Active Contour Based Object Detection

Falk Lempelius and Josef Pauli
Lehrstuhl für Kognitive Systeme, Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel

1 Introduction

A usual way to detect the contour of an object is to extract the edges of the whole image using the magnitude of the grey level gradient, then to match an edge model of the object with the edge image and to locate it by looking for the strongest correlation. The gradient based edges come from grey level fluctuations within the object and from the object border. Often the edges within the object are undesirable due to shading caused by nonperfect lighting conditions. Instead of that the edges at the object border are stable and object detection should mainly take these into account.
Therefore a useful approach to detect border edges is to use a priori knowledge for distributing a set of points roughly around the object in a first step. In the second step the procedure iteratively moves the points towards the object to stop finally if all points are laying directly at the border. The polygon received by connecting every point of the sequence with its successor is an approximation of the object contour. This way of proceeding first was discussed by Kass et al. [1]. Such a sequence of points that change their location by time based on several constraints is called an *active contour* or figuratively, a *snake*.
The kind of constraints which let the snake move (or more exactly the points) is a characteristic property of every snake algorithm. There exist different snake algorithms, which minimize the so-called *snake energy*. The snake energy represents the grade of the approximation of the object border, it can depend on image grey levels, curvature, smoothness, snake length etc. (see [1], [2]).
In this paper two calculation rules are presented for forcing a snake to approximate the contour of a selected object or the outer contour of an assembly of objects. Actually this second capability of extracting the contour of an object assembly characterises the novelty of the approach. The snake works on an image of flat scene objects. Therefore the border of the real 3D object can easily be reconstructed from the object contour in the image. It was intented to develop a system which works with nearly arbitrarily chosen initial snake points, to get a snake which "finds" the object even if the initial points are put far away from the desired object. A further goal was to close gaps in the outer contour of an object assembly if the space between two objects has to be bridged respectively.

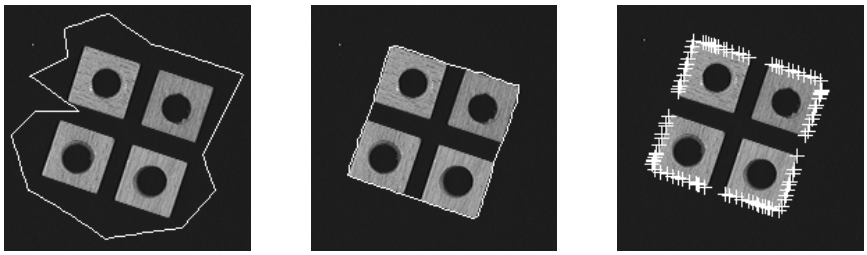
2 Calculation Rules

With regard to objects and object assemblies it is not possible for the snake to decide if a nonconvex object should be mould or a gap between two objects should be bridged. For this reason there are two different algorithms which serve the two contradictory requirements and the user has to select the proper algorithm.
The two calculation rules are based on the analysis of the 8-neighbourhoods of each snake point (Figure 2a). The algorithms run in a loop of iterations and in each iteration the sequence of all points is considered. Each point is moved in a way that the energy of the

snake can be reduced maximally. The loop stops when the energy has converged to a minimal value.

2.1 Minimization of the Snake Length

Using the snake length as the main constraint for appoximating the object, it is obviously possible to close gaps in the contour, in the same way as to span a bigger distance between two objects (Figure 1). The minimization takes place in the following way: Start by taking



(a) (b) (c)

Figure 1: (a) start polygon, (b) end polygon, (c) distribution of the points at the end.

two successive points P_{i-1} and P_i into consideration. Look for the point $P_{i_{min}}$ in the 8-neighbourhood of P_i which minimizes the euclidean distance to P_{i-1}. If the replacing of P_i by $P_{i_{min}}$ is done for all points of the sequence iteration by iteration the snake polygon begins to shrink. Actually, the points rotate around the object and come closer and closer (Figure 2b). However, the replacement of point P_i by $P_{i_{min}}$ only makes sense if the grey level jump is smaller than a given threshold.

