수정된 캐니 에지 맵으로부터 만들어진 LOD 에지 맵을 이용한 물체 추적 및 소거

박 지 헌[†] · 장 영 대[††] · 이 동 훈[††] · 이 종 관[††] · 함 미 옥[†††]

요 약

본 논문은 하나의 움직이는 카메라와 수시로 바뀌는 배경을 가진 환경 하에서 파라미터를 사용하지 않는 외곽선을 사용한 움직이는 물체의 외곽을 추적하고, 추적된 물체의 외곽을 다른 장면에서 가져온 배경으로 대체하여 추적물체를 제거하는 기법을 제안한다. 먼저 캐니 에지 이미지(map)를 수정하여 만들어 내고, 이들 에지들의 강도에 대한 정보를 LOD (Level-of-Detail) 로 만든 결과 LOD 캐니 에지 이미지(map)을 생성한다. 이들 LOD 캐니 에지 이미지 화소에 대해 그래프를 사용한 경로 설정 방법을 사용한다. 이 작업으로 결정되는 외곽선을 이용하여 추적 대상이 되는 물체를 다른 이미지에서부터 얻은 배경이미지로 대체함으로써 제거한다. 우리의 물체 추적을 위한 방법은 LOD 수정된 캐니에지 이미지를 위주로 이루어진다., 추가 에지 정보를 얻기 위해 LOD 계층에 따라서 자세한 외곽선 정보를 얻는다. 우리의 경로 설정 방법은 보다 강한 이미지 차에서 만들어진 에지 화소를 선호하는 것이다. 이 방법은 이전 외곽선 정보를 최소한으로 참고하기 때문에, 이전 외곽선 정보를 새로운 외곽선을 생성하는데 있어서 가중치를 사용 이전 외곽선을 포함시키는 방법에 비해 탁월하다. 외곽선 추적 후 추적 물체를 배경으로 대체하는데, 첫 이미지 배경은 이후에 나타나는 이미지로부터 추적 물체에 대해 가려진 배경정보를 가져오는 카메라 운동법이라 부르는 방법에 의하여 계산되어진다. 첫 프레임을 위한 배경 계산이 완료되면, 다음 이미지의 배경 계산은 첫 프레임의 배경에 의존한다. 본 논문에서 제시된 방법을 사용할 경우, 추적 물체의 형상 변화가 극심하지 않고, 카메라의 움직임이 매우 빠르지 않을 경우 성공적으로 추적할 수 있었다.

키워드 : 물체 추적, LOD 캐니 에지 맵, 움직이는 카메라, 물체 제거

Object Tracking And Elimination Using Lod Edge Maps Generated from Modified Canny Edge Maps

Jihun Park[†] · Yung-Dae Jang[††] · Dong-Hun Lee[††] · Jong-Kwan Lee[††] · Miok Ham[†††]

ABSTRACT

We propose a simple method for tracking a nonparameterized subject contour in a single video stream with a moving camera and changing background. Then we present a method to eliminate the tracked contour object by replacing with the background scene we get from other frame. First we track the object using LOD (Level-of-Detail) canny edge maps, then we generate background of each image frame and replace the tracked object in a scene by a background image from other frame that is not occluded by the tracked object. Our tracking method is based on level-of-detail (LOD) modified Canny edge maps and graph-based routing operations on the LOD maps. We get more edge pixels along LOD hierarchy. Our accurate tracking is based on reducing effects from irrelevant edges by selecting the stronger edge pixels, thereby relying on the current frame edge pixel as much as possible. The first frame background scene is determined by camera motion, camera movement between two image frames, and other background scenes are computed from the previous background scenes. The computed background scenes are used to eliminate the tracked object from the scene. In order to remove the tracked object, we generate approximated background for the first frame. Background images for subsequent frames are based on the first frame background or previous frame images. This approach is based on computing camera motion. Our experimental results show that our method works nice for moderate camera movement with small object shape changes.

Key Words : Object Tracking, Level-Of-Detail Canny Edge Maps, Moving Camera, Object Elimination

1. Introductions and Related Works

The tracking of moving subjects is a hot issue because of a wide variety of applications in motion capturing for computer animation, video coding, video surveillance, monitoring, and augmented reality. We track a highly textured subject moving in a complex scene compared to a relatively simple subject tracking done by others. We mean *complex* because both tracked subject and back-

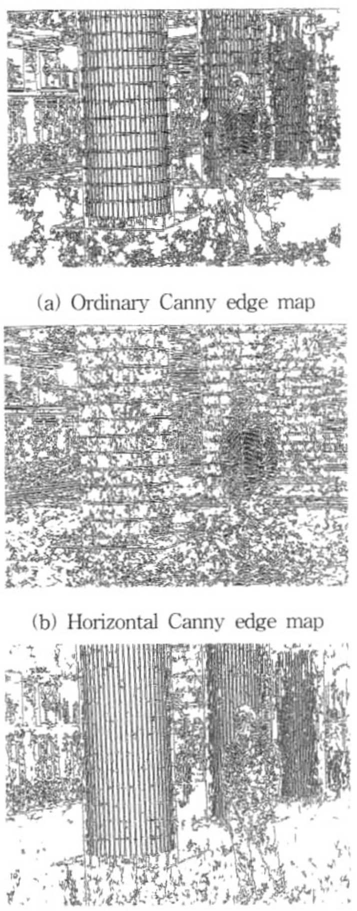
[†] 정 회 원 : 홍익대학교 컴퓨터공학과 교수(교신저자)
[††] 준 회 원 : 홍익대학교 컴퓨터공학과 석사과정
[†††] 준 회 원 : 홍익대학교 컴퓨터공학과 박사과정
논문접수 : 2006년 6월 23일, 심사완료 : 2007년 3월 26일

ground scene leave many edges after the edge detection. We assume our subject is never occluded by any background objects, but it occludes other objects in the background. Our background generation assumes all background objects are static. We can classify the methods of representing a subject contour into two categories depending on the method used; parameterized contour or nonparameterized contour. In tracking a parameterized contour, a subject contour estimating the motion is represented by using parameters. In general, these methods use the Snake model[1]; Kalman Snake[2] and Adaptive Motion Snake[3] are popular Snake models.

In the method of tracking a nonparameterized contour, a subject contour as a subject border is represented. The contour created by these algorithms is represented as a set of pixels. Paragios's algorithm[4] and Nguyen's algorithm[5] are popular in these approaches. Recently, Nguyen proposed a method[5] for tracking a non-parameterized subject contour in a single video stream with a moving camera and a changing background. Nguyen's approach combined the outputs of two steps: creating a predicted contour and removing background edges. Nguyen's background edge removal method of leaving many irrelevant edges is subject to inaccurate contour tracking in a *complex* scene because removing the background edges is difficult. Nguyen's method[5] of combining the predicted contour computed from the previous frame accumulates tracking error. In Nguyen's algorithm[5], a watershed line that was determined by using the watershed segmentation[6] and the watershed line smoothing energy[5,7] becomes the new contour of a tracked subject. Nguyen's approach removed background edges by computing subject motion. But Nguyen's approach left many irrelevant edges that prohibit accurate contour tracking because removing the background edges is difficult. The watershed line is generated by combining the previous frame contour and the current frame Canny edges that do not always make a closed edge contour. In this way, tracking errors are accumulated by always including the previous contour regardless of the intensity of the current Canny edges. Predicted contour that is computed from the previous frame is usually different from the exact contour for the current frame. A big change between the previous and current contour shapes makes this kind of contour tracking difficult. The non-parametric contour tracking research presented in this paper is improvement on our previous works of parametric contour tracking[8] and non-parametric contour tracking[9]. We replaced parametric contour tracking of our old work[8] to avoid tracking error accumulation. This paper has no big difference in object tracking of our old work[9] but we extend our research in tracked object elimination. This technique can be basic in editing a movie compared to a popular image editing. Our object elimination by background is basically similar to other works[10].

