DETECTION AND TRACKING OF HUMAN MOTION USING ACTIVE CONTOUR MODELS

Theofrastou Nikolaos

A Thesis submitted for the Degree of Diploma
in the Department of Electronic & Computer Engineering

Technical University of Crete

Supervisory Committee

Professor Michalis Zervakis (Supervisor)
Professor Nikolaos Sidiropoulos
Associate Professor Evripides Petrakis

April 2004

Acknowledgements

I wish to express my sincere appreciation to Professor Michalis Zervakis for his valuable assistance and excellent cooperation in every single stage of this work.

In addition, special thanks to Professor Nikolaos Sidiropoulos and to Associate Professor Evripides Petrakis for their participation in the supervisory committee.

Finally, I would like to thank my family and my friends for their love and support throughout all the years of my studies.

Abstract

Visual tracking is a domain of computer vision with many promising applications. An important field of this domain involves the tracking of human motion with interesting applications, including “smart” surveillance systems, virtual interfaces, character animation and gesture-driven control. This work intends to implement a semi-automatic, non-real-time tracking system of humans by using active contour models (snakes). Snakes are energy-minimizing splines that are moving towards a target of interest under the influence of internal and external forces. The modular character of the snake energy functional allows the inclusion of many different image cues (edges, color, shape) and corresponding terms, according to the specific application. The snake model offers a flexible and unified way of extracting and representing a desired image feature. Emphasis is given in the confrontation of two major, intrinsic problems of snakes: the dependence on the initial contour and the weakness in outlining concave shapes. Moreover, this work examines the details of adapting the snake model to the tracking system. The overall algorithm is tested on sequences of grey-scale images under some specific stipulations and has encouraging results.

Table of Contents

Introduction

The term ‘visual tracking’ may be generally defined as the problem where- given an identified target in an initial image I_0 of a sequence of images (I_0, I_1, I_2, ...), and a corresponding initial configuration (or state), S_0, of this target- we must produce a series of target state estimates (S_1, S_2, …). At this generic definition we may add some stipulations in order to produce a more specific visual tracking system that covers our needs at each time. Such stipulations may concern the ‘real-time’ or ‘non-real-time’ operation of the system, the range of the conditions under which the target is expected to be tracked, the dominant cue(s) that distinguish the target of interest (e.g. shape, color, appearance), or some prior information that can be assumed. In this thesis, the goal is to produce a semi-automatic, ‘non-real-time’ system of tracking moving humans in sequences of gray-scale images.

For a long time, the mainstream of tracking has been the “ blob” tracking algorithm. It can be said that it is a ‘segmentation-based’ approach, in the sense that it is actually a way of optimizing a pixel selection based on a given cue. This algorithm though relies on some (usually) undesirable assumptions, for example the fact that there is only one target in the region of interest and that this target remains roughly constant in size from frame to frame. The same limitation in the appearance of the object of interest between frames somewhat also exists in the “template-based” region-tracking algorithm. The basic idea of this approach is that of matching the direct appearance of the target from frame to frame, using some variation in correlation. This thesis uses a third approach that can be referred to as “snake-tracking”. The state estimates of the target are represented as contours of discrete points; the given initial contour in the first image converges to the target through an energy-minimization process, whereas the initial contour of the subsequent states is generated from the previous state(s) and converges with the same process. This approach can give quite accurate results by low computational work.

Snakes were proposed by Kass et al. in 1988, as energy-minimizing splines in an image domain, with the ability to move under the influence of internal forces, coming from within the curve itself, and external image forces, computed from the image data. They may also be guided by external constrained forces. The internal forces aim to enforce a piecewise smoothness constraint. The external image forces attract the snake toward salient image features like edges, lines and subjective contours. Finally, the external constraint forces serve to put the snake near the desired local minimum and can, for example, come from user interaction or high-level interpretations. All these forces form an energy functional, which has to be minimized in order to achieve a convergence of an initial contour to the object of interest. This minimization exhibits a dynamic behavior and that is why snakes are regarded to be active models.

In the original definition of snakes, the minimization of the energy functional is achieved by a variational approach (Euler – Lagrange equations). This approach requires the computation of higher order derivatives of the discrete data, may have problems of numerical instability and do not allow direct and natural addition of external constraints, as these need to be differentiable. Amini et al. presented a dynamic programming algorithm for energy-minimization, which allows the enforcement of hard constraints for a more desirable behavior of the snakes. Nevertheless, this method is very slow and computationally expensive in memory requirements. Williams et al. proposed a fast algorithm, widely known as “greedy snake”. This method was derived from the dynamic programming method, but, unlike the latter, does not use exhaustive search. Its searching strategy is based instead on the connective property of the eight neighbours of a snake point. Furthermore, the greedy snake retains the privilege of the dynamic programming method to allow hard constraints. Therefore, it is a method that combines speed, flexibility and simplicity and that is why is chosen in this thesis.

However, irrespectively of the method used for the minimization of the energy functional, snakes tend to present some generic problems. Mainly, if the initial contour is not close to the target in both position and shape, the convergence process may end up in the wrong result due to the lack of attracting forces or because the snake is trapped by local minima. Even in the case of a quite close initial contour, snakes usually have difficulties in handling concave/convex contours. Many methods have been proposed towards the solution of these problems, such as the “gradient-vector-flow” (GVF) snake, in the area of the variational approach, and the “attractable snake model”, which is based on the greedy algorithm. Some of these methods solve some problems, usually in specific cases, but sometimes other problems appear. It is hard to say that there is a globally optimal solution.

This work intends to examine the effectiveness of snakes in outlining and tracking of humans and proposes some modifications on the previous work. Chapter 1 is an introduction to visual tracking and a statement of the assumptions and stipulations of our desired tracking system. Chapter 2 introduces the original snake model of Kass (variational approach), continues with the dynamic programming approach and concludes to the greedy algorithm. Chapter 3 describes two major problems of snakes and outlines some of the proposed solutions. Chapter 4 proceeds to the implementation details and results of the snake model and the tracker of the current thesis. Finally, chapter 5 gives the conclusion and some possible directions of future work.

Chapter 1: Visual tracking

1.1 What is visual tracking? - Definition and applications

A simple and theoretical definition of visual tracking could be given through the following problem [1]:

Given:

1. an identified target (or object of interest) in an initial image, I_0,
2. a corresponding initial configuration (or state), S_0, of this target and
3. a series of subsequent images, $I_1, I_2, \ldots$,

produce a corresponding series of state-estimates $S_1, S_2, \ldots$ of the target.

Visual tracking could also be defined mathematically, in a probabilistic framework [2]. In a dynamic system, the states of the target and image observations are represented by $\boldsymbol{X}_t$ and $\boldsymbol{Z}_t$ respectively. The tracking problem could be formulated as an inference problem with the prior $p(\boldsymbol{X}_{t+1} \mid \boldsymbol{Z}_{t,}\boldsymbol{Z}_{t-1,\ldots})$, which is a prediction density. We have:

$$p(\boldsymbol{X}_{t+1} \mid \boldsymbol{Z}_{t+1,}\boldsymbol{Z}_{t,\ldots}) \propto p(\boldsymbol{Z}_{t+1} \mid \boldsymbol{X}_{t+1}) p(\boldsymbol{X}_{t+1} \mid \boldsymbol{Z}_{t,}\boldsymbol{Z}_{t-1,\ldots}) \tag{1.1}$$

$$p(\boldsymbol{X}_{t+1} \mid \boldsymbol{Z}_{t,}\boldsymbol{Z}_{t-1,\ldots}) = \int p(\boldsymbol{X}_{t+1} \mid \boldsymbol{X}_t) p(\boldsymbol{X}_t \mid \boldsymbol{Z}_{t,}\boldsymbol{Z}_{t-1,\ldots}) d\boldsymbol{X}_t \tag{1.2}$$

where $p(\boldsymbol{Z}_{t+1} \mid \boldsymbol{X}_{t+1})$ represents the measurement or observation likelihood and $p(\boldsymbol{X}_{t+1} \mid \boldsymbol{X}_t)$ is the dynamic model. This probabilistic formulation could be represented by a graphical model that is similar to the hidden Markov model (Figure 1.1). At time t, the observation $\boldsymbol{Z}_t$ is independent of previous states, $(\boldsymbol{X}_{t-1,}\boldsymbol{X}_{t-2,\ldots})$ and previous observations, $(\boldsymbol{Z}_{t-1,}\boldsymbol{Z}_{t-2,\ldots})$, given the current state, $\boldsymbol{X}_t$, i.e. $p(\boldsymbol{Z}_t \mid \boldsymbol{X}_{t,}\boldsymbol{X}_{t-1,\ldots}, \boldsymbol{Z}_{t-1,}\boldsymbol{Z}_{t-2,\ldots}) = p(\boldsymbol{Z}_t \mid \boldsymbol{X}_t)$, and the states have Markov property, i.e. $p(\boldsymbol{X}_t \mid \boldsymbol{X}_{t-1,}\boldsymbol{X}_{t-2,\ldots}) = p(\boldsymbol{X}_t \mid \boldsymbol{X}_{t-1})$.

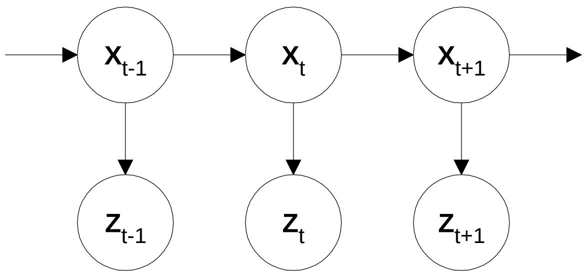


Figure 1.1: The tracking problem through a graphical model [2].

However, in many practical cases, this strict definition is of small significance. On the other hand, the first definition is quite generic. In order to describe accurately a real visual tracking system we have to add some stipulations to this definition.

1.1.1 A real visual tracking system

The stipulations and assumptions that define the form of a real visual tracking system could be derived from the following set of questions [1]:

- What is/are the dominant cue(s) (e.g. edges, colour, appearance, etc.) that discriminate the target from other objects and consequently what is the target representation (e.g. parameterised shapes, colour distribution, image-templates, etc.)? Usually there are good 'local' approximations rather that a globally unique answer.
- What is the range of viewing conditions under which we expect to track our target? Is the target constant in shape or is it deformable (and then what are the expected changes in pose)? What are the changes in lighting conditions? Is the system able to handle occlusion (i.e. the case where other objects 'hide', partially or totally, the target of interest)?
- Relevant to the issue of the viewing conditions are also some features of the camera (or generally of the sensor), principally its multiplicity and its mobility; does the tracker use a single camera or multiple cameras, which can resolve some ambiguous poses that may occur from monocular vision? Is/are the camera(s) stationary or moving?
- What is the desired dimensionality of the tracking space? Does the tracker need a quite accurate and compact representation of physical space (3-D approach) or a less precise representation of the target is sufficient?

- What kind of prior information can be assumed? For example, is the 3-D structure (in 3-D approaches) of the object already known? Is it possible to know *a priori* the deformation pattern of the target?
- What level and frequency of information is needed? There is a big difference between expecting 60 Hz pose estimates down to seconds of accuracy, and just coarsely knowing where something is going. By sub-sampling the image or the frame rate, we can gain in speed but lose in accuracy. What is more important?
- Do we need a 'real-time' tracking operation or not?
- What grade of user interference is allowed? For example, a user could provide the initial state or interfere by correcting an estimate.

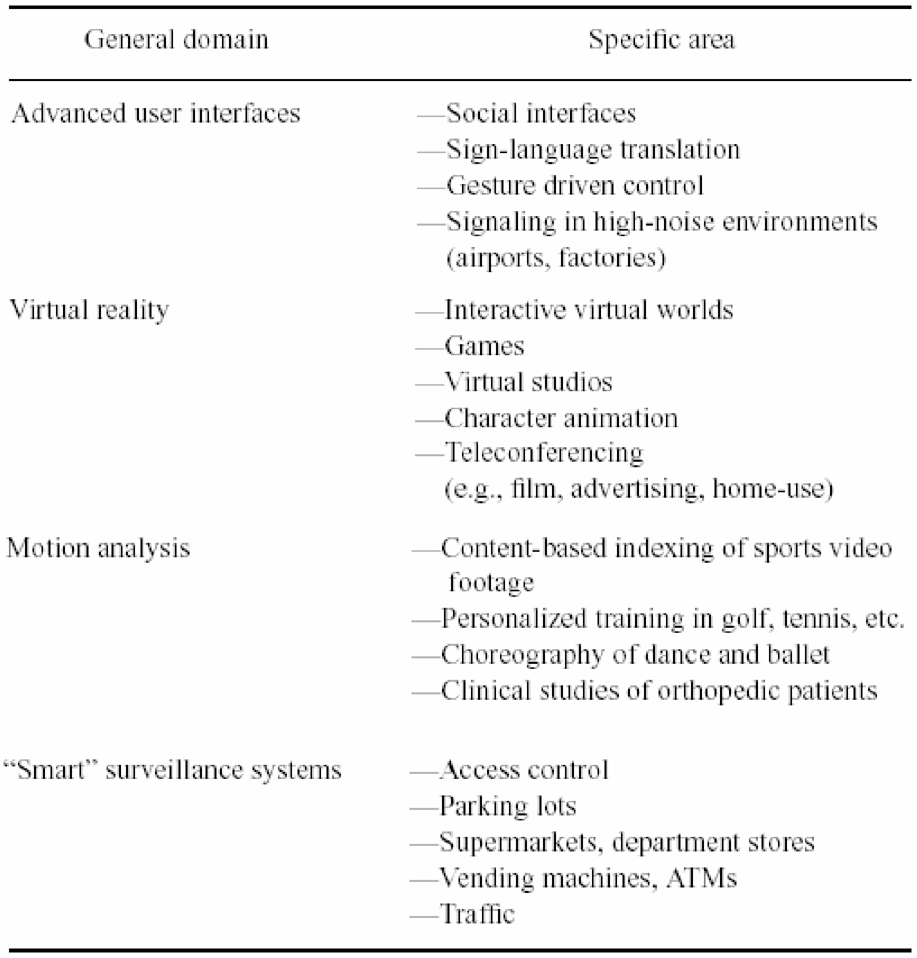
Some of these issues are not necessarily independent of one another. Moreover, thinking about them, may lead us to some other useful questions. In each case, the answers to these questions depend, more or less, on the application of the specific tracker.

1.1.2 Applications

In the last few years, a new application domain has emerged in computer vision. This domain works on the analysis of images involving humans, covering, among others, issues like, hand gesture recognition, lip tracking and whole-body tracking [3]. The tracking of human motion could be put in a general framework of human motion analysis [4]. A step of motion analysis involving human body parts may precede the tracking-phase, providing to it some low-level information (e.g. body part segmentation or joint detection and identification) that may be useful during the tracking-phase. Finally, a higher-level task of recognizing human activities may follow a successful tracking stage, completing the procedure of human motion analysis. There are many interesting and promising applications in this area. For a summary, see Table 1.1 [3].

More specifically, human motion analysis can help in the development of advanced social interfaces, where computer-generated characters may interact with the user in a more friendly way, using human-like behaviours [5]. Furthermore, a speech-guided interface can use computer vision, either in order to detect the presence of a user and commence an interaction, either in order to recognize a user, distinguish multiple users and guide the dialogue in a more proper way, or finally in order to enable a more robust recognition of speech in the presence of acoustic noise (e.g. lip tracking [6]). Other interesting applications in this domain are sign-language interpretation [7], gesture-driven control for people with disabilities [8] and signalling in high-noise environments, such as airports and factories.

The development of interactive virtual worlds is also relevant to the above application domain. The tracking of the human body may help in the creation of a human presence in a virtual space, whereas the tracking of hand gestures may be useful in finding a natural way to interact with virtual objects. Other applications in the domain of virtual reality are games [9], virtual studios and character animation [10].



General domain	Specific area
Advanced user interfaces	—Social interfaces
	—Sign-language translation
	—Gesture driven control
	—Signaling in high-noise environments (airports, factories)
Virtual reality	—Interactive virtual worlds
	—Games
	—Virtual studios
	—Character animation
	—Teleconferencing (e.g., film, advertising, home-use)
Motion analysis	—Content-based indexing of sports video footage
	—Personalized training in golf, tennis, etc.
	—Choreography of dance and ballet
	—Clinical studies of orthopedic patients
"Smart" surveillance systems	—Access control
	—Parking lots
	—Supermarkets, department stores
	—Vending machines, ATMs
	—Traffic

Table 1.1: Applications of human tracking [3].

Moreover, visual-based human motion analysis can be applied in personalized training systems for various activities, like sports and dance. It can also help the clinical research in medical branches, like orthopedics. Another possible application could be the content-based indexing of sports video footage, that would decrease the browsing-effort through a large data set, for example in a query like “give me all the cases of action X of the player Y” [3].

Another important application domain is that of “smart” surveillance. Applications may range from detection of human presence and motion to face recognition for the purpose of access control or the observation of human actions and suspicious behaviours. These applications are useful in areas such as parking lots, airports, department stores or traffic management systems. Of course the matter of privacy rights must be taken into account in these cases [3].

Especially in traffic management systems, it is usually desirable to track, apart from humans, other objects as well, for example, vehicles, obstacles and traffic signs. The goal is usually the maintenance of a secure distance of the pedestrian or vehicle from static or moving obstacles and the observance of traffic laws [11]. Furthermore, visual systems that track only vehicles are also useful in applications such as the measurement of traffic flow or the computation of parameters like the average vehicle speed and spatial occupancy [12].

Visual tracking is also applicable in areas where the human motion is not involved in a direct way like in the previous applications, or is not involved at all. Medicine is one of them and relevant application is the tracking of biological structures in MR images [13].

Robotic applications are another domain. These can include mobile robot navigation [14], machine-learning [15] and visual servo [16].

Finally, tracking techniques are often applied in the area of model-based coding in order to accomplish low bit-rate video compression [17].

1.2 Tracking approaches

It is obvious that visual tracking is a very intriguing and quite complex problem of computer vision with numerous and interesting applications. Therefore, during the last years a lot of research has been done on this subject. The proposed approaches usually depend on the specific application and the stipulations that govern the tracker. Although there is no globally adopted classification of the existing tracking techniques, we can consider two distinct classes [18], [19]:

- *Model-based* approaches that impose high-level semantic representation and *a priori* knowledge of the 2-D or 3-D structure of the object.
- *View-based* approaches that do not use shape models but rely on heuristic assumptions to find correspondences of the target.

The *view-based* approaches are relatively fast but cannot handle non-rigid movements of objects as easily as the *model-based* approaches; on the other hand the latter have higher computational cost. In both classes, tracking is performed by information provided by geometrical properties of the target. We can define two different sub-classes according to the type of this information:

- *Feature-based* approaches, which depend on the information provided by the target’s features, such as points, lines and, especially, boundaries (*boundary-based or edge-based* approaches).
- *Region-based* approaches, which rely on the information provided by the whole region.

Another, more general classification states that there are two kinds of approaches to the visual tracking problem [2]:

- *Bottom-up* approaches, which try to determine the target state by simply analysing the content of images.
- *Top-down* approaches, which generate candidate hypotheses (predictions) from the previous frame and then measure and verify these hypotheses against the image observations.

The first class might be computational efficient, but, since tracking depends only on the content of the image, its robustness is strongly affected by the viewing and analysis conditions. On the other hand, *top-down* approaches are less dependent on image analysis but the need for methods of generating and verifying hypotheses imposes larger computational effort.

Obviously it is not possible to outline all the existing tracking approaches at this point, however we can overview three typical approaches: “Blob” tracking, “Template-Based” region tracking and “snake” tracking.

1.2.1 “Blob” tracking

Basic “blob” tracking has been for quite a long time the mainstream of tracking. It can be considered to be a region-based approach according to the above classification. More specifically, it can be characterized as “segmentation-based” approach since it groups similar (in the sense of a given cue) image pixels into blobs in order to estimate the position and shape of the target. Thus, it relies on the analysis of the image and can be regarded as a bottom-up approach. Some of the cues that can be used for the grouping of blobs are intensity, colour, motion, texture and depth. The basic algorithm goes roughly as follows [1]:

1. Identify an image (binary) segmentation function π (based on the selected cue) and an initial region of interest R_0 in the initial image I_0. Let the location of the center of R_0 (with respect to the entire image) be u_{-1}.
2. For every image of the video stream $I_0, I_1, \ldots$
   i. acquire R_i about u_{i-1} in I_i,
   ii. compute the new blob $B_i = \pi(R_i)$,
   iii. compute the new center $u_i = u_{i-1} + \text{centroid}(B_i)$.

This basic form of the algorithm promises quite efficient and accurate tracking but only under some restrictive assumptions. Such assumptions are the usage of a constant segmentation function and the presence, in the region of interest, of only one target, which remains almost constant in size and stays into the region of interest between frames. Even then, this algorithm is very sensitive to unstructured environments, where large appearance variability occurs, due to pose, lighting or occlusion.

Furthermore, the segmentation process can be computationally time-consuming. A good idea to improve the algorithm's performance would be to use lower image resolution (sub-sampling) and lookup-tables for the segmentation function.

1.2.2 "Template-based" region tracking

The basic idea of this approach is to match the direct appearance of the target from frame to frame. Thus, it could be roughly considered as a top-dawn approach (the candidate hypotheses are all the possible locations of the template and the matching based on image features works as the verification phase). A simple form of this algorithm would be [1]:

1. Choose a region of interest R_0 in the initial image I_0. Let the location of R_0 in I_0 be u_0.
2. For every subsequent image I_1, I_2, ...
    i. use some variation on correlation to find the location u_i of the best match to R_i in I_i,
    ii. sample R_i about u_i in I_i.

The above "greedy" operation of correlation, which demands searching for the target at each possible location of the image, is expected to lead to a prohibitive computational cost. Some algorithmic tricks and data structures, such as the resolution pyramids, could improve the performance. In a resolution pyramid, successive levels are smaller versions of the image with fewer pixels to be processed.

As far as the accuracy is concerned, this basic algorithm relies on the assumption that the target remains almost constant in appearance between frames. Likewise the "blob" approach, it cannot handle occlusion and rapid changes in pose and illumination.

The problem of the algorithms that use simplified matching methods, like SSD (Sum of Squared Difference) minimization or cross-correlation maximization, is that they model the motion of the target region as pure translation in the image plane.

However, as mentioned above, in most real cases of tracking a target through a large sequence of images, changes of illumination and image distortions (e.g. rotation or scaling), may occur. Thus, more efficient and robust algorithms of region tracking may be accomplished by modelling these changes in a unified framework [20].

1.2.3 “Snake” tracking

Snakes were introduced by Kass et al. [21] as energy-minimizing splines, which tend to lock onto nearby edges. They are active contour models with the ability to represent the boundary of a target and keep updating it dynamically between frames. Hence, they can be used in a *boundary-based* tracking approach. The basic form of such an algorithm is:

1. Identify an initial contour, C_0, in the initial image, I_0.
2. Let it converge iteratively to the target boundary through the energy-minimization process, giving you the optimised contour $C_0^{'}$.
3. For every subsequent image I_i (i = 1, 2 ,…),
   1. compute the initial contour, Ci, of the image I_i, based on the optimised contour, $C_{i-1}^{'}$, of the previous image I_{i-1} ,
   2. let C_i converge iteratively to the target boundary through the energy-minimization process, giving you the optimised contour $C_i^{'}$.

The initial contour, C_0, in the first image of the sequence, can be either provided by the user or acquired through a high-level process. In both cases, tracking should proceed automatically (i.e. with no further input) and therefore the initial contour, C_i, in the subsequent images, I_i (i = 1, 2 ,…), has to be computed.

There are more than one possible ways to compute C_i, based on the optimised contour, $C_{i-1}^{'}$, of the previous image. A very ‘naïve’ approach is to use $C_{i-1}^{'}$ directly as C_i (Figure 1.1). This is a really simplified method that could work only under certain circumstances. For example, the interior of the target boundary has to be smooth (without edges) and the displacements of the target between frames should be relatively small.

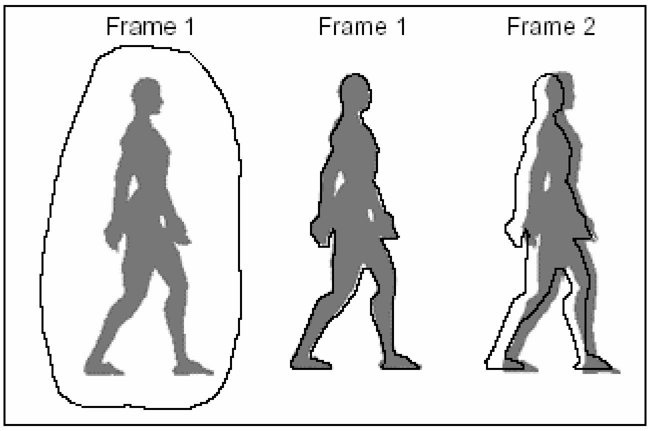


Figure 1.1: The initial contour is provided by the user and its optimisation is used as starting contour for the next frame (naïve approach) [22].

A similar, but less simple, method computes C_i using an inflation factor in order to expand the previous optimised contour $C_{i-1}^{'}$ about its centroid. One drawback of this method is that improper inflation factor may lead to a starting contour, which is either too far away from the real boundary and cannot be attracted to it, or intersects the boundary of adjacent objects.

A third possible approach is to take as starting contour, C_i, the motion-compensated version of the optimised $C_{i-1}^{'}$ (using optic flow for example). This third approach is usually the most efficient but undoubtedly increases the computational cost.

From the above we can claim that snake tracking is somewhat an operation where predictions (naïve or more complex) are made and updated against image observations through the energy-minimization process. Thus, it can be considered as a *top-down* approach.

Regarding accuracy, the basic algorithm as it stands, can promise quite efficient but still not very precise tracking of a target contour at all points, since snakes are actually an interpolation through a discrete set of points. Furthermore, the boundary-based character of snakes makes them sensitive to lighting changes and occlusion. A more robust tracking requires some addition to the basic algorithm. Nevertheless, snakes are generally a good and computationally easy way of tracking.

Finally, it is worth to mention that, some research, inspired from Kass' snakes, has moved on to statistical frameworks and spline-based ideas. A least-squares-style snake algorithm (a Kalman filter) [23] and a probabilistic model of object shape and motion [24] are some of these approaches.

1.3 Visual tracking in the current thesis

This thesis intents to build a quite efficient and robust non-real-time visual system for tracking a single human target through a video stream. The tracking approach that is followed is snake-based. The starting contour of the initial image is a rectangular that is provided by the user and encloses the human target. The user actually defines the rectangular by 'clicking' on the image the upper-left and lower-right angles. This is the only user interference; the following tracking is curried out automatically using motion compensation for the determination of the following starting contours.

The representation of the target is performed in the 2-D space using a contour and the dominant cue of tracking is edges. The system does not use any kind of shape model or other prior knowledge. The target is expected to be deformable (e.g. movement of arms, gait). Small changes in lighting are tolerable but occlusion is not handled. The tracking system is tested on gray-scale images, taken from one single and stationary camera.

More implementation details of the tracker will be stated in chapter 4. Until then we have to take a closer look at snakes.

Chapter 2: Snakes: active contour models

2.1 Introduction

Snakes were introduced by Kass et al. [21], as energy-minimizing splines guided by internal spline forces and external, image and constraint, forces. They are contour models that can provide a unified account of a number of visual problems, which used to be examined separately, such as detection of edges, lines and subjective contours, stereo matching and motion tracking. The iterative and dynamic minimization of their energy functional makes these models active. The name "snakes" was given to them because the movement of the contour through this dynamic minimization looks like the slithering of the homonym reptile.

