Real-time Tracking Using Level Sets

Yonggang Shi, W. Clem Karl [*]
Electrical and Computer Engineering Department
Boston University
Boston, MA 02215
email:{yshi,wckarl}@bu.edu

Abstract

In this paper we propose a novel implementation of the level set method that achieves real-time level-set-based video tracking. In our fast algorithm, the evolution of the curve is realized by simple operations such as switching elements between two linked lists and there is no need to solve any partial differential equations. Furthermore, a novel procedure based on Gaussian filtering is introduced to incorporate boundary smoothness regularization. By replacing the standard curve length penalty with this new smoothing procedure, further speedups are obtained. Another advantage of our fast algorithm is that the topology of the curves can be controlled easily. For the tracking of multiple objects, we extend our fast algorithm to maintain the desired topology for multiple object boundaries based on ideas from discrete topology. With our fast algorithm, a real-time system has been implemented on a standard PC and only a small fraction of the CPU power is used for tracking. Results from standard test sequences and our real-time system are presented.

1. Introduction

Real-time tracking of object boundaries is an important task in many vision applications such as video surveillance, video conferencing, and human-computer interaction, etc. Parametric contours have been applied successfully to achieve real-time performance[12, 7, 6, 19], but they have difficulties in handling topological changes such as the merging and splitting of object regions. For this purpose, the level set method[15] is a more powerful technique and various models have been proposed[2, 16, 4, 14, 9, 21]. But the high computational cost of the level set method has limited its popularity in real-time scenarios. In this paper, we propose a novel approach to implement the level set method. Our approach does not need to solve any partial differential equations (PDEs), thus reducing the computation dramatically compared with optimized narrow band techniques proposed before. With our approach, *real-time level-set-based* video tracking can be achieved.

Tracking models using level sets can be classified as edge-based or region-based. In [16], a geodesic model that combines motion and edge information was proposed. Such edge-based models can be traced back to the snake model in [13]. Based on morphing images, an early work on region-based tracking was proposed in [2]. Using the difference between the current frame and a reference background, a region-based model was proposed in [4]. By generalizing the region competition[22] idea, statistical approaches become quite popular recently. In [14, 21], the feature distributions of both the object and background regions were used for tracking. In [9], a predefined distribution for the object region was tracked by minimizing a Kullback-Leibler or Bhattacharyya distance.

Considerable work has been done to improve the speed of level-set-based curve evolution. The basic idea has been to limit the solution of the level set PDE to a narrow band around the zero level set[1, 17, 20, 16], but issues such as narrow band construction, reinitialization and step size control still make existing level set methods computationally too expensive for real-time tracking.

The novel implementation we propose in this paper avoids the above problems. Our approach is based on the key observation that implicitly represented curves can be moved by simply switching elements between two linked lists. We also propose a Gaussian filtering process to replace curvature-based smoothing, and this further speeds up our algorithm. Another advantage of our method is that ideas for controlling topological changes in [11] can be incorporated into our algorithm easily, which can be important for the tracking of multiple objects and medical imaging applications. With our fast level set implementation, we have

[*]This work was partially supported by The Air Force Office of Scientific Research under Grant F49620-03-1-0257, The National Institutes of Health under Grant NINDS 1 R01 NS34189, and The Engineering research centers program of the NSF under award EEC-9986821

developed a region-based real-time video tracking system requiring only a small fraction of the CPU power on a standard PC, leaving computational power available for other tasks, such as recognition, trajectory analysis, etc.

The rest of the paper is organized as follows. In Section 2, we present the tracking model used in this paper. Our fast level set implementation for the two-region case is then presented in Section 3. For the case of multiple object regions, we extend our fast algorithm to incorporate topology control capabilities in Section 4. Real-time tracking results are presented in Section 5. Finally conclusions are made in Section 6.

2. Tracking Model

In this paper, we use a region-based tracking model based on the region competition[22] idea, but our fast level set implementation is also applicable for edge-based models.

We assume each scene of the video sequence is composed of a background region Ω_0 and M object regions $\Omega_1, \Omega_2, \cdots, \Omega_M$. The boundaries of these M object regions are denoted as $C_1, C_2, \cdots, C_M$. We model each region with a feature distribution $p(\underline{v}|\Omega_m)(m = 0, 1, \cdots, M)$, where $\underline{v}$ is the feature vector defined at each pixel. For example, the features can be the color, the output of a filter bank designed to model textures, or other visual cues. Assuming that the feature distribution at each pixel is independent, the tracking result of each frame is the minimum of the following region competition energy:

$$E = \underbrace{-\sum_{m=0}^{M} \int_{\Omega_m} \log p(\underline{v}(\underline{x})|\Omega_m) d\underline{x}}_{E_d} + \lambda \underbrace{\sum_{m=1}^{M} \int_{C_m} ds}_{E_s} \quad (1)$$

where E_d is the data fidelity term that represents the likelihood of the current scene, E_s is for smoothness regularization and is proportional to the length of all curves, and λ is the non-negative regularization parameter.

