with ordered keypoints and the terminating point (marked as 'x'). A λ value of 30 was used for the algorithm. Figure 8b shows that the complete curve is not detected and the algorithm stops after the



(a) (b) (c) (d) (e) (f)

Fig. 8: (a) Image with complex topological curve and initial point p. (b) Curve, ordered keypoints and terminating point (marked by 'x') detected by *GeneralCurveDetection*. (c) Minimal path between terminating point q and $m(m(q))$. (d) Incremental keypoint r and Fast Marching boundary of $FMM(S, \lambda)$ (e) Minimal path between $m(q)$ and r. (f) Complete curve detected by *RobustCurveDetection* with ordered keypoints and terminating point marked by 'x'.

Algorithm *RobustCurveDetection*
Input: Image Im, potential function Φ and initial source point set S.
Output: Detected Curve C

1. Start with initial source point set S containing an arbitrary point p on the curve, end point set $E = \emptyset$, branch point set $B = \emptyset$.
2. Set $StopDetection = FALSE$.
3. Use $FMM(S, \lambda)$ to find keypoint k_1.
4. Run $MinimalPathbk(S, k_1)$ and initialize curve C to contain the minimal path between S and k_1.
5. Set $S \leftarrow S \cup k_1$.
6. Set $m(k_1) = p$ and $m(p) = k_1$.
7. Set $m_1(k_1) = 1$ (no. of neighbors is given by the map m_1.) and $m_1(p) = 1$.
8. Set $E \leftarrow p \cup k_1$.
9. **while** $StopDetection = FALSE$
10. **do** Run $FMM(S, \lambda)$ to find new keypoint k_2.
11. Store *STATUS* of all points after $FMM(S, \lambda)$.
12. Run $MinimalPathbk(S, k_2)$ and find the origin point s^* in set S.
13. Set curve C' to contain the minimal path between S and k_2.
14. Get point $s' = m(s^*)$.
15. Use $FMM(s', k_2)$ to calculate $L(s', k_2)$.
16. **if** $|L(s', k_2) - 2\lambda| < \epsilon$
17. **then** $S \leftarrow S \cup k_2$ and $m(k_2) = s^*$.
18. Set $m_1(k_2) = 1$ and $m_1(s^*) \leftarrow m_1(s^*) + 1$.
19. **if** $s^* \in B$ (branch-points set)
20. **then** $C \leftarrow C \cup n(s^*)$, $B = B - s^*$.
21. **if** $m_1(s^*) > 2$
22. **then** $n(k_2) = C'$ and $B = B \cup k_2$
23. **else** $C \leftarrow C \cup C'$, $E \leftarrow E \cup k_2$, $E \leftarrow E - s^*$.
24. **else** $C'' = C'$, $q = k_2$ and $m(q) = s^*$.
25. Use $FMM(q, \lambda, \text{STATUS})$ to compute incremental keypoint r.
26. Set curve C' to contain the minimal path between q and r.
27. Use $FMM(m(q), r)$ to calculate $L(m(q), r)$.
28. **if** $|L(m(q), r) - 2\lambda| < \epsilon$
29. **then** $S \leftarrow S \cup q \cup r$, $C \leftarrow C \cup C'' \cup C'$ and $m(r) = q$.
30. Set $m_1(r) = 1$, $m_1(q) = 2$ and $m_1(m(q)) \leftarrow m_1(m(q)) + 1$.
31. $E \leftarrow E \cup r$, $E \leftarrow E - m(q)$.
32. **else** $StopDetection = TRUE$.
33. **for** all pairs of points e_i, e_j in E,
34. **if** $L(e_i, e_j) < 2\lambda$
35. Run $MinimalPathbk(e_i, e_j)$ and initialize C' to contain the minimal path between e_i and e_j.
36. **if** $|L(e_i, m(e_j)) - L(e_i, e_j) - \lambda| < \epsilon/2$
37. **then** $C \leftarrow C \cup C'$
38. **repeat**