A snake is actually defined as a sequence of (x,y) points in an image, named "control points", but is usually drawn with lines that connect the adjacent control points, giving the impression of the snake-shape (Figure 2.1). The control points are also called "snaxels" (Snake Elements, according to pixels). Every control point has an energy level, which is desirable to be minimized at each iteration, in order to minimize the energy functional of the whole contour. Depending on the implementation, the minimization process makes each snaxel move, either asynchronously to the other snaxels (affecting the energy level of adjacent snaxels) or synchronously to them. When all snaxels have their energy minimized once, a new iteration begins and the procedure goes on until the total energy stops decreasing or when a more complex criterion is satisfied.

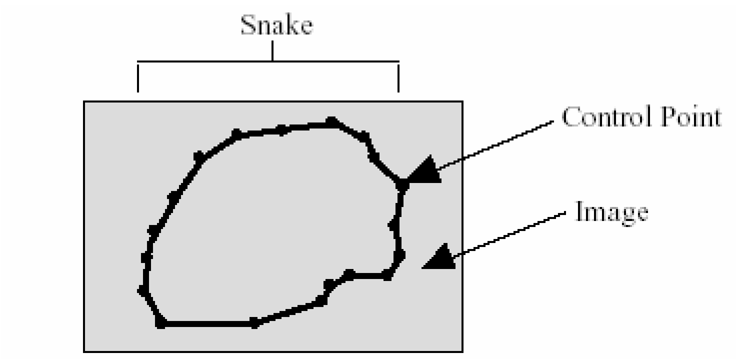


Figure 2.1: A snake as a sequence of control points in an image [22].

There are three main implementations of the snake model according to the formulation of the energy functional and the techniques used for the energy minimization; Kass' original model adopts a variational approach, Amini et. al use dynamic programming and Williams and Shah propose a greedy algorithm.

2.2 Original snake model – a variational approach

Representing the position of a snake parametrically by *$\mathbf{v}(s) = (x(s), y(s))$* (where s is proportional to the contour arc length and $s \in [0,1]$), the energy functional of the original snake model can be written as:

$$E^{*}_{snake} = \int_{0}^{1} E_{snake}(\mathbf{v}(s))ds = \int_{0}^{1} (E_{int}(\mathbf{v}(s)) + E_{image}(\mathbf{v}(s)) + E_{con}(\mathbf{v}(s)))ds \qquad (2.1)$$

where E_{int} is the internal energy of the spline, E_{image} is the energy from the exerted image forces and E_{con} is due to the constraint forces. The minimization of E_{int} aims to impose a piecewise smoothness constraint to the curve, whereas the image forces pull the snake towards salient image feature. The external constraint forces may be exerted through user-interference or high-level interpretations, and serve in putting the snake near the desired local minima. The image and constraint forces are both considered as external forces, unlike the internal forces that come from the spline itself.

2.2.1 Internal energy

The internal spline energy consists of a weighted first-order term and a second-order term, also controlled by some weight:

$$E_{\text{int}} = \frac{1}{2}(\alpha(s)|\mathbf{v}_{s}(s)|^{2} + \beta(s)|\mathbf{v}_{ss}(s)|^{2}) \qquad (2.2)$$

where $\mathbf{v}_{s}(s)$ and $\mathbf{v}_{ss}(s)$ denote the first and second derivatives of $\mathbf{v}(s)$ with respect to s and can be approximated in a discrete form by the backward finite differences

$\left|\boldsymbol{v}_i - \boldsymbol{v}_{i-1}\right|$ and $\left|\boldsymbol{v}_{i-1} - 2\boldsymbol{v}_i + \boldsymbol{v}_{i+1}\right|$ respectively ($\boldsymbol{v}_{i-1}$ and $\boldsymbol{v}_{i+1}$ are the preceding and following snaxels of the snaxel $\boldsymbol{v}_i$ in the sequence of the snake points). Adjusting the two weights, *α(s)* and *β(s)*, results in controlling the relative significance of the above terms.

The first term, also called "elasticity term" or "elastic energy", makes the snake act like a membrane. It is a measure of the distance of a snaxel from the previous snaxel in the sequence of the snake points, and therefore decides how much a control point should be pulled towards its neighbour or pushed away from it. Actually, the influence of this term can be expanded to include also the distance of the snaxel to its following (*i+1*) neighbour, as in equation (2.13). The 'direction' of this movement (push or pull) is defined by the polarity of *α(s)*; if *α(s)* is positive, the minimization of the elastic energy of a snaxel is achieved by pulling this snaxel towards its neighbours. When this pull is applied to all snaxels in one iteration step, the snake appears to contract. On the contrary, when *α(s)* becomes negative, every snaxel moves away from its neighbours and contraction converts to expansion. This can be useful, for example, when we need to detect the contour of an object with a smooth interior (few internal edges) in a complex environment. In this case, we can initialize the snake into the object and then let it expand, avoiding the misleading external edges.

The magnitude of *α(s)* is also significant. The new position of a snaxel in the current iteration of the algorithm is related by the coefficient *α(s)* to the positions that this snaxel and its two neighbours (previous and next) had in the previous iteration step (equation 2.13). Therefore, the magnitude of *α(s)* controls the expansion or contraction rate. Larger values of *α(s)* make the snake shrink (or expand) more rapidly.

The second-order term measures the bending ability of the snake, in other words, its ability to form corners. It is also known as "bending term" or "bending energy". The minimization of this energy tends to smooth the snake curve, disallowing large curvatures. That is why this term is said to make the snake act like a thin metal strip [22]: if the two ends of the strip are not connected (correspondingly, if the snake is open), the strip tends to open out into a straight line; if they are joined together (correspondingly, if the snake is closed), the strip becomes a smooth curve almost without corners (Figure 2.2). Setting the parameter *β* of a control point equal to zero eliminates the bending term, allowing the snake to become second-order discontinuous and form a corner at that point.

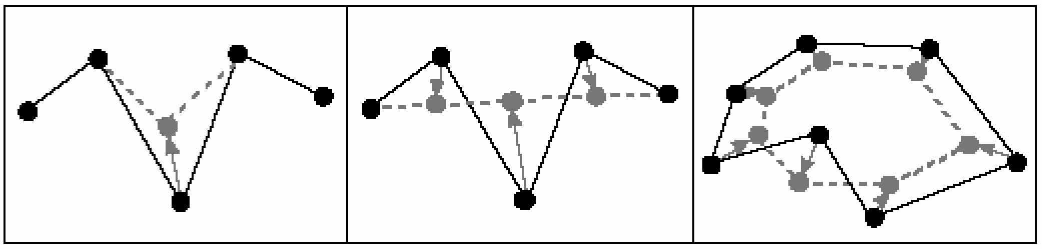


Figure 2.2: The minimization of the bending energy: the effect on a single snaxel [left], on an open snake [middle] and on a closed snake [right], after a number of iterations [22].

From the above it is obvious that both elasticity and bending terms are affected by the relative positions of the control points along the snake and not by the image features. This fact justifies the characterization “internal”, which is given to the corresponding energies.

2.2.2 Image energy

Unlike the internal energy, this energy term relies on the information obtained from the image area around the snake and is irrelevant to the position of the control points. The minimization of the image energy aims to attract the snake towards salient features. The original model examines the cases of attraction to lines, edges and terminations, and in this direction proposes the following formulation of the image energy term:

$$E_{image} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term} \tag{2.3}$$

where E_{line} is the line-energy, E_{edge} is the edge-energy, E_{term} is the termination-energy, and $w_{line}, w_{edge}, w_{term}$ are the corresponding weights. By adjusting properly these weights we can obtain different snake behaviour.

The line-energy is the simplest term of image energy and is actually the image intensity itself:

$$E_{line} = I(x, y) \tag{2.4}$$

This energy term can attract a snake either to dark lines (small values of intensity) or bright lines (large values of intensity), depending on the sign of the weight w_{line}. If w_{line} is positive, then the minimization of a positive quantity is achieved by going to smaller values of intensity, hence darker areas. Conversely, a negative weight leads to brighter areas.

The second term of equation (2.3) desires to guide the snake towards edges; this can be done using a simple gradient function:

$$E_{edge} = -|\nabla I(x, y)|^2 \tag{2.5}$$

The gradient of image intensity, $\nabla I(x, y)$, has large values at points that belong in strong edges; the stronger is the edge, the larger the gradient. This fact imposes the negative sign in equation (2.5). If the negation were not present, the control points would be repelled from strong edges during the energy-minimization.

The original model also proposes a slightly different form of the edge term:

$$E_{edge} = -(G_\sigma * \nabla^2 I)^2 \tag{2.6}$$

where G_σ is a Gaussian of standard deviation σ. Minima of this energy correspond to zero-crossings of $G_\sigma * \nabla^2 I$, which define edges. This term can be used in a scale-space continuation approach, where firstly the snake is let to converge on a blurry version of the image and then the blurring is being reduced gradually. This blurring effect enlarges the capture range around an object of interest. Figure 2.3 shows an example:

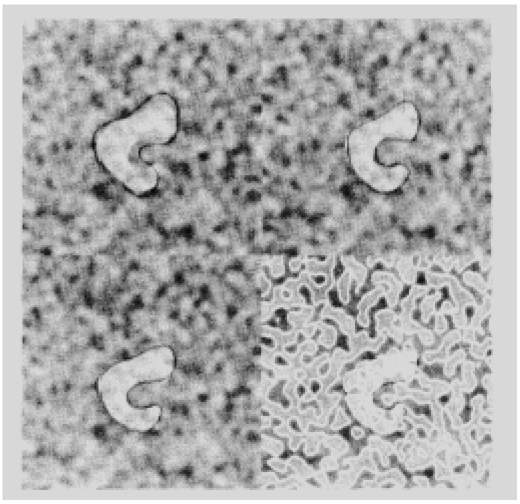


Figure 2.3: Scale-space continuation on edge-energy [21].

The upper-left image shows the snake in equilibrium at a coarse scale. The edge localization is not that accurate but the goal of approaching to the contour is achieved. Reducing the blurring gradually, leads the snake to the position shown in the upper-right image (intermediate scale) and finally to a very accurate localization shown in the lower-left image (no blurring at all). The lower-right image superimposes on the lower-left image the zero-crossings of $G_\sigma * \nabla^2 I$ that corresponds to the energy term of this image. It is worth to note that this last image appears to have many zero-crossings that could mislead the snake. This does not happen though, due to the fact that the edge-energy is only a term in the whole energy functional.

Finally, the termination-energy is useful in finding terminations of line segments and corners. A line segment may appear to terminate either because another line or edge (probably from another object) occludes it, or because it changes direction, forming a corner [22]. In either case, line terminations correspond to important features of a target shape; therefore it is sometimes useful to detect them. The termination term is computed in a slightly smoothed version of the image, $C(x, y) = G_\sigma(x, y) * I(x, y)$. Smoothing is used to reduce the image noise; noise could be misleading, for example, in cases where it degrades a line segment, making it appear like two distinct lines. If we denote by $\theta = \tan^{-1}(C_y / C_x)$ the gradient angle and by $\boldsymbol{n} = (\cos\theta, \sin\theta)$ and $\boldsymbol{n}_\perp = (-\sin\theta, \cos\theta)$ the unit vectors along and perpendicular to the gradient direction respectively, then we have:

$$\begin{aligned} E_{term} &= \frac{d\theta}{d\boldsymbol{n}_\perp} \\ &= \frac{\partial^2 C / \partial \boldsymbol{n}_\perp^2}{\partial C / \partial \boldsymbol{n}} \\ &= \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{\left(C_x^2 + C_y^2\right)^{3/2}} \end{aligned} \tag{2.7}$$

The significance of the termination-term could be shown through the following example. Figure 2.4, shows a "subjective" triangle occluding two lines and three circles. If a snake were initialized into this "invisible" triangle, it would probably manage to converge to its subjective contour. Due to the combination of E_{edge} and E_{term}, some snaxels would be attracted to the existing edges, line-terminations and corners, while the internal energy would smooth the snake, allowing to the rest of the snaxels to rest on the subjective edges of the triangle.

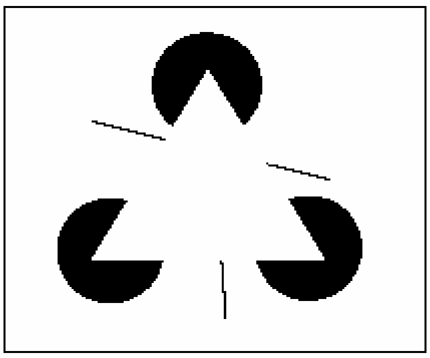


Figure 2.4: A subjective contour [22].

2.2.3 External constraint energy

Kass et al. created a user-interface for snakes on a Symbolics Lisp Machine (Figure 2.5), in order to examine the results of exerting interactively constraint forces to a snake. The user was able to "push" a snake near a feature of interest, letting the rest of the energy terms to continue the snake's movement. Furthermore, the user was able to attach a spring to a snaxel and hence constraint its movement. The other side of the spring could be free, so that the user could drug it around (using the mouse). It could also be connected to a fixed position in the image or to another snaxel.

Another external constraint force proposed in the original model is a repulsion force of the form $\frac{1}{r^2}$, also controllable by the mouse. Due to numerical instability near $r = 0$ ($\frac{1}{r^2} \to \infty$ when $r \to 0$), this energy functional is clipped near zero and that is why the resulting potential is depicted as a volcano (Figure 2.5).

The computation of the constraint energy term is not fully documented in the original model. It is stated, for example, that a spring connecting two snaxels x_1 and x_2 would cause the addition of the term $-k(x_1 - x_2)^2$ to the external constraint energy (where k is the spring constant); there is no reference though to the case of a spring dragged around by the mouse. In fact, it is difficult, if not impossible to model such an interaction as an energy term. However, the original model includes E_{con} in the total snake energy.

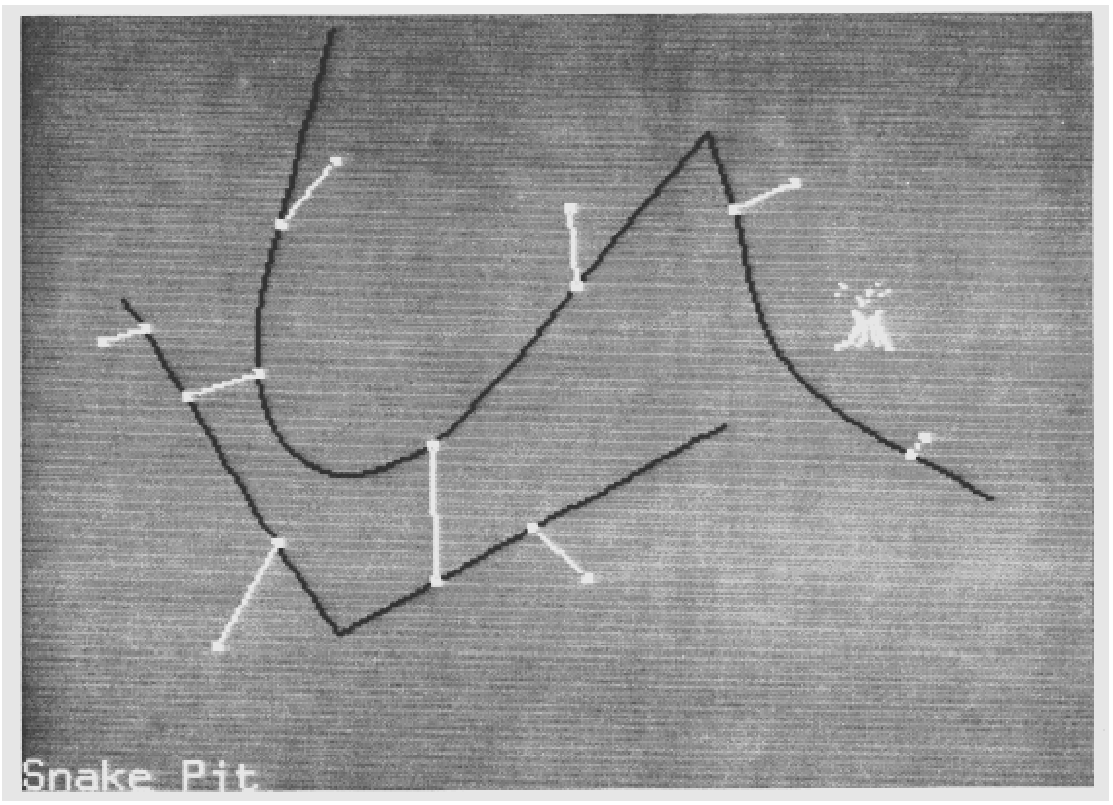


Figure 2.5: The Snake Pit user-interface. Snakes are shown in black, springs and the volcano in white [21].

An example of applying external constraint forces to a snake is shown in figure 2.6 [22]. A snake has relaxed on a contour including two different objects (1). This fact is undesirable, so the user pushes a snaxel towards the desired target (2). Due to the internal forces of the snake, the neighbours of the above snaxel are moving along with it (3,4,5). The snake finally fits on the desired target, influenced by the other terms as well (6).

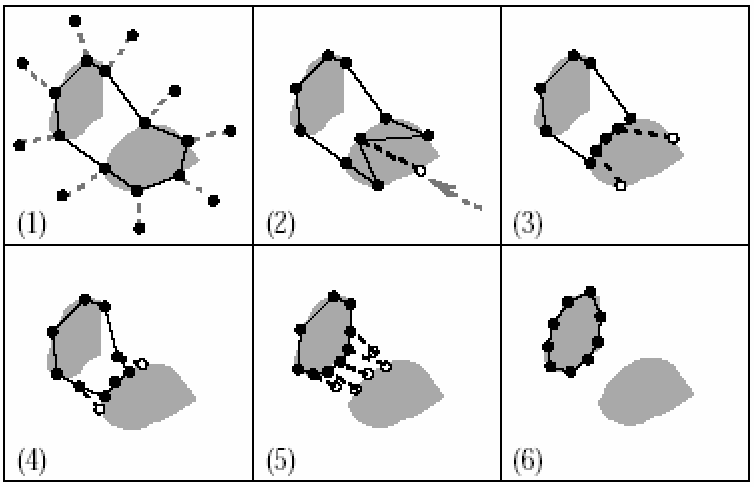


Figure 2.6: User-interacted movement of a snake. Solid lines denote the snake and dashed-lines denote snaxels' movement [22].

2.2.4 Energy minimization

As mentioned before, both image forces and constraint forces are external, therefore the sum of their corresponding energies can be written as $E_{ext} = E_{image} + E_{con}$. Hence equation (2.1) becomes:

$$\begin{aligned} E^*_{snake} &= \int_0^1 (E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s)))ds \\ &= \int_0^1 (\frac{1}{2}(\alpha(s)\,|\,\mathbf{v}_s(s)\,|^2 + \beta(s)\,|\,\mathbf{v}_{ss}(s)\,|^2) + E_{ext}(\mathbf{v}(s)))ds \end{aligned} \quad (2.8)$$

Assuming constant weights $\alpha(s)=\alpha$ and $\beta(s)=\beta$, the above equation gives two independent Euler-Lagrange equations:

$$-\alpha\frac{\partial^2 x}{\partial s^2}+\beta\frac{\partial^4 x}{\partial s^4}+\frac{\partial E_{ext}}{\partial x}=0 \qquad (2.9)$$

$$-\alpha\frac{\partial^2 y}{\partial s^2}+\beta\frac{\partial^4 y}{\partial s^4}+\frac{\partial E_{ext}}{\partial x}=0 \qquad (2.10)$$

When $\alpha(s)$ and $\beta(s)$ are not constant, it is simpler to use a discrete form of equation (2.8):

$$E^*_{snake}=\sum_{i=1}^{n}(E_{int}(i)+E_{ext}(i)) \qquad (2.11)$$

where *n* is the number of the control points. Approximating the derivatives with finite differences and using vector notation $\boldsymbol{v}_i=(x_i,y_i)=(x(ih),y(ih))$, the internal energy is written as:

$$E_{int}(i)=\alpha_i\,|\boldsymbol{v}_i-\boldsymbol{v}_{i-1}|^2/2h^2+\beta_i\,|\boldsymbol{v}_{i-1}-2\boldsymbol{v}_i+\boldsymbol{v}_{i+1}|^2/2h^4 \qquad (2.12)$$

where $\boldsymbol{v}_0=\boldsymbol{v}_n$ and $\boldsymbol{v}_{n+1}=\boldsymbol{v}_1$ (in a closed snake) and h is the distance between subsequent snaxels (it can be included in the values of *α* and *β* and disappear from the equation). Let $f_x(i)=\partial E_{ext}/\partial x_i$ and $f_y(i)=\partial E_{ext}/\partial y_i$, where the derivatives are approximated by finite differences if they cannot be computed analytically. Then the corresponding Euler equation is:

$$\begin{aligned}\alpha_i(\boldsymbol{v}_i-\boldsymbol{v}_{i-1})-\alpha_{i+1}(\boldsymbol{v}_{i+1}-\boldsymbol{v}_i)&+\\ \beta_{i-1}[\boldsymbol{v}_{i-2}-2\boldsymbol{v}_{i-1}+\boldsymbol{v}_i]&-\\ 2\beta_i[\boldsymbol{v}_{i-1}-2\boldsymbol{v}_i+\boldsymbol{v}_{i+1}]&+\\ \beta_{i+1}[\boldsymbol{v}_i-2\boldsymbol{v}_{i+1}+\boldsymbol{v}_{i+2}]&+\\ (f_x(i),f_y(i))&=0\end{aligned} \qquad (2.13)$$

This equation can be written in matrix form as:

$$\boldsymbol{Ax} + \boldsymbol{f_x(x,y)} = 0 \tag{2.14}$$

$$\boldsymbol{Ay} + \boldsymbol{f_y(x,y)} = 0 \tag{2.15}$$

where $\boldsymbol{A}$ is a $n \times n$ pentadiagonal banded matrix. If α and β are constant, then

$$\boldsymbol{A} = \begin{bmatrix} 2\alpha+6\beta & -\alpha-4\beta & \beta & 0 & \cdots & \cdots & 0 & \beta & -\alpha-4\beta \\ -\alpha-4\beta & 2\alpha+6\beta & -\alpha-4\beta & \beta & 0 & \cdots & \cdots & 0 & \beta \\ \beta & -\alpha-4\beta & 2\alpha+6\beta & -\alpha-4\beta & \beta & 0 & \cdots & \cdots & 0 \\ 0 & \beta & -\alpha-4\beta & 2\alpha+6\beta & -\alpha-4\beta & \beta & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & \cdots & \cdots & 0 & \beta & -\alpha-4\beta & 2\alpha+6\beta & -\alpha-4\beta & \beta \\ \beta & 0 & \cdots & \cdots & 0 & \beta & -\alpha-4\beta & 2\alpha+6\beta & -\alpha-4\beta \\ -\alpha-4\beta & \beta & 0 & \cdots & \cdots & 0 & \beta & -\alpha-4\beta & 2\alpha+6\beta \end{bmatrix}$$

A similar form is used when α and β are variable for each snaxel.

In order to solve the above equations, the snake is made dynamic by treating $\boldsymbol{x}$ and $\boldsymbol{y}$ as functions of time. Then the right-hand sides of these equations are set equal to the product of a step size, γ, and the negative time derivatives of the left-hand sides:

$$\boldsymbol{Ax}_t + \boldsymbol{f_x}(\boldsymbol{x}_{t-1}, \boldsymbol{y}_{t-1}) = -\gamma(\boldsymbol{x}_t - \boldsymbol{x}_{t-1}) \tag{2.16}$$

$$\boldsymbol{Ay}_t + \boldsymbol{f_y}(\boldsymbol{x}_{t-1}, \boldsymbol{y}_{t-1}) = -\gamma(\boldsymbol{y}_t - \boldsymbol{y}_{t-1}) \tag{2.17}$$

We can now solve for position vectors iteratively:

$$\boldsymbol{x}_t = (\boldsymbol{A} + \gamma\boldsymbol{I})^{-1}(\gamma\boldsymbol{x}_{t-1} - \boldsymbol{f_x}(\boldsymbol{x}_{t-1}, \boldsymbol{y}_{t-1})) \tag{2.18}$$

$$\boldsymbol{y}_t = (\boldsymbol{A} + \gamma\boldsymbol{I})^{-1}(\gamma\boldsymbol{y}_{t-1} - \boldsymbol{f_y}(\boldsymbol{x}_{t-1}, \boldsymbol{y}_{t-1})) \tag{2.19}$$

The matrix $(\boldsymbol{A} + \gamma\boldsymbol{I})$ is also a pentadiagonal and banded matrix, so it can be inversed by LU decompositions in *O(n)* time. When the snake reaches its equilibrium, the time derivative vanishes and the above equations come to solution.

2.2.5 Advantages and disadvantages

It was shown that snakes are active contour models with the ability to handle many different visual problems in a unified way. This is mainly due to the fact that snakes are based on the minimization of a modular energy; many different energy terms may be added to the total energy functional, according to the specific application. For example, if we want the snake to be attracted to areas of high green colouring, we can simply add an appropriate term that has small values in “high-green” pixels.

Another advantage of the snake model, as it was proposed by Kass et al., is that it allows the user to contribute to the snake movement. The user can place one or more snakes near target objects and interfere to their movement if needed. Thus a more efficient detection can be achieved.

Nevertheless, the ability of user-interaction could also be regarded as disadvantage. The original intention might be a user-guided model, however there are many applications that require unaided visual systems.

Even in the case though that the external constraint forces are tolerable, it is necessary to be able to model them in a functional. Moreover, they also have to be differentiable, due to the variatonal approach to energy-minimization. The same condition stands for the image forces too.

Furthermore, there is no clear way to define the weights and hence the relative importance of the energy terms. A bad adjusting could result, for example, in a snake that keeps contracting (or expanding) even after “finding” the target.

The fact that the movement of a snaxel is proportional to the magnitude of the internal energy could also create a problem. For example, if a snaxel has large elastic energy, it will jump to its new position and overrun an edge that may lie in between.

Finally, the energy-minimization process is rather slow. This makes difficult the real-time implementation of the model in the case of tracking.

2.3 A dynamic programming approach

The problems related to the variational approach of the original model created the necessity of finding new methods for energy minimization. Amini et al. proposed a discrete multistage-decision process, using dynamic programming [25]. This approach concerns only the minimization task; the snake energy to be minimized is still given by equation (2.8):

$$E^*_{snake} = \int_0^1 (E_{int}(\boldsymbol{v}(s)) + E_{ext}(\boldsymbol{v}(s)))ds = \int_0^1 (\frac{1}{2}(\alpha(s)\,|\,\boldsymbol{v}_s(s)\,|^2 + \beta(s)\,|\,\boldsymbol{v}_{ss}(s)\,|^2) + E_{ext})ds$$

2.3.1 Energy minimization

The discrete form of the above equation can be written as:

$$E^*_{snake} = \sum_{i=1}^{n} \left(\frac{1}{2}(\alpha_i\,|\,\boldsymbol{v}_i - \boldsymbol{v}_{i-1}\,|^2 + \beta_i\,|\,\boldsymbol{v}_{i-1} - 2\boldsymbol{v}_i + \boldsymbol{v}_{i+1}\,|^2) + E_{ext}(\boldsymbol{v}_i) \right) \tag{2.20}$$

The minimization problem can be solved through a discrete multistage-decision process: starting from the initial snaxel on the contour, at each one of a finite set of stages $(i_1, i_2, ..., i_n)$ (that correspond to the *n* control points), a decision has to be made from a finite set of possible decisions.