2. Our Approach

In this paper, we remove redundant edges by modifying Canny edge generation. To overcome Nguyen's two problems, difficulty in removing noisy background edges and accumulating tracking errors, we propose a new method to increase the subject tracking accuracy by using LOD Canny edge maps in predicted contour normal direction. We compute a predicted contour as Nguyen does. But, we use two major approaches. First, in order to reduce side-effects caused by irrelevant edges, we generate Canny edge maps around the predicted contour in the contour


(a) Ordinary Canny edge map

(b) Horizontal Canny edge map

(c) Vertical Canny edge map

(Fig. 1) Effect of computing Canny edge maps according to the contour direction normal direction. Second, we start our basic tracking contour using *simple (strong)* Canny edges generated from large image intensity gradients, called *Scanny* edges.

(Fig. 1) (a) shows an ordinary Canny edge map, and (Fig. 1) (b,c) show modified Canny edge maps generated assuming horizontal and vertical contour direction respectively. It is quite easy to modify the Canny edge generator by considering the computed predicted contour and computing the image intensity derivatives in the contour normal direction. As can be found from the figures, the contour direction effect on generating Canny edge maps is removing redundant edges generated in an ordinary Canny edge map.

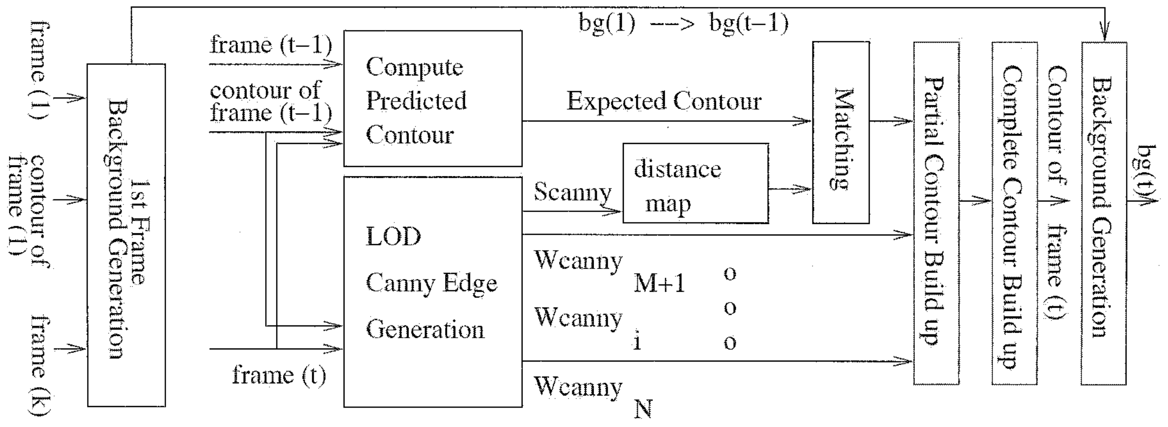
A *strong Canny edge map* is generated by a pixel-wise union of the simplest Canny edge maps out of various scaled Canny edge maps. Contrary to Nguyen's approach, we do not remove the background edges that are difficult to remove. Our new method selects only the Canny edges with large image intensity gradient values, *Scanny* edges. A *Scanny* edge map does not have noisy background edges and looks simple, meaning there are less edges in the Canny edge map of the scene. Working on Scanny has an effect of background removal. Our accurate tracking is based on reducing the effects from irrelevant edges by only selecting strongest edge pixels, and relying on the current frame edge pixels as much as possible contrary to Nguyen's approach of always combining the previous contour.

For Canny edge maps generated with smaller image intensity gradient values, we call $Wcanny_i, i = M+1,\cdots,N$ where N is the number of LOD Canny edge maps, M is the number of Canny edge maps used in computing *Scanny* edge map. $Wcanny_{M+1}$ has the simplest Canny edges generated from a set of large *(strongest)* intensity gradient value edges. $Wcanny_N$ has the most detailed Canny edges generated by an accumulation from largest *(strongest)* till to the smallest *(weakest)* intensity gradient valued edges. Basically, we rely only on a *Scanny* edge map and a predicted contour from the previous frame to find reference pixels, called *selected Scanny* pixels, for building a basic (but not closed) tracked contour frame. Then, we seek additional edge pixels from $Wcanny_i$s according to the descending sequence of multi-level detailed edge pixels, following LOD in edge maps. These *selected* Scanny pixels become start nodes and end nodes in routing. LOD Canny edge pixels become nodes in routing, and LOD values of adjacent edge pixels determine routing costs between the nodes. We mean *adjacent* to be four-neighbor connected. From a set of adjacent *selected* Scanny edge pixels, we find segments of contours, called *partial contour*. In finding a *partial contour*, we find the *best* route to follow Canny edge pixels favoring stronger Canny edge pixels.

We consider *Scanny* edges around a predicted contour, computed from the previous frame contour, to likely be a part of the new contour. To make a closed contour, we do a final routing using the above segments of *partial contours* and *Scanny* edges around the predicted contour. We do a routing between two disconnected *Scanny* edge pixels using LOD *Wcanny* edge maps favoring stronger edge maps. The disconnected contour is connected using Dijkstra's minimum cost routing.

3. Overview of Our System

(Fig. 2) shows an overview of our system for tracking and eliminating an object (to make a background image) in a single image frame. First, we generate the first frame background scene that will be explained in Section 3. Then we compute a tracked object contour for the next frame. As inputs to compute an object contour, we



(Fig. 2) Overview of our single frame tracking and background generation get a previous image frame, denoted as $frame\ (t-1)$ and the corresponding tracked subject contour of input $frame\ (t-1)$, and a current image frame, denoted as $frame\ (t-1)$. From $frame\ (t-1)$, contour of $frame\ (t-1)$, and $frame\ (t-1)$, we compute a predicted contour, $\partial\Omega^{(p,t)}$, for $frame\ (t)$ using subject motion[5]. Then, we generate various detailed levels of modified Canny edge image maps for the input $frame\ (t)$. We select *Scanny* edges from the LOD Canny edge maps. From a *Scanny* edge map, we derive a corresponding distance map. Using the predicted contour, the best matching is then found between the predicted contour and the *Scanny* distance map. *Scanny* edge pixels matching with the predicted contour become the frame of the contour build up. We call these pixels *selected Scanny contour pixels. Selected Scanny contour pixels*, generated using *Scanny* and predicted contour, are the most reliable (but not closed) contour pixels to start building a closed tracked contour, and are stored in the *selected Scanny found list.* We then route a path to connect adjacent *selected Scanny contour pixels* in the found list using LOD Canny edge pixels. If we finish connecting every adjacent *selected Scanny contour pixel* pair, we get a set of *partial contours* although not guaranteed to be the *best* closed contour. We mean *best* because the contour is four-neighbor connected and follows every possible *Scanny* edge. To build a *best* closed contour for the $frame\ (t)$, we use LOD Canny edge maps around the predicted contour. We run a final routing using the computed segments of partial contours and *Scanny* edges around it to find the best contour. In this process, we fix the incorrectly computed basic contour. The resulting contour becomes the contour of $frame\ (t)$, and it is used to generate background of $frame\ (t)$.