detection of the terminating point, which is located on a branch with a sharp corner. This is because though the potential image (the image intensity value is used as potential for the image) provides a good contrast between the curve and the background, there exists a shorter path between the terminal point, which is called q, and $m(m(q))$ that does not pass through $m(q)$. This fact is shown in Figure 8c where $m(m(q))$ is point '11' and $m(q)$ is point '12'. Hence, $L(m(m(q)), q)$ is not close to 2λ and (21) is violated and the *GeneralCurveDetection* algorithm is terminated. q, $m(q)$ and $m(m(q))$ correspond to k_2, s^* and s' in (21). To fix this problem, we introduced an extra condition for algorithm termination. We compute an additional keypoint r and call it the *incremental keypoint*. This point r is computed using the Fast Marching procedure with the terminal point q as the source point. The incremental keypoint r is the first point for which the Euclidean distance L is greater than λ. However, this keypoint r should also lie outside the Fast Marching boundary derived from the procedure $FMM(S, \lambda)$. Recall that $FMM(S, \lambda)$ is the Fast Marching procedure used to compute the terminal keypoint q in the *GeneralCurveDetection* algorithm where S is the source point set containing all keypoints apart from q. The incremental point r and the Fast Marching boundary for the procedure $FMM(S, \lambda)$ is shown in Figure 8d. In the figure, the region inside the Fast Marching boundary is scaled to a lower intensity value for better illustration. The status of all the grid points in an image after the execution of the $FMM(S, \lambda)$ Fast Marching procedure is stored in an array called *STATUS*. Points inside the Fast Marching boundary of $FMM(S, \lambda)$ correspond to the Solved points in the Fast Marching algorithm described

in Section 2. Hence, the incremental point r should not be a Solved point in the *STATUS* array. This condition ensures that the new keypoint r originates from the terminal point q and does not correspond to other keypoints on the curve that are already computed. The Fast Marching procedure to compute the incremental keypoint r is called $FMM(q, \lambda, \text{STATUS})$. If a condition similar to (21) given by

$$L(m(q), r) \approx L(m(q), q) + L(q, r), \tag{24}$$

is satisfied then the curve detection algorithm is not terminated. This condition ensures that if the incremental keypoint r lies along a branch of the curve as shown in Figure 8e, the curve detection procedure continues. Since the new point r is restricted to lie outside the Fast Marching boundary of $FMM(S, \lambda)$, we found that a lower tolerance value should be used for the validating Condition (24) compared to Condition (21). According to the results of the quantitative validation study that is described in detail in Section 6.2, we chose the tolerance value to be $\epsilon/2 = 0.1\lambda$, half of the value of Condition (21). In Figure 8e, the incremental keypoint r satisfies Condition (24) because the minimal path between r and $m(q)$ passes through the point q. The algorithm can therefore proceed till Condition (24) is not satisfied for a terminating point q. The complete curve detected using the modified algorithm is shown in Figure 8f. Figure 8f also shows all the ordered keypoints and the terminating point (point for which Condition (24) is not satisfied). The robust curve detection algorithm that was designed with the two modifications discussed in this section is described in *RobustCurveDetection*.

6 Experimental Results and Analysis

In order to analyze the novel self-terminating minimal path algorithm explained in Section 5 (*RobustCurveDetection*), the algorithm was tested extensively on both real and synthetic experimental data sets. This section presents a detailed analysis of these experiments. We used the true positive rates (TP), false positive rates (FP) and false negative rates (FN) as defined in [13] to evaluate the accuracy of our algorithm. For real images that have features like cracks, ground truth accuracy itself is suspect and it is difficult to accurately quantify the performance of a crack detection algorithm. To overcome this problem, Kaul et al. [15] devised the scaled buffered distance (SBD) metric. The SBD is a scaled version of the buffered Hausdorff distance measure described in [15]. The buffered Hausdorff distance is given by given by $BD(A, B)$ where A and B are coordinate locations of the curve in the ground truth image and the processed image respectively.

$$BD(A, B) = \max(h(A, B), h(B, A)), \tag{25}$$

where

$$h(A, B) = \frac{1}{m} \sum_{a \in A} sat_L \min_{b \in B} ||a - b||. \tag{26}$$

$BD(A, B)$ lies between 0 and L and we normalize this value to lie between 0 and 1 to get scaled buffered distance $SBD(A, B)$. The value of SBD approaches 0 if the curve detected by the algorithm and the curve in the ground truth image are close to each other.

$$SBD(A, B) = BD(A, B)/L. \tag{27}$$

The value of L was chosen to be 50 for our experiments.

In this section, we first present the experimental results of our algorithm on crack images in pavements and concrete structures. The consolidated quantitative validation of several crack images is also presented. Next, we present the results of a detailed sensitivity study on the effect of algorithm parameters λ and ϵ and effect of image noise on curve detection. Synthetic images with varying background intensities were used to conduct the sensitivity analysis.