We can start with the comprehension of the problem, making an assumption for the shake of simplicity; if we assume that the snake energy does not include the bending-energy (the second term of equation (2.20)) then the minimization problem can be resembled to the minimization of a function like the following:

$$E(u_1, u_2, ..., u_n) = E_1(u_1, u_2) + E_2(u_2, u_3) + ... + E_{n-1}(u_{n-1}, u_n) \tag{2.21}$$

where each variable is allowed to take only one of *m* possible values. In a discrete dynamic programming approach, every u_i corresponds to the state variable in the *i*th decision stage. The solution yields a sequence of functions of one variable (optimal value functions), $\{s_k\}_{k=1}^{n-1}$, where for obtaining each s_k, a minimization is performed over a single dimension. In the case of equation (2.21), s_k is of the general form:

$$s_k(u_{k+1}) = \min_{u_k}\{s_{k-1}(u_k) + E_k(u_k, u_{k+1})\} \tag{2.22}$$

For example, if $n=5$ in equation (2.21), then (2.22) gives:

$$\begin{aligned}
&s_1(u_2) = \min_{u_1} E_1(u_1,u_2) \\
&s_2(u_3) = \min_{u_2}\{s_1(u_2) + E_2(u_2,u_3)\} \\
&s_3(u_4) = \min_{u_3}\{s_2(u_3) + E_3(u_3,u_4)\} \\
&\min_{u_1,\dots,u_5} E(u_1,u_2,u_3,u_4,u_5) = s_4(u_5) = \min_{u_4}\{s_3(u_4) + E_4(u_4,u_5)\}
\end{aligned} \tag{2.23}$$

In the same way, the equation (2.20) without the bending-energy term, gives a set of equations of the following general form:

$$s_k(\mathbf{v}_{k+1}) = \min_{\mathbf{v}_k}\{s_{k-1}(\mathbf{v}_k) + \frac{\alpha_k}{2}|\mathbf{v}_{k+1} - \mathbf{v}_k|^2 + E_{ext}(\mathbf{v}_k)\} \tag{2.24}$$

The process uses two matrices for storing the results of each stage. An energy-matrix for the optimal value functions, s_k, and a position-matrix for the values of $\mathbf{v}_k$ that minimize (2.24). After the formulation of the two matrices, the contour of minimum energy is found by using the backward method of solution for discrete dynamic programming problems: the energy of the resulting contour is $E_{\min}(t) = \min_{\mathbf{v}_n} s_{n-1}(\mathbf{v}_n)$. Going back in the position-matrix, the contour is found. This procedure corresponds to a single iteration. The iterative process continues until $E_{\min}(t)$ stops changing with time. Supposing n control points and m possible directions (decisions) for each point, the time complexity is $O(nm^2)$ for each iteration and the storage requirements are $n \times m$.

An important advantage of this algorithm is that it allows the enforcement of hard constraints to the solution (they are called "hard" because they cannot be violated). An example of such a constraint is a condition that demands that two adjacent snaxels cannot come closer to each other than a distance d (inequality constraint). In this case, the distance between adjacent points is also computed during the process; if a point $\mathbf{v}_k$ minimizes the function $s_k(\mathbf{v}_{k+1})$, but does not satisfy the constraint, it is rejected and substituted by the next best point that satisfies the condition. If there is no such a point, the algorithm terminates.

Having the simple case (where the bending-energy is missing) in mind, we can proceed to the solution of the overall energy of equation (2.20). Now, a two-element vector of state variables, $(\boldsymbol{v}_{k+1}, \boldsymbol{v}_k)$, is fixed and the general case of the optimal value function is:

$$s_k(\boldsymbol{v}_{k+1}, \boldsymbol{v}_k) = \min_{\boldsymbol{v}_{k-1}} \{ s_{k-1}(\boldsymbol{v}_k, \boldsymbol{v}_{k-1}) + \frac{\alpha_k}{2} | \boldsymbol{v}_k - \boldsymbol{v}_{k-1} |^2 + \\ \frac{\beta_k}{2} | \boldsymbol{v}_{k+1} - 2\boldsymbol{v}_k + \boldsymbol{v}_{k-1} |^2 + E_{ext}(\boldsymbol{v}_k) \} \tag{2.25}$$

In this case, the dynamic programming table has m^2 entries for each stage; these correspond to the m^2 possible two-element vectors of state variables. Thus, the time complexity increases to $O(nm^3)$ and the memory requirements become $n \times m^2$. Since there is no interaction between the m^2 entries of each stage, their energy values could be computed in parallel. This would require m^2 processors but would reduce complexity to $O(n)$.

2.3.2 Advantages and disadvantages

The above "time-delayed" discrete dynamic programming algorithm has a significant advantage over the previous variational approach: there is no need of computing high-order derivatives. The only ones required are the first-order and second-order derivatives of the internal energy and the gradient in the image energy functional. This fact implies numerical stability. Furthermore, the external forces don't have to be differentiable.

Continuing the comparison, we can note that the Amini's method works directly on the discrete grid, whereas the original model computes new point coordinates as real numbers. The latter allows points to fall between the discrete coordinates.

Moreover, the dynamic programming approach allows hard constraints to be easily included in the model. When these constraints are violated, the algorithm stops, so they can be used in order to prevent undesirable situations.

On the other hand, the computational cost of the algorithm is very high. Given a snake of *n* control points and a window of *m* possible positions, the complexity is $O(nm^3)$ in time and $O(nm^2)$ in memory requirements.

Finally, some drawbacks of Kass's model still exist. There is no guideline at all for the evaluation of the energy weights. Furthermore, the elasticity energy is prone, due to its formulation, to make the snake shrink (or expand) continuously, as it minimizes (or maximizes) the distance of adjacent snaxels.

2.4 "Fast snake" – a greedy approach

Williams and Shah introduced a similar (to the dynamic-programming algorithm) approach [26], in the sense that the new positions of the control points are sought into a window around them. During each iteration of the algorithm, each snaxel moves to that position of its neighborhood (window), which minimizes its energy level. Unlike the previous methods, the snaxels move asynchronously, i.e. the movement of each snaxel takes place during the iteration rather at the end of it, affecting, in this way, the movement of the following snaxels. Despite being greedy, the proposed algorithm ensures faster convergence than the dynamic-programming approach, while in the same time, preserves the advantages of numerical stability and inclusion of hard constraint. The energy-model does not include terms due to external constrained forces (although it would be able to do so) and the basic snake energy is now written as:

$$\begin{aligned} E^*_{snake} &= \int_0^1 (E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s)))ds \\ &= \int_0^1 (\alpha(s)E_{cont}(\mathbf{v}(s)) + \beta(s)E_{curv}(\mathbf{v}(s)) + \gamma(s)E_{image}(\mathbf{v}(s)))ds \end{aligned} \tag{2.26}$$

where $\alpha(s)$, $\beta(s)$ and $\gamma(s)$ are weights that control the relative influence of each term. Thus, their relative rather than absolute values are important.

2.4.1 Internal energy

The first two terms of equation (2.26) form the internal energy of the snake and they correspond to the elastic and bending energy of the original model respectively.

The first-order term in the integrand of (2.8), i.e. the elastic energy, which in discrete form is written as $|\boldsymbol{v}_i - \boldsymbol{v}_{i-1}|^2$, has a serious drawback. As mentioned before, the minimization of this term is equivalent to the minimization of the distance between subsequent snaxels, causing the snake to shrink consecutively. This is getting even worse in the greedy approach, as the movement is asynchronous and is based on local decisions.

In this model, the above term, which is called here "the continuity energy", is made to include the average distance, $\bar{d}$, between the adjacent control points. For each snaxel $\boldsymbol{v}_i$ we have:

$$E_{cont}(\boldsymbol{v}_i) = \bar{d} - |\boldsymbol{v}_i - \boldsymbol{v}_{i-1}| \tag{2.27}$$

This equation still satisfies the desirable first-order continuity of the curve but in the same time ensures that the adjacent snaxles are almost equidistant. The average distance, $\bar{d}$, is updated at the end of each iteration and is affected by the overall movement of the snake due to the minimization of the total energy (equation (2.26)). Thus, depending on the value of $\bar{d}$, the snake can still expand or contract, but in a more controllable way, preventing the bunching-up on strong segments of the contour.

The second term of equation (2.26) corresponds to the second-order term in the integrand of equation (2.8) and is actually a measure of the curvature of the contour (in this model it is called "the curvature energy", E_{curv}). The mathematical definition of curvature is $\frac{d\theta}{ds}$, where θ is the angle between the positive x-axis and the tangent vector to the curve.

There are many different ways to approximate the curvature of a discrete contour [26]. The greedy snake model uses the square of the difference of the vectors $\vec{\boldsymbol{u}}_i = (\boldsymbol{v}_i - \boldsymbol{v}_{i-1}) = (x_i - x_{i-1}, y_i - y_{i-1})$ and $\vec{\boldsymbol{u}}_{i+1} = (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) = (x_{i+1} - x_i, y_{i+1} - y_i)$, where $\boldsymbol{v}_i$

is the snaxel under consideration and $\boldsymbol{v}_{i-1}$ and $\boldsymbol{v}_{i+1}$ are the preceding and following snaxels respectively. Thus, the curvature energy is written:

$$E_{curv}(\boldsymbol{v}_i) = |\vec{\boldsymbol{u}}_{i+1} - \vec{\boldsymbol{u}}_i|^2 = |\boldsymbol{v}_{i-1} - 2\boldsymbol{v}_i + \boldsymbol{v}_{i+1}|^2 \tag{2.28}$$

This is a reasonable and quick measure of curvature. Note that it takes into account, not only the difference in the directions of the two vectors, but also the difference in length. Thus, if the above three snaxels are not evenly spaced, the curvature will be larger.

Another approach, which eliminates this last feature, uses the difference of the normalized vectors:

$$curv(\boldsymbol{v}_i) = \left| \frac{\vec{\boldsymbol{u}}_{i+1}}{|\vec{\boldsymbol{u}}_{i+1}|} - \frac{\vec{\boldsymbol{u}}_i}{|\vec{\boldsymbol{u}}_i|} \right|^2 \tag{2.29}$$

The length of $curv(\boldsymbol{v}_i)$ is given by $2\sin(\theta/2)$, where θ is the difference between the directions of the two vectors (Figure 2.7). Now the curvature depends only on this difference, but its computation is slower than the one used in (2.28).

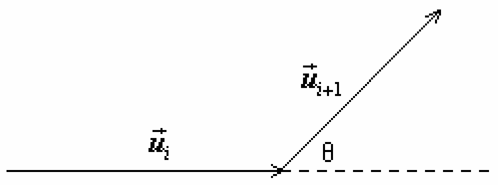


Figure 2.7: Difference in direction of two vectors $(\theta \in [0, \pi])$.

For every snaxel, both E_{cont} and E_{curv} are computed for every possible position in the neighborhood of this snaxel and then the values are normalized to [0, 1] by dividing by the largest respective value in the neighbourhood. This normalization is made so that the range of both terms of the internal energy is the same. In this way, the relative importance of the two terms is absolutely defined by the ratio of α and β.

The polarity of these weights has the same significance as in section 2.2.1. Their magnitude though affects the gravity of each term but not the rate of movement.

2.4.2 Image energy

Unlike the original snake model, the image energy of the greedy snake consists of only one term: the magnitude of the gradient of the image. For the computation of the gradient any appropriate mask and smoothing filter can be used.

The value of image energy must also be normalized to [0, 1] in consistency with the terms of internal energy. The magnitude of the gradient is an integer value between 0 and 255. The normalization of this value by dividing by 255 or by the largest value in the neighbourhood isn't very accurate. If, for example, the largest value in the neighborhood is 240 and another position has a gradient of 225, the division with 240 will give 1 and 0.94 respectively, which doesn't reflect the exact (quite large) difference between 240 and 225. The model uses a more efficient normalization formula:

$$P'_{image}(\mathbf{v}_i) = \frac{P_{image\_min} - P_{image}(\mathbf{v}_i)}{P_{image\_max} - P_{image\_min}} \tag{2.30}$$

where $P_{image}(\mathbf{v}_i)$ is the real value of the gradient magnitude, $P'_{image}(\mathbf{v}_i)$ is the normalized value and P_{image_min}, P_{image_max} are the minimum and maximum values of gradient magnitude in the neighborhood. In the above case, supposing a minimum value of 140, this formula will give -1 and -0.85 respectively. Note that equation (2.30) yields a negative image energy, which has smaller values in larger values of gradient, exactly as in the original model. Furthermore, the algorithm sets *min* equal to *max-5* when $max - min < 5$. This prevents large differences when close values of gradient are normalized. For example, for three subsequent values 50, 51 and 52 (in a supposed neighbourhood of three positions), the direct application of (2.31) will give 0, -0.5 and –1 respectively, i.e. quite large differences; if we set $min = max - 5 = 47$, we will get –0.6, -0.8 and –1 respectively, i.e. smaller differences.

Finally, the weighting coefficient *γ* in equation (2.26) has a behaviour similar to that of weights *α* and *β*. The ratio of *α*, *β* and *γ* defines the gravity of each term in the overall energy. Usually γ is set larger than the other weights. Setting γ equal to 0 results in a snake not affected by the image forces.

2.4.3 Energy minimization

The minimization of the snake energy is performed locally and asynchronously. At each iteration of the algorithm, starting from the first snaxel, a square neighbourhood, which has this snaxel as center, is considered (Figure 2.8). The three energy terms are computed and normalized for every possible position in this neighbourhood. The normalized values are then multiplied by the respective weights and the addition of the results gives the total energy for each position. The position with the lower overall energy is found and the snaxel is moved there. This process goes on for each subsequent control point. As the computation of the internal energy takes into account also the preceding and following snaxels, we have to note that in closed snakes (the usual case) $\boldsymbol{v}_0 = \boldsymbol{v}_n$ and $\boldsymbol{v}_{n+1} = \boldsymbol{v}_1$, where n is the number of snaxels (modulo n index arithmetic). The first snaxel is reexamined at the end, in order to take into account the movement of the last snaxel.

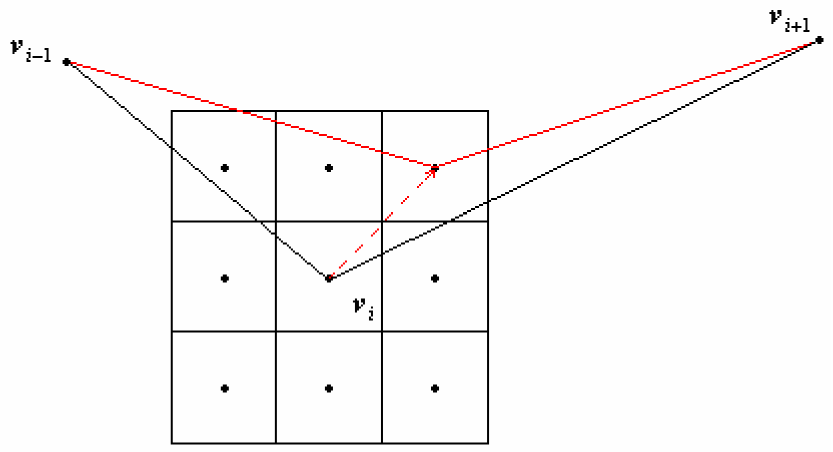


Figure 2.8: Snaxel $\boldsymbol{v}_i$ is moving to that position of its neighbourhood that minimizes its total energy. The new contour segment is in red.

At the end of each iteration the algorithm includes a significant and rather simple method for choosing an appropriate value of β_i for each snaxel $\boldsymbol{v}_i$ in the next iteration step. Firstly, the curvature of the new snaxels is computed using equation (2.29). As mentioned before, this measure of curvature is computationally slow, but it's only used n times at each iteration. Furthermore, this measure offers an easy way to define a meaningful threshold for the size of curvature, as its length is given by $2\sin(\theta/2)$. A good choice for this threshold is 0.25, which corresponds approximately to 29^{o}. If the curvature of a control point is greater than this threshold and greater than the curvature of the points before and after it in the contour, and moreover, if the gradient of the point is above a gradient-threshold (i.e. on a strong edge), then β is set equal to 0, for the specific control point, in the next iteration. This allows the snake to form a corner at that point in the next iteration, by eliminating from the overall energy the curvature term, which is expected to be large. This process is repeated at the end of the next iteration and if the above conditions aren't satisfied, β gets back its initial value. The overall algorithm terminates when the number of the control points moved during an iteration is smaller than a threshold. A summary of the algorithm is given below in pseudo-code:

```
Initialize $\alpha_i$, $\beta_i$ and $\gamma_i$ for all $i$
do
  // loop for moving the snaxels to new locations
  for i = 1 to n+1        // where $\boldsymbol{v}_{n+1} = \boldsymbol{v}_1$. The first snaxel is first and last one processed
       $E_{min}$ = BIG
       for j = 1 to m    // m is the size of neighbourhood
            $E_j = \alpha_i E_{cont,j} + \beta_i E_{curv,j} + \gamma_i E_{image,j}$
            if $E_j < E_{min}$ then
                 $E_{min} = E_j$
                 $j_{min} = j$
            end
       end
       if $j_{min}$ not current location then
            move snaxel $\boldsymbol{v}_i$ to location $j_{min}$
            ptsmoved + = 1 // count points moved
       end
  end
  // decision for the value of β
  for i=1 to n
        $c_i = | \vec{u}_{i+1} / | \vec{u}_{i+1} | - \vec{u}_i / | \vec{u}_i | |^2$
  end
  for i = 1 to n
        if $c_i > c_{i-1}$ and $c_i > c_{i+1}$          // if curvature is greater than neighbours
              and $c_i$ > threshold1              // and curvature is greater than threshold1
              and mag($\boldsymbol{v}_i$) > threshold2       // and magnitude is above threshold
        then
              $\beta_i = 0$
        end
  end
until ptsmoved < threshold3
```

2.4.3 Advantages and disadvatages

The greedy snake retains all the advantages of both the original model and the dynamic programming approach. The modular energy function allows the usage of any helpful term (differentiable or not), including hard constraints and even user-guided constraints. The greedy snake model though can work efficiently without the influence of constraint forces coming from the user and this makes it more suitable for unaided visual systems. Furthermore, like the dynamic programming approach, it doesn't require the computation of high-order derivatives, operates on the discrete grid and avoids overruning phenomena due to its window searching strategy.

Additionally, the greedy model makes easier the decision for the appropriate values of the weights *α, β* and *γ*. Only a zero weight is important as absolute value, since it eliminates the corresponding term in the energy functional. In all other cases, only the ratio of the parameters is important as it defines the gravity of each term. Two different sets of coefficients with the same ratio, for example, $\alpha_i = 0.7, \beta_i = 0.5, \gamma_i = 1$ and $\alpha_i = 1.4, \beta_i = 1, \gamma_i = 2$ will have the same effect on the movement of snaxel $\boldsymbol{v}_i$. In the original model though, the second set would move the snaxel to a different position than the first set would. Moreover, the greedy algorithm implements a simple but efficient method for getting a variable *β*. This is very useful for handling in a unified way target shapes that have both corners and smooth areas.

Another advantage over the other two approaches comes from the different formulation of the continuity energy. The contraction/expansion of the snake due to this energy is more controllable and it leads to a contour with almost equidistant control points.

It is important that all the above advantages are achieved by an algorithm that is fast, in spite of its greedy character. The complexity in time is *O(nm)*, much faster than the $O(nm^3)$ of dynamic-programming approach, where *n* is the number of control points and *m* is the size of the neighbourhood. Moreover the memory requirements are only *O(m)*, as the snaxels are moving asynchronously.

Unfortunately, the greedy algorithm doesn’t solve some generic problems of snakes. The most important of them, the sensitivity in the original contour and the handling of concave contours, will be examined in detail in the following chapter.

Chapter 3: Two major problems of the basic snake algorithm

3.1 Introduction

After the general definition of the snake model and the examination of the three basic approaches to energy formulation and minimization, a closer look at the effectiveness of snakes must be taken. There are two basic problems that exist in all these approaches: sensitivity to the position and shape of the original contour, and disability of coping with concavities. The first problem may be due to the lack of desirable features (e.g. edges) or the existence of misleading features between the initial and the target contour. The second one is caused by the lack or weakness of forces that could push the snake towards concave areas.

The two problems may become clearer through the following simple example (Figure 3.1). The target is a line-drawing of a U-shaped object on a smooth background (1). Note the boundary concavity on the top. The potential external force field that corresponds to an external energy of the form $-\nabla(G_\sigma(x,y) * I(x,y))$, where G_σ is a Gaussian of standard deviation σ and $\sigma=1$, is shown in (2). A close-up of this field at the concave area is taken in (3). Finally, the subsequent curves of the snake towards the final contour are superimposed on the original image (4). In (2) we can see that the external forces die out quite rapidly far away from the target boundary; if an initial contour is placed far from the target, it is unlikely to converge under the lack of image forces. A greater value of σ would increase the capture range but it would also lead to a deformation of the shape (blurring). But even if the contour is initialized into the capture range, it won't be able to converge accurately to the concave part, as shown in (4). An explanation can be found by observing the close-up in (3): within the concave region, the forces point towards the sides of the concavity rather than towards the bottom of it. Hence, only an initial contour that is close enough to the target in both position and shape (i.e. having points within the concavity as well) can result in a more accurate fit. This is very restrictive though.

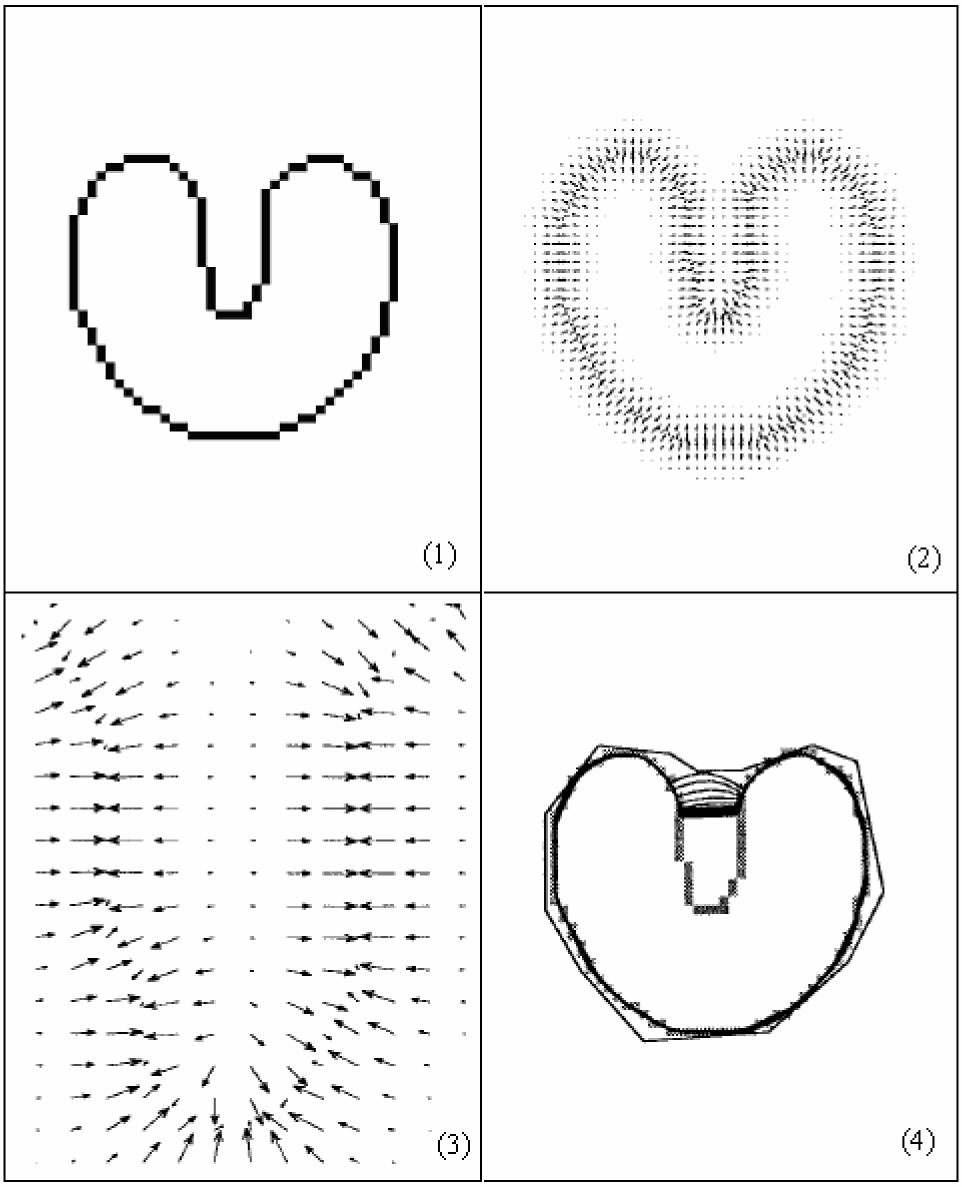


Figure 3.1: An inappropriate force field can lead to a poor convergence [27].

Many methods have been proposed for the solution of the above problems and the most of them were developed in the variational framework. The computational cost of the dynamic programming approach has limited the research for solutions that use dynamic programming. Therefore, in this chapter, we will see some methods proposed in the variational framework, and we will examine an interesting proposal, applied on the greedy snake model. This will serve as an introduction to the implementation details of the snake model of the current thesis, in the following chapter.

3.2 Variational approach framework

Most of the proposed solutions were developed in the variational framework. Some of them are outlined in this section.

Cohen [28] copes with the first problem by introducing a 'balloon model', where inflation/deflation normal forces push/pull constantly the snake towards the target object. These forces have the form $F = k_1\vec{\boldsymbol{n}}(s) - k\frac{\nabla P}{\|\nabla P\|}(\boldsymbol{v}(s))$, where $\vec{\boldsymbol{n}}(s)$ is the unit vector normal to the curve at point $\boldsymbol{v}(s)$, k_1 is the amplitude of the force, k is a constant, slightly larger than k_1 so that an edge point can stop the inflation force, and $P = -|\nabla I|^2$. In an extended version of the model [29], a Euclidean (or Chamfer) map is used to increase the capture range of the snake (Figure 3.2), as referenced in [27]. This model though doesn't solve the problem of fitting to concave shapes.

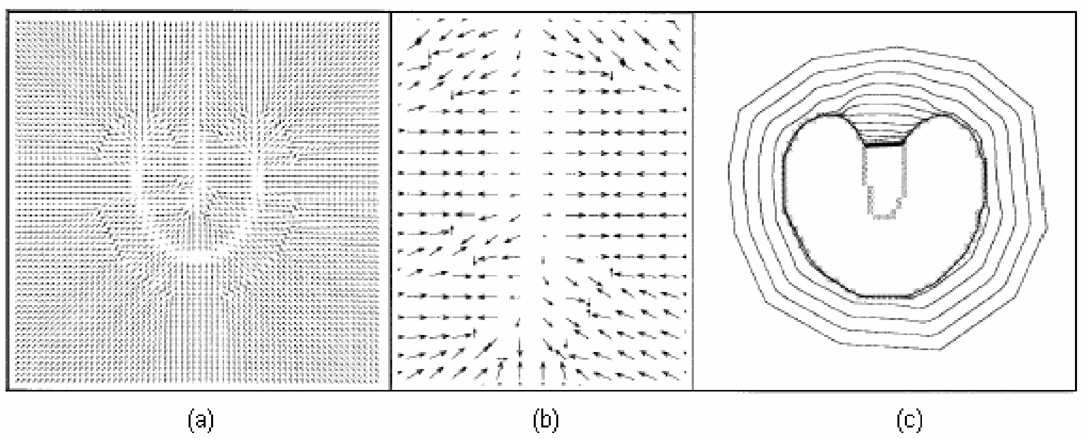


Figure 3.2: The capture range is improved (a), but the potential forces into the region of concavity (b), still cannot guide the snake to the bottom of it (c) [27].

Other similar approaches add to the snake model constant forces in the normal direction of the snake, as noted in [30]. Xu et al. [31] uses such a force to help the evolution of the snake towards the target object, after suppressing its internal resistance in the normal direction. Berger [32] constructs a "wrapping" force in an open snake model in order to make the snake move towards the target after "locally growing" from a given starting point. However, all these forces are constant normal forces that depend only on the particular geometric property of a snake, i.e. either its delimited area or its distance to the target contour. Hence, they may be too weak to reach the target or too strong, resulting in edge-overruns.