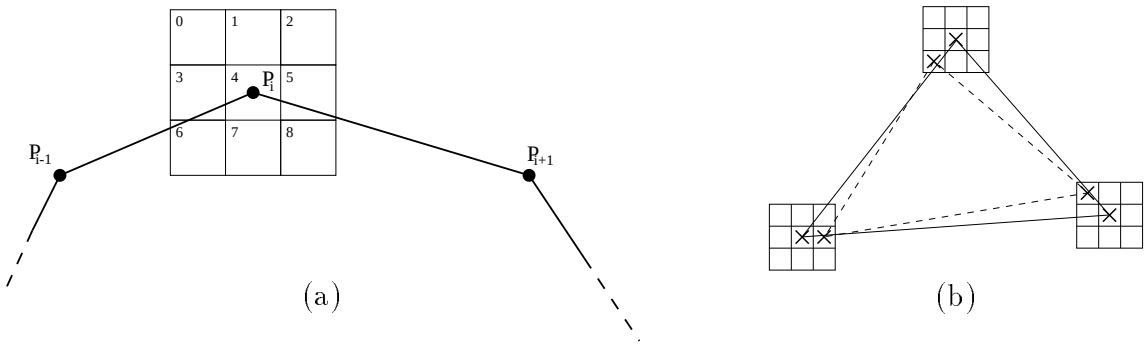


Figure 2: (a) 8-neighbourhood, (b) length minimization.

Applying this procedure the snake tends to approximate only convex objects or forms the convex hull of several included objects. The rotating behaviour of points can be used to make the algorithm more robust against salt and pepper disturbances in the image. Everytime when an edge is detected (because the grey level jump exceeds the threshold) a so-called hit is counted. Thus it can be introduced another parameter value for the maximum number of edge hits. A point does not have to stay constantly at an edge when it is detected for the first time. Rather, in combination with the rotating behaviour a snake point can pass by isolated points in the image that do not belong to the desired contour. All together the snake points approach the object, run along the object contour and stay at remarkable places of

the contour, e.g. edges (Figure 1c). To avoid too big gaps between points of the snake, a maximum value for the distance between the snake points can be used.

2.2 Minimization of the Snake Polygon Area

As mentioned in the previous section, the minimization of the snake length is only a useful approach for an object assembly or a convex object. However, to mould nonconvex areas of an object a calculation rule is needed that works mainly by minimizing the area of the snake polygon. Instead of minimizing the whole area of the polygon many times in an iteration (equal to the number of snake points), it is more efficient to calculate the triangle areas (Figure 3a) spaned by the vectors

$$V_{1_i} := P_{i+1} - P_i \quad \text{and} \quad V_{2_i} := P_{i+2} - P_i, \quad \text{for} \quad P_i = \begin{pmatrix} x_i \\ y_i \\ 0 \end{pmatrix} \quad (\text{for} \quad i = 0, \cdots, n-2), \quad (1)$$

by using the following formula:

$$V_{triangle_i} := \frac{V_{1_i} \times V_{2_i}}{2}, \qquad A_{triangle_i} := ||V_{triangle_i}|| = |\pi_3(V_{triangle_i})|. \qquad (2)$$

The $\times$ denotes the cross product and π_3 the projection on the third component of the vector. The $V_{triangle_i}$ is a 3-dimensional vector, and in our special case the first two components are obviously zero because V_{1_i} and V_{2_i} are laying in the x-y-plane.