6.1 Results on Crack Images

Cracks in structures can be modeled as curves that are darker in the intensity than the background. Recent studies [18] show that automatic detection of cracks in these structures is very challenging because of multiple textures, shadows, variable lighting, irregular background and high noise present in the images. Most algorithms use multiple heuristic parameters that help to detect cracks for only a small data set of images. Hence, it is useful to have an algorithm that can detect cracks in local regions using only one user defined crack point. The objective of the experimental tests was to detect cracks of different types in images with different backgrounds (both asphalt and concrete) and lighting conditions using an arbitrary user defined point on the crack. We used 120 images provided by Georgia Department of Transportation (GDOT) and Dr Ezzatollah Salari from University of Toledo to conduct our study. Figure 9 presents the algorithm results on 5 pavement images that have different lighting conditions and crack types, which include longitudinal, transverse, diagonal, complex crack with branches and a complex crack with branches and closed cycles, and 1 concrete structure image with a crack. The *RobustCurveDetection* algorithm was used on all the crack images. Some of the processed images have branch-points that were discussed as part of the *RobustCurveDetection* algorithm. The robust algorithm avoids false positives associated with these branch-points. Figure 9h, 9i, 9k and 9l contain branch-points that have no crack detection associated with them. The state of the art keypoint detection algorithm devised by [3] will require a prior knowledge of a minimum of 3 endpoints in Figure 9d and 7 endpoints in 9d to detect cracks with similar accuracy to the proposed algorithm. The qualitative analysis of the algorithm on the 120 pavement and concrete structure images gave very good results for crack detection. A λ value of 50 and an ϵ value of 0.2λ were used for all the crack images in the data set. Similar results were obtained for a large λ range between

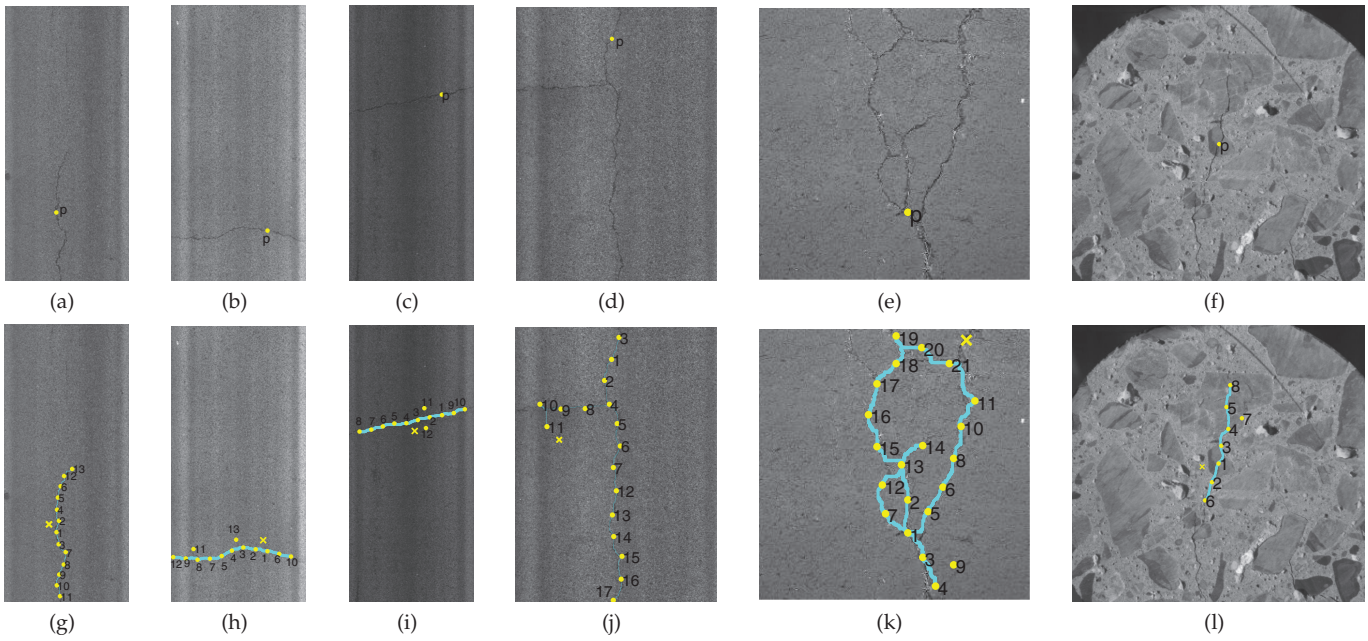


Fig. 9: (a) Longitudinal crack. (b) Transverse crack. (c) Diagonal crack. (d) Complex crack with multiple branches. (e) Complex crack with multiple branches and closed cycles. (f) Concrete structure crack. (g) Longitudinal crack detected with ordered keypoints and terminating point (marked by 'x'). (h) Transverse crack detected with ordered keypoints (including branch-points) and terminating point (marked by 'x'). (i) Diagonal crack detected with ordered keypoints (including branch-points) and terminating point (marked by 'x'). (j) Complex crack detected with ordered keypoints and terminating point (marked by 'x'). (k) Complex crack containing branches and closed cycles detected with ordered keypoints (including branch-points) and terminating point (marked by 'x'). (l) Concrete structure crack detected with ordered keypoints (including branch-points) and terminating point (marked by 'x').