Wong et al. [33] proposes a “blown” force, which is also dependent on the specific geometric property, but both its amplitude and direction are determined adaptively. In this model, a first estimation of the target contour is found using the original snake model and then this estimation is corrected through a split-and-merge process. During the split process, the estimated contour is divided into two types of segments: these that are regarded to be located on the target boundary and those that are regarded to be far from it and need further movement. Each of the segments of the second type forms an open snake with fixed end-points, and an additional force is applied to this open snake, in order to further push it towards the target boundary. The direction of the force is determined by comparing the local regions on both sides of the segment; the force has to point towards the region with the greater image energy. The amplitude of the force is designated through a scaling function in a way that it is maximum at the mid-point of the segment and decreases gradually on both sides, resulting to be zero at the fixed end-points. When the split process is completed, old and new segments are re-connected in a merge process. The new contour is examined again and the split-and-merge process is repeated, if needed. The whole method has encouraging results but also some drawbacks. For example, the split-and-merge process is quite complex and the determination of the direction of the force is not always accurate.

Xu and Prince [27] introduce a model that replaces the original image force by a new type of external force field, which is called the “gradient vector flow” (GVF) field $\mathbf{v}(x,y)=(u(x,y),\upsilon(x,y))$. This field is defined by minimizing the energy functional:

$$\varepsilon=\iint \mu({u_x}^2+{u_y}^2+{\upsilon_x}^2+{\upsilon_y}^2)+|\nabla f|^2|\mathbf{v}-\nabla f|^2\, dxdy \tag{3.1}$$

In the above integrand, μ is a regularization parameter governing the trade-off between the two terms, $u_x, u_y, \upsilon_x, \upsilon_y$ are the partial derivatives of u and υ, and f is any possible edge map that has large values near the edges. In almost homogenous regions of the original image, where the image intensity is almost constant, the gradient of edge map, ∇f, has small values and the energy in (3.1) is dominated by the sum of the squares of the partial derivatives of the vector field (first term). This yields a slowly varying, non-zero field in smooth areas of the image. On the other hand, in regions of strong edges, ∇f has large values and the energy, ε, is minimized by setting $\mathbf{v} = \nabla f$. These two cases lead together to an increment of the capture range. Furthermore, from the Euler equations derived from (3.1),

$$\mu \nabla^2 u - (u - f_x)(f_x^2 + f_y^2) = 0 \tag{3.2}$$

$$\mu \nabla^2 \upsilon - (\upsilon - f_x)(f_x^2 + f_y^2) = 0 \tag{3.3}$$

we can see that, in homogenous regions (where the partial derivatives of the edge map, f_x, f_y, are zero), u and υ are both determined by Laplace's equation. As referenced in [27], the result is that the GVF field is interpolated from the region's boundary, reflecting a kind of competition among the boundary vectors, and yielding vectors that can point to boundary concavities (Figure 3.3).

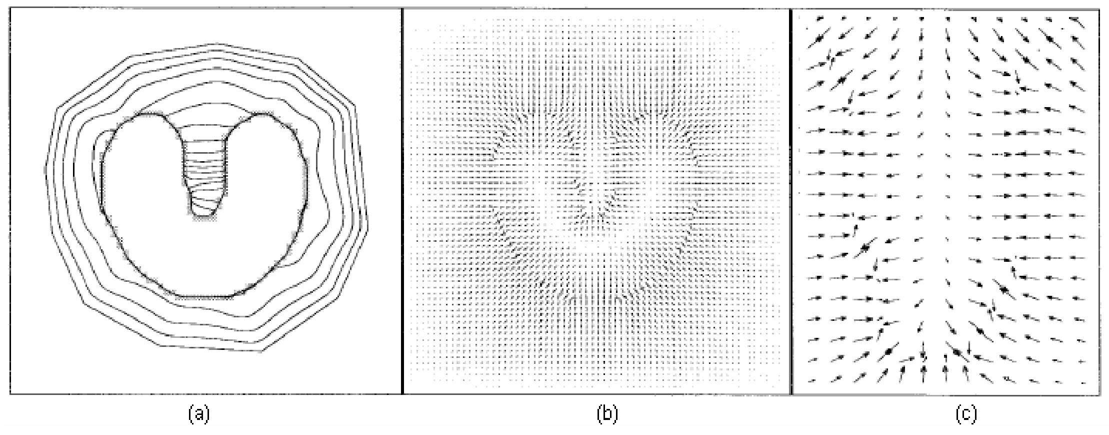


Figure 3.3: (a) Convergence of a snake using GVF external forces. (b) The potential force field. (c) A close-up of (b) within the boundary concavity [27].

The resulting potential field seems quite functional in the above figure but in real cases, where many objects and noise are present, it can be really misleading. Furthermore the computation of the GVF field is very time-consuming.

3.3 Greedy snake framework: Attractable snakes

L. Ji and H. Yan [30] proposed a snake model, based on the greedy algorithm, with promising results in handling both of the known problems. The core of the model is a direct feedback mechanism, which introduces an additional adaptive force. This force manages to move the snake to the target when the other (internal and external) forces seem inadequate. The proposed model also includes an efficient edge detector for the computation of the potential external energy field, a composite convergence criterion for the minimization process, and an adaptive interpolation scheme that increases the flexibility of the snake contour.

3.3.1 Energy formulation

The attractable snake model is based on the greedy algorithm. The energy to be minimized is given, for a discrete contour, by the equation:

$$E_{snake}(\mathbf{v}) = \sum_{i=1}^{n} \left[\alpha(i) E_{cont}(\mathbf{v}_i) + \beta(i) E_{curv}(\mathbf{v}_i) + \gamma(i) E_{image}(\mathbf{v}_i) - f_{db}(i) \nabla P_{field}(\mathbf{v}_i) \boldsymbol{n}(i) \right] \qquad (3.4)$$

The three first terms in the above sum constitute the energy functional of the original greedy snake model. E_{cont}, E_{curv} and E_{image} are computed and normalized as mentioned in the previous chapter. The new term implements the feedback mechanism, where ∇P_{field} is the difference in the potential energy of a desired image feature (e.g. edge) between the target and the snake, f_{db} is a weight that controls the gain of the feedback mechanism, and $\boldsymbol{n}(i)$ is the projection of the normal direction of the snake at a snaxel. By determining the orientation of the normal direction, we can make the snake either contract or expand. The value of ∇P_{field} is computed and normalized as follows:

$$\nabla P_{field}(\mathbf{v}_i) = \begin{cases} \dfrac{P_{\max} - \sum_{j=1}^{8} P_{field}(\mathbf{v}_{i,j})/8}{P_{\max}}, & if\, P_{field}(\mathbf{v}_{i,0}) < P_{level} \\ 0 & , \ if\, P_{field}(\mathbf{v}_{i,0}) \geq P_{level} \end{cases} \qquad (3.5)$$

where P_{field} is the potential energy field and P_{max} and P_{level} are its maximum and threshold values respectively. In [30] it is implied that the potential field may result from a pre-processing stage or prior knowledge in order to include information about the desired image feature (e.g. edges) of only the target object. When such a prior knowledge is not possible the potential field involves the available image. For example, the attractable snake model in [30] uses as P_{field}, the magnitude of the intensity gradient of the entire image. Furthermore, it proposes an edge detector scheme for the computation of this gradient (see next section for details). In the above equation we assume a 3x3 neighbourhood, where $\boldsymbol{v}_i = \boldsymbol{v}_{i,0}$ and $\{\boldsymbol{v}_{i,j} \mid j = 1,2,...,8\}$ are the eight neighbours of $\boldsymbol{v}_i$.

A closer look at equation (3.5) reveals the functionality of the feedback mechanism and the adaptive character of the 'force' that this mechanism introduces. When a given snaxel is far away from the target object, the potential field has a small value at this point, smaller than the threshold P_{level}. The potential field is also small (approaching to zero) at the eight neighbours of the snaxel. Hence, ∇P_{field} approaches to its maximum value, which, in combination with the negative sign of the feedback term (eq. 3.4), gives a maximum push along the normal direction of the snake at the given snaxel (Figure 3.4a and 3.4b). As the snaxel gets closer to the target contour, its neighbourhood starts to include some desired features (e.g. edges) and so the sum in equation (3.5) gets larger values. This leads to smaller values of ∇P_{field} and consequently to a reduction of the push (Figure 3.4c). Finally, when the P_{field} on the snaxel position is greater than P_{level} (i.e. when the snaxel is very close or on the target contour), ∇P_{field} is zero and there is no push at all. In general, we can say that both the image and the feedback term are based on the potential image energy (edges); the difference is that the image term uses only this potential energy, whereas the feedback term uses the difference in potential energy between the snake and the desired target contour, and combines this difference with a geometrical property of the snake, that is the normal direction of the snake at a snake point.

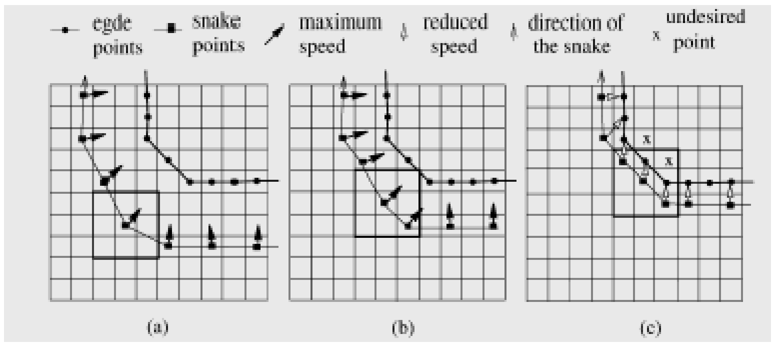


Figure 3.4: Movement of a snake under the control of the feedback mechanism [30].

3.3.2 Potential field: edge image

The potential field of an image is determined by the image feature that is responsible to attract the snake to the target object. In the case where this feature is the object boundary (the case in this thesis), the potential field is actually an edge image. An effective edge image requires both good edge-detection and noise suppression. The attractable snake model proposes a new edge detector, which seems to be more efficient than the usual edge detectors [30].

In the Canny detector, the gradient is calculated using the derivative of the intensity or the derivative of a Gaussian filter, and the result is smoothened via two 1D Gaussian filters. The method uses then two thresholds, to detect strong and weak edges, and recognizes weak edges only if they are connected to strong edges. The detector is good at noise suppression but not at the weak edge identification. Furthermore, it gives a one-pixel-strength response to a step edge, which can lead to overrunning problems.

The Sobel detector uses two separable filters in order to detect edges in x and y directions:

$$\begin{aligned} S_x &= D_x \cdot A_y = [-1\ 0\ 1]^{\mathrm{T}} \cdot [1\ 2\ 1] \\ S_y &= A_x \cdot D_y = [1\ 2\ 1]^{\mathrm{T}} \cdot [-1\ 0\ 1] \end{aligned} \tag{3.6}$$

where D_x and D_y are the 1D simple difference operators and A_x and A_y are the inherent average smoothers of the Sobel detector, applied in the orthogonal direction. The detector can give a two or three-pixel response to a step edge but is not so good at noise suppression. Noise can be reduced by using a 2D Gaussian smoother before or after the Sobel detector, but this will also lead to image blurring.

The proposed edge detector is actually a combination of the above schemes. The two separate filters of (3.6) are first applied on the image, giving $ImgS_x$ and $ImgS_y$ respectively. Then, a 2D Gaussian filter splits into two directions x and y:

$$\begin{aligned} G_x &= \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \\ G_y &= \frac{1}{\sqrt{2\pi}\sigma} e^{-y^2/2\sigma^2} \end{aligned} \tag{3.7}$$

and the two smoothers are applied on the orthogonal directions of $ImgS_x$ and $ImgS_y$. Hence the intensity gradient is:

$$|\nabla I| = \sqrt{(ImgS_x \otimes G_y)^2 + (G_x \otimes ImgS_y)^2} \approx |ImgS_x \otimes G_y| + |G_x \otimes ImgS_y| \tag{3.8}$$

Noise can be further suppressed using a threshold:

$$P_{level} = Tg(|\nabla I|_{\max} - |\nabla I|_{\min}) + |\nabla I|_{\min} \tag{3.9}$$

where $|\nabla I|_{\max}$ and $|\nabla I|_{\min}$ are the maximum and minimum values of gradient respectively and T_g controls the trade-off between noise suppression (larger values of T_g) and identification of weak edges (smaller values of T_g). A value between 0.1 and 0.4 is good for coping with both cases. Note that P_{level}, as defined in (3.9), is also used in equation (3.5) for deciding whether a snaxel is close to the target contour or not.

The proposed scheme is claimed [30] to achieve larger edge strength over two or three pixels, greater noise suppression and better identification of weak edges than the Canny edge detector. An example is shown in figure 3.5. Furthermore, the blurring effects from the scale factor of the Gaussian filter are much less intense (Figure 3.6).

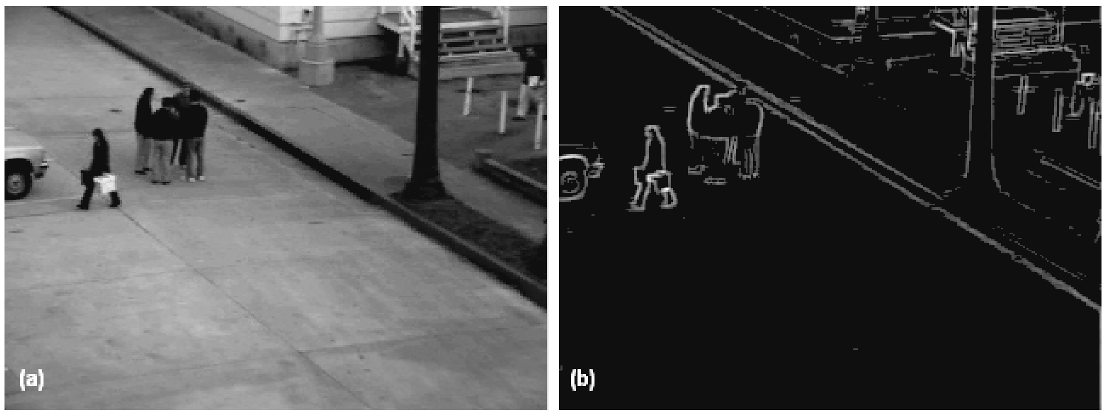


Figure 3.5: The proposed edge detector: (a) original intensity image and (b) edge image ($\sigma = 1.0$, $T_g = 0.2$, $P_{level} = 46.6$).

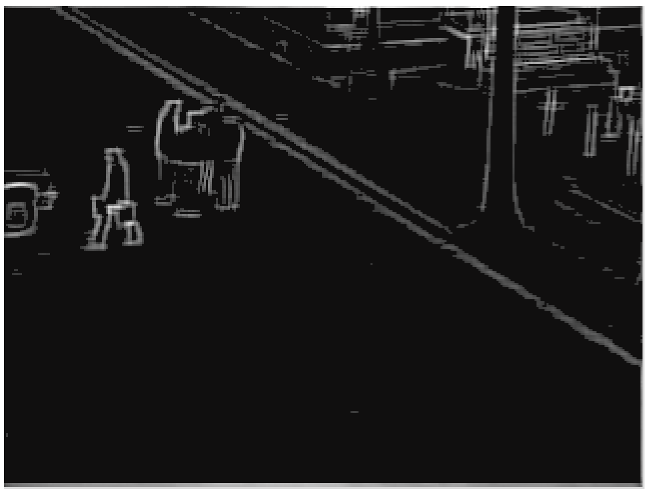


Figure 3.6: The proposed edge detector with $\sigma = 2.0$ ($T_g = 0.2$, $P_{level} = 40.2$).

3.3.3 Convergence criterion

As mentioned in the previous chapter, the original greedy snake model uses a quite simple convergence criterion. The algorithm terminates when the number of points moved in a single iteration becomes smaller than a predefined threshold (a small non-zero value). The attractable snake model proposes a compound-parameter criterion, which is based, apart from the number of the moved points, on two additional parameters: the snake contour length and the snake potential energy.

By intuition, as the snake approaches its equilibrium, it tends to be static (i.e. the number of points that have been moved tends to be zero), the contour length stops changing and the overall image energy of the snake tends to remain constant and large. Therefore, the proposed criterion needs to compute the variation of these parameters during the snake movement, i.e. the standard deviation of the parameters within a number of successive iterations of the algorithm. This number is given by a step size, T_{step} (in [30] a value around 10 is claimed to be a good value for T_{step}). The criterion is shown in the block diagram of the following figure:

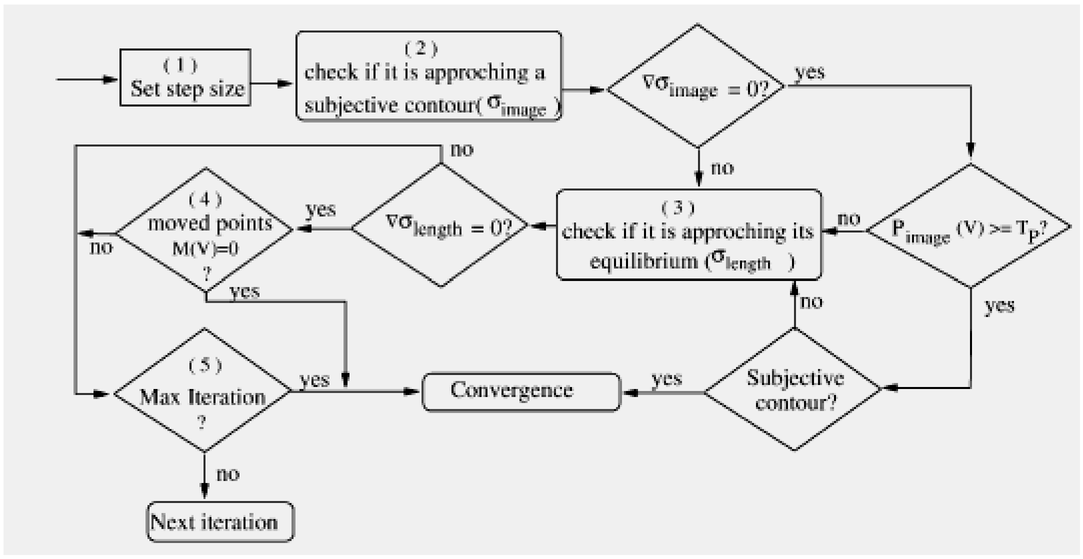

(1) Set step size
(2) check if it is approching a subjective contour(σ_{image})
$\nabla\sigma_{image} = 0?$
yes
no
$P_{image}(V) >= T_p?$
no
(3) check if it is approching its equilibrium (σ_{length})
$\nabla\sigma_{length} = 0?$
yes
no
(4) moved points M(V)=0 ?
no
yes
yes
no
Subjective contour?
yes
Convergence
yes
(5) Max Iteration ?
no
Next iteration

Figure 3.7: A compound-parameter convergence criterion [30].

This criterion also allows us to choose between a convergence to a subjective contour and a convergence to a concave/convex contour (Figure 3.8). When the snake is far away from the target, its image energy is very small and starts to increase as the snake approaches to the target. If the variation of the snake image energy is not negligible then the variation of the other two parameters is being checked; note that the number of maximum acceptable iterations of the algorithm is also added as a final condition. On the contrary, if the image energy is almost constant, then the snake is likely to lie on a subjective contour. [30] mentions that, based on the human visual perception, a snake with overall image energy, $P_{iamge}(V)$, equal or greater than a threshold T_p, which is equal to the 60% of the maximum possible image energy of the snake ($T_p = 0.6P_{image_{\max}}(V)$), is likely to lie on a subjective contour. Due to the normalization of the image energy to [0,1], we can say that $P_{image_{\max}}(V) \approx \sum_{i=1}^{n} \gamma(i)$, where n is the number of points of the snake V and γ is the weighting coefficient of the image energy. So if $P_{image}(V) \geq T_p$ and furthermore we have predefined that we're interested in a subjective contour, the algorithm terminates (convergence). Otherwise, the check moves on to the other parameters, as before.

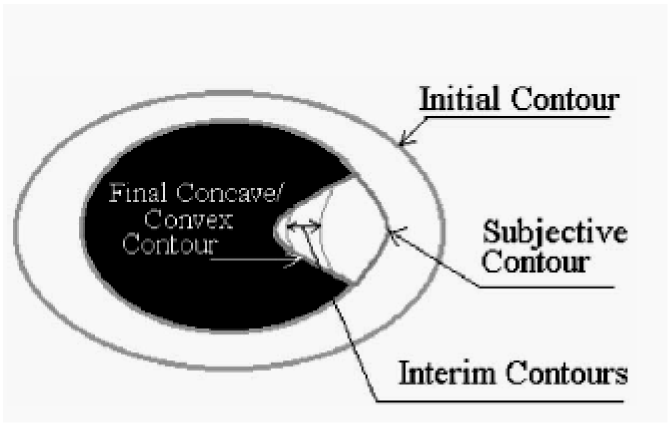


Figure 3.8: The convergence process of a snake [30].

3.3.4 Adaptive interpolation scheme

The attractable snake model uses a dynamic interpolation scheme for the discretization of the snake contour. The snake can start from a simple initial contour, consisted of a few snaxels, and during its deformation it can be re-sampled via a linear interpolation. In particular, before each iteration step of the algorithm, the average distance, $\overline{d}$, of subsequent snaxels is examined against a desired distance T_{gap} and if it's larger than T_{gap}, the contour is linearly interpolated. Every new snaxel retains the parameter settings of the old snaxels, between which it is inserted. The fact that the interpolation takes place at the beginning of each iteration step offers a dynamic way to take into account the geometric property of the snake at each time, which is expressed by $\overline{d}$. The value of threshold T_{gap} is crucial: if we want a coarse representation of the target contour or a snake that lies on a subjective contour then T_{gap} should be rather large, whereas lower values of T_{gap} are suitable for handling more complex shapes.

Chapter 4: Implementation details and results

4.1 Introduction

In this chapter we proceed to the implementation details of the tracker defined in chapter 1. Firstly, having only a single image in mind, we propose a modified snake model, which is based on the greedy algorithm and aims to achieve a tolerable convergence to the target contour. Then, we examine the details of applying this model to the whole process of tracking a target in a sequence of images, under the given stipulations of our tracker.

The algorithm is tested on the two video streams shown in figures 4.1 and 4.2, which, for now on, we shall call *video A* and *video B* respectively. Note that in the first video, the people are moving perpendicular to the camera, whereas in the second, the movement is in general parallel to the camera. Hence, *video A* gives postures with larger concave areas due to the gait, as in frames 311 and 321 (Figure 4.1). On the other hand, in *video B*, the movement takes place in a highly cluttered background, with many misleading edges.

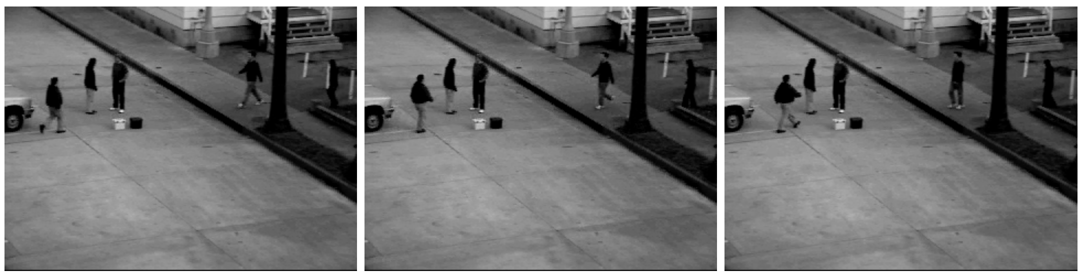


Figure 4.1: Three frames of *video A*.

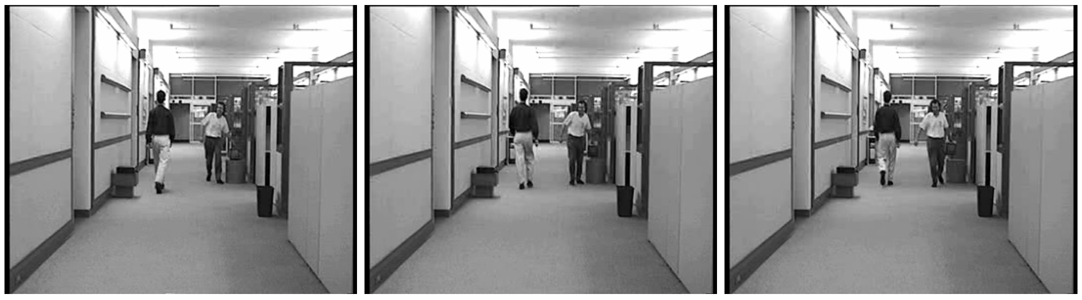


Figure 4.2: Three frames of *video B*.

4.2 The snake model

As mentioned in chapter 1, the initial contour is a rectangle around the target, defined by the user-specified upper-left and lower-right corners. Starting from these two points we get the rest of the initial snaxels by taking three equidistant points on each of the smaller sides of the rectangle and five equidistant points on each of the larger sides of it, concluding to 12 initial snaxels (Figure 4.3a). In figure 4.3b, it is depicted the initial contour that comes up by connecting the subsequent points. A more natural representation of the contour is given by a parametric curve interpolation (Figure 4.3c), where the parameter value t(i) for the point $\mathbf{v}_i$ is chosen by Eugene Lee's centripetal scheme [34], i.e. as accumulated square root of chord length:

$$t(i) = \sum_{j<i} \sqrt{\left\| \mathbf{v}_{j+1} - \mathbf{v}_j \right\|_2} \tag{4.1}$$

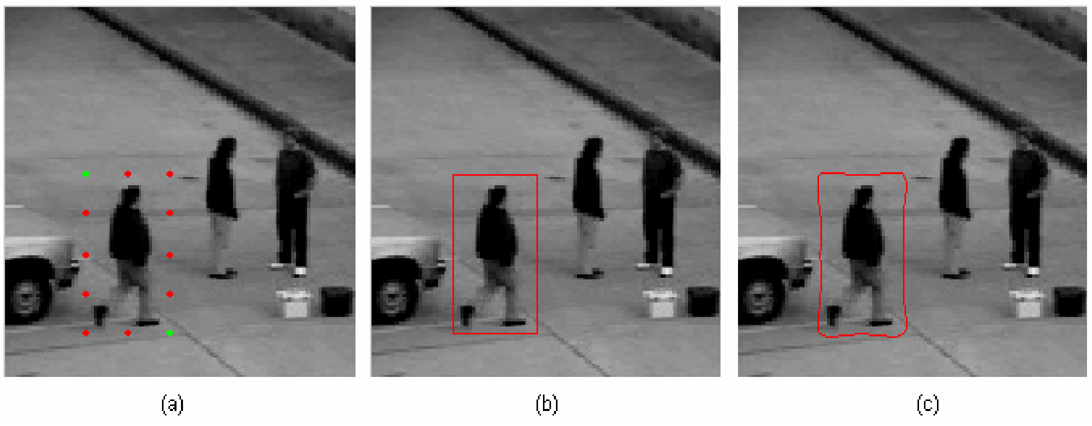


Figure 4.3: Initial snaxels. The user-defined corners are shown in green (a). Two different representations of the initial contour (b) and (c).

From figure 4.3 it is obvious that, even if the rectangle is chosen close to the target, it's not totally close in position (e.g. near the corners) and is definitely not close in shape (especially near the concavity). Therefore, the two problems mentioned in the previous chapter are still present. In the previous chapter we also saw that "attractable snakes" [30] offer promising solutions to problems of this form. Here we shall see that this algorithm generally has satisfying results, but in some cases there are some problems with its application to the given test videos. That is why we shall also examine a second solution, which is based on the original greedy snake model but uses a scale-space continuation technique.