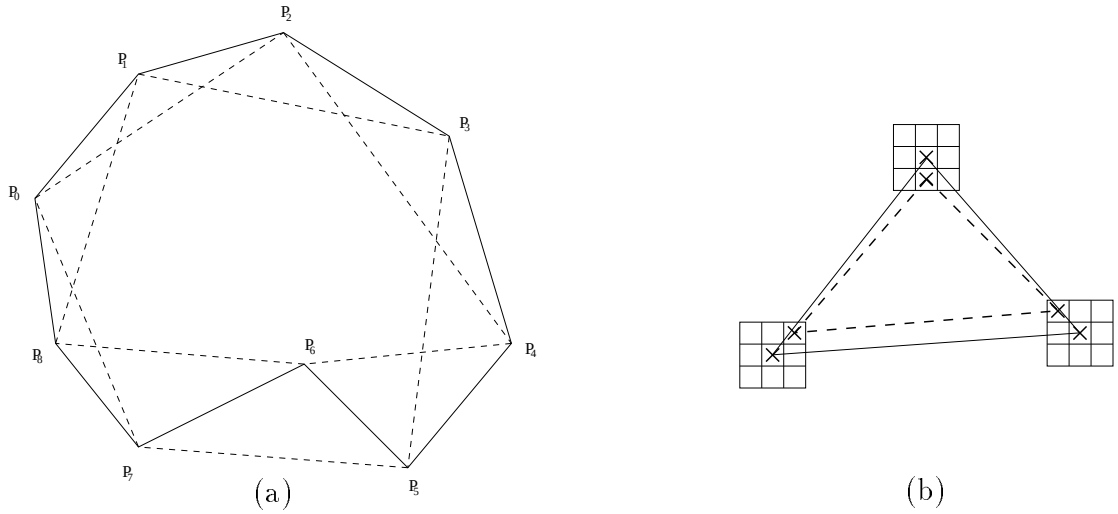


Figure 3: (a) single triangle minimization, (b) area minimization.

The third component represents the area of the triangle and its sign is important for deciding if the triangle belongs to the polygon area or not. The sign depends on enumerating the points P_i clockwise or counterclockwise. For example the area of the triangle that is described by P_5, P_6, and P_7 in Figure 3b does not belong to the area of the polygon and its sign differs from the sign of all other triangle areas. So the area $A_{triangle_i}$ cannot be minimized because it is unknown whether $\pi_3(V_{triangle_i})$ is positiv or negativ. To manage this problem a value σ is used,

$$\sigma := \text{sgn}(\pi_3(\sum_{i=0}^{n-2} \begin{pmatrix} x_{i+1} - x_0 \\ y_{i+1} - y_0 \end{pmatrix} \times \begin{pmatrix} x_{i+2} - x_0 \\ y_{i+2} - y_0 \end{pmatrix})) \qquad (3)$$

with which each single triangle area is normalized in some way. The sum of cross products calculates a vector with the (double) area of the snake polygon in its third component. The variable σ is positive if the points P_i of the polygon run counterclockwise. Consequently the expression

$$\sigma \cdot \pi_3(V_{triangle_i}), \tag{4}$$

is received, which describes one single triangle area that has to be minimized. That means, in every iteration of the algorithm it is tested which point in the 8-neighbourhood of P_i minimizes the area of the triangle area $\sigma\pi_3(V_{triangle_{i-1}})$ optimally. The grey level jump is tested in the same way as in the previous algorithm. The snake polygon shrinks in the way as shown in Figure 3b.

3 Experiments

We demonstrate exemplarily a combined use of the two calculation rules. The first calculation rule minimizes the contour length of an arch shaped object. Figure 4 shows the snake at every 10 iterations of the loop. The algorithm stops with a nearly rectangular contour, and all snake points are located at the convex part of the object border. If we now use these snake points as the initial distribution for the second calculation rule, also the nonconvex part of the object border can be approximated (before starting the algorithm the gap is automatically filled with equidistantly distributed points). Figure 5 shows the snake at every five iterations of this second loop.

The depicted objects are real objects and the image background has small grey level fluctuations. The initial distance of the snake points was five pixels and to detect edges a threshold of 20 grey levels was used. The maximum distance between a point and its successor was unlimited and the allowed number of edge hits was 25.