40 and 60. A more rigorous, quantitative validation using FN, FP, TP and SBD measures was conducted on a diverse and representative data set containing 44 pavement and concrete structure images with variable texture and lighting conditions.

TABLE 1: Quantitative validation of crack images.

Image	TP(%)	FP(%)	FN(%)	SBD
Figure 9a	91.37	15.31	8.73	0.094
Figure 9b	91.21	3.1	8.79	0.0364
Figure 9c	91.38	13.52	8.62	0.1121
Figure 9d	92.35	18.1	7.65	0.121
Figure 9e	81.16	10.80	18.84	0.0693
Figure 9f	88.55	5.59	11.45	0.092
Average (48 images)	90.52	14.83	10.52	0.0683

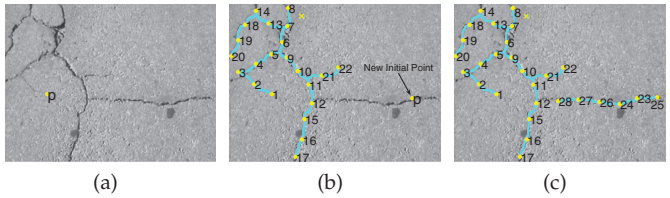


Fig. 10: (a) Complex crack and initial point p. (b) Crack detection results, ordered keypoints, terminating point (labeled by 'x') and new initial point p on undetected curve portion (c) Final crack detection with ordered keypoints and final terminating point labeled by 'x'.

For some images that have complex cracks with substantial breaks between separate sections of the cracks, the complete crack is not captured by the algorithm as illustrated in Figure 10a and 10b. For such cases, the self-terminating algorithm also provides the user an option of interactively providing additional inputs. The algorithm can use the curve detection results from the first run of the algorithm and an additional user point in the undetected portion of the curve as starting conditions for next run of the algorithm. Figure 10c shows the crack detection results after the user provides an additional point. We also used our algorithm to

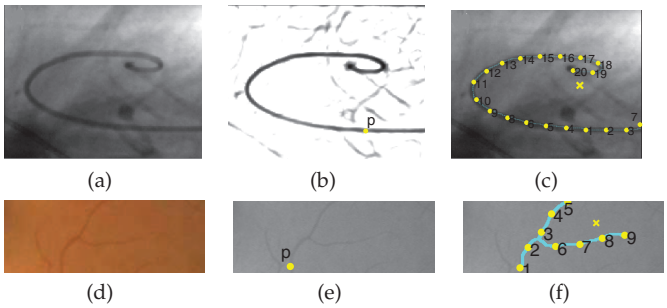


Fig. 11: (a) Catheter tube image. (b) Edge based potential image. (c) Final catheter tube detection with ordered keypoints and terminating point (marked by 'x'). (e) Color Retinal image. (f) Grayscale image. (g) Final optical nerve detection with ordered keypoints and terminating point (marked by 'x').

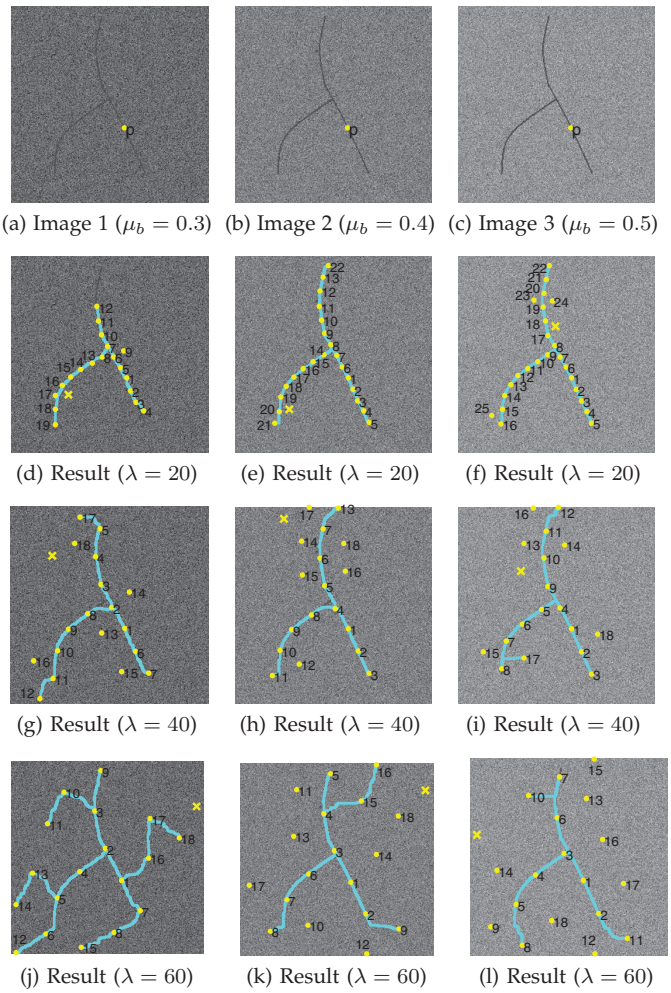
detect features or objects in medical images that can be modeled as thin curves. Figure 11a shows a medical image containing a catheter tube. A potential function based on the Laplacian is used to provide contrast between the desired feature and the background. Figure 11b illustrates the potential function image. Figure 11c shows the detected curve (catheter tube), keypoints and