By computing the first variation of this energy, the curve evolution equation for the minimization of this energy is:

$$\frac{dC_m}{dt} = (F_d + F_s)\vec{N}_{C_m} \quad (m = 1, 2, \cdots, M) \qquad (2)$$

where $\vec{N}_{C_m}$ is the normal of C_m pointing outward, and F_d and F_s are the speed resulting from E_d and E_s, respectively. The speed F_d represents the competition between two regions and it is $F_d = \log[p(\underline{v}(\underline{x})|\Omega_m)/p(\underline{v}(\underline{x})|\Omega_{out})]$, where Ω_{out} denotes the region outside C_m at $\underline{x} \in C_m$. The speed F_s makes the curve smooth and it is $F_s = \lambda\kappa$, where κ is the curvature.

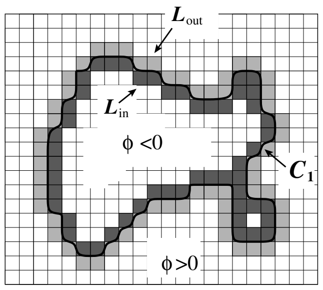

Figure 1. The implicit representation of the curve C_1 and the two lists L_{in} and L_{out} in the neighborhood of C_1.

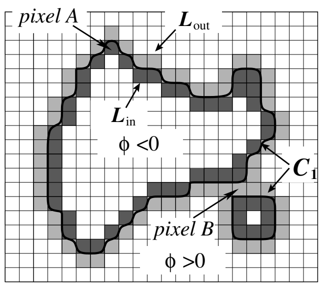

Figure 2. Illustration of the motion of the curve C_1 by switching pixels between L_{in} and L_{out}.

Since the focus of this paper is on real-time level set implementation for video tracking, we use a simple tracking strategy as follows. For each frame, we use the tracking results from the last frame as the initial curves, and then evolve each curve according to (2) to locate the object boundaries in the current frame. Once it stops, we move on to track the next frame of the video sequence.

3. Fast Level Set Implementation

In this section, we present a fast level set implementation of the curve evolution process in (2) when the scene is composed of only the background Ω_0 and a single object region Ω_1. Extensions are then made to track multiple objects with different feature distributions in the next section.

To represent the background Ω_0 and the object region Ω_1, we use a level set function ϕ which is negative in Ω_1 and positive in Ω_0. Based on this representation, we define two lists of neighboring pixels L_{in} and L_{out} of C_1 as shown in Fig. 1. Formally, they are defined as:

$$L_{out} = \{\underline{x}|\phi(\underline{x}) > 0 \; and \; \exists \underline{y} \in N_4(\underline{x}) \text{ such that } \phi(\underline{y}) < 0\},$$
$$L_{in} = \{\underline{x}|\phi(\underline{x}) < 0 \; and \; \exists \underline{y} \in N_4(\underline{x}) \text{ such that } \phi(\underline{y}) > 0\}$$

where $N_4(\underline{x})$ is a 4-connected discrete neighborhood of a pixel $\underline{x}$ with $\underline{x}$ itself removed.

In conventional implementations of the level set method, an evolution PDE is solved either globally on the whole domain or locally in a narrow band to evolve the curve according to (2). Our fast level set implementation is based on the key observation that the implicitly represented curve C_1 can be evolved at pixel resolution by simply switching the neighboring pixels between the two lists L_{in} and L_{out}. For example, as we show in Fig.2, the curve C_1 moves outward at pixel A and shrinks and splits at pixel B compared with the curve shown in Fig.1. This motion can be realized by simply switching pixel A from L_{out} to L_{in}, and pixel B from L_{in} to L_{out}. By doing this for all the pixels in L_{in} and L_{out}, the curve can be moved inward or outward for one pixel everywhere in one scan. Since the curve is still represented implicitly, topological changes can be handled automatically. With this idea, we can achieve level-set-based curve evolution at pixel resolution and this is usually enough for many imaging applications. Next we present the details of our fast algorithm.

3.1 Basic Algorithm

The data structure used in our implementation is as follows:

- An array for the level set function ϕ;
- An array for the evolution speed F;
- Two bi-directionally linked lists of neighboring pixels: L_{in} and L_{out}.