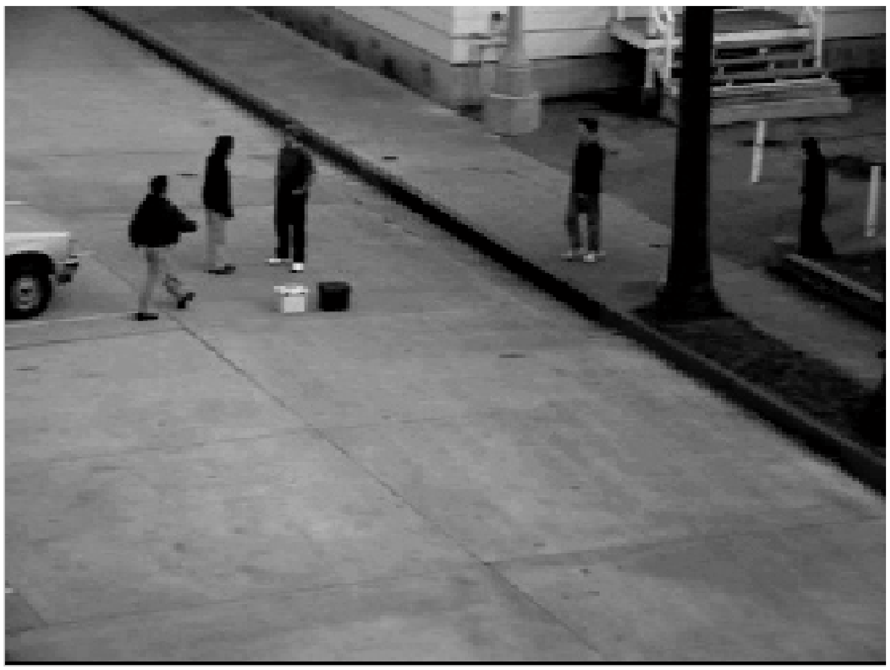
4.2.1 The "attractable snake" model

In [30] the attractable snake model is applied successfully on some concave/convex objects and MR images. Here we try to apply this model on the above test images, where the target object is human. Some points of this implementation, including some modifications of the original algorithm, are given below:

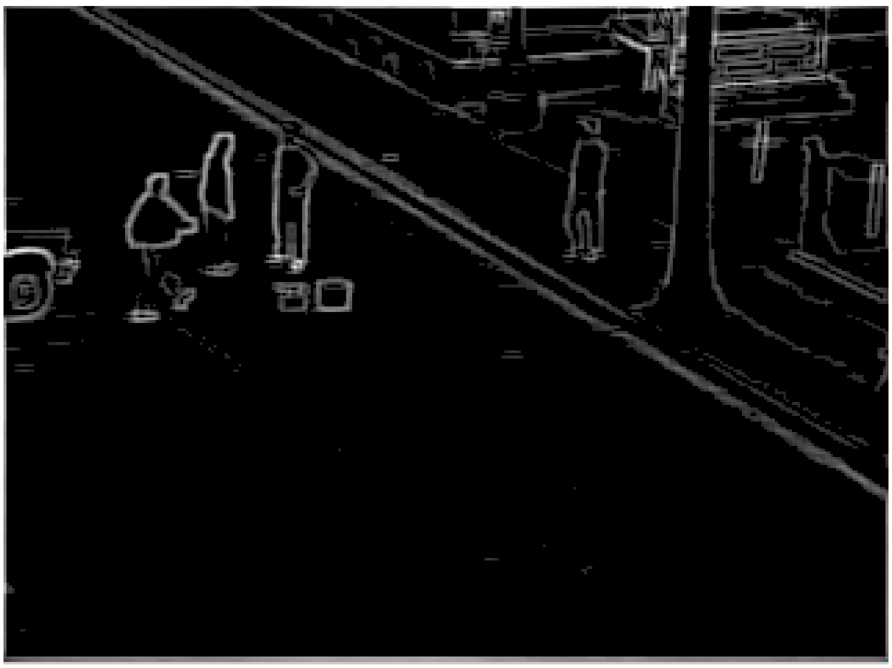
- The edge image, used in the image energy term, is computed as mentioned in the previous chapter, i.e. by a combination of the Sobel and Canny edge detectors. Furthermore, it is thresholded, according to the equation (3.8):

$$P_{level} = Tg(|\nabla I|_{\max} - |\nabla I|_{\min}) + |\nabla I|_{\min}$$

  Having no prior information about the desired potential field, we can use as P_{field} the above edge image. In our case, we can take advantage of the rectangle initial contour in order to seek for the maximum and minimum values of gradient into this rectangle instead of the whole image. Of course this can only be done at the first frame, where the initial contour is rectangle. Some examples are shown in the following figures.

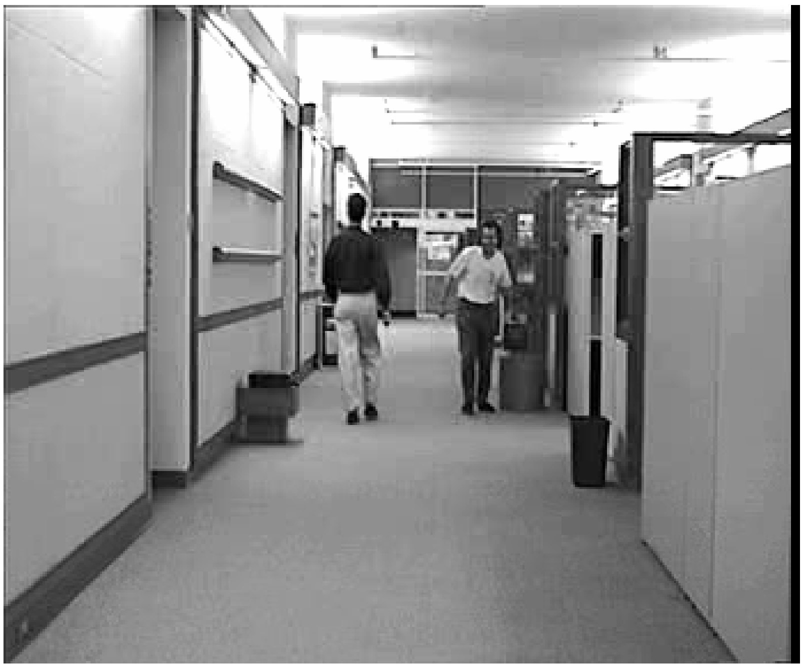


(a)

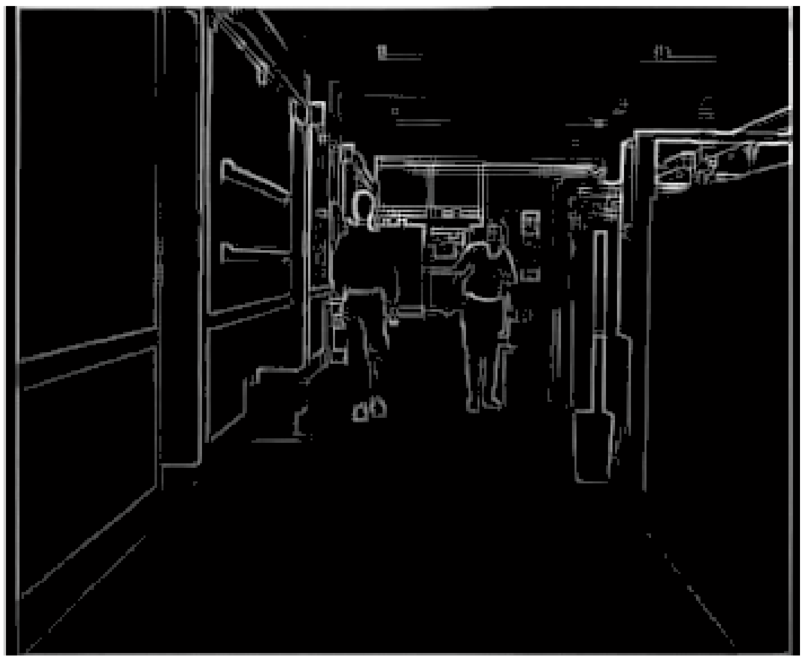


(b)

Figure 4.4: Original image (a) and thresholded edge image (potential field) (b)
($T_g = 0.2$, $P_{level} = 36.2$, Gaussian $\sigma = 1.0$)



(a)



(b)

Figure 4.5: Original image (a) and thresholded edge image (potential field) (b) ($T_g = 0.2$, $P_{level} = 48$, Gaussian $\sigma = 1$).

- The interpolation scheme proposed in [30] states that a new point is interpolated between each pair of snaxels at the beginning of each iteration step, when the average distance of the snake points is above a threshold T_{gap}. However, another re-sampling scheme is suggested in [35], which seems to be more adaptive. A new point is added in the middle of a snake segment defined by two subsequent snaxels, if the length of this segment is larger than a threshold l_{max}. In other words, the segment splits into two segments of equal length and the number of snaxels increases by one (Figure 4.6a). On the other hand, two subsequent snaxels that get closer than a threshold l_{min} are replaced by a single snaxel in the middle of the segment they define. Hence the segment is removed and the number of snaxels decreases by one (Figure 4.6b). At the beginning of each iteration of the snake algorithm, the re-sampling process consists of the removal-step followed by the split-step. In order to avoid an oscillatory behaviour, where segments are repeatedly removed and reinserted, we have to choose carefully the ratio of l_{max} and l_{min}. In general, the condition $l_{max} > 2l_{min}$ must be satisfied. We can consider that the values of l_{max} and l_{min} both depend on another parameter: the approximate desired length of the snake segments, l_{des}, which has significance similar to that of T_{gap}. Therefore, we can define only l_{des} and let l_{max} and l_{min} be proportional to the desired length, in a way, of course, that they satisfy the above inequality. A good choice is:

$$l_{min} = \frac{1}{2} l_{des} \tag{4.2}$$

$$l_{max} = \frac{3}{2} l_{des} \tag{4.3}$$

Such a choice actually means that the desired segment length can approximately vary between $\frac{1}{2} l_{des}$ and $\frac{3}{2} l_{des}$.

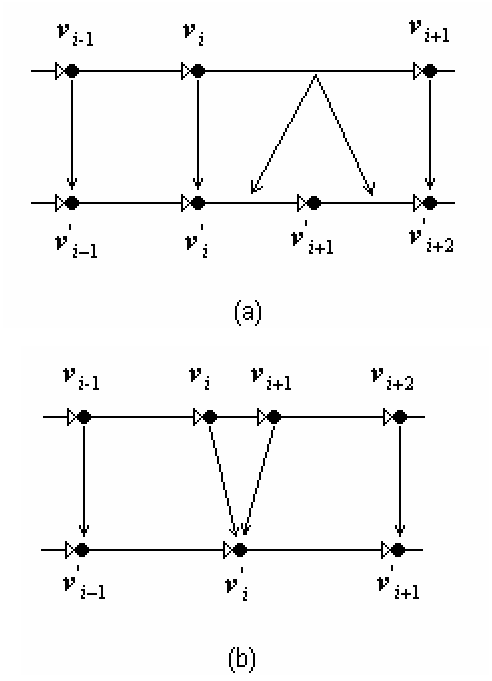


Figure 4.6: The resampling steps: (a) split and (b) remove.

- In the previous chapter we saw that the compound-parameter convergence criterion of the attractable snake model allows an option between a convergence to a subjective contour and a convergence to a concave/convex contour. In the edge image of Figure 4.4b, assuming that our target is the man at the left of the image, we can consider as a subjective contour the one that outlines the man without entering the area between the legs that is formed due to the gait. Obviously, we would like a snake that would not stop at this subjective contour but would move on to outline the present concavity. At the same time though, the resemblance in the illumination between the left leg and the background, in combination with the thresholding of the edge image, give a rather false sense of subjective contour at this area. Hence, in general we cannot decide whether we want the convergence process to stop on a subjective contour or not. If we omit the part that refers to the subjective contour, the proposed convergence criterion becomes:

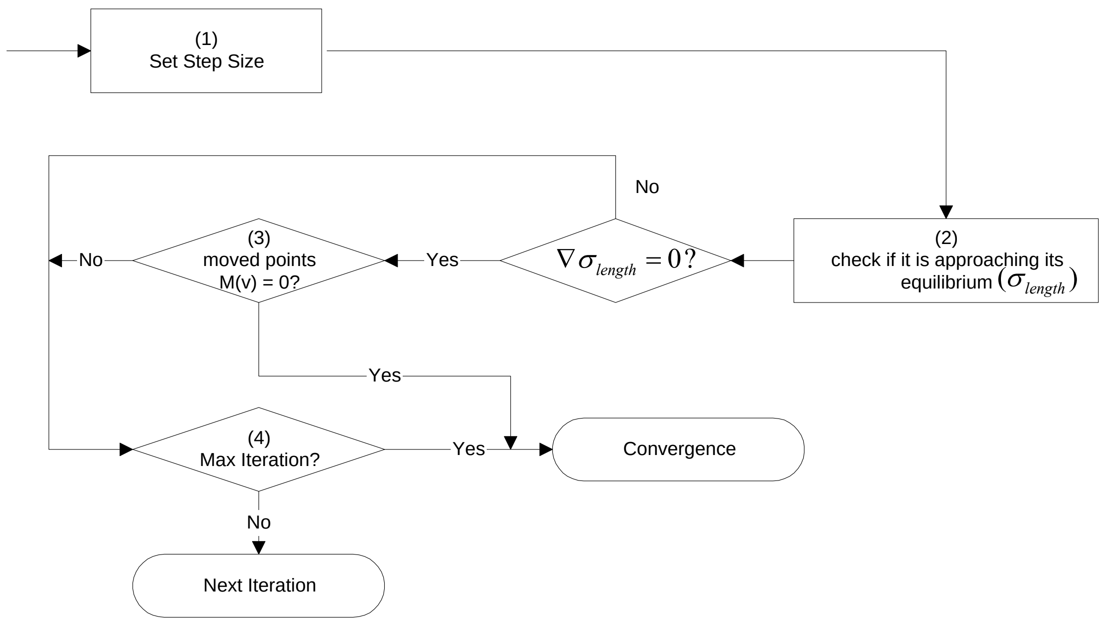


Figure 4.7: The new compound-parameter convergence criterion.

The use of a step size in the above criterion imposes that the iterations of the algorithm are multiple of this step size, even if the snake converges at an earlier iteration as well. Therefore we can use a simpler convergence criterion that examines at the beginning of each iteration only two parameters: the number of the points moved in the previous iteration against a threshold, and the number of the current iteration against a maximum number of iterations (Figure 4.8). The threshold for the points' movement, T_{moved}, should be a small non-zero value, which should be dependent on the total number of snaxels and should be updated after each re-sampling of the snake. A good choice for T_{moved} is:

$$T_{moved} = n/10 + \sigma \tag{4.4}$$

where n is the total number of snaxels and σ is the scale factor of the Gaussian smoother. As we will show later in this section, this convergence scheme has almost the same results as the one of figure 4.7.

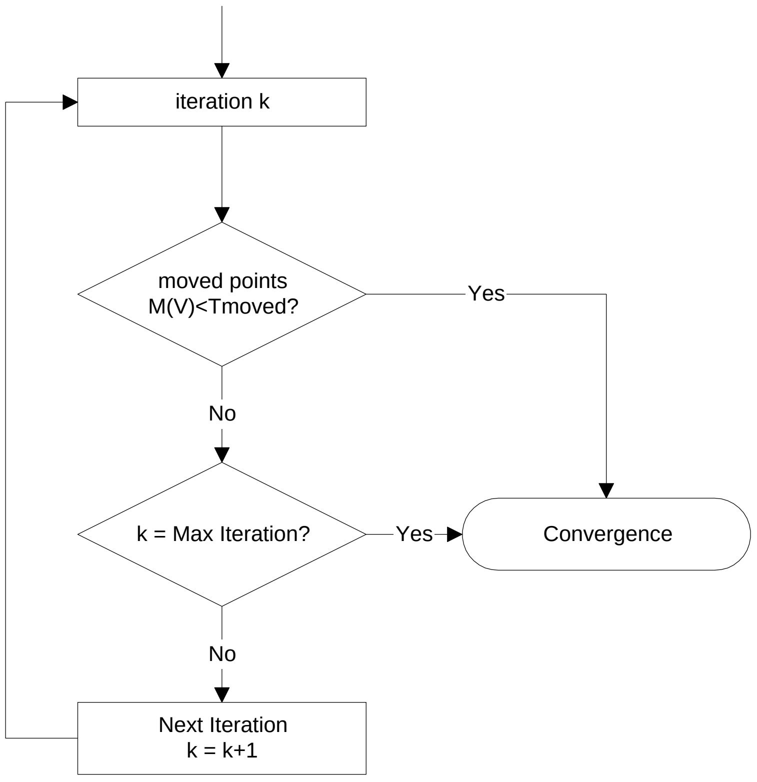


Figure 4.8: A simpler convergence criterion.

- The energy terms are calculated and normalized as mentioned in the previous chapter. The continuity, curvature and image terms are computed for each position of the neighbourhood around a given snaxel. For the feedback term though, the insertion of the normal direction in the computation of the term, is not very clear in [30]. Here, the difference in the potential, ∇P_{field}, is calculated for the given snaxel as in equation (3.5):

$$\nabla P_{field}(\mathbf{v}_i) = \begin{cases} \dfrac{P_{\max} - \sum_{j=1}^{8} P_{field}(\mathbf{v}_{i,j})/8}{P_{\max}} &, \; if\, P_{field}(\mathbf{v}_{i,0}) < P_{level} \\ 0 &, \; if\, P_{field}(\mathbf{v}_{i,0}) \geq P_{level} \end{cases}$$

where $\boldsymbol{v}_{i,0} = \boldsymbol{v}_i$ is the given snaxel and P_{field} is the magnitude of the above computed image gradient. This value of ∇P_{field}, multiplied by $-f_{db}$, is assigned to that point of the neighbourhood of $\boldsymbol{v}_i$, which lies on the normal direction of the contour at $\mathbf{v}_i$. All the other points in the neighbourhood have zero feedback energy. Thus, the additional push towards the target is given by means of energy minimization, only at the contour normal direction. The strength of the additional push depends on how far from the target the given snaxel currently is. In particular, when the given snaxel is already on an edge (or at least very close to an edge), its gradient magnitude is above P_{level} and we have a zero push. In the opposite case, the further the snaxel is from the target, the smaller is the sum of gradient magnitudes at the snaxel's eight neighbours and so the stronger is the push.

➢ The normal vector of the snake at a given point is calculated as the perpendicular vector to the tangent vector at that point. The tangent vector is given by the following equation:

$$\boldsymbol{t}_i = \begin{bmatrix} \cos(\theta_{t_i}) \\ \sin(\theta_{t_i}) \end{bmatrix} \tag{4.5}$$

where θ_{t_i} is the angle between the tangent at the given point $\mathbf{v}_i$ and the x-axis. This angle can be basically calculated in three ways: using the backward difference, the forward difference or the centered difference:

$$\theta_{t_i} = \tan^{-1}\frac{y_i - y_{i-1}}{x_i - x_{i-1}} \tag{4.6}$$

$$\theta_{t_i} = \tan^{-1}\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{4.7}$$

$$\theta_{t_i} = \frac{1}{2}(\tan^{-1}\frac{y_i - y_{i-1}}{x_i - x_{i-1}} + \tan^{-1}\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{4.8}$$

The first two equations give tangent vectors that have the direction of the contour (clockwise), as shown in figures 4.9a and 4.9b respectively. The centered difference gives a more realistic calculation of the tangent vector, as shown in Figure 4.9c, but the orientation does not always agree with the contour direction, since it's the average of the other two measures. In this work, we chose the third way of calculating the tangent vector, but we have to check the orientation of the tangent against the direction of the contour. This can be simply done by computing the dot products of the resulting tangent vector from equation (4.8) and each of the resulting tangent vectors from equations (4.6) and (4.7) (which always have the same orientation as the contour). If any of these dot products is negative, then the proper angle is $\theta'_{t_i} = \theta_{t_i} + \pi$. Once the correct angle is computed, the normal vector is:

$$\boldsymbol{n}_i = \begin{bmatrix} \cos(\theta_{n_i}) \\ \sin(\theta_{n_i}) \end{bmatrix} \tag{4.9}$$

where $\theta_{n_i} = \theta_{t_i} - \frac{\pi}{2}$ is the angle between the normal vector and the x-axis. In other words, the normal vector is the tangent vector rotated clockwise (i.e. at the contour direction) by $\frac{\pi}{2}$.

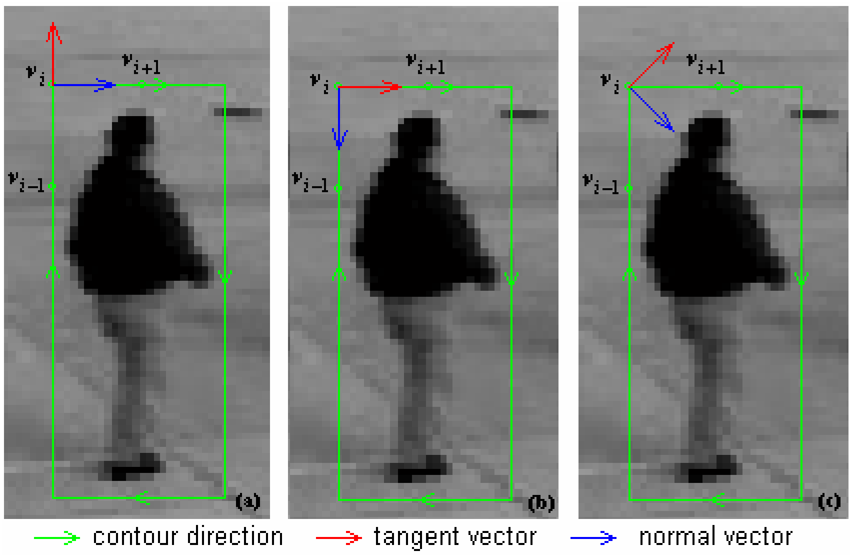


Figure 4.9: The tangent and normal vectors using (a) the backward, (b) the forward and (c) the centered difference.

- Even if the significance of the coefficients α, β, γ and f_{db} as control parameters of the four energy terms is clear, the optimal choice of their values is not an easy task. Actually, there is no ratio of these parameters that is optimal for all postures and for all parts of the target (e.g. weak edges, strong edges, concavities etc.) at the same time. The fact that, during an iteration step, the movement of each snaxel is affected by the so far moved snaxels (asynchronous movement), makes things more difficult. For example, if a snaxel locks onto a noisy point of high gradient far away from the target, the internal energy will probably keep the following snaxel(s) near to this snaxel and far from the target as well. In this implementation we attempt to keep a (almost) constant parameter ratio (with small variations in some cases), based on the common sense and on trials, and moreover we intent to correct some problems that may arise in some cases from such a general parameter definition. The common sense dictates that the weight of the image term, γ, should be greater that the other weights, since the image feature (edge) is the dominant cue that defines the target. The control of the continuity term is slightly more important than that of the curvature term, especially when we are interested in a rather coarse description of the target shape, so α should be grater than β. The feedback weight, f_{db}, should be usually between α and β, so that the effect of the feedback force would be limited in cases of "edge gaps". A set of parameters that agrees with the above statements is:

  $$\alpha = 0.7, \beta = 0.5, \gamma = 1, f_{db} = 0.5$$

  Furthermore, the choice of T_g for the computation of the edge threshold P_{level} is also crucial: T_g should be large enough to suppress noise and at the same time small enough to maintain weak edges. Our choice is a value of T_g around 0.2, which results in a value of P_{level} around 40 in the range [0,255]. Sometimes though, this "trade-off" value doesn't completely remove the image noise. In these cases we may observe snaxels being trapped in places far away from the target (Figure 4.10a). Note that two non-subsequent snaxels come very close to each other, forming a narrow contour "strip". Cases like these are undesirable anyway, but they can further create another problem:

such snaxels, apart from being very close to each other, have also normal directions with opposite orientations; hence, they may move in such a way that they form a “closed-loop” (Figures 4.10b and 4.10c). A closed-loop is also created when two non-subsequent snaxels coincide (Figure 4.10d).

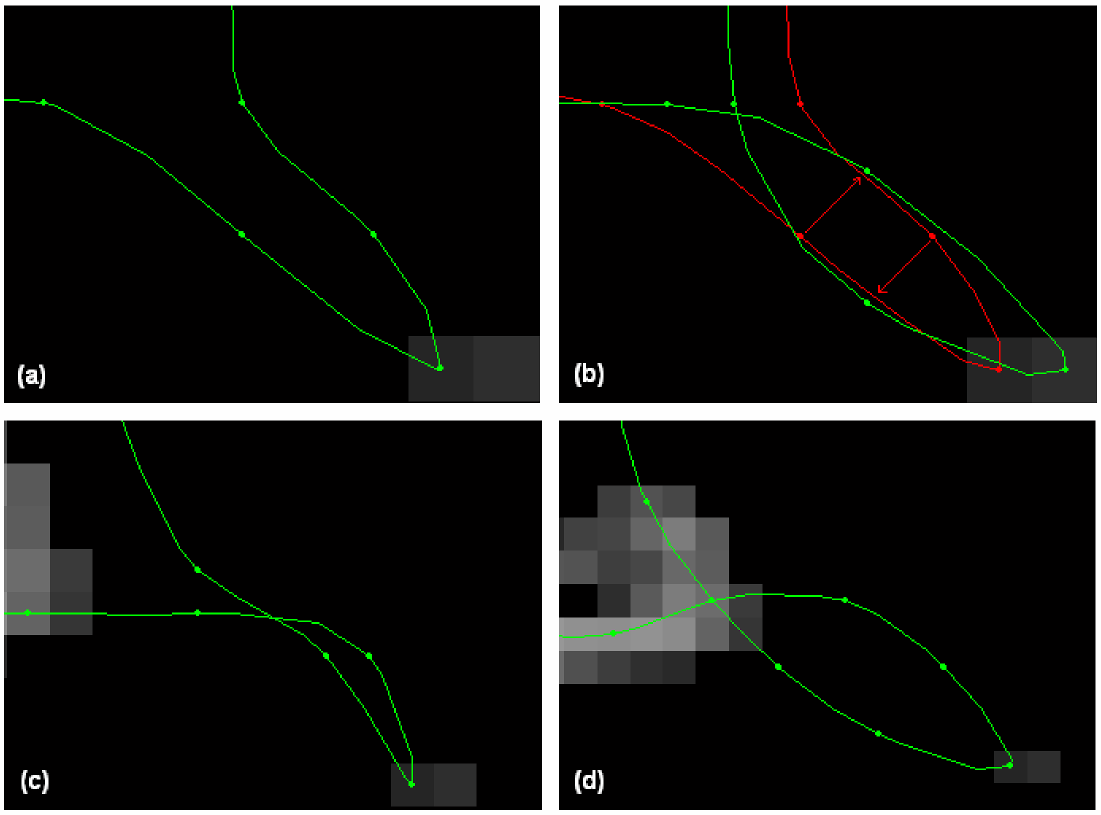


Figure 4.10: Some undesirable contour deformations. (a): A snaxel is trapped far away from the target, forming a narrow contour “strip”. (b): The opposite movement of the two snaxels (red arrows) create a closed-loop (the contour in the previous iteration step is in red). (c) and (d): Two other cases of closed-loops.