terminating point for the catheter tube image. We also applied our algorithm to 3 retinal images containing optical nerves, one of which is shown in Figure 11d. The color image is converted to grayscale and shown in Figure 11e. An intensity based potential function was used as input for the *RobustCurveDetection* algorithm. Figure 11f shows the detected curve (optical nerve), keypoints and terminating point for the retinal image. These examples demonstrate the wide applicability of our algorithm.

A quantitative validation was carried out for 48 images (44 crack images, 3 retinal images and 1 catheter tube image). The ground truth for crack data was determined by GDOT engineers for 44 pavement images. They were asked to represent the crack with zero-width curves in the local neighborhood of the initial source point p. Table 1 presents the consolidated validation results of the crack images shown in this chapter. The validation results include the TP, FP, FN and SBD values. The average validation measure values for the 48 images, which were evaluated, is provided in Table 1. The results show that our algorithm achieves consistently high TP rates. The FP rates and FN rates arise because of the fact that the algorithm adds a minimal path of length λ at every iteration step and this tends to slightly overestimate or underestimate the crack or the desired feature. The SBD value, which was established in [15] as an excellent measure for quantitative validation of cracks, shows consistently good results for all images. The total average value of SBD is **0.0721**, which corresponds to an absolute buffered distance (BD) value of **3.41** out of the maximum of 50. As ground truth accuracy of crack and retinal images is suspect because it is derived from manual hand markings, the SBD value indicates the good performance of our algorithm. In the future, more crack images can be tested to validate our algorithm.

6.2 Sensitivity Analysis of Parameters λ and ϵ

We conducted an extensive quantitative study on sensitivity of parameter λ using synthetic images and kept parameter ϵ constant at 0.4λ. Four basic synthetic images that contain a simple curve, closed curve, curve with multiple branches and a complex topology curve (similar to Figures 4a, 5a, 6a and 8a used for algorithm illustration) were used for this validation study. The curves were chosen to be of mean intensity $\mu_c = 0.2$ and variance $\sigma_c = 0.05$. As we used a potential function based on intensity for our algorithm, we varied the mean intensity μ_b of the background to see its impact on the curve detection rate. The background variance σ_b was chosen to be 0.1. We found that when $\mu_b > 0.5$, the curve detection was very accurate and there was no impact of varying μ_b beyond a value of 0.5. Therefore, we chose μ_b to be 0.3,0.4 and 0.5 for our study. To determine the impact of the Euclidean length parameter λ on the curve detection, we varied λ between 20 to 60. μ_b and λ were varied for each of the four synthetic images and the



(a) Image 1 ($\mu_b = 0.3$) (b) Image 2 ($\mu_b = 0.4$) (c) Image 3 ($\mu_b = 0.5$)

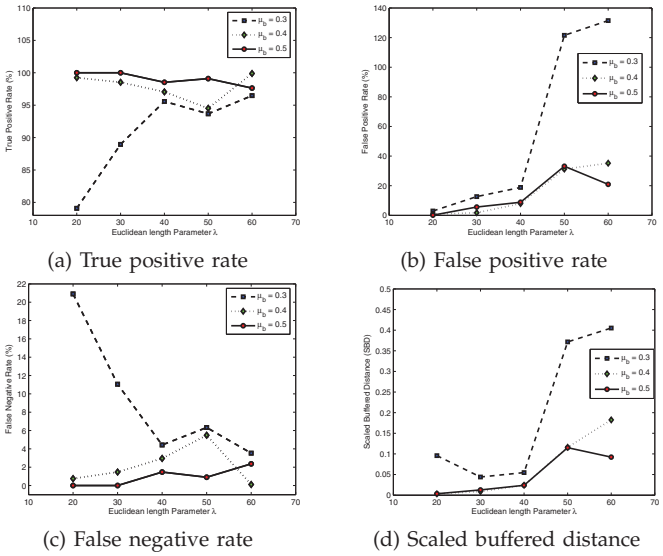
(d) Result ($\lambda = 20$) (e) Result ($\lambda = 20$) (f) Result ($\lambda = 20$)