Besides the inside and outside neighboring pixels contained in L_{in} and L_{out}, we call those pixels inside C_1 but not in L_{in} as *interior pixels* and those pixels outside C_1 but not in L_{out} as *exterior pixels*. For faster computation, we define ϕ as follows:

$$\phi(\underline{x}) = \begin{cases} 3 & \text{if } \underline{x} \text{ is an exterior pixel,} \\ 1 & \text{if } \underline{x} \in L_{out}, \\ -1 & \text{if } \underline{x} \in L_{in}, \\ -3 & \text{if } \underline{x} \text{ is an interior pixel.} \end{cases} \tag{3}$$

To switch pixels between L_{in} and L_{out}, we define two basic procedures on our data structure.

The procedure $switch\_in()$ for a pixel $\underline{x} \in L_{out}$ moves the curve outward one pixel at $\underline{x}$ by switching it from L_{out} to L_{in} and adding all its neighboring exterior pixels to L_{out}. Formally this procedure is defined as follows:

$switch\_in(\underline{x})$:

- Step 1: Delete $\underline{x}$ from L_{out} and add it to L_{in}. Set $\phi(\underline{x}) = -1$.
- Step 2: $\forall \underline{y} \in N_4(\underline{x})$ satisfying $\phi(\underline{y}) = 3$, add $\underline{y}$ to L_{out}, and set $\phi(\underline{y}) = 1$.

Similarly, the $switch\_out()$ procedure that moves the curve inward one pixel at $\underline{x} \in L_{in}$ is defined as follows:

$switch\_out(\underline{x})$:

- Step 1: Delete $\underline{x}$ from L_{in} and add it to L_{out}. Set $\phi(\underline{x}) = 1$.
- Step 2: $\forall \underline{y} \in N_4(\underline{x})$ satisfying $\phi(\underline{y}) = -3$, add $\underline{y}$ to L_{in}, and set $\phi(\underline{y}) = -1$.

To track the object boundary, we compute the speed at all pixels in L_{out} and L_{in} and store their sign in the array F. We first scan through the list L_{out} and apply a $switch\_in()$ procedure at a pixel if $F = +1$. After this scan, some of the pixels in L_{in} become interior pixels and they are deleted. We then scan through the list L_{in} and apply a $switch\_out()$ procedure for a pixel with $F = -1$. Similarly, exterior pixels in L_{out} are deleted after this scan. At the end of this iteration, a stopping condition is checked. If it is satisfied, we stop the evolution; otherwise, we continue this iterative process. In our implementation, the following stopping condition is used:

Stopping Condition. *The curve evolution algorithm stops if either of the following conditions is satisfied:*

(a) The speed at each neighboring pixel satisfies:

$$\begin{aligned} F(\underline{x}) &\leq 0 \ \ \forall \underline{x} \in L_{out}; \\ F(\underline{x}) &\geq 0 \ \ \forall \underline{x} \in L_{in}. \end{aligned} \tag{4}$$

(b) A pre-specified maximum number of iterations is reached.

The condition in (4) is very intuitive in the sense of region competition. When the curve is on the object boundary, all the pixels in L_{out} are in the background and all the pixels in L_{in} are in the object region. When (4) is satisfied, they disagree with each other on which direction to move the curve and convergence is reached. When the data is noisy or there is clutter, regularization is necessary and (4) may not be always satisfied in the final curve. Thus part (b) of the condition is also necessary to stop the evolution.

The above algorithm can be applied to arbitrary speed fields and speeds up the evolution process in (2) dramatically compared with previous narrow band techniques based on solving the level set PDE. For the curve evolution equation in (2), we can achieve a further speedup by introducing a novel scheme that separates the evolution driven by the data dependent speed F_d and the smoothing speed F_s into two different cycles. In spirit, this idea is similar to the work in [10] which proposed a fast method to implement the Chan-Vese model[5] over the whole domain, but the two-cycle algorithm we present next is still based on updating the two linked lists L_{in} and L_{out} to evolve the implicitly represented curve.

3.2 Two-cycle Algorithm

In the curve evolution equation in (2), the speed F_s for smoothness regularization is a function of the curvature, which is computationally quite expensive to evaluate generally. However when the level set function is chosen as the signed distance function, the curvature takes a simpler form and equals the Laplacian of ϕ. From the theory of scale-space[18], we know that evolution of a function according to its Laplacian is equivalent to Gaussian filtering the function. Motived by this observation, we propose to incorporate smoothness regularization using a Gaussian filtering process to further speed up our algorithm.

Let us denote an isotropic Gaussian filter of size $N_g \times N_g$ as G. To smooth the zero level set, we compute the response of ϕ to the Gaussian filter only at pixels in the two lists L_{in} and L_{out}. Due to the need of maintaining our data structure, we do not take the output of the Gaussian filter directly. Only when the sign of the output is different from the original value of ϕ at a pixel, we apply a $switch\_in()$ or $switch\_out()$ procedure to move the zero level set; otherwise, the original value of ϕ is kept.