The problems depicted in Figure 4.10 may (or may not) be avoided with a specific parameter choice, but since we decided to use a general, constant parameter ratio, we have to find a way to eliminate these problems whenever they are created. A simplified but quite effective way to do so is the following. At the end of each iteration we find all the pairs of non-subsequent snaxels, between which the distance is 0, 1 or $\sqrt{2}$. For each pair $(\boldsymbol{v}_i, \boldsymbol{v}_j)$, there is a group of snake points that lie clockwise between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ and a group of points that lie counter-clockwise between these two snaxels. In order to tackle the above undesirable deformations, we eliminate all the snaxels of the smallest group, including $\boldsymbol{v}_j$, if $|\boldsymbol{v}_j - \boldsymbol{v}_i| < l_{\min}$. We can also use a threshold so that only relatively small loops are eliminated. The images in Figure 4.10 are close-ups of some snapshots during a real convergence process. Using the initial contour that is shown, superimposed on the original and the edge image, in figures 4.11a and 4.11b, and the parameter set: $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $f_{db} = 0.5$, $l_{des} = 3$, $T_g = 0.2$, we get the contour of Figure 4.11c. During the execution of the algorithm, a lot of closed loops are being formed, blocking the smooth convergence of the snake. The algorithm terminates at the predefined maximum iteration, without achieving a tolerable fit to the target. Using the above algorithm for the elimination of "closed-loops", we get the result in Figure 4.11d. In the following figure we use the centripetal interpolation scheme that was mentioned at the beginning of this section, since it gives a more natural contour representation.

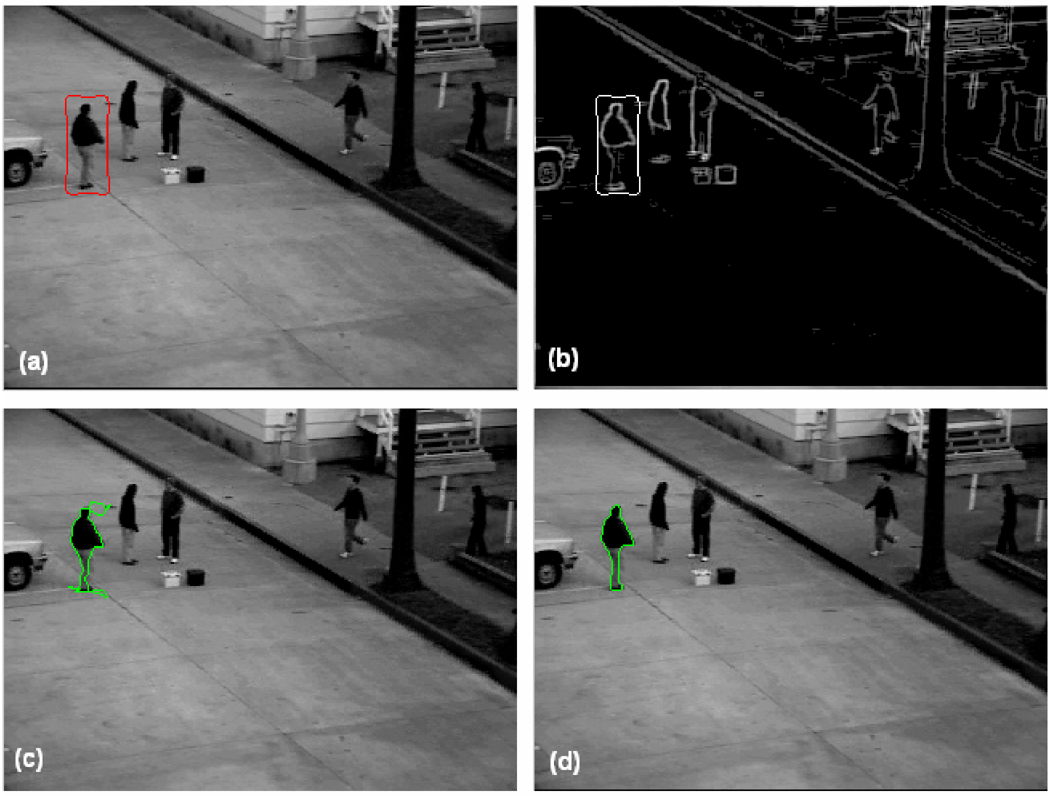


Figure 4.11: The initial contour on the original (a) and on the gradient image (b). The final contour without (c) and with (d) the "closed-loop elimination" algorithm.

- The final contour in Figure 4.11d reveals the effectiveness of both the "closed-loop elimination" algorithm and the feedback mechanism. In the specific example, the target object doesn't have large concavities but the initial contour is not very close to it, in both position and shape. If we don't use the feedback mechanism the snake will fail to converge to the target, as it's shown in Figure 4.12, where the resulting final contour (b) is compared to the contour that results if we use the feedback mechanism along with the algorithm for elimination of "closed-loops" (a). We get these results using the initial contour of Figure 4.11a and the same parameter set. We also use the simplified convergence criterion of Figure 4.8. This criterion terminates the algorithm after the 18th iteration, during which, 1 of overall 42 points was moved, for Figure 4.12a, and after the 9th iteration, during which, 6 of overall 51 points were moved, for figure 4.12b. Using the compound-parameter criterion of figure 4.7, we get almost the same results (figures 4.10c and 4,10.d) but after more iteration steps: 30 iterations and 1 of 42 points moved for figure 4.12a, and 30 iterations and 0 of 50 points moved for figure 4.12d.

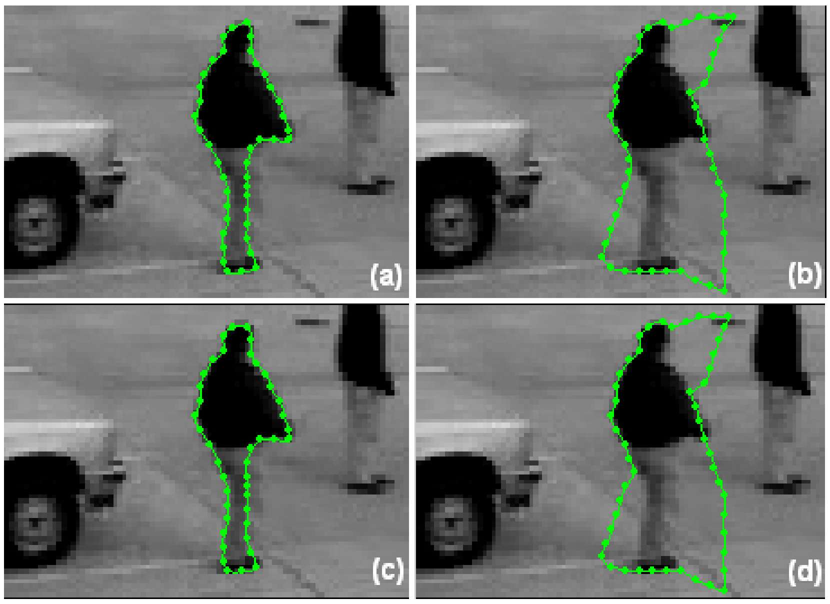


Figure 4.12: The resulting contours with (a,c) and without (b,d) the feedback mechanism, using the simplified (a,b) or the compound-parameter (c,d) convergence criterion.

However, the effectiveness of the feedback mechanism in handling the two major problems of the previous chapter should also be examined on targets with larger concavities. Such a target is the one depicted in figure 4.13a. We use the initial contour that is superimposed on this figure, and the same parameter set ($\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $f_{db} = 0.5$, $l_{des} = 3$, $T_g = 0.2$). Obviously due to the fact that the continuity and the curvature term dominate the feedback term, the snake oscillates around some point into the concavity and finally the algorithm terminates at the maximum number of iterations, giving the result of figure 4.13d. Note that the compound-parameter criterion terminates the algorithm earlier (30 instead of 50 iterations) with exactly the same result. Hence, we can say that this criterion achieves faster convergence than the simplified criterion, when the snake converges in an oscillatory way. We can see that the snake does not totally outlines the concavity, but the final contour is definitely better than the one that comes up without using the feedback mechanism (Figure 4.13c).

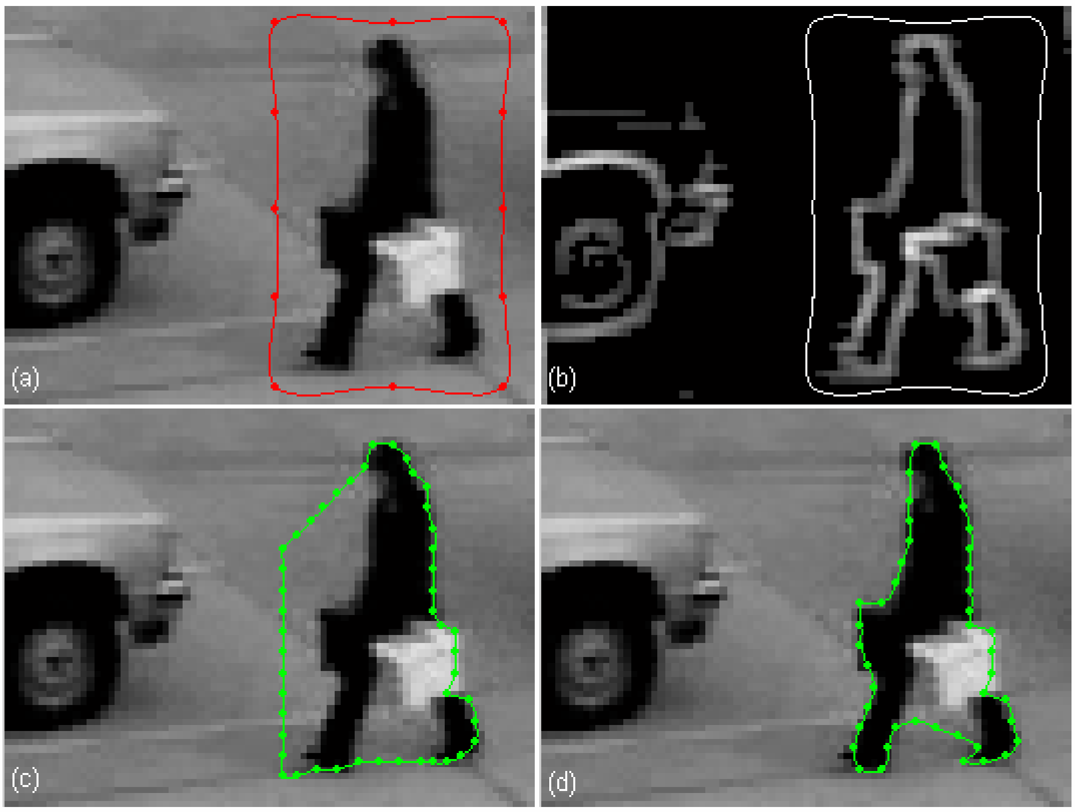


Figure 4.13: The initial contour on the original (a) and edge image (b). The resulting contours with (d) and without (c) the feedback mechanism.

Having in mind what we have said above about the weight f_{db}, i.e. that $\beta \leq f_{db} \leq \alpha$, we can further improve the final contour by choosing $f_{db} = \alpha = 0.7$ (Figure 4.14). This increment of f_{db} is allowed because the specific target has an edge contour without gaps (Figure 4.13b), unlike the target of Figure 4.11.

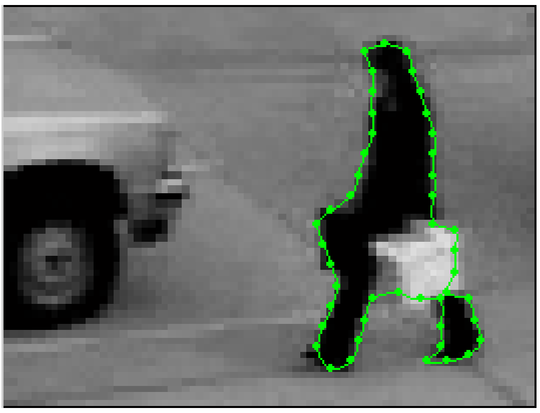


Figure 4.14: The final contour using f_{db} = 0.7.

The model becomes less effective though for targets with similar concavities and "edge gaps" as well (Figure 4.15a). With $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $l_{des} = 3$, $T_g = 0.2$ ($P_{level} = 37$) and $f_{db} = 0.5$, we get the result of figure 4.16a. Obviously the small values of f_{db} and P_{level} stop the evolution of the snake into the concavity. Furthermore, the edge gap at the bottom of the right foot (4.15b) makes the snake overrun this "subjective" edge. The increment of f_{db} to 0.7 doesn't improve the result into the concavity and moreover leads to more edge overruns (Figure 4.16b).

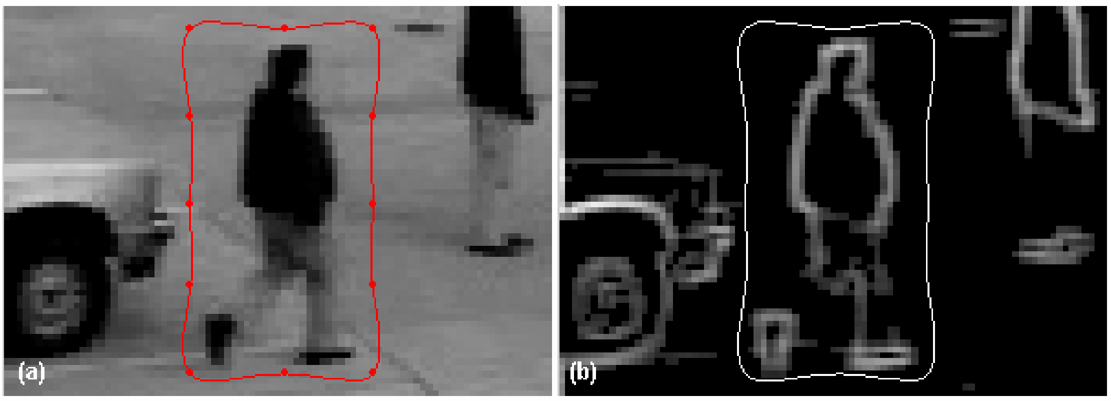


Figure 4.15: A target with "edge gaps" and large concavities. The initial contour on the original (a) and edge image (b).

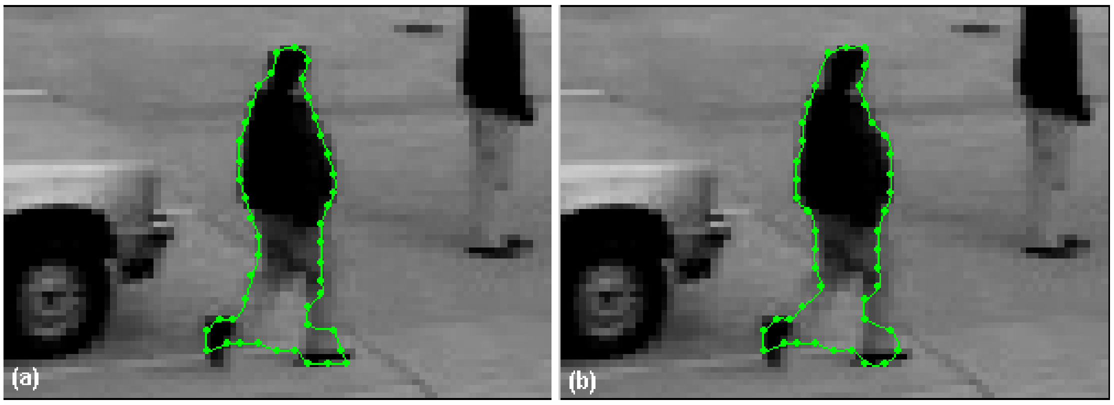


Figure 4.16: The final contour with f_{db} = 0.5 (a) and f_{db} = 0.7 (b).

A greater value of T_g and hence of P_{level}, for example 0.22 and 40.7 respectively, can further suppress noise (Figure 4.17a) and lead to a slightly better description of the concave area but, of course the handling of edge gaps is still problematic (Figure 4.17b).

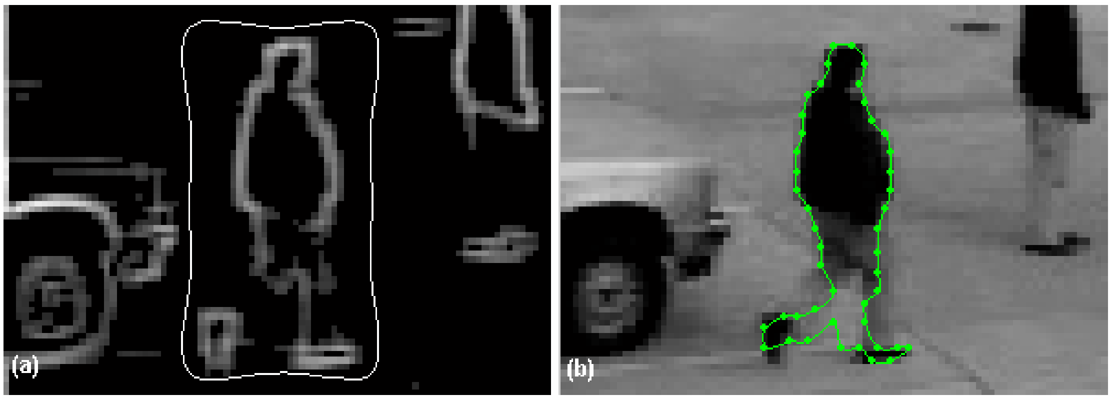


Figure 4.17: The edge image (a) (P_{level} = 40.7) and the final contour (b) (f_{db} = 0.7).

- In all the previous examples the area between the initial contour and the target object was almost clear of edges coming from noise or other objects, which could block the evolution of the snake. Unfortunately, this is not always the case. The following frame (Figure 4.18) of *video B* is a representative example. The background edges around the target (Figure 4.18b) stop the snake far away from the real contour, as we can see in figure 4.18c, where α = 0.7, β = 0.5, γ = 1, f_{db} = 0.5, l_{des} = 4 and T_g = 0.2 (P_{level} = 48). A lower value of

T_g (T_g = 0.1/ P_{level} = 24) just makes the effect of 'edge-bypass' less intense (Figure 4.18d).

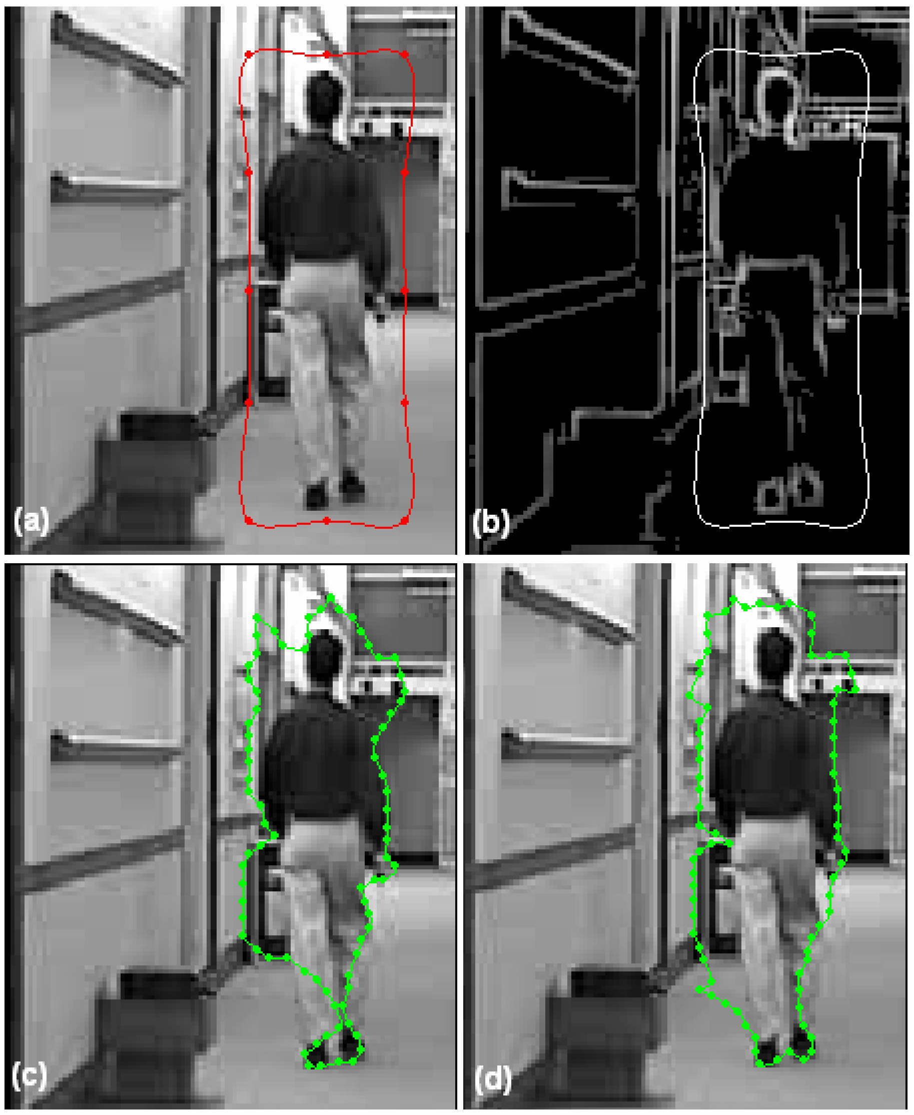


Figure 4.18: A cluttered background: the initial contour (a,b) and the final contour for two different values of T_g (c,d).

There is definitely no parameter set that could achieve a tolerable convergence in this case, as the problem arises from the cluttered background. Therefore, the solution should be sought in the improvement of the edge image. A simple way to approximate the background is to calculate the average of the intensity images of the entire video stream (Figure 4.19a). Then, an improved edge image can be taken by simply differencing the edge image of the background (Figure 4.19b) from the edge image of the current frame.

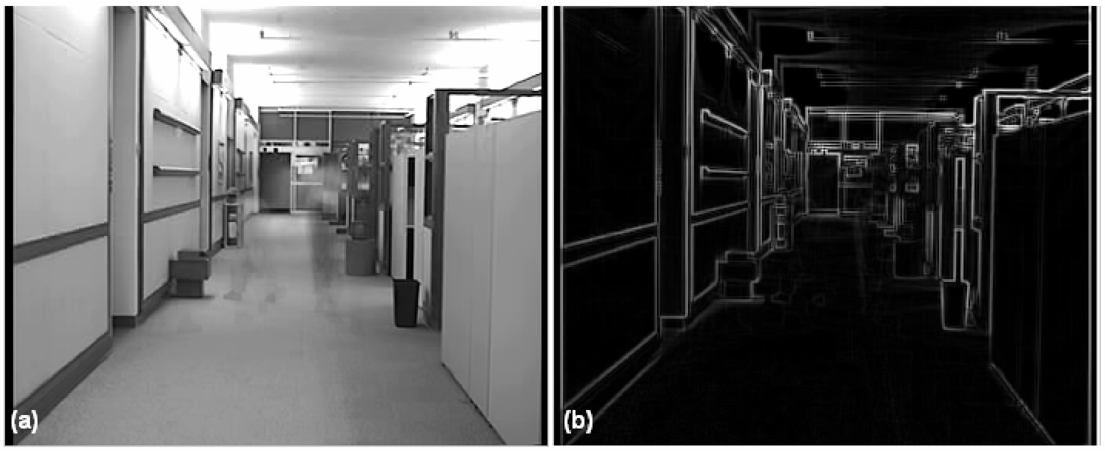


Figure 4.19: A background approximation (a) and its edge image (b).

The resulting image is also thresholded in order to further suppress noise. An example is shown in the following figure. In this case we choose a rather large value of T_g (T_g = 0.2, P_{level} = 42.6) in order to remove the noisy edges due to the differencing, and a small value of f_{db} (f_{db} = 0.3), in order to avoid the 'bypass' of subjective edges. We also use fewer snaxels (l_{des} = 5), as the target shape is not very complicated. This parameter set gives a quite tolerable final contour (Figure 4.20b).

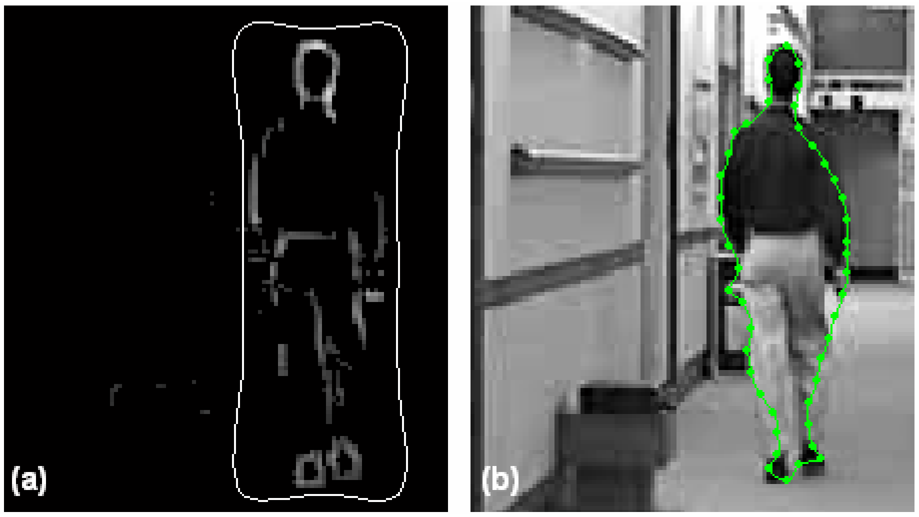


Figure 4.20: Using the difference of edge images: the initial contour on the resulting edge image (a) and the final contour (b)

Another way of detecting only the moving objects of an image is referenced in [36]. This algorithm combines the spatial and the temporal information of an image in order to distinguish the moving objects from the background. In particular, it finds the static edges of a moving object using the spatial gradient of both the intensity image and the background (spatial information). Then it uses the information coming from the motion vectors and the first derivative of intensity in time, $I_t(x,y,t) = |I(x,y,t) - I(x,y,t-1)|$, in order to detect the moving edges of the object (temporal information). The final edge image comes from the combination of the detected static and moving edges, followed by noise suppression (median filter). An example is shown in the following figure.

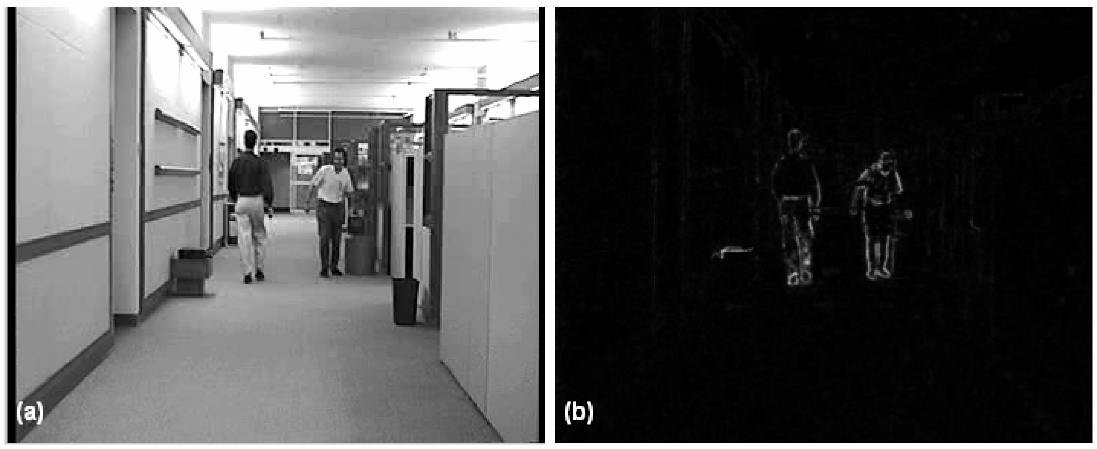


Figure 4.21: The edge detection scheme of [35]: (a) Original intensity image and (b) edge image.

Now we can apply the above attractable snake model on this improved edge image. By setting $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $f_{db} = 0.3$, $l_{des} = 4$ and $T_g = 0.14$ ($P_{level} = 48$), we get a satisfactory final contour (Figure 4.22b).