(g) Result ($\lambda = 40$) (h) Result ($\lambda = 40$) (i) Result ($\lambda = 40$)

(j) Result ($\lambda = 60$) (k) Result ($\lambda = 60$) (l) Result ($\lambda = 60$)

Fig. 12: Synthetic Images corresponding to different background mean intensity $\mu_b = 0.3, 0.4, 0.5$ and algorithm results for $\lambda = 20, 40, 60$.

average TP, FP, TN and SBD values for the detected curves were calculated. The plots for TP, FP, TN and SBD values are shown in Figure 13. Figure 12 shows one of the four synthetic images that contains a curve with multiple branches. The curve detection results for λ values 20, 40 and 60 are illustrated for each of the three images generated using different μ_b. The plots and Figure 12 indicate that for higher μ_b values (0.4 and 0.5), a smaller λ value gives better curve detection results as indicated by the TP, FP, TN and SBD values, though the overall results are fairly good even for higher λ values. This is because the potential provides enough contrast between the curve and the background and a smaller λ can help to terminate the *RobustCurveDetection* algorithm near the endpoints of the curve. Recall that the Euclidean length λ is the Euclidean distance L between detected keypoints and the curve is detected in minimum increments of λ. However, when μ_b is 0.3, a smaller λ value of 20 leads to a high FN rate. The reason for this is that a lower μ_b value provides poor contrast between the curve and the background, and a small interval λ is not sufficient for the keypoints to lie on the desired curve. Hence, Equations (21) and (24) are not satisfied



(a) True positive rate (b) False positive rate

(c) False negative rate (d) Scaled buffered distance

Fig. 13: Plots indicating variation of true positive, false positive and false negative rates, and scaled buffered distance with changes in parameter λ and background intensity μ_b

and the algorithm is terminated before the entire curve is detected. On the other hand as we increase the value of λ, the FN rate reduces and the FP rate increases. For $\mu_b = 0.3$ and λ values 50 and 60, the curve detection algorithm shows very high FP rates. The SBD plot in Figure 13d indicates that a value of λ between 30 and 40 gives the lowest value of SBD for $\mu_b = 0.3$. Therefore, for images with a poor potential function that does not provide enough contrast between the curve and the background, an optimal λ that minimizes SBD can be picked. In general, the results indicate that a fairly good potential function that provides enough local contrast between the curve and the background is essential for the high accuracy of our algorithm. This is true for all minimal path based techniques.

We also assessed the impact of the tolerance value ϵ on the curve detection algorithm. Recall that we used a tolerance ϵ for Condition (21) and (23) and tolerance $\epsilon/2$ for the incremental keypoint Condition (24). Our sensitivity study showed that ϵ values between 0.2λ and 0.6λ had very little impact on Conditions (21) and (23). On the other hand, ϵ value had an impact on Condition (24) because the incremental keypoint r in the (*RobustCurveDetection*) algorithm is restricted to originate only from the previous keypoint q. Hence, a higher ϵ value leads to more false positives because many points that were not on the actual curve were able to meet a relaxed tolerance value for Condition (24).