3. LOD Canny Edge Maps and Matching for Selecting Reference Contour Pixel

A *strong Canny edge map* is generated by a pixel-wise union of the simplest Canny edge maps out of various scaled Canny edge maps. Contrary to Nguyen's approach, we do not remove the background edges that are difficult to remove. Our new method selects only the Canny edges with large image intensity gradient values, *Scanny* edges. A *Scanny* edge map does not have noisy background edges and looks simple, meaning there are less edges in the Canny edge map of the scene. Working on *Scanny* has an effect of background removal. Our accurate tracking is based on reducing the effects from irrelevant edges by only selecting strongest edge pixels, and relying on the current frame edge pixels as much as possible contrary to Nguyen's approach of always combining the previous contour. For Canny edge maps generated with smaller image intensity gradient values, we call $Wcanny_i, i=M+1,\cdots,N$ where N is the number of LOD Canny edge maps, M is the number of Canny edge maps used in computing *Scanny* edge map. $Wcanny_{M+1}$ has the simplest Canny edges generated from a set of large *(strongest)* intensity gradient value edges. $Wcanny_N$ has the most detailed Canny edges generated by an accumulation from largest (*strongest*) till to the smallest (*weakest*) intensity gradient valued edges.

By varying control parameters, we can get various Canny edge maps of different scales given a single image. The resulting Canny edge maps are mainly affected by the image intensity changes between pixels. We take advantage of the fact that we can get various Canny edge maps by varying these control parameters. Usually, very detailed Canny edge maps confuses us in finding the exact outline, but simple Canny edge maps generated from large image intensity changes do not have enough detail to make a closed contour for the tracked subject. But simple Canny edge maps are very reliable because they are generated only if there are big intensity changes in the image. We need both simple and detailed Canny edge maps for the best subject tracking. Various detailed Canny edge maps are generated by varying the values of control parameters. We totally order the resulting Canny edge maps by counting the number of edge pixels in each edge map.

Let $\Phi_i^{(I,t)}$, where $i=1,\cdots,N$, be a totally ordered set of Canny edge maps of an input image $frame(t)$. The ordering is done by counting the number of edge pixels. $\Phi_1^{(I,t)}$ has the smallest number of edge pixels while $\Phi_N^{(I,t)}$ has the largest number of edge pixels. N is the total number of Canny edge maps generated for the input image. Then, we take the top 10 percent to 30 percent of the simple Canny edge maps and union into pixel-level to make a *Scanny* edge map, $S\Phi^{(I,t)}$. M is the total number of Canny edge maps used to make a $S\Phi^{(I,t)}$. The rest of the Canny edge maps are used to generate $Wcanny_i$, $W\Phi_i^{(I,t)}$.

$$S\Phi^{(I,t)} = \bigcup_{i=1}^{M} \Phi_i^{(I,t)} \qquad (1)$$

$$W\Phi_i^{(I,t)} = S\Phi^{(I,t)} \bigcup \left(\bigcup_{j=M+1}^{i} \Phi_j^{(I,t)} \right), \qquad i=(M+1),\cdots,N$$

where $\bigcup$ is pixel-wise union of bitmaps.

$Wcanny_{M+1}$ is a pixel-wise union of *Scanny* and the next detailed sets of Canny edge maps. $Wcanny_i$ is generated by unioning $Wcanny_{(i-1)}$ and the next detailed sets of Canny edge maps, etc. $Wcanny_N$ has the union of all levels of detail Canny edges generated by an accumulation from *highest-to-lowest* intensity gradient value edges. (Fig. 3) (a,b) shows an example of *Scanny* and $Wcanny_N$ Canny edge maps respectively.

LOD Canny edge map, $L\Phi^{(I,t)}$, is generated using $S\Phi^{(I,t)}$ and $W\Phi_i^{(I,t)}$s edge pixels around $\partial\Omega^{(p,t)}$. $\Gamma(L\Phi^{(I,t)}(x,y))$ is a function returning an LOD value given an edge pixel *(x,y)* of a LOD edge map, $L\Phi^{(I,t)}$. To build a $L\Phi^{(I,t)}$, we search $S\Phi^{(I,t)}$ and $W\Phi_i^{(I,t)}$s from the simplest edge map to the most detailed edge map.

- $\Gamma(L\Phi^{(I,t)}(x,y))=1$ if $S\Phi^{(I,t)}(x,y)$ is an edge pixel and around $\partial\Omega^{(p,t)}$.
- $\Gamma(L\Phi^{(I,t)}(x,y))=i+1$ if $W\Phi_i^{(I,t)}(x,y)$ is an edge pixel and around $\partial\Omega^{(p,t)}$ and $\Gamma(L\Phi^{(I,t)}(x,y))$ is not initialized any value where $i=M,\cdots,N-1$.
- $\Gamma(L\Phi^{(I,t)}(x,y))=255$ if *pixel(x,y)* does not belong to any edge pixel or is not around $\partial\Omega^{(p,t)}$.