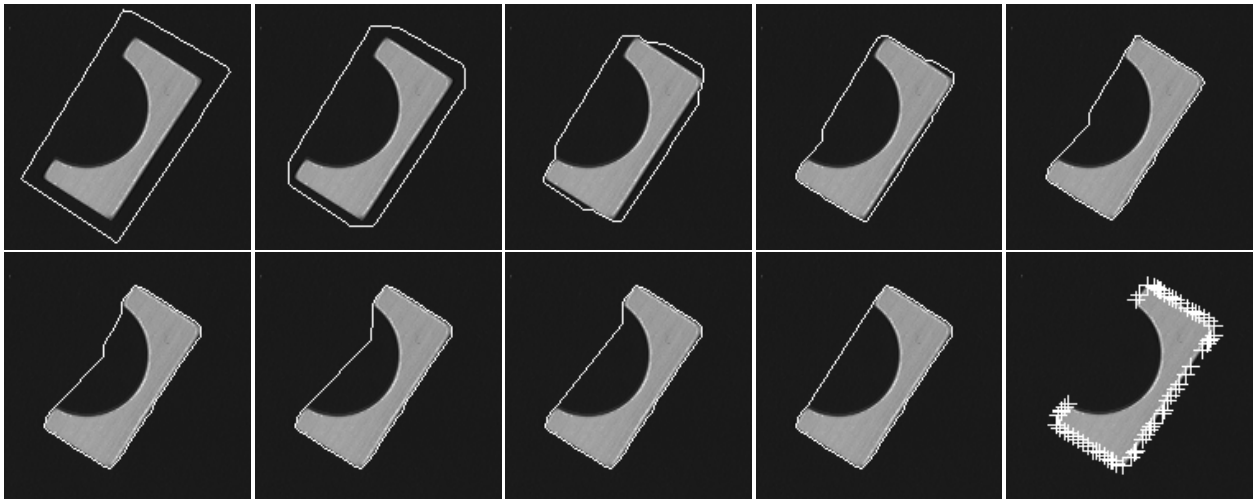


Figure 4: Approximation of the snake polygon to an object by length reduction, every image after 10 iterations. Upper left: initial polygon with equidistantly distributed points. Lower right: end positions of the snake points.

Figure 5: Minimization of the included area, starting with a polygon similar to the end polygon in figure 4 but with equidistantly distributed points.

4 Applications

Using a simple camera arrangement, where the optical axis is directed perpendicular to the flat objects, it is easy to get a relation between the final snake points in the image and the real world coordinates by a simple affine transformation. Thus a robot arm could handle some tightly laying objects as a single unit, e.g. to weld them together (the received snake polygon would be the welding seam). Actually we implemented a system where the end-effector of a robot arm runs along the border of objects which are determined using the active contour approach.

5 Conclusion

So far our snake algorithm does not make use of gradient based information. However, a gradient based algorithm like the one presented in [3] could optimize the grade of approximation. Therefore such an algorithm could use our final points as initial points. Rather, our snake uses a threshold for the grey level jump to detect edges if they are not too smooth. This strategy is useful if the initial contour is far away from the object border. Until now the initial snake points are set via mouse clicks. For an automatical setting several approaches are conceivable, e.g. learning a point distribution model (see [4]) or decomposing an image into objects and background.

References

[1] M. Kass, A. Witkin, D. Terzopoulos, Snakes: Active Contour Models, *International Journal of Computer Vision*, pp. 321-331, 1988.

[2] D.J. Williams, M. Shah, A Fast Algorithm for Active Contours and Curvature Estimation, *Image Understanding*, Vol. 55. No 1. January, pp. 14-26 1992.

[3] D. Young, Active Contour Models (Snakes), March 1995, *http://www.cogs.susx.ac.uk/users/davidy/teachvision/vision7.html.*

[4] T.F. Cootes, C.J. Taylor, D. H. Cooper, J. Graham, Active Shape Models - Their Training and Application, *Computer Vision and Image Understanding*, Vol. 61, No 1, January, pp. 38-59, 1995.