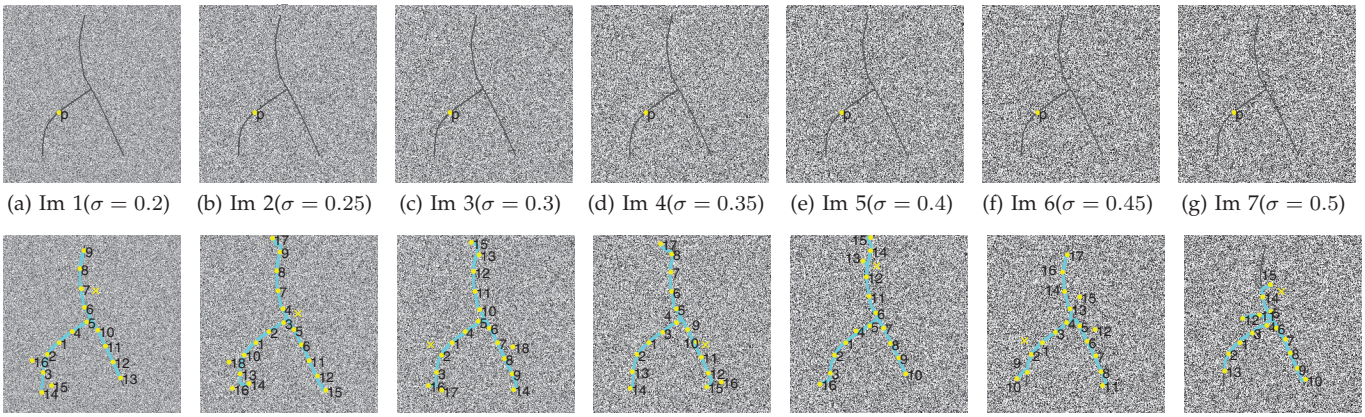
6.3 Noise Analysis

We also examined the effect of noise variance σ_b on the algorithm results. 3 synthetic images were used that had intensity values in the range $[0, 1]$, desired curve mean intensity μ_c of 0.2, noise mean intensity μ_b of 0.5. Gaussian noise variance σ_b was varied between 0.2 to 0.5 for each of the three synthetic images. One of the synthetic images that was used in the tests is shown in Figure 14. The algorithm results are very accurate for σ between 0.2 and 0.45. Even for $\sigma = 0.5$, a large portion of the curve is detected. Similar results were obtained for the other two synthetic images. The results demonstrate that the algorithm is robust to noise variance and can detect curves fairly accurately for images with low Signal-to-Noise (SNR) ratio.

7 Conclusion

The primary objective of the paper was to develop an algorithm that detects complex curves with less user interaction and prior knowledge compared to the prior state of the art minimal path methods, most of which require all desired curve enpoints as initial input for open curves. The proposed algorithm requires the knowledge of just *one arbitrary point* (not necessarily an endpoint) to achieve the same results. In addition, the proposed algorithm, with no additional input, is also able to capture complex curves containing both closed cycles and multiple branches, something which the prior state of the art could not accomplish with a significant amount of additional prior knowledge to separate the open and closed parts of the curve. Experimental results that contain quantitative evaluation for 48 images that include crack images and retinal images demonstrate the efficacy of the algorithm. In addition, the algorithm results are fairly robust to the addition of noise in the image. Sensitivity studies on the algorithm parameter λ in Section 6.1 indicate that a good potential function that provides enough local contrast between the curve and the background is essential for high accuracy of our algorithm. This limitation also applies, however, to prior state of the art minimal path methods. Pre-processing of the image and better segmentation of the potential function can help to overcome this limitation.

There are several future research directions that can be pursued. First, the self-terminating minimal path algorithm can be extended to detect higher dimensional curves. In particular, it will be useful to detect tubular structures like vessels where these structures are modeled as 3D and 4D curves [13] (2D structures as 3D curves and 3D structures as 4D curves). An extra dimension is introduced to model the width of tubular structure. Even wide cracks can be modeled as higher dimensional curves to capture crack width information. Currently, further user interaction is needed to detect these structures using the method formulated by Hua and Yezzi [13]. Our novel algorithm has the potential to substantially reduce this user interaction. Second, work can be done on complete automation of the curve detection process where the user does not have to supply any point on the curve. More extensive testing on image that have complex structural cracks and features like thin capillaries and bone cracks in medical images can be conducted. Finally, code optimization can be done to improve the computational speed of the current algorithm. The current algorithm implementation does not



(a) Im 1($\sigma = 0.2$) (b) Im 2($\sigma = 0.25$) (c) Im 3($\sigma = 0.3$) (d) Im 4($\sigma = 0.35$) (e) Im 5($\sigma = 0.4$) (f) Im 6($\sigma = 0.45$) (g) Im 7($\sigma = 0.5$)

(h) Im 1($\sigma = 0.2$) (i) Im 2($\sigma = 0.25$) (j) Im 3($\sigma = 0.3$) (k) Im 4($\sigma = 0.35$) (l) Im 5($\sigma = 0.4$) (m) Im 6($\sigma = 0.45$) (n) Im 7($\sigma = 0.5$)

Fig. 14: Synthetic images and algorithm results corresponding to different noise variance σ_b.

exploit redundant information between different iterations. This redundant information can be used to make the algorithm faster.

ACKNOWLEDGMENTS

The authors thank Prof. Laurent Cohen at the University of Dauphine for helpful conversations, Dr Ezzatollah Salari from University of Toledo for the raw images in Figures 9e and 10a, and the Georgia Department of Transportation for their support.