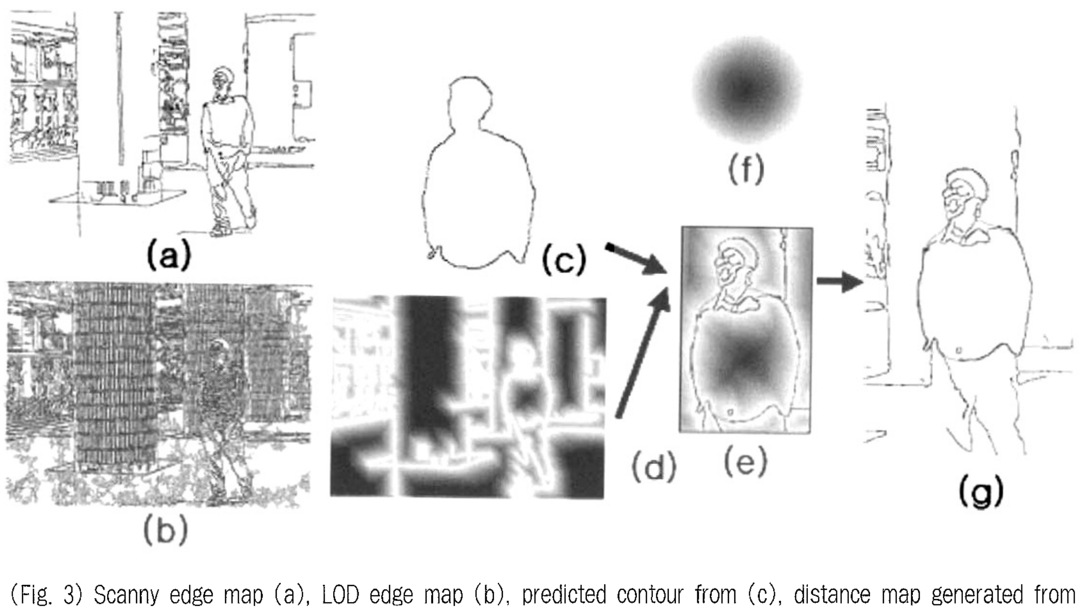
Basically, we rely only on a *Scanny* edge map and a predicted contour from the previous frame to find reference pixels, called *selected* Scanny pixels, for building a basic (but not closed) tracked contour frame. Then, we seek additional edge pixels from $Wcanny_i$s according to the descending sequence of multi-level detailed edge pixels, following LOD in edge maps. These *selected* Scanny pixels become start nodes and end nodes in routing. LOD Canny edge pixels become nodes in routing, and LOD values of adjacent edge pixels determine routing costs between the nodes.

Nguyen[5] removed background edges using *object motion*. But, Nguyen's approach left many irrelevant edges which prohibit accurate contour tracking.

We do not remove any background edges. Removing background edges is not easy. (Fig. 3) (a,b) shows an example of *Scanny* and $Wcanny_N$ Canny edge maps. Rather than removing background edges, we start with a *Scanny* edge map, as presented in (Fig. 3) (a), that has simple edges in a scene.

(Fig. 3) shows a process of computing *selected Scanny pixels*, and the selection result is presented in (Fig. 3) (e). *Selected Scanny pixels* are denoted as green pixels in (Fig. 3) (e), along the predicted contour, while red pixels mean a failure in finding a matching *Scanny* pixel. By using an image matching as used by others[5], we can get a predicted contour, $\partial\Omega^{(p,t)}$, as presented in (Fig. 3) (c). Then, we generate a distance map of *Scanny*, $DS\Phi^{(I,t)}$, as in (Fig. 3) (d).

Given a pixel (x_{ref}, y_{ref}) on $\partial\Omega^{(p,t)}$, we find the corresponding *Scanny* edge pixel, if one exists, by finding the *best* matching between the predicted contour and the dis-



(Fig. 3) Scanny edge map (a), LOD edge map (b), predicted contour from (c), distance map generated from Scanny (d), matching between predicted contour and Scanny distance map (e), circular distance map used in matching (f), final routing result favoring *Scanny* (g) tance map. In computing the matching, we use the *best* that we can find $(\Delta x, \Delta y)$ minimizing equation (2).

$$\sum_{y=0}^{SY-1}\sum_{x=0}^{SX-1} w(x_{ref},y_{ref})\left(DS\Phi^{(I,t)}(x,y)-\partial\Omega^{(p,t)}(x+\Delta x,y+\Delta y)\right)^2 \quad (2)$$

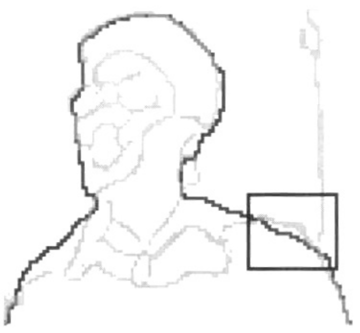
where $w(x_{ref}, y_{ref})$ is a weight function such as a circular distance map as presented in (Fig. 3) (f), *SX* and *SY* is the width and height of the input image. The center of the circular distance map is positioned at the reference pixel on the predicted contour. The $(\Delta x, \Delta y)$ minimizing equation (2) is denoted as $(\Delta x_{\min}, \Delta y_{\min})$. If the matching pixel, pixel$(x_{ref}+\Delta x_{\min}, y_{ref}+\Delta y_{\min})$, corresponds to a *Scanny* edge pixel, then the pixel $S\Phi^{(I,t)}(x,y)$ is selected. We call this pixel a *selected Scanny contour pixel.* Tracing along $\partial\Omega^{(p,t)}$, we get a set of *selected Scanny contour pixels.* These pixels are totally ordered in terms of $\partial\Omega^{(p,t)}$, and stored in the *found list.*

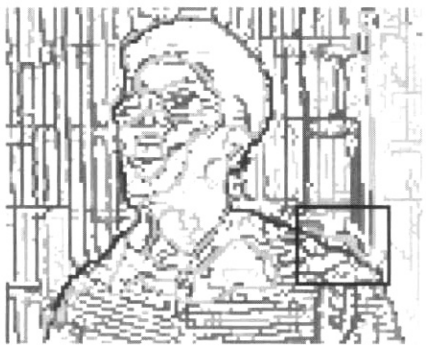
(Fig. 3) (e) shows an example of the best matching with the reference contour pixel point (marked as red cross). The green contour denotes the predicted contour, while black edge pixels denote *Scanny* edge pixels. Gray levels are shown because of a distance map of *Scanny* edge map. *Selected Scanny contour pixels* are the reference pixels to start building a segment of a tracked contour and are stored in the *selected Scanny found list.* These pixels are usually not connected as a four-neighbor connection but are most likely to become part of the new contour to be computed.

4. Scanny Contour Pixel Connection-Local Routing

We route a path to connect *adjacent selected Scanny contour pixels* in the *found list* using LOD Canny edge pixels, $LS\Phi^{(I,t)}$. We mean *adjacent* to be adjacent in the *found list.* If we finish connecting every adjacent *selected Scanny contour pixel* pairs, we get a set of *partial contours* although they are not guaranteed to be complete. We mean *complete* because the contour is four-neighbor connected and follows every possible *Scanny* edge. These *selected Scanny contour pixels* become start and end nodes in routing.

LOD Canny edge pixels become nodes in routing, and LOD values of adjacent edge pixels determine routing costs between the nodes. In finding a *partial contour,* we find the *best* route to follow Canny edge pixels favoring stronger Canny edge pixels. We mean *best* because


(a)


(b)

(Fig. 4) Close-up of selected Scanny pixels after matching between the predicted contour and the current frame Scanny edge map (a), selected Scanny pixels as well as accumulation from $Wcanny_{M+1}$ until $Wcanny_N$ edge pixels (b)

building an optimal partial contour route by taking possible strongest Canny edges (minimizing routing cost) according to the descending sequence of multi-level detailed edge pixels, following LOD in edge maps.

(Fig. 4) shows a close up of a matching result between the predicted contour and the current frame *Scanny* edge map. The green pixels were stored in the selected *Scanny* found list. We need to connect adjacent *selected Scanny contour pixels,* stored in the *found list,* to build a new closed contour. We mean adjacent, adjacent in the *found list.* This computed contour will be the *basic* tracked subject contour for *frame* (t).