To combine this smoothing process with the evolution of the data dependent speed F_d, we propose a two-cycle algorithm. In the first cycle of our algorithm, we evolve the curve according to F_d using the fast algorithm proposed in Section 3.1. In the second cycle, we apply the Gaussian filtering process to incorporate smoothness regularization. The details of this algorithm are listed in Table 1.

Two factors make this two-cycle algorithm faster than curvature-based regularization. When the noise is low, we can reduce the parameter N_g or increase the parameter N_a to reduce the percentage of computation allocated for smoothness regularization. The second reason is that we can implement the Gaussian filtering process with integer operations since we only care about the sign of its output.

To conclude, we will use the two-cycle algorithm in our level-set-based tracking. With this fast implementation, real-time tracking can be achieved.

4. Tracking Multiple Objects

In this section, we extend our fast algorithm to track multiple objects with different feature distributions.

For the representation of multiple object regions, we use two functions: one region indication function ψ and one level set function ϕ. This simple representation is motivated by the work in [8]. The region indication function is defined as follows:

$$\psi(\underline{x}) = m, \quad \text{if } \underline{x} \in \Omega_m (m = 0, 1, \cdots, M). \tag{5}$$

For the tracking of each frame, we assume the initial curves for all the object regions are separated by the background

Table 1. The Two-cycle Fast Algorithm

- Step 1: Initialize the array ϕ, F_d, the two lists L_{out} and L_{in}.
- Step 2(cycle one): For i=1:N_a do
  - Compute the speed F_d for pixels in L_{out} and L_{in};
  - For each pixel $\underline{x} \in L_{out}$, $switch\_in(\underline{x})$ if $F_d(\underline{x}) > 0$;
  - For each pixel $\underline{x} \in L_{in}$, if $\forall \underline{y} \in N(\underline{x}), \phi(\underline{y}) < 0$, delete $\underline{x}$ from L_{in}, and set $\phi(\underline{x}) = -3$.
  - For each pixel $\underline{x} \in L_{in}$, $switch\_out(\underline{x})$ if $F_d(\underline{x}) < 0$;
  - For each pixel $\underline{x} \in L_{out}$, if $\forall \underline{y} \in N(\underline{x}), \phi(\underline{y}) > 0$, delete $\underline{x}$ from L_{out}, and set $\phi(\underline{x}) = 3$.
  - Check the stopping condition. If it is satisfied, go to Step 3; otherwise continue this cycle.
- Step 3(cycle two): For i=1:N_g do
  - For every pixel $\underline{x}$ in L_{out}, compute $G \otimes \phi(\underline{x})$. If $G \otimes \phi(\underline{x}) < 0$, $switch\_in(\underline{x})$;
  - For each pixel $\underline{x} \in L_{in}$, if $\forall \underline{y} \in N(\underline{x}), \phi(\underline{y}) < 0$, delete $\underline{x}$ from L_{in}, and set $\phi(\underline{x}) = -3$.
  - For every pixel $\underline{x}$ in L_{in}, compute $G \otimes \phi(\underline{x})$. If $G \otimes \phi(\underline{x}) > 0$, $switch\_out(\underline{x})$.
  - For each pixel $\underline{x} \in L_{out}$, if $\forall \underline{y} \in N(\underline{x}), \phi(\underline{y}) > 0$, delete $\underline{x}$ from L_{out}, and set $\phi(\underline{x}) = 3$.
- Step 4: If the stopping condition is satisfied in cycle one, terminate the algorithm; otherwise, go back to Step 2.

region, but arbitrary topology is allowed in the final tracking result. The level set function ϕ is negative inside all the object regions and positive in the background. For each object region $\Omega_m (1 \leq m \leq M)$, two lists of neighboring pixels L_{in}^m and L_{out}^m can be defined as we did for the case of single object region. Here the *interior pixels* are those pixels inside the object regions but not contained in any L_{in}^m, and *exterior pixels* are those pixels in the background but not in any L_{out}^m. Similar to the two region case, the level set function is defined as:

$$\phi(\underline{x}) = \begin{cases} 3 & \text{if } \underline{x} \text{ is an exterior pixel,} \\ 1 & \text{if } \underline{x} \in L_{out}^m \text{ for any } 1 \leq m \leq M, \\ -1 & \text{if } \underline{x} \in L_{in}^m \text{ for any } 1 \leq m \leq M, \\ -3 & \text{if } \underline{x} \text{ is an interior pixel.} \end{cases} \tag{6}$$

To evolve the M curves $C_1, C_2, \cdots, C_M$ and keep track of all the regions, we evolve ϕ and ψ simultaneously while keeping all the object regions from merging with each other. This kind of topology control requirements can be realized easily in our fast algorithms by incorporating ideas in discrete topology[3, 11]. In the following, we introduce related concepts in the context of video tracking. For more detailed discussion about topological numbers and discrete topology, see [3].