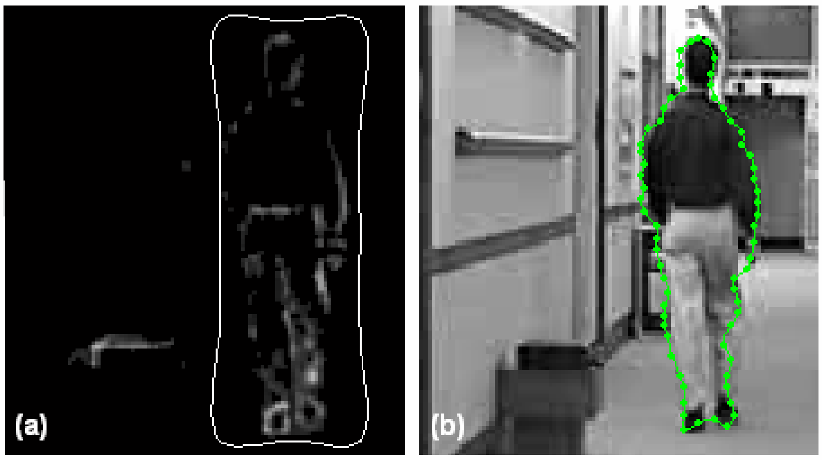


Figure 4.22: Attractable snake and edge detection using spatio-temporal information.

However, neither of the improved edge images can prevent the blocking of the snake movement, when an object that is not moving temporarily is very close to the target object, as in figure 4.23. For a given value of T_g (T_g = 0.2, P_{level} = 35.2), a small value of f_{db}, e.g. f_{db} = 0.4, allows the snake to adapt to the subjective contour, but at the same time cannot "unlock" the snake from the adjacent object (Figure 4.23c). On the other hand, a greater value of f_{db}, e.g. f_{db} = 0.7, has exactly the opposite result (Figure 4.23d).

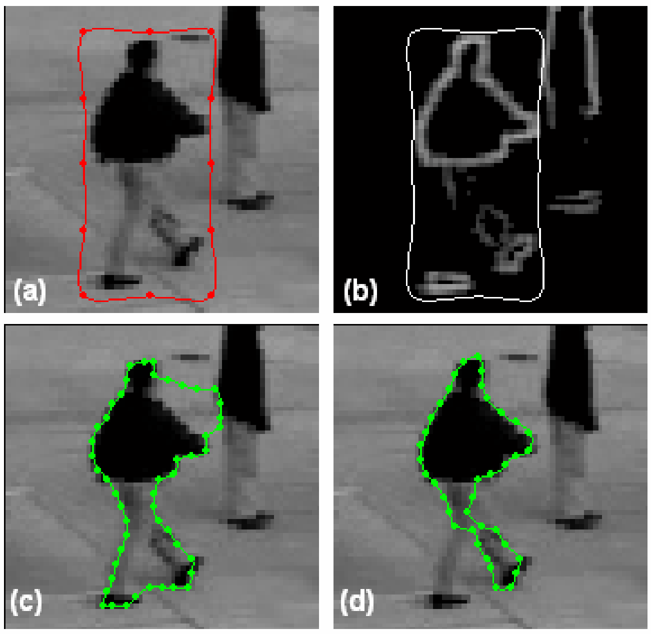


Figure 4.23: The initial contour is very close to a background edge (a,b). The parameter f_{db} should be neither small (c), nor large (d).

A quite efficient convergence can be achieved by a lower value of T_g (T_g = 0.11, P_{level} = 19.36) and, at the same time, by a higher value of f_{db} (f_{db} = 0.9) (figure 4.24). However, this "golden mean" can only be found after a "trial and error" process.

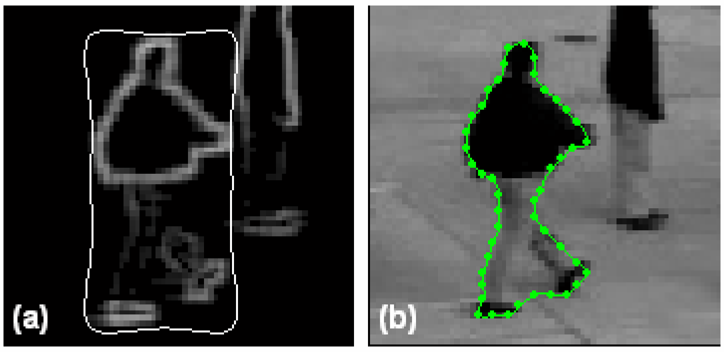


Figure 4.24: The edge image (a) and the final contour (b).

4.2.3 The "scale-space" model

We have seen that the additional force introduced by the feedback mechanism can become very helpful when the initial contour is far from the target in both position and shape. In some cases though, e.g. in edge gaps, it may become undesirable and may force the snake to 'overrun' a subjective contour. In this section we examine yet another way of attracting the snake to the target when the initial contour is far away from it. The proposed model uses the energy functional of the original greedy snake (i.e. without the feedback term) but lets the snake move sequentially on edge images of subsequently decreasing Gaussian scale factor. All the other features of the model of section 4.2.1, i.e. the interpolation scheme, the simplified convergence criterion and the "closed-loop" elimination algorithm, are maintained in this model too.

For the computation of the edge image we use the following edge detector. A 2D Gaussian filter splits into two directions, x and y, and the first derivatives of the 1D filters (D_x and D_y in equation 4.11) are applied on the intensity image, giving the directional derivatives in the x and y directions:

$$\begin{aligned} ImgD_x &= I \otimes D_x \\ ImgD_y &= I \otimes D_y \end{aligned} \tag{4.10}$$

$$D_x = -\frac{x}{\sqrt{2\pi}\sigma^3} e^{-x^2/2\sigma^2}$$
$$D_y = -\frac{y}{\sqrt{2\pi}\sigma^3} e^{-y^2/2\sigma^2} \tag{4.11}$$

where *σ* is the scaling factor of the Gaussian filter. Then, the two 1D Gaussian filters (*G_x* and *G_y* in equation 4.13) are applied on the orthogonal directions of the results of equation (4.10), giving the intensity gradient:

$$|\nabla I| = \sqrt{(ImgD_x \otimes G_y) + (G_x \otimes ImgD_y)} \tag{4.12}$$

$$G_x = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}$$
$$G_y = \frac{1}{\sqrt{2\pi}\sigma} e^{-y^2/2\sigma^2} \tag{4.13}$$

Finally, the resulting gradient is thresholded in order to further suppress noise. The threshold value is given, as above, by:

$$P_{level} = Tg(|\nabla I|_{\max} - |\nabla I|_{\min}) + |\nabla I|_{\min}$$

This edge detection scheme is quite good at noise suppression and gives a more-than-one-pixel strength response to a step edge (fig 4.25). However, its most important feature, for the current application, is that, the increment of the scaling factor, *σ*, results in the blurring effect of figure 4.26. On one hand, this effect causes an undesirable image deformation (blurring), but on the other hand increases the capture range of the object.

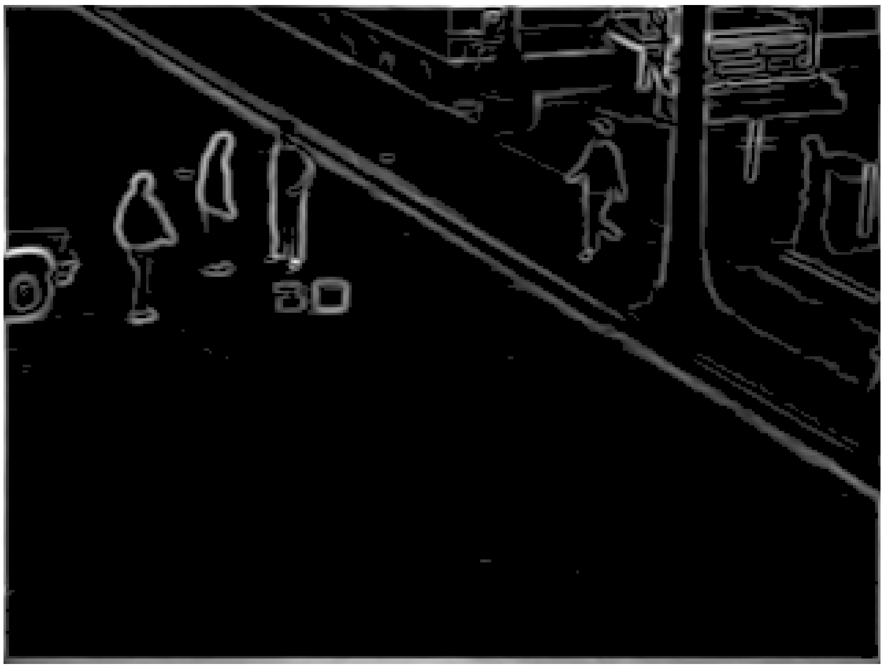


Figure 4.25: The edge image with scale factor $\sigma = 1$ ($T_g = 0.22$, $P_{level} = 38.28$)

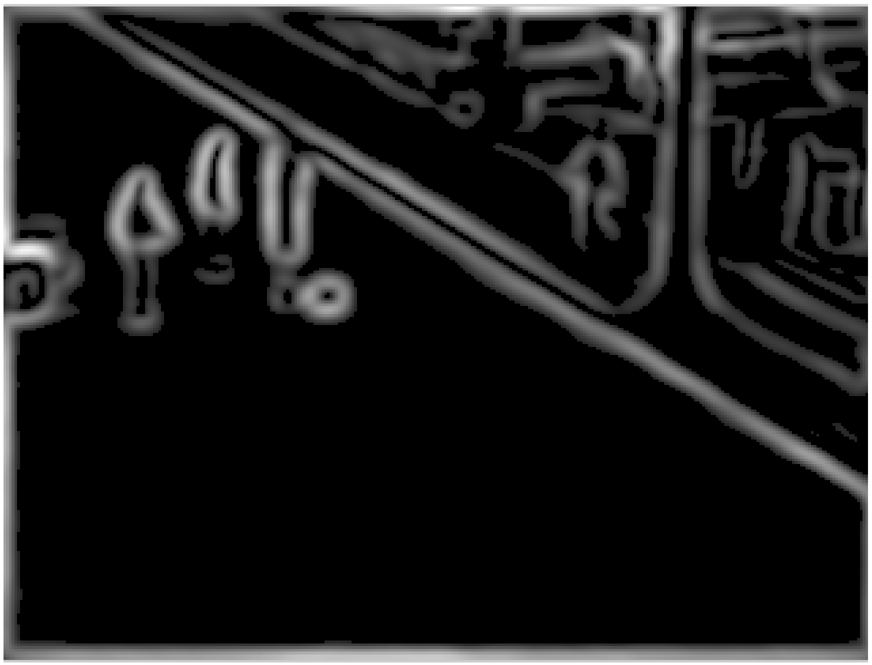


Figure 4.26: The edge image with scale factor $\sigma = 3$ ($T_g = 0.22$, $P_{level} = 37.4$)

In the following figure, firstly we let the snake converge onto the edge image of $\sigma = 3$; then we execute the snake algorithm in the edge image of $\sigma = 1$, using as initial contour the final contour of the previous stage (fig 4.27b). The final result is shown in figure 4.27d. For this execution of the algorithm we used the following parameter set: $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $l_{des} = 3$ and $T_g = 0.22$.

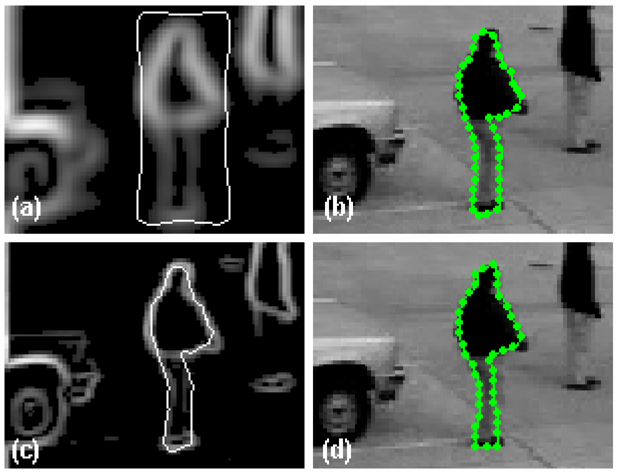


Figure 4.27: The snake converges subsequently on edge images of different scale factor ($\sigma = 3$ in (a) and $\sigma = 1$ in (c)). The intermediate result is in (b) and the final contour in (d).

It is obvious that by setting $\sigma = 3$, we increase the capture range of the target but we also get a coarse representation of the shape, loosing in details. This can cause some problems. For example, in the above figure, the deformation at the human hand is quite intense and hence the snake concludes in the interior of the real target at that area, as it is depicted in the close up of figure 4.28a. Then the scale factor decreases to 1 (fig. 4.28b) but the particular part of the snake is already away from the angle that the hand forms. That is why the snake slightly misses the human hand in figure 4.27d.

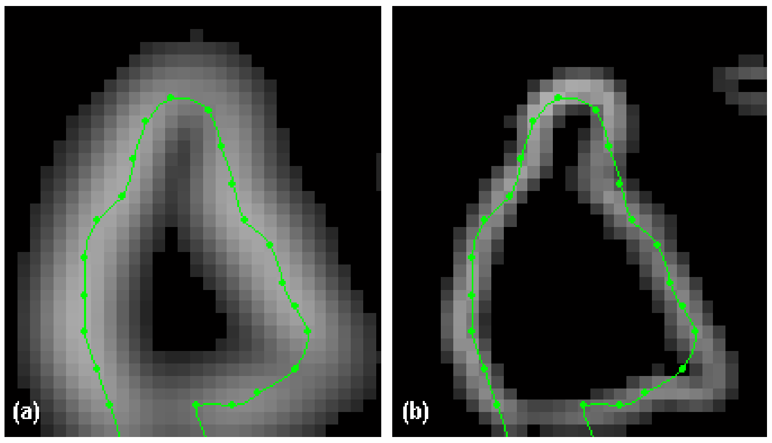


Figure 4.28: Close-ups of the two edge images.

This effect can be improved by substituting the edge image of $\sigma = 3$ with a combination of the edge images of $\sigma = 3$ and $\sigma = 1$. For example, if we add the two edge images we get a new image of increased capture range and, at the same time, of smaller deformation (Figure 4.29).

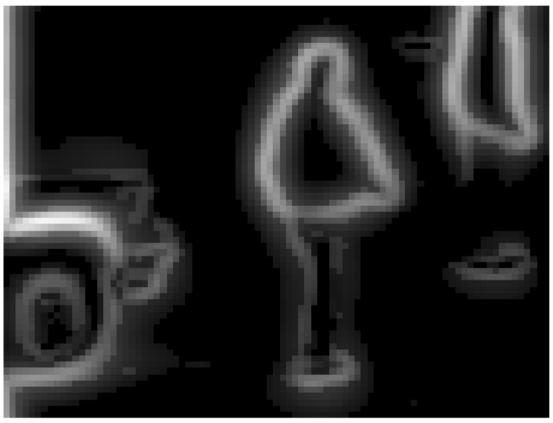


Figure 4.29: The result of adding the two edge images ($\sigma = 3$ and $\sigma = 1$).

Note that he edge detector with $\sigma = 1$, unlike the one with $\sigma = 3$, cannot suppress the non-target line at the upper right of the target. This line is also present in the edge image of figure 4.29 and pulls the snake to the right, far away from the target. This feature of the image is undesirable and can be eliminated by the background-subtraction technique of the above section (for both edge images before the addition). The final result is shown in figure 4.30d. It is clear that this final contour outlines the target better than the one of figure 4.27d. We can also notice that the difference between the intermediate and the final contour is rather small. In this sense we could omit the execution of the algorithm on the edge image of $\sigma = 1$. However, this stage of the algorithm is usually short and improves (even slightly) the final contour.

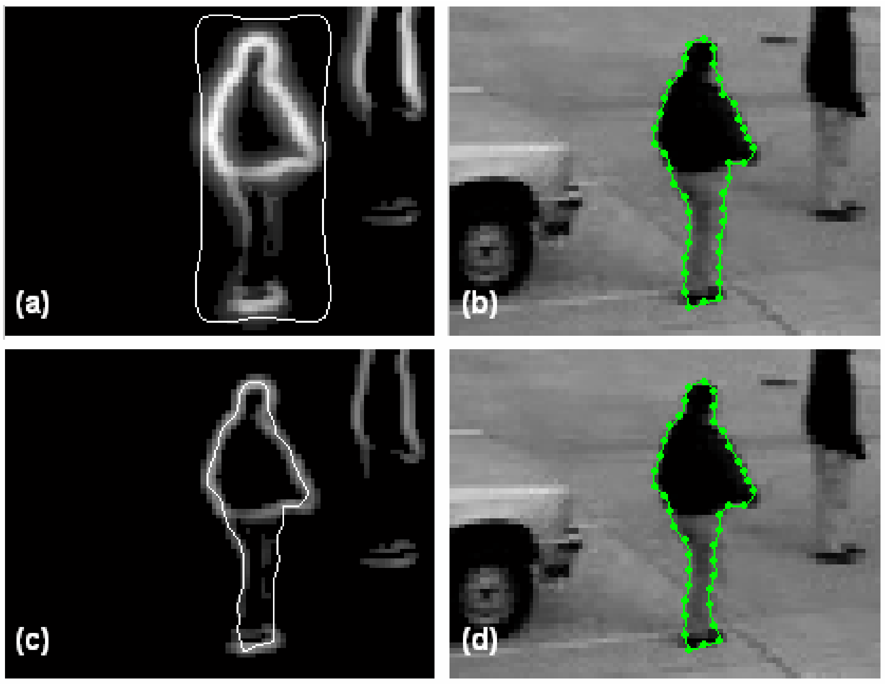


Figure 4.30: Using the combination of the edge images along with the background extraction (a). The intermediate result is shown in (b), the gradient with $\sigma = 1$ in (c) and the final contour in (d).

The application of this model to a more complicated shape gives us the result of the following figure. The parameter set is: $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $l_{des} = 3$ and $T_g = 0.12$.

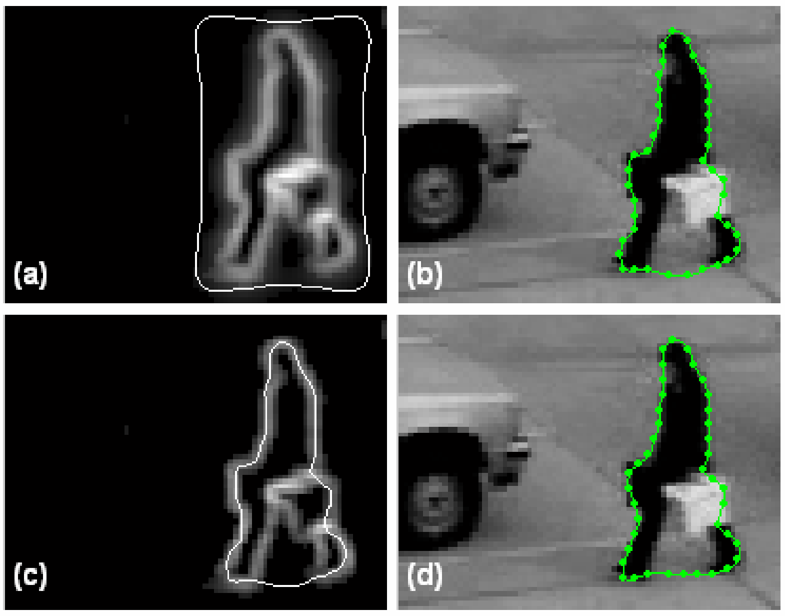


Figure 4.31: The application of the “scale-space” model on a more complex posture.

It is clear that the snake fails to enter the concavity, due to the absence of image forces in the area. Hence, the proposed model resolves only one of the two major problems of snakes.

Finally, the "scale-space" model has quite good results when applied to targets with many edge gaps in a highly cluttered background, as it is demonstrated in the following figure. The parameter set at this example is: $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $l_{des} = 4$ and $T_g = 0.16$.

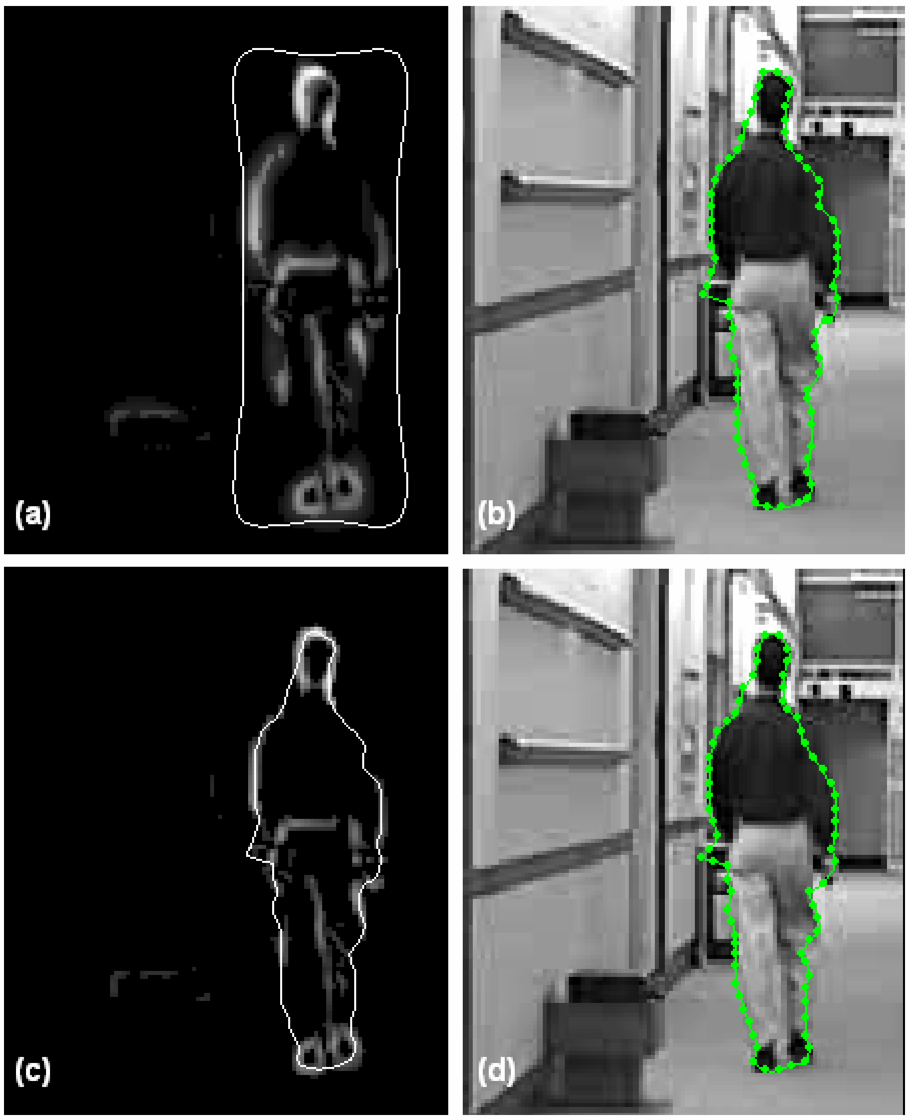


Figure 4.32: The application of the "scale-space" model on a target with many edge gaps, in a highly cluttered background.

4.3 The tracking algorithm

In this section we present the details of embedding the snake model in the overall tracking algorithm. The basic form of this algorithm is the following:

1. Identify an initial contour, C_0, in the initial image, I_0.
2. Let it converge through the snake algorithm, giving you the optimised contour $C_0^{'}$.
3. For every subsequent image I_k (k = 1, 2, …),
   i. use as initial contour, C_k, of the image I_k, a motion-compensated version of the optimised contour, $C_{k-1}^{'}$, of the previous image I_{k-1} ,
   ii. let C_k converge to the target boundary through the snake algorithm, giving you the optimised contour $C_k^{'}$.

A simple and fast way to find a motion-compensated version of the previous optimised contour is to use the velocity of each snaxel. Let's say that we attempt to find the initial contour, C_k, in the current frame, I_k. The velocity of a single snaxel can be expressed by the signed distance, in x and y-axis, between the final positions of this snaxel in the two previous frames, I_{k-1} and I_{k-2}. Hence, the initial contour C_k can be found by moving each snaxel of the optimised contour $C_{k-1}^{'}$ according to its previous velocity. In other words, each snaxel of C_k is given by the following equation:

$$\mathbf{v}_i^{(k)} = \begin{bmatrix} x_i^{(k)} \\ y_i^{(k)} \end{bmatrix} = \begin{bmatrix} x_i^{'(k-1)} \\ y_i^{'(k-1)} \end{bmatrix} + \left\lceil \delta \cdot \begin{bmatrix} x_i^{'(k-1)} - x_i^{'(k-2)} \\ y_i^{'(k-1)} - y_i^{'(k-2)} \end{bmatrix} \right\rceil \tag{4.14}$$

where the weight δ defines the contribution of the previous velocity to the calculation of the new contour ($\delta \in (0,1]$), and the symbol $\lceil \ \rceil$ denotes rounding towards $+\infty$. The effectiveness of this simplified algorithm requires that the movement of the target object is constant in velocity and orientation. If the target motion is not abrupt, the variations in speed and orientation are small and the snake algorithm is likely to correct an inaccurate estimation of the initial contour. Furthermore, it is obvious that this algorithm can be applied only from the third frame onwards, hence we should consider as initial contour for the second frame, the optimised contour of the first frame. This consideration cannot be very harmful, for the same reason.

4.3.1 A tracker that uses the attractable snake model

As it is mentioned in [30], the movement of the snake can be either contraction or expansion, according to the orientation of the normal direction. In all the experiments of section 4.2.1 we have chosen an inward orientation, since the starting, rectangle contour is initialised outside the target object. However, the above simple way of motion-compensation may give an initial contour that lies partially on the target interior. If a snaxel of the initial contour occur in the interior of the target and the contribution of its feedback energy term to the overall point energy is large enough, this snaxel will move further inwards. This incorrect movement will be accumulated from frame to frame and will probably result in a complete convergence failure, sooner or later. An example is shown in the following figure. The initial contours are in red (left) and the resulting contours are in green (right), for 4 consecutive frames. Note that the initial contour of the second frame (fig. 4.33c) is exactly the final contour of the first frame (fig. 4.33b), as there is no former information about the movement of the object. However, the snake manages to converge to the target. The initial contour of the third frame (fig. 4.33e), which comes from the motion-compensation of the contour in figure 4.33d, outlines sufficiently the target object, leading to a good final contour (fig. 4.33f). The initial contour in the fourth frame though contains closed loops and snaxels within the object interior (fig. 4.33g), due to the motion-compensation algorithm. Inevitably, the snake fails to converge satisfactorily (fig. 4.33h). By setting $\delta = 0.1$, we note that the compensated initial contours become closer to the real target contours. In this case, the term $\left\lceil \delta \cdot \begin{bmatrix} x_i^{'(k-1)} - x_i^{'(k-2)} \\ y_i^{'(k-1)} - x_i^{'(k-2)} \end{bmatrix} \right\rceil$ of equation (4.14) actually cuts off all the non-zero differences ($x_i^{'(k-1)} - x_i^{'(k-2)}$ and $y_i^{'(k-1)} - y_i^{'(k-2)}$) to –1 or 1. Eventually the snake also results in a convergence failure, but after 5 frames (fig. 4.34). Thus the parameter δ may be used to control convergence, but essentially there is a need of some way to decide effectively between contraction and expansion.