Our *Wcanny* Canny edge tracing to find a route to connect *selected Scanny contour pixels* is done using the concept of LOD. The LOD Canny edge maps consist of various levels in Canny edge generation. The *Scanny* edge pixels are assigned LOD value one, LOD value *i+1* for $Wcanny_i$ edge pixels, and LOD value *(N+1)* for $Wcanny_N$ and so on. LOD value 255 is reserved for pixels with no edge, but number 255 does not have any special meaning. The LOD function is presented in Section 5.

We take a part of the LOD Canny edge map around two adjacent *selected Scanny contour pixels*. Pixels of the LOD map become nodes, and we determine costs between *adjacent* pixels. We mean *adjacent* to be four-neighbor connected. We determine costs between adjacent pixels using a Canny edge LOD value of each pixel. We favor traversing the most simple (stronger) edge pixels in the map rather than the most detailed (weaker) edge pixel in LOD. We assign the lowest cost between two adjacent *Scanny* edge pixels to encourage *Scanny*-based routing.

An LOD edge map I is a pair(Γ,Y) consisting of a finite set Γ of pixels, and a mapping Υ that assigns to each pixel t in Γ an LOD edge pixel value $\Upsilon(t)$ ranging from 1 to 255. An adjacency relation A is an irreflexive binary relation between pixels of Γ. The LOD edge map I can be interpreted as a directed graph with nodes that are the LOD edge pixels and with arcs that are the pixel pairs in A. sAt depends only on the four-connected neighbor of the pixels in the LOD map, and $(s,t)\in\Gamma\times Y$. A path is a sequence of pixels $\pi=<t_1,t_2,\cdots,t_k>$, where $(t_i,t_{i+1})\in A$ for $1\le i\le k-1$. t_1 is the origin, and t_k is the destination of the path. We assume given a function f that assigns to each path π a path cost $f(\pi)$, in some totally ordered set of cost values. The set of cost values contains a maximum element denoted by $+\infty$. The additive cost function satisfies

$$f_{\text{sum}}(\pi\cdot\langle s,t\rangle)=f_{\text{sum}}(\pi)+w(s,t)$$

where $(s,t)\in A$, π is any path ending at s, and $w(s,t)$ is a fixed nonnegative weight assigned to the arc (s,t).

$$w(s,t)=\begin{cases}+\infty & \text{if } s \text{ and } t \textit{ are} \text{ not adjacent pixels}\\ Y(\mathrm{s})*Y(\mathrm{t})*Y(\mathrm{t}) & \text{if s and } t \textit{ are adjacent} \text{ pixels}\end{cases}$$

This weight function guarantees to take stronger Canny edges in the optimum path routing. If there is no edge pixel present, the routing takes ordinary pixels with Υ value 255 to make a closed contour. The routing is done using Dijkstra's minimum cost routing algorithm. We route a path to connect each adjacent *selected Scanny contour pixels* pair in the *found list*. If we finish connecting all adjacent *selected Scanny contour pixels* pairs, we get a basic contour although it is not guaranteed to make a closed contour for the tracked subject.

5. Scanny Contour Pixel Connection-Final Routing

To build a closed and complete contour for the current frame, we use *Scanny* edge maps around the predicted contour as well as a set of *partial contours* computed from *selected Scanny edge pixels*. The resulting contour becomes the contour of the current frame. We run a final routing using the computed basic contour and *Scanny* edges around it to find the best contour. These pixels are usually not connected as a four-neighbor connection, but these pixels will most likely become part of the new contour to be computed. To get a *globally best* contour, we mean *best* that the contour is four-neighbor connected, closed, and follows every possible *Scanny* edges. We run a final routing using the computed basic contour and *Scanny* edges around the computed contour. We mean *global* considering the entire contour rather than considering a part of the edge map. The resulting contour becomes the contour of $frame\ (t)$. The major reason for considering only Scanny edge pixels excluding *Wcanny* pixels is because of computational complexity, $O(n^2)$. As the number of pixels involved in the final routing grows, the computation slows down. In computing the final contour, we consider *Scanny edge* pixels rather than all LOD edge pixels to reduce the number of nodes in the routing computation. For the final contour routing, Γ consists of *Scanny* pixels as well as the computed *partial contour* pixels, the pixels found from the routing between adjacent *selected Scanny contour pixels*, and that $\Upsilon(t)$ has dual values each for *Scanny* and the computed contour pixels. Υ values for *Scanny* edge pixels have value one, and the computed partial contour pixels have value two. The weight function for the final routing is as follows:

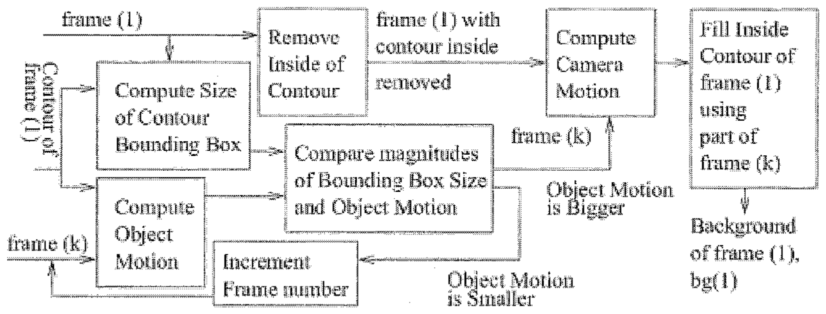
$$w(s,t)=\begin{cases}+\infty & \text{if s and } t \text{ are not adjacent } \pi\text{xels}\\ Y(\mathrm{s})*Y(\mathrm{t})*Y(\mathrm{t}) & \text{if s and t are adjacent and } Y(s)\neq Y(t)\\ 1 & \text{if s and } t \text{ are adjacent and } Y(s)=Y(t)\end{cases}$$

We assign cost one between adjacent *Scanny* pixels, while there are higher costs between pixels of the computed basic contour. This has an effect of favoring *Scanny* edges rather than computed contour pixels. If there is no route made by *Scanny* pixels for a special part of an edge map, then a corresponding segment of the computed partial contour is selected.

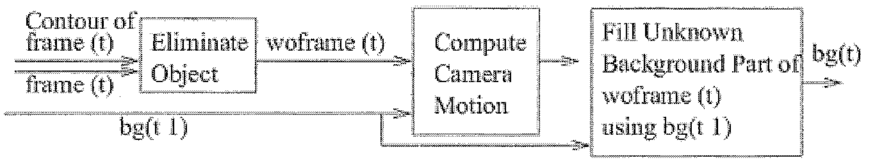
6. Object Elimination and Background Generation

(Fig. 5) shows a process to determine the first frame background given a sequence of video stream. As inputs, we get the *k*th frame denoted as *frame* (*k*), the first image frame denoted as *frame* (1) and the corresponding tracked subject contour of input *frame* (1). The first frame consists of the tracked object as well as background. The *k*th frame is the earliest frame, in video sequence, that has the background information for the part occluded by the tracked object in *frame* (1). From *frame* (1) and contour of *frame* (1), we compute a size of the bounding box of the tracked contour, and remove inside of the tracked object contour, the part of the image frame occluded by the tracked object. By filling the occluded/removed part using background image from *frame* (*k*), we build a background image of *frame* (1), denoted as *bg* (1). The process to determine the exact frame to fill occluded part is as follows. First we try with an arbitrary frame, say *frame* (*k*). In order to verify that the frame actually contains the missing background part of the first image frame, we compute object motion between *frame* (1) and *frame* (*k*)[5]. If the object motion magnitudes in both x and y direction are bigger than the width and height of the bounding box of the first frame respectively, we are done in finding the exact frame to fill the missing part of *frame* (1). Otherwise we try with the next image frame until the exact frame is found. Then we compute *camera motion* between the first frame and the *k*th frame, and the computation result is used in generating a background image of the first frame denoted as bg(1). *woframe* (1) denotes the first frame with the contour inside removed. In order to compute *camera motion*, we find the best matching displacement between *woframe* (1) and *frame* (*k*). In order to fill the occluded part of *woframe* (1), we use computed *camera motion* and take corresponding image part from *frame* (*k*). As a result, we get the background image of the first frame, denoted as *bg* (1).