Let $N_8(\underline{x})$ denote the 3×3 neighborhood centered at a pixel $\underline{x}$ but with $\underline{x}$ removed, and Ω_{obj} denote the union of all the object regions $\Omega_m (1 \leq m \leq M)$. The topological number of $\underline{x}$ with respect to Ω_{obj} is the number of 4-connected components in the set $\Omega_{obj} \bigcap N_8(\underline{x})$ and we denote it as $T_{obj}(\underline{x})$. The topological number of $\underline{x}$ with respect to the background region Ω_0 is the number of 8-connected components in the set $\Omega_0 \bigcap N_8(\underline{x})$ and we denote it as $T_{bg}(\underline{x})$. According to [3], the pixel $\underline{x}$ is a *simple point* if both $T_{obj}(\underline{x}) = 1$ and $T_{bg}(\underline{x}) = 1$. For a simple point, its removal or addition to Ω_{obj} will not change the topology of Ω_{obj}.

Based on the idea of *topological numbers*, a topology preserving level set method was developed in [11] which does not permit any splitting or merging of the curve. Here our requirement is more challenging. We want to allow each region $\Omega_m (1 \leq m \leq M)$ to have more than one connected components and they can merge and split at will if they belong to the same object region, but we also want to prohibit the merging of two connected components if they are from different object regions. To achieve this goal, we propose the concept of *the relaxed topological number* for a pixel $\underline{x} \in L_{out}^m (1 \leq m \leq M)$ as follows:

Definition. *For a pixel $\underline{x} \in L_{out}^m (1 \leq m \leq M)$, if the number of object regions that intersect with $N_8(\underline{x})$ is $\alpha(\underline{x})$, its topological number with respect to Ω_{obj} is $T_{obj}(\underline{x})$, and its topological number with respect to Ω_0 is $T_{bg}(\underline{x})$, then* **the relaxed topological number** *of $\underline{x}$ is defined as:*

$$T_r(\underline{x}) = \min(\alpha(\underline{x}), \max(T_{obj}(\underline{x}), T_{bg}(\underline{x}))).$$

If $\underline{x}$ is a simple point, then $T_r(\underline{x}) = 1$ and its addition to an object region will not cause two different object regions to merge. If $\underline{x}$ is not a simple point but $\alpha(\underline{x}) = 1$, we still have $T_r(\underline{x}) = 1$. This means that the addition of $\underline{x}$ to an object region will cause the merge of connected components from the same object region. For all the other cases, we have $T_r(\underline{x}) > 1$, thus the relaxed topological number at a pixel can be used to prevent the merging of different object regions while allowing flexible topological changes within each object region. Based on this idea, we modify the $switch\_in()$ and $switch\_out()$ procedure to update ϕ and ψ to track the M object regions as we evolve the zero level set. The modified procedures are listed as follows:

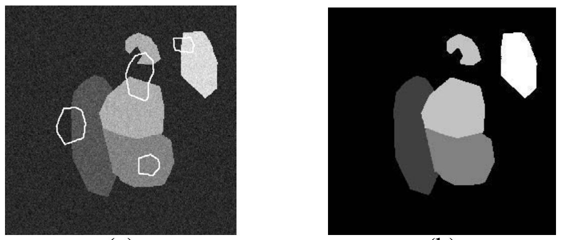
(a) (b)

Figure 3. (a) The original image and the initial curves(in white). One initial curve is used for each object region. (b) The segmentation result is shown in the region indication function.

$switch\_in(\underline{x})$ for $\underline{x} \in L_{out}^m (1 \leq m \leq M)$:

- Step 1: Compute $T_r(\underline{x})$. If $T_r(\underline{x}) = 1$, continue to step 2; otherwise, exit the procedure.
- Step 2: Delete $\underline{x}$ from L_{out}^m and add it to L_{in}^m. Set $\phi(\underline{x}) = -1$ and $\psi(\underline{x}) = m$.
- Step 3: $\forall \underline{y} \in N_4(\underline{x})$ satisfying $\phi(\underline{y}) = 3$, add $\underline{y}$ to L_{out}^m, and set $\phi(\underline{y}) = 1$.

$switch\_out(\underline{x})$ for $\underline{x} \in L_{in}^m (1 \leq m \leq M)$:

- Step 1: Delete $\underline{x}$ from L_{in}^m and add it to L_{out}^m. Set $\phi(\underline{x}) = 1$ and $\psi(\underline{x}) = 0$.
- Step 2: $\forall \underline{y} \in N_4(\underline{x})$ satisfying $\phi(\underline{y}) = -3$, add $\underline{y}$ to L_{in}^m, and set $\phi(\underline{y}) = -1$.