REFERENCES

[1] D. Adalsteinsson and J. A. Sethian, "A fast level set method for propagating interfaces," *J. of Computational Physics*, vol. 118, no. 2, pp. 269 – 269, 1995.

[2] R. Ardon, L. D. Cohen, and A. Yezzi, "Fast surface segmentation guided by user input using implicit extension of minimal paths," *J. of Mathematical Imaging and Vision*, vol. 25, no. 3, pp. 289–305, 2006.

[3] F. Benmansour and L. D. Cohen, "Fast object segmentation by growing minimal paths from a single point on 2D or 3D images," *J. of Mathematical Imaging and Vision*, vol. 33, no. 2, pp. 209–221, 2009.

[4] S. Bonneau, M. Dahan, and L. D. Cohen, "Single quantum dot tracking based on perceptual grouping using minimal paths in a spatiotemporal volume," *IEEE Transactions on Image Processing*, vol. 14, no. 9, pp. 1384–95, 2005.

[5] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," in *IEEE International Conference on Computer Vision*, 1995.

[6] L. D. Cohen, "Multiple contour finding and perceptual grouping using minimal paths," *J. of Mathematical Imaging and Vision*, vol. 14, no. 3, pp. 225–236, 2001.

[7] L. D. Cohen and R. Kimmel, "Global minimum for active contour models: a minimal path approach," *International J. of Computer Vision*, vol. 24, no. 1, pp. 57–78, 1997.

[8] N. Cornea, D. Silver, X. Yuan, and R. Balasubramanian, "Computing hierarchical curve-skeletons of 3d objects," *The Visual Computer*, vol. 21, pp. 945–955, 2005.

[9] P.-E. Danielsson and Q. Lin, *Image Analysis*, ser. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2003, ch. A Modified Fast Marching Method, pp. 631–644.

[10] T. Deschamps, J. M. Letang, B. Verdonck, and L. D. Cohen, "Automatic construction of minimal paths in 3D images: an application to virtual endoscopy," in *Computer Assisted Radiology and Surgery.*, 1999, pp. 151–5.

[11] M. S. Hassouna and A. A. Farag, "Multistencils fast marching methods: A highly accurate solution to the eikonal equation on cartesian domains," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 1563–1574, 2007.

[12] M. Hassouna and A. Farag, "On the extraction of curve skeletons using gradient vector flow," in *Proc. of IEEE International Conference on Computer Vision*, 2007.

[13] L. Hua and A. Yezzi, "Vessels as 4-D curves: global minimal 4-D paths to extract 3-D tubular surfaces and centerlines," *IEEE Transactions on Medical Imaging*, vol. 26, no. 9, pp. 1213–23, 2007.

[14] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models," *International J. of Computer Vision*, vol. 1, no. 4, pp. 321 – 31, 1987.

[15] V. Kaul, Y. Tsai, and R. Mersereau, "A quantitative performance evaluation of pavement distress segmentation algorithms," *Transportation Research Record: J. of Transportation Research Board*, 2010.

[16] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Gradient flows and geometric active contour models," in *IEEE International Conference on Computer Vision*, 1995.

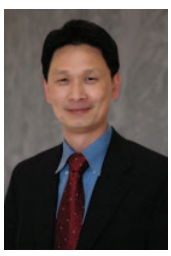
[17] J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.

[18] Y. Tsai, V. Kaul, and R. Mersereau, "Critical assessment of pavement distress segmentation methods," *ASCE J. of Transportation Engineering*, vol. 136, pp. 11–19, 2010.

[19] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1528–1538, 1995.

Vivek Kaul Vivek Kaul received his PhD degree in Electrical and Computer Engineering from Georgia Institute of Technology in 2010. He also holds Masters degrees in Mathematics, Electrical and Computer Engineering, and Aerospace Engineering from Georgia Tech. He currently works in the statistical risk detection for PayPal Inc. His interests lie primarily within the fields of computer vision, machine learning and pattern recognition.

Anthony Yezzi Anthony Yezzi received his Ph.D. in Electrical and Comptuer Engineering from the University of Minnesota. After completing a post-doctoral research appointment at Massacusetts Institute of Technology he joined the faculty at Georgia Tech where he directs the Lab for Computational Computer Vision. His research lies primarily within the fields of image processing and computer vision with an emphasis on shape optimization and partial differential equations.



Yichang (James) Tsai Yichang (James) Tsai received the PhD degree from in 1996. After working ten years in GIS center, he joined the faculty of the School of Civil and Environmental Engineering, Georgia Institute of Technology, in 2007, where he currently holds the position of associate professor. His research lies primarily within the applications of image processing and laser technology to infrastructure condition assessment, and spatial optimization.