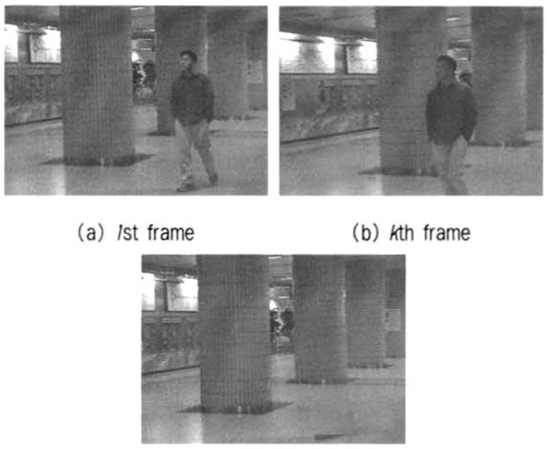
(Fig. 6) shows a process to determine the *t*th frame background. As inputs, we get a *t*th image frame, denoted as *frame* (*t*), the corresponding tracked subject contour of input *frame* (*t*), and the computed background image of *frame* ($t-1$), denoted as *bg* ($t-1$). Given *frame* (*t*) and the contour of *frame* (*t*), we eliminate inside the contour, the tracked object. The resulting image frame is denoted as *woframe* (*t*). Using *woframe* (*t*) and *bg* ($t-1$), we compute the *camera motion* between the *frame* ($t-1$) and the *frame* (*t*). Using the computed *camera motion*, we fill the occluded part of *woframe* (*t*) using *bg* ($t-1$). As a result, we get the background image for *frame* (*t*), denoted as *bg* (*t*). (Fig. 7) shows an example of generating the background image for the first frame. The inputs were a sequence of video, the first frame, and the contour for the full tracked body of the first frame. (Fig. 7) (a) shows the first frame, (Fig. 7) (b) is the selected *k*th frame which has the background image for the occluded part of the first frame, and (Fig. 7) (c) is the computed background image for the first frame. As you may find, there are some dark image areas that do not have any corresponding background image available.


frame (1)
Contour of frame (1)
Compute Size of Contour Bounding Box
Remove Inside of Contour
frame (1) with contour inside removed
Compute Camera Motion
Fill Inside Contour of frame (1) using part of frame (k)
frame (k)
Compare magnitudes of Bounding Box Size and Object Motion
Object Motion is Bigger
Compute Object Motion
frame (k)
Increment Frame number
Object Motion is Smaller
Background of frame (1), bg(1)

(Fig. 5) Process of generating the first frame background



(Fig. 6) Process of generating the *t*th frame background using *(t-1)*th frame background


(a) *l*st frame (b) *k*th frame

(c) background of *l*st

(Fig. 7) The result of generating background of the first frame

7. Experimental Results on Tracking while Handling Occlusion

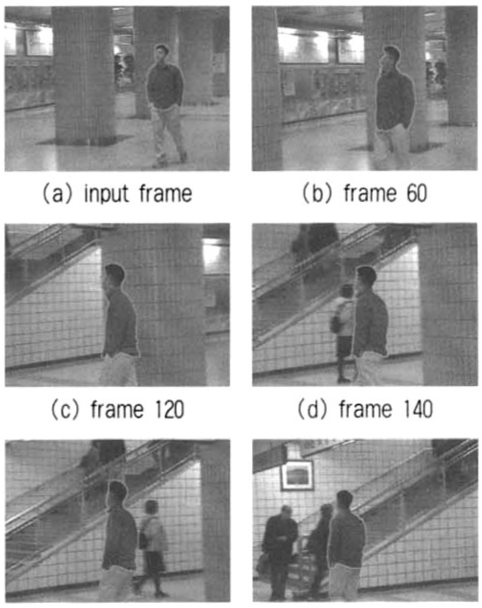
7.1 Experimental Environment

We have experimented with easily available video sequences either available on the Internet or generated with a home camcorder, SONY DCR-PC3. We have generated 64 different LOD Canny edge maps, ordered them according to the number of Canny edge pixels, and union simplest 18 (top 30 percent) Canny edge maps to make *Scanny* Canny edge map. It does not take a long computational time to generate 64 Canny edge maps, and it is not necessary to keep 64 different levels. We may vary the percentage of Canny edge maps in determining a *Scanny* edge map, but the percentage is not critical in tracking performance as far as we take 10 percent to 30 percent of the Canny edge maps. (Fig. 8[a-j]) show a man walking in a subway hall, and (Fig. 8[k-t]) are corresponding background generated images. The hall tiles as well as a cross stripe shirt generate many complicated Canny edges. The tracked contour shape and color changes as the man with a cross stripe shirt rotates from facing the front to the back as he comes closer to a camera and then moves away from it. To make tracking more difficult, the face color of the tacked subject is similar to the hall wall color (Fig. 8[c,e]) while his shirt color is similar to that of stairs (Fig. 8[i,j]), and tracked body black hair is interfered with by a walking woman in (Fig. 8(f,g)) and a man with a black suit in (Fig. 8(k-r)). Stair colors in (Fig. 8(j-s)) are similar to the tracked subject shirt color. We tracked the upper body of the man because his pants color is similar to that of the subway station floor (Fig. 8[a]).

There are many edge pixels in the background and the subject has many edges inside the tracked contour. There are other people moving in different directions (Fig. 8[f-h]), in the background, causing errors in background image generation(Fig. 8[o-s]). To make tracking more difficult, the face color of the tacked subject is similar to the hall wall color (Fig. 8[a-c]) while his shirt color is similar to that of stairs (Fig. 8[f,g]), and tracked body black hair is interfered with by a walking woman in (Fig. 8(f,g)) and a man with a black suit in (Fig. 8(h)). Our tracked contour is bothered by these interferences, but recovers as soon as we get *Scanny* edges for the interfered part. Even under this complex circumstance, our boundary edge-based tracking and background generation was successful.