Similar to the case of a single object region, we use a two-cycle algorithm to track multiple object regions. Compared with the algorithm used to track one object, we need to scan through $2M$ lists L_{in}^m and $L_{out}^m (m = 1, 2, \cdots, M)$ in each iteration of cycle one and two to track M objects. After the two-cycle algorithm finishes, we perform a likelihood test for each pixel in $L_{out}^m (1 \leq m \leq M)$ with $T_r(\underline{x}) > 1$ and set its value in the region indication function as:

$$\psi(\underline{x}) = \underset{m \in S_M}{\text{argmax}}\, p(\underline{v}(\underline{x}) | \Omega_m) \tag{7}$$

where

$$S_M = \{1 \leq m \leq M | \Omega_m \bigcap N_4(\underline{x}) \neq \emptyset\}.$$

Here $p(\cdot | \Omega_m)$ is the feature distribution for each object region used in (1). By performing this likelihood test, we can assign pixels enforced as background before to proper object regions and enable the final tracking result to have arbitrary topology, such as multiple object regions connecting with each other. With all the regions labeled, the final

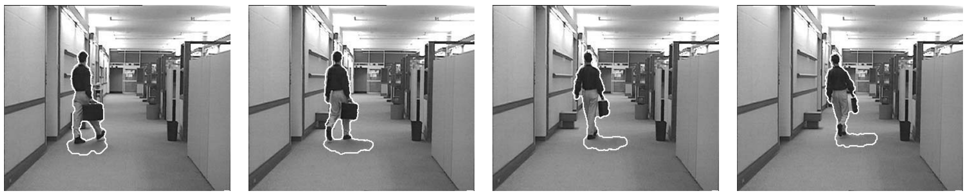
Figure 4. Tracking results of the *Hall monitor* sequence. Frame 41,49,57,65 are shown.

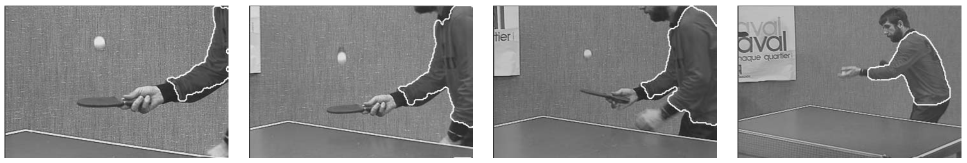
Figure 5. Tracking results of the *Tennis* sequence. Frame 1,38,45,65 are shown.

tracking result is saved in ψ for the current frame and we move on to the next frame.

In Fig. 3, we illustrate an image segmentation example using our algorithm. The noisy original image shown in Fig. 3(a) is composed of the background region and four object regions. We start with four initial curves to localize the four object regions of different intensities. With our method, successful segmentation is obtained and we show the region indication function in Fig. 3(b). Note that only one of the curve is able to split into two parts to localize two connected components that belong to the same object region. This example demonstrates the power of our method to represent and localize multiple regions with the desired topology.

5. Results

In this section, we present some real-time tracking results using our fast level set implementation. All the results presented in this section were run on a 1.7GHz PC.

5.1 Results of Test Sequences

We first present tracking results from three standard test sequences.

In the first example, we track a person in the *Hall monitor* sequence in CIF format from frame 41 to 65. The first frame of this sequence is used as a reference frame. The feature $\underline{v}$ used here is the magnitude of the intensity difference between the current frame and the reference frame. A Gaussian distribution is assumed for both the object and background region. As we can see from the tracking results shown in Fig. 4, we successfully tracked the person and his shadow, and topological changes of the curves are handled automatically as he walks in the hallway. For this sequence, the average tracking time is 0.0069s per frame.

In the second example, we show the tracking results of 65 frames from the *Tennis* sequence in SIF format. The Y and V components of the YUV color at each pixel are used as the feature. The feature distribution of the object and background region is modeled with a 32×32 histogram. Using the initial curve, we learn the histogram for each region in the first frame of the video sequence. As shown in Fig. 5, we successfully tracked the player as he moves into the scene. The average tracking time for each frame is 0.005s.

In the third example, we track the face of the character in the *Carphone* sequence in QCIF format for 40 frames. The hue and saturation components of the color in the HSV space at each pixel are used as the feature. The transformation of color space from YUV to HSV is performed online. Given the initial curve in the first frame, we learn the distribution of the feature for both the face and the background region and store the results in a histogram of size 32×32. Even though the head of the character moves dramatically in a cluttered background in this sequence, we are still able to track the face successfully as shown in Fig. 6. The average tracking time for this sequence is 0.0066s per frame.