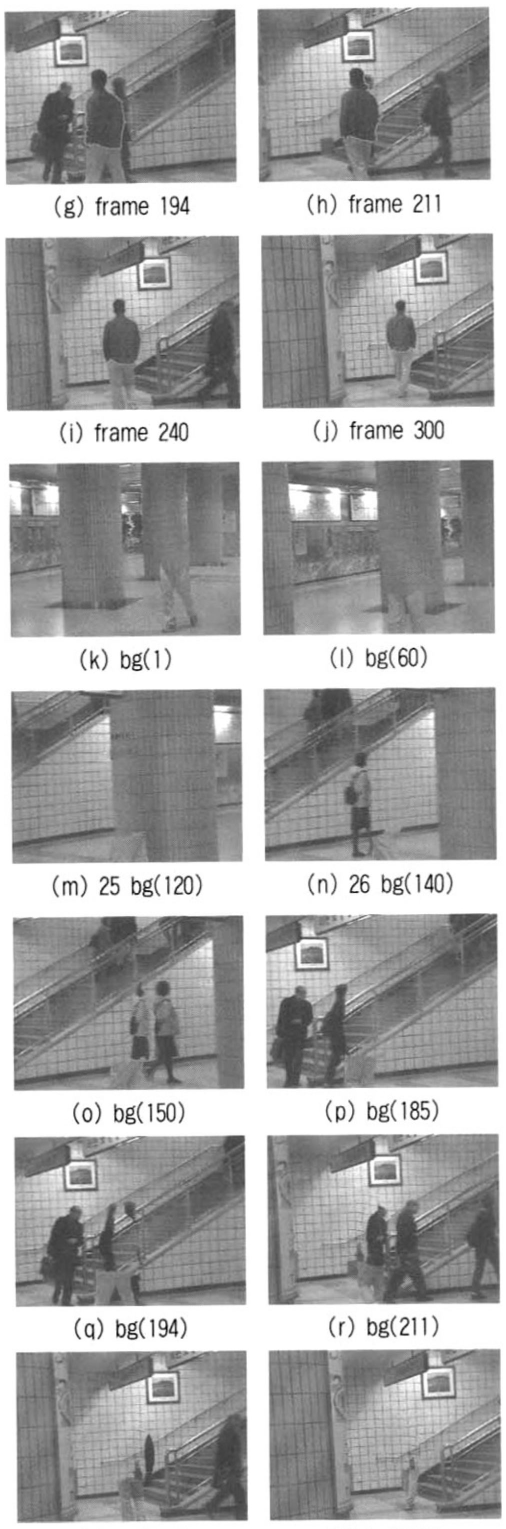
7.2 Handling Occlusion

We assume our subject is never occluded by any background objects, but it occludes other objects in the background. Our tracking condition is tougher to track than the experimental environment by Nguyen[5]. A series of occlusions occur in frames (Fig. 8(f-r)). We suffer serious interference whenever similar colored moving objects are occluded by the tracked subject. The occlusion in frames at (Fig. 8(f-h)) could be easily handled because the white color of the woman moving in the background is distinct from the tracked subject. There are strong Canny edges generated around the tracked subject contour. According to the matching, we get enough set of *selected Scanny points* around our


(a) input frame (b) frame 60

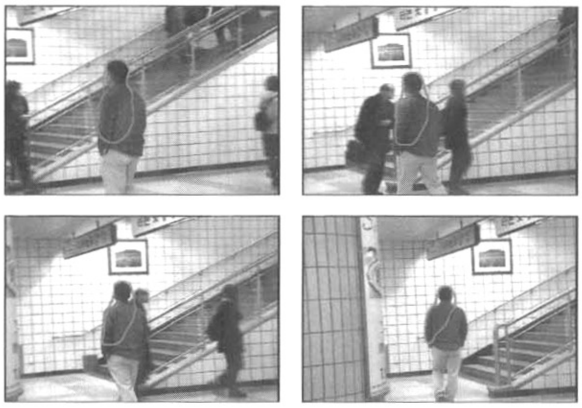
(c) frame 120 (d) frame 140

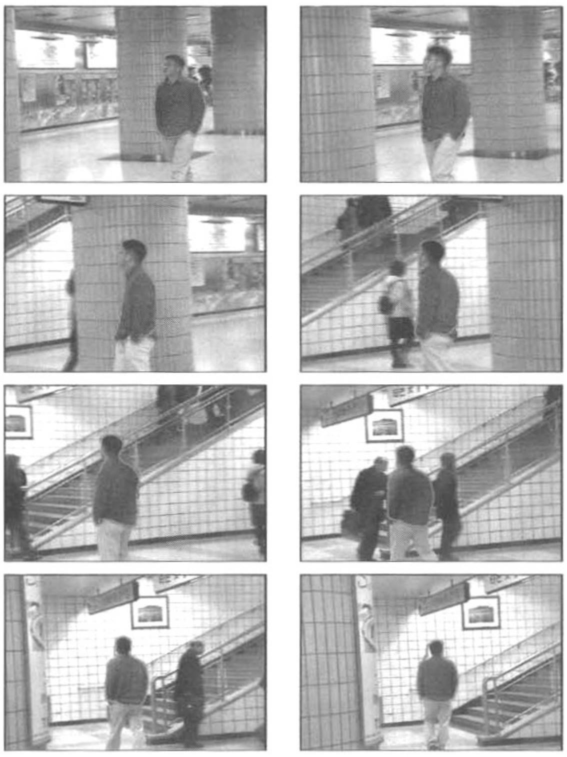
(e) frame 150 (f) frame 185

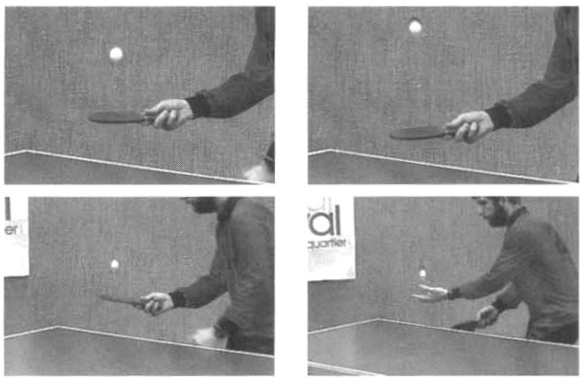


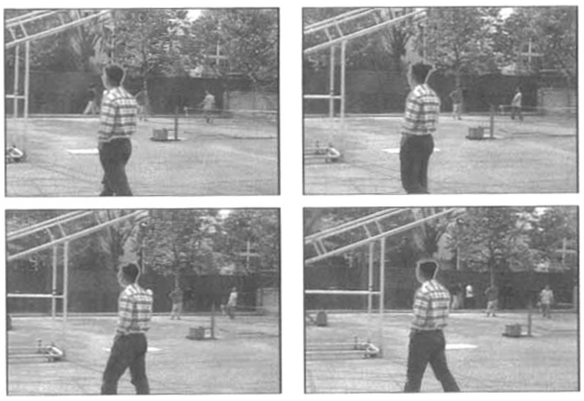
(Fig. 8) Tracking and background generated(upper body removed) result