From the results of these sequences, we can see that our level set implementation is able to track objects in a video sequence at a rate much faster than the real-time requirement. This makes our algorithm ready to be incorporated into real-time contour tracking systems where only a small fraction of the CPU time can be allocated to the tracking algorithm.

5.2 Results from Our Real-time Tracking System

As a demonstration, we have implemented a real-time tracking system which includes video capturing, tracking, motion analysis and the displaying of tracking results. This system runs at 24 frames per second on a 1.7GHz PC.

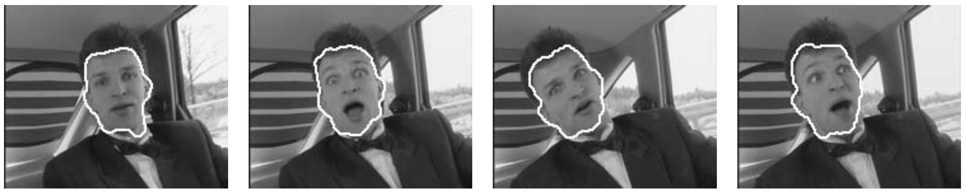

Figure 6. Tracking results of the *Carphone* sequence. Frame 1,20,30,40 are shown.

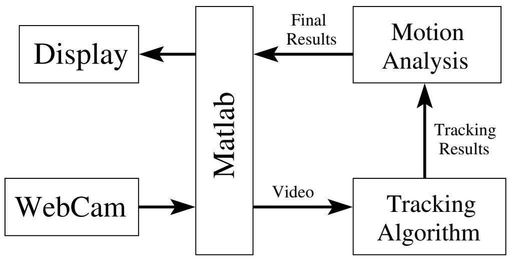

Figure 7. The block diagram of our tracking system.

A block diagram of our system is shown in Fig. 7. We use the image acquisition toolbox of Matlab to capture video sequences from a WebCam. The captured frame is sent to the tracking algorithm implemented in C++. Tracking results from this module are forwarded to an optional motion analysis algorithm once it is available. After that, the final results are sent back to Matlab for displaying. Then the next frame is captured from the WebCam to continue the tracking process. Currently we use color as the feature for tracking.

We show two examples from our tracking system. For both examples, the size of each frame is 352×288. In the first example, we track the motion of two hands as shown in Fig. 8. This sequence includes various topological changes, such as the merging of curves, the creation of holes, and the splitting of curves. With this example, we demonstrate that complicated topological changes can be tracked in real-time with our system.

In the second example, we track two objects of different color distributions using the algorithm for the tracking of multiple objects developed in Section 4. In the motion analysis module, we implemented a simple collision detection algorithm by looking for pixels with $T_r(\underline{x}) > 1$. As we can see from the results shown in Fig. 9, when the two objects collide, the boundary between these two regions is labeled in white color. By tracking this interface, we can detect and follow the collision process. Meanwhile the deformation and movement of the two objects are tracked successfully in real-time.

6. Conclusion

In this paper, we have proposed a fast level set implementation without the need of solving any PDEs. Real-time results have been achieved for the tracking of both single and multiple objects. With our approach, a real-time video tracking system has been implemented on a standard PC.

References

[1] D. Adalsteinsson and J. Sethian, “A fast level set method for propagating interfaces,” *Journal of Computational Physics*, vol. 118, pp. 269–277, 1995.

[2] M. Bertalmio, G. Sapiro, and G. Randall, “Morphing active contours: A geometric approach to topology-independent image segmentation and tracking,” *Proc. ICIP*, vol. III, pp. 318–322, 1998.

[3] G. Bertrand, “Simple points, topological numbers and geodesic neighborhoods in cubic grids,” *Pattern Recognition Letters*, vol. 15, pp. 1003–1011, 1994.

[4] S. Besson, M. Barlaud, and G. Aubert, “Detection and tracking of moving objects using a new level set based method,” *Proc. ICPR*, vol. 3, pp. 1100 – 1105, Sept 2000.

[5] T. Chan and L. Vese, “Active contours without edges,” *IEEE Trans. Imag. Process.*, vol. 10, no. 2, pp. 266–277, Feb. 2001.

[6] Y. Chen, Y. Rui, and T. S. Huang, “JPDAF based HMM for real-time contour tracking,” *Proc. CVPR*, vol. 1, pp. 543–550, 2001.

[7] D. Comaniciu, V. Ramesh, and P. Meer, “Kernel-based object tracking,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 25, no. 5, pp. 564–577, May 2003.

[8] H. Feng, D. Castañón, and W. Karl, “A curve evolution approach for image segmentation using adaptive flows,” in *Proc. ICCV*, 2001, pp. 494–499.

[9] D. Freedman and T. Zhang, “Acitve contours for tracking distributions,” *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 518–526, Apr 2004.