subject and can find a correct routing around the tracked subject contour using LOD Canny edges. Whenever *Scanny edges* are missing from the basic edge map, we refer to detailed *Wcanny edge* maps with a penalty to favor stronger Canny edges. A series of serious interference occurs starting at (Fig. 8(i)). The hair color of a woman in the background is the same as that of the tracked subject, and the contour is disturbed as she moves to the right. But, the following bold-haired man seriously interferes the tracked subject. He generates many strong Canny edge maps, and the tracked contour is seriously deformed because of the similar color with the tracked subject. When the background object moves away from the tracked subject, we get strong Canny edges back between the tracked subject and the background object, and we get a tracked contour that is heavily deformed. Whenever the background subject is gone, there is another strong Canny edge map generated by wall tiles. But, the tracked contour because of the wall tile has similar colors around the inside/outside of the tracked contour. Because our contour routing favors short routes, the tracked contour successfully shrinks to our tracked subject in several tracking frames. We received source codes from Dr. Nguyen to compare his tracking performance with our tracking performances. Because Dr. Nguyen's algorithm is not designed for a highly textured environment, the tracking result is poorer than that of our result as can be found in (Fig. 9) (Fig. 10) shows our parametric contour tracking result mainly coded by T. Kim[9]. Please note the first frames of (Fig. 9) and (Fig. 10) (these results are based on parametric contour tracking) are later than our test result presented in (Fig. 8) which is based on non-parametric contour. This is due to parametric approaches[5,9] cannot handle object boundaries of similar colors with their background. (Fig. 11) shows our result of tracking a pingpong ball. (Fig. 12) shows our result of tracking a man wearing a shirt with very strong textures. Our method degrades by contour shrinking while tracking an object with strong textures because they generate strong Canny edges.


(Fig. 9) Upper body tracking result by Nguyen's code[5], images were generated by T. Kim


(Fig. 10) Upper body tracking result by T. Kim at al's code[9]


(Fig. 11) Pingpong ball tracking result by our approach


(Fig. 12) Our result of Tracking a man with very strong texture shirts (thanks to input image provided by T. Kim)

8. Conclusion

In this paper, we proposed a brand-new method of improving accuracy in tracking a highly textured object and eliminating it to generate corresponding background scene. We start by selecting a boundary edge pixel from the *simple (strong) Canny edge map*, referring to the most detailed edge map to get edge information along the LOD Canny edge maps. Our basic tracking frame is determined from the strong Canny edge map, and the missing edges are filled by the detailed Canny edges along the LOD hierarchy. Even though detailed Canny edges are noisy, our basic tracking frame is determined from the *Scanny*, and is not disturbed by noisy edges. This has an effect of Nguyen's background noisy edge removal. Another major contribution of our work is not accumulating tracking errors. We minimize the possibility of accumulated tracking errors by relying on the current Canny edge map only. In Nguyen's approach[5], a new contour is determined by mixing the current image edge map with the previous contour. If there is no edge present, we may have a tracking error for the part. Whenever we get *Scanny* edge information, the tracking error disappears, and we can restart accurate tracking for the erroneous part. Our tracking condition is tougher to track compared to Nguyen's. The problem with our approach is that we need edge information as every other edge-based approach does. If there is no edge information available because of the same color with the background, our tracking performance degrades heavily, and this is inevitable for all approaches. But, our tracking performance recovers whenever we get edge information back. By using our novel method, our computation is not bothered by noisy edges resulting in a robust tracking. Our experimental results show that our tracking approach is reliable enough to handle a sudden change of the tracked subject shape in a complex scene.

Acknowledgement

This work was supported by 2006 Hongik University Research Fund.

Additional expenses of this research were supported by Seoul Research and Business Development Program (10555).

References

[1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," International Journal of Computer Vision Vol.1, No. 4, pp. 321-331, 1987.

[2] N. Peterfreund, "Robust tracking of position and velocity with kalman snakes," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.21, pp. 564-569, 1999.

[3] Y. Fu, A. T. Erdem, and A. M. Tekalp, "Tracking visible boundary of objects using occlusion adaptive motion snake," IEEE Trans. on Image Processing, Vol.9, pp. 2051-2060, 2000.

[4] N. Paragios and R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.22, pp. 266-280, 2000.

[5] H. T. Nguyen, M. Worring, R. van den Boomgaard, and A. W. M. Smeulders, "Tracking nonparameterized object contours in video," IEEE Trans. on Image Processing, Vol.11, pp. 1081-1091, September 2002.

[6] J. B. T. M. Roerdink and A. Meijster, "The watershed transform: Definition, algorithms and parallelization strategies," Fundamenta Informaticae, Vol.41, pp. 187-228, 2000.

[7] H. T. Nguyen, M. Worring, and R. van den Boomgaard, "Watersnakes: energy-driven watershed segmentation," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.25, pp. 330-342, 2003.

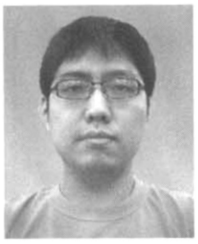
[8] J. Park, "Contour tracking using modified Canny edge maps with level-of-detail," Lecture Notes in Computer Sciences, Vol.3691, pp. 1-8, 2005.

[9] M. Roh, T. Kim, J. Park, and S. Lee, "Accurate Object Contour Tracking Based on Boundary Edge Selection," Pattern Recognition, Vol. 40, No. 3, pp. 931-943, March 2007.

[10] A. Bartoli, N. Dalal, B. Bose and R. Horaud, "From Video Sequences to Motion Panoramas," Proceedings of Workshop on Motion and Video Computing, pp. 1-7, 2002.

박 지 헌

e-mail : jhpark@hongik.ac.kr
1983년 서울대 공대 기계설계학과(학사)
1985년 한국과학기술원 전산학과(석사)
1990년 University of Texas at Austin 전산학과(석사)
1994년 University of Texas at Austin 전산학과(박사)
1985년~1986년 한국전자통신연구원
1986년~1993년 부산외국어대학 컴퓨터 공학과 조교수
1994년~현재 홍익대학교 컴퓨터 공학과 교수
관심분야 : 컴퓨터 그래픽스, 이미지 처리



장 영 대

e-mail : ferthona@cs.hongik.ac.kr
2006년 홍익대 공대 컴퓨터공학과(학사)
2006년~현재 홍익대학교 컴퓨터 공학과 석사과정
관심분야 : 비전 컴퓨팅 시스템, 실시간 영상 처리

이 동 훈

e-mail : spitter@cs.hongik.ac.kr
2004년 홍익대 공대 컴퓨터공학과(학사)
2005년~현재 홍익대학교 컴퓨터 공학과 석사과정
관심분야 : 컴퓨터 비전, 컴퓨터 그래픽스

이 종 관

e-mail : bellres@cs.hongik.ac.kr
2005년 홍익대 공대 컴퓨터공학과(학사)
2005년~현재 홍익대학교 컴퓨터 공학과 석사과정
관심분야 : 이미지 프로세싱, 컴퓨터 비전

함 미 옥

e-mail : moham@hanmail.net
2000년 가톨릭대학교 대학원 전자계산학 전공(석사)
2003년~현재 홍익대학교 컴퓨터 공학과 박사과정
관심분야 : 영상 처리, 컴퓨터 비전