[10] F. Gibou and R. Fedkiw, “A fast hybrid k-means level set algorithm for segmentation,” in *Proceedings of the 4th Annual Hawaii Int’l Conf. Statistics and Mathematics*, Jan. 2005, pp. 281–291.

[11] X. Han, C. Xu, and J. Prince, “A topology preserving level set method for geometric deformable models,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 25, no. 6, pp. 755–768, Jun 2003.

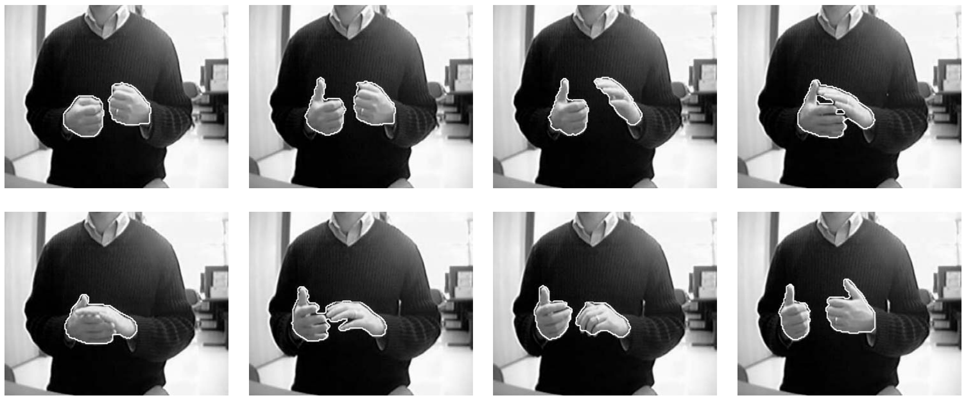
Figure 8. The results of tracking hands using our system. Frame 1, 40, 60, 81,95, 130, 150,and 200 of a sequence are shown.

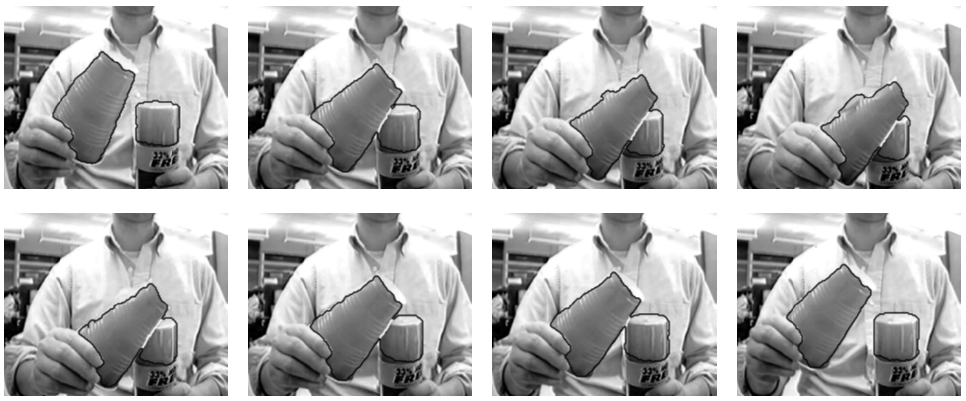
Figure 9. The results of tracking the collision of two objects with different color distribution using our system. Frame 41, 44, 52, 60, 70, 76, 78, and 80 of a sequence are shown.

[12] M. Isard and A. Blake, "ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework," *Proc. ECCV*, pp. 767–781, 1998.

[13] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes:active contour models," *International Journal of Computer Vision*, pp. 321–331, 1988.

[14] A. Mansouri, "Region tracking via level set PDEs without motion computation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, no. 7, pp. 947–961, Jul 2002.

[15] S. Osher and J. Sethian, "Fronts propagation with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations," *Journal of computational physics*, vol. 79, pp. 12–49, 1988.

[16] N. Paragios and R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 266–280, Mar 2000.

[17] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics*, vol. 155, pp. 410–438, 1999.

[18] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 7, Jul 1990.

[19] F. Precioso, M. Barlaud, T. Blu, and M. Unser, "Smoothing B-spline active contour for fast and robust image and video segmentation," *Proc. ICIP*, Sept.

[20] R. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int'l Journal of Computer Vision*, vol. 29, no. 3, pp. 203–231, OCT 1998.

[21] A. Yilmaz, X. Li, and M. Shah, "Contour-based object tracking with occlusion handling in video acquired using mobile cameras," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 26, no. 11, pp. 1531–1536, NOv 2004.

[22] S. Zhu and A. Yuille, "Region competition: unifying snake/balloon, region growing, and bayes/mdl/energy for multi-band image segmentation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, no. 9, pp. 884–900, Sept 1996.