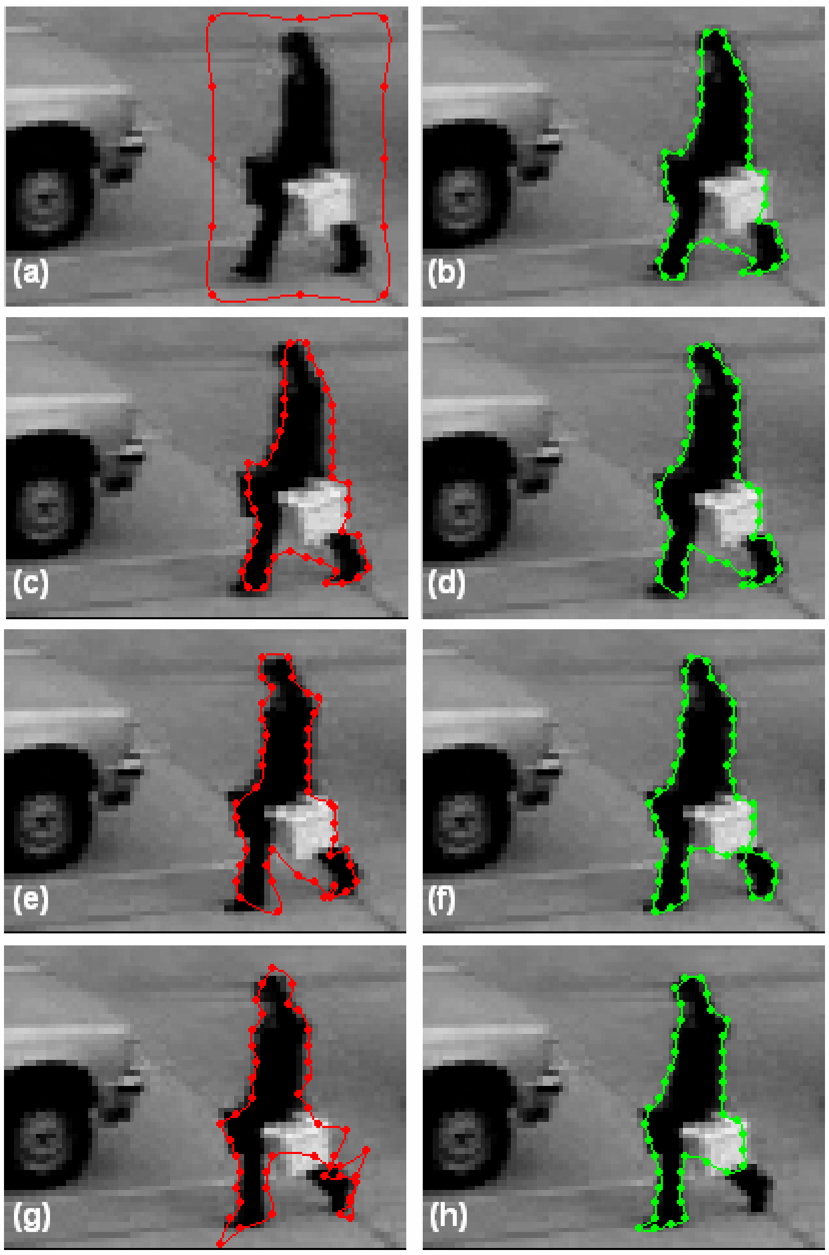


Figure 4.33: The results of tracking an object in a sequence of images, using the attractable snake model and the above motion-compensation algorithm (δ=1). The snake “misses” the target after three frames.

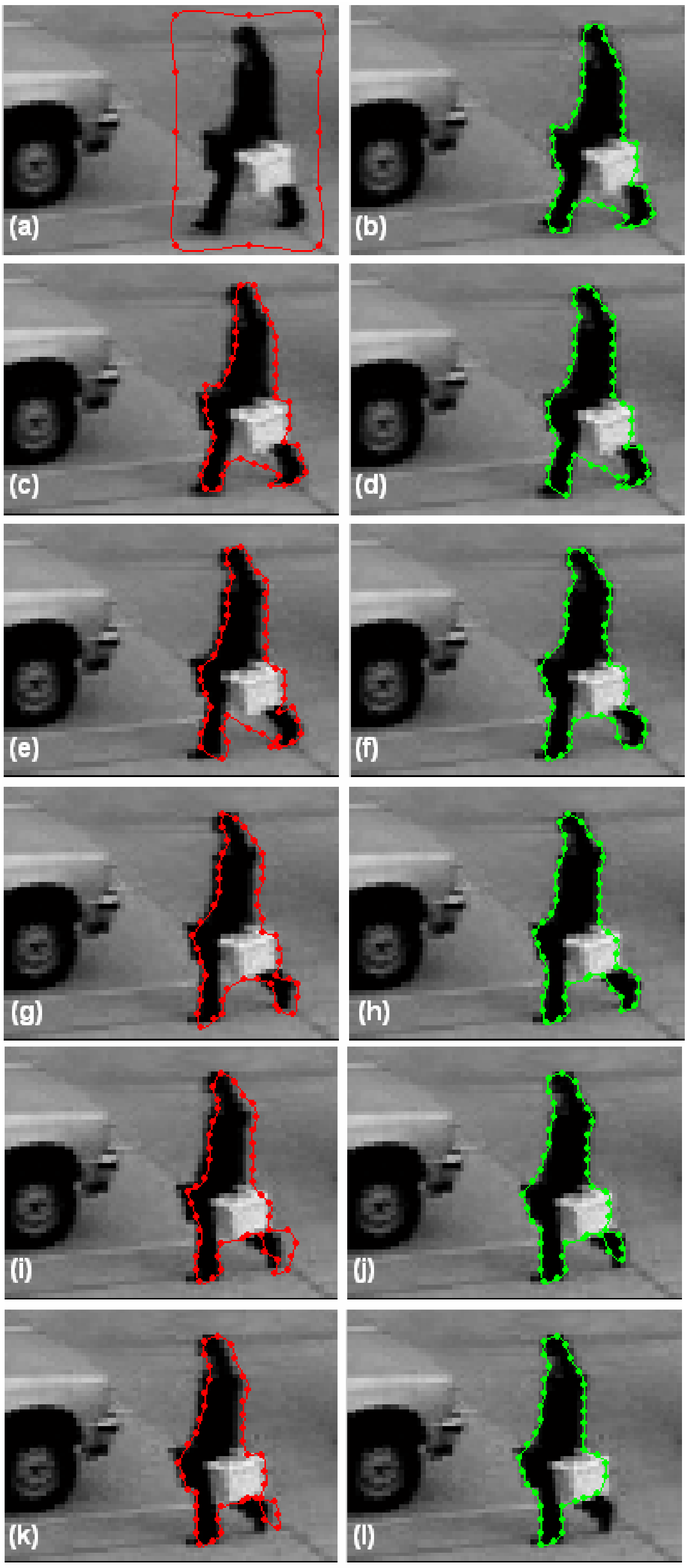
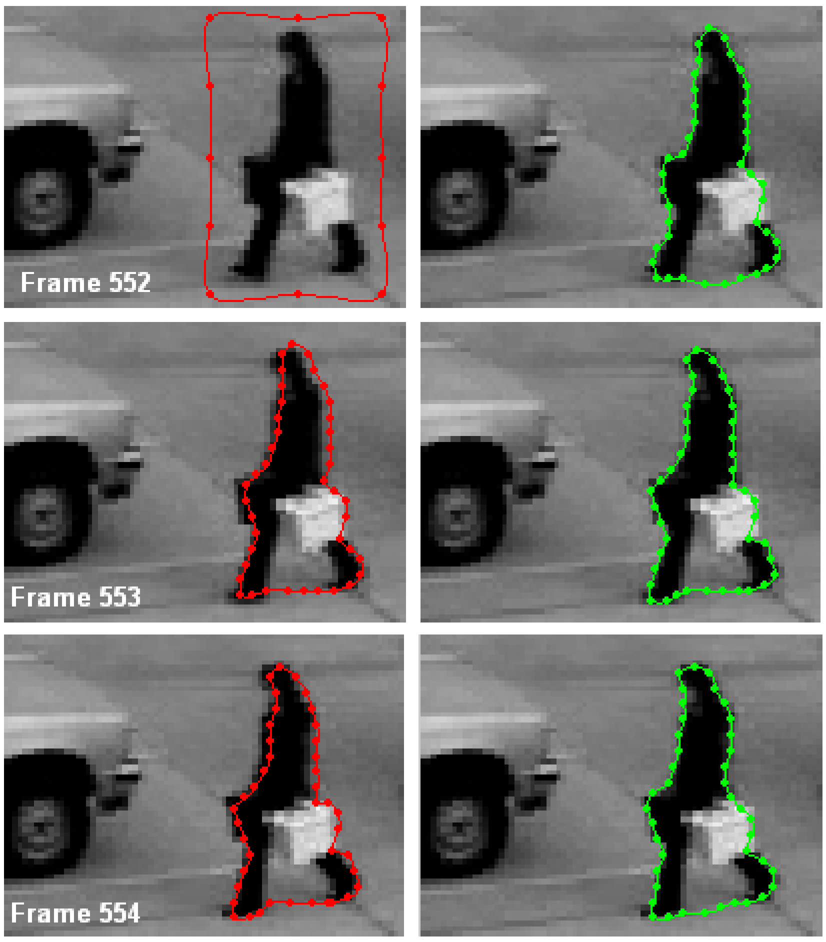


Figure 4.34: By decreasing the 'compensation weight', δ, to 0.1, the snake "misses" the target after 5 frames.

4.3.2 A tracker that uses the "scale-space" model

In the previous subsection we saw that the simplified algorithm of motion compensation, even with a small compensation weight (δ = 0.1), may lead to an inaccurate initial contour, partially away from the target, either in or out of the target shape. Such cases usually result in a convergence failure when using the attractable snake model, mainly due to the action of the feedback mechanism. In the "scale-space" model the feedback force does not exist and moreover, the convergence process starts in an edge image of increased capture range. Thus the snake model is more likely to correct an inaccurate estimation of the initial contour. The corresponding results of using this tracking algorithm on the sequence of images of figure 4.34 are shown in the following figure. The parameter set is: $\alpha = 0.7$, $\beta = 0.5$, $\gamma = 1$, $l_{des} = 3$, $T_g = 0.12$ and $\delta = 0.1$.



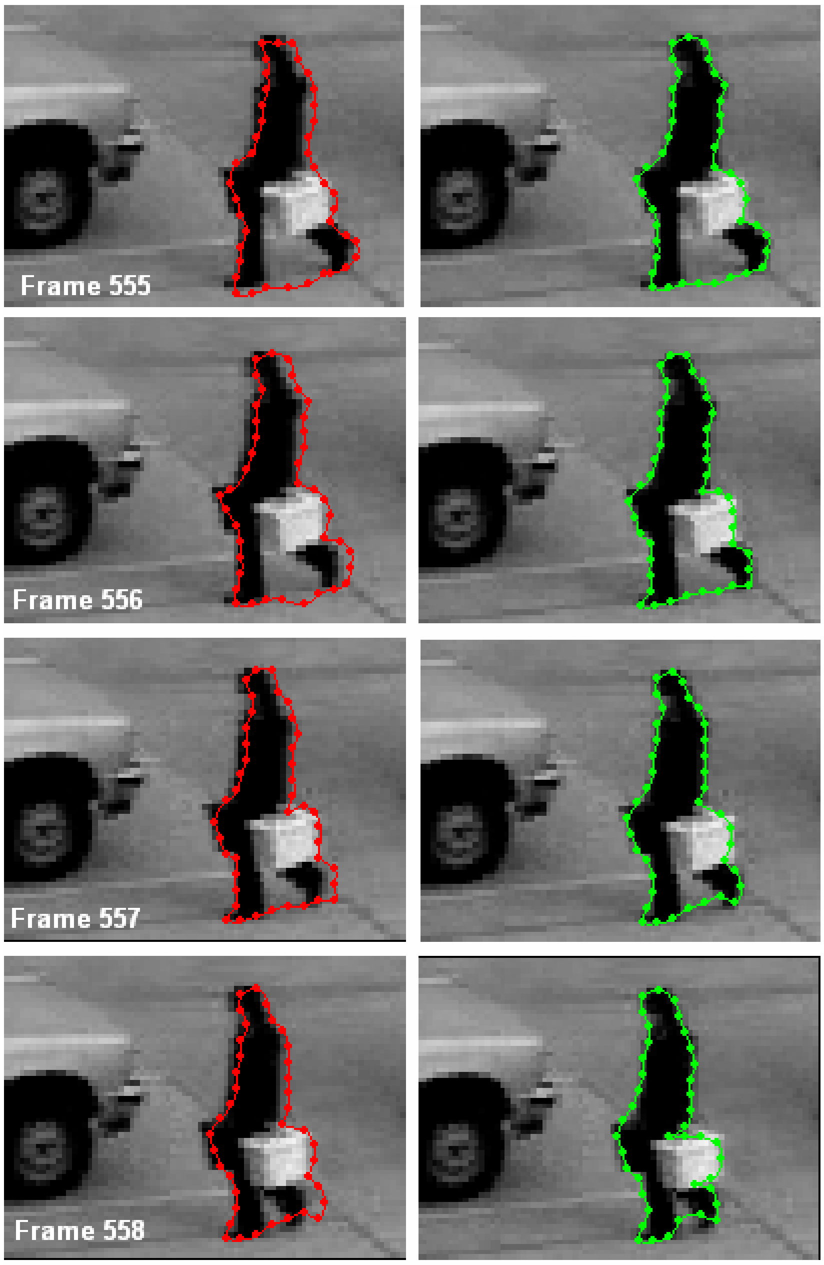
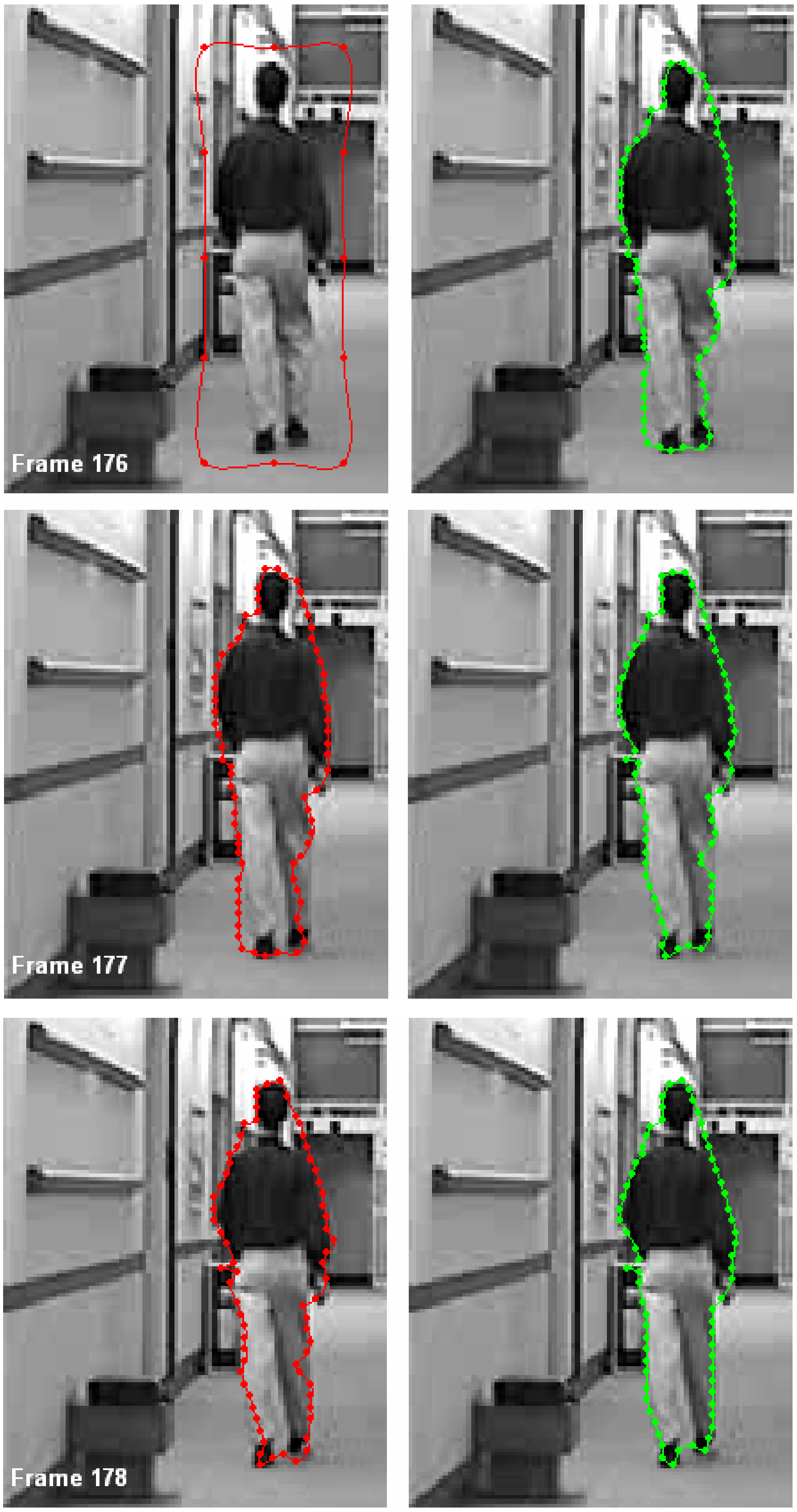
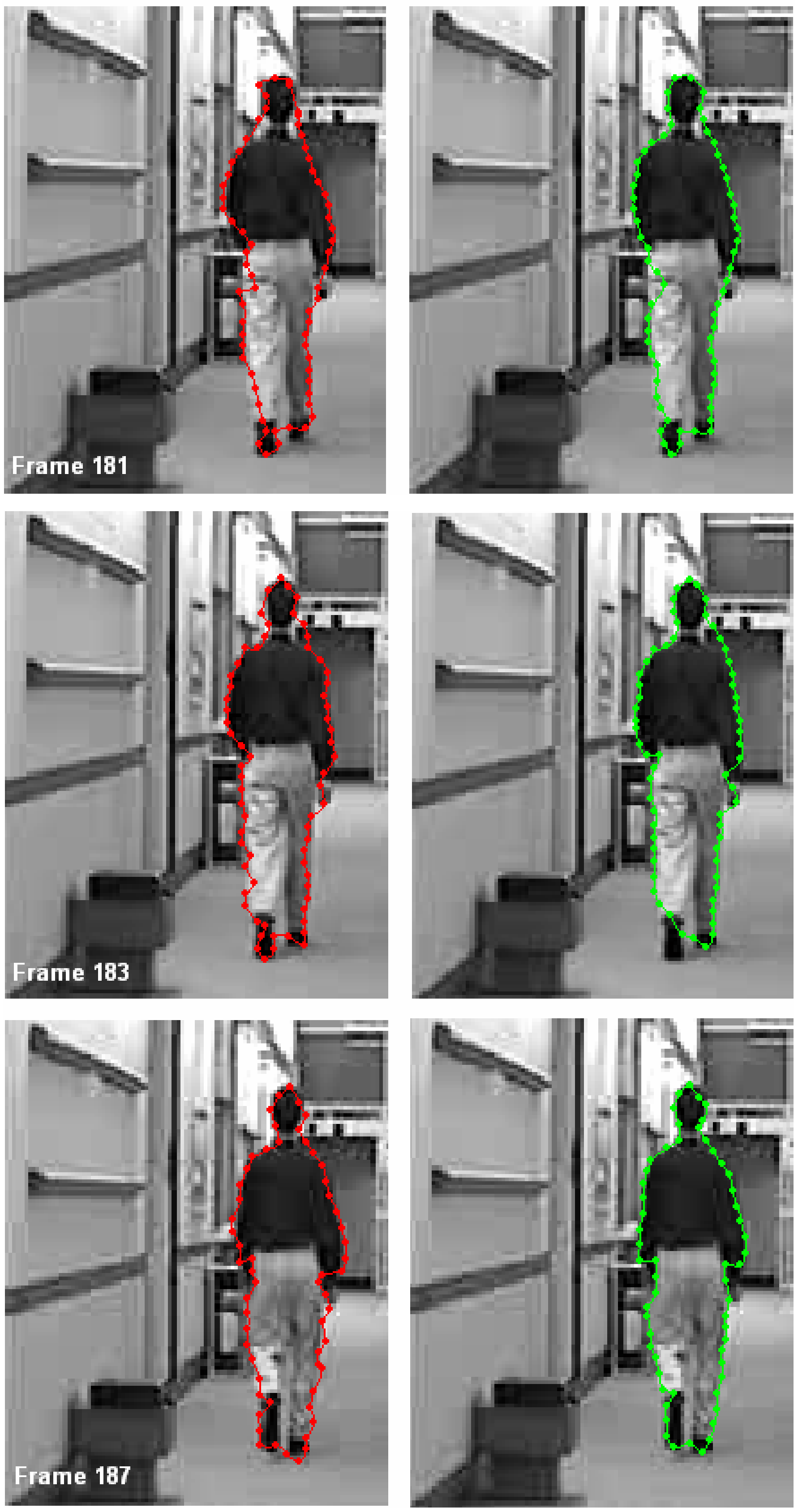


Figure 4.35: Tracking of object using the “scale-space” model (video A).

It is clear that, if we can tolerate a coarse target outline (i.e. an outline that does not enter the concavity), the tracker works quite effectively. Another example is shown in the following figure.


Frame 176
Frame 177
Frame 178


Frame 181
Frame 183
Frame 187

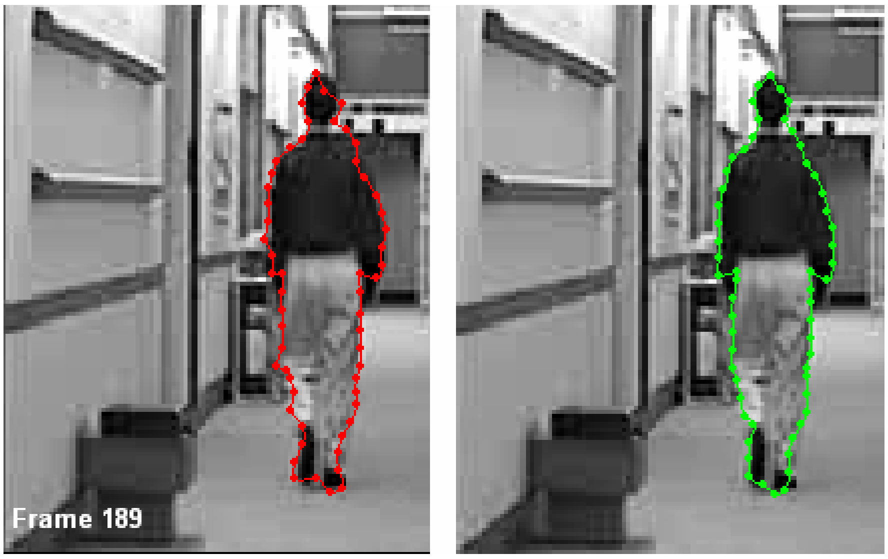


Figure 4.36: Tracking of object using the "scale-space" model (video B).

We can see that the snake misses some parts of the target contour in some frames (e.g. frames 181 or 183), but the convergence failure is usually not propagated to the following frames as it is in the case of the attractable snake model; on the contrary, in some cases we can observe a "recovery" of the "missing" parts (e.g. frame 187).

As a conclusion, we can say that there is a trade-off between accuracy in the target representation and effectiveness in the tracking process. This implementation of the tracker is less accurate in outlining the target that the implementation with attractable snakes but has lower sensitivity in bad initial-contour estimations, and hence better results in tracking the target in a sequence of images. An appropriate combination of the two methods would probably eliminate the disadvantages and maintain the advantages of the two sides. For example, after the application of the scale-space model we can find the snaxels of the final contour that haven't reached the target (e.g. by examining their gradient value against a threshold); then we can further move towards the target only the contour segments that these snaxels form, by using the attractable snake model. The fact that the feedback force will not be applied on all snaxels and from the beginning of the process will probably eliminate the above-mentioned problems of the feedback term. Nevertheless, it is hard to decide whether a snaxel of low gradient is 'in front of' a concavity, and requires further movement, or just on a subjective contour, and does not need correction. Therefore this possible improvement of the algorithm has to be examined further.

Chapter 5: Conclusions and future work

5.1 Conclusions

In this work we attempted to examine the possibility of using active contour models (snakes) in order to implement a semi-automatic visual tracker. The system performs a target-state-estimation, followed by a target-state-verification, for each one of a series of images. The state-estimation (initial state) is provided by the user for the first image of the sequence and is generated from the previous movement of the target for the second image onwards. For the state-verification (final state) we tested two different implementations of the snake model: the attractable snake model and the scale-space model.

The addition of the feedback mechanism makes the attractable snake model very effective in outlining the target, as it resolves both of the major problems of the original snake model. On the other hand, the scale-space model can also cope with an initial contour far away from the target in position and shape, but it cannot outline large shape concavities. Both methods are parameter-dependent, in the sense that there is no global parameter set that gives the best results in all cases. However, in this work we propose a parameter set that can work, with small variations, for most cases.

The system was tested on images with noise and cluttered background, i.e. features that can distract the snake from fitting on the target. Techniques like noise suppression and background extraction minimize, but not totally eliminate, this problem. The attractable snake model is more effective than the scale-space model in these cases, as the additional feedback force points towards the target. On the other hand this force may have undesirable effect in the contour “edge gaps” that occur when the edge detector fails to detect target edges with intensity very close to the background intensity; the snake is likely to bypass the “subjective contour” and move towards the target interior, under the influence of the feedback force. The absence of this force in the scale-space model allows the internal energy to restrain the snake in these cases and therefore the scale-space model is more effective when “edge gaps” are present. However it is, in general, slower than the attractable-snake algorithm, as it is actually executed both times for each case.

The simplified way of motion-compensation that is used in this work generally leads to inaccurate initial contour estimations, since it assumes that the movement of the target is almost constant in velocity and orientation. The attractable snake model, due to the action of the feedback force, is more susceptible to a bad estimation, especially when the estimated contour falls within the target interior. This fact reduces the number of subsequent frames on which the tracking algorithm can be applied. On the contrary, the increased capture range of the scale-space method can handle more easily such cases.

As a general conclusion, we could say that the attractable-snake model achieves a satisfactory contour fit on a single frame, but cannot follow the target for a large sequence of images, whereas the scale-space model offers a coarser target representation, but is more effective in the tracking stage.

5.2 Future work

The tracking system of this thesis obtains some good results but also indicates some problems of tracking moving object using snakes. Therefore, there are some improvements that should be made in order to build a more effective tracker.

An important point of the snake algorithm is the construction of an appropriate edge image, since edges are the dominant feature of the snake attraction. Problems like noise, surrounding edges and "edge gaps", should be resolved more efficiently. The edge detection scheme that uses spatio-temporal information, and was mentioned in chapter 4, is a step towards this direction. However, a further processing of the detected edges, such as edge linking, is needed in order to prevent the snake from overrunning the real target contour.

Future work may also take advantage of the modular snake energy in order to add other useful terms in the energy functional. For example, a possible addition could be a second image term, based on the temporal image gradient ($\frac{\partial I(x,y,t)}{\partial t} = \left| I(x,y,t) - I(x,y,t-1) \right|$), instead of the spatial image gradient. This term would actually generate an additional attracting force towards edges that are moving between frames.

The above improvements refer to the “snake-part” of the implementation. In addition, future work should also concern the “tracking-part”. The estimation of the initial contour could be made in a more effective way, for example by using optical flow. In this case though, it is important to note that there is a trade-off between accuracy and speed.

Another potential improvement could be the following. For each frame (from the second onwards), the initial contour could be generated from the relaxed contour of the previous frame, by some way of motion-compensation (snaxel by snaxel). Then, the final positions of the new snaxels (old snaxels after the motion-compensation) could be found by minimizing, not only the energy of the new snake in the current frame, but also the energy of the old (relaxed) snake in the previous frame. This could work as feedback information for the correction of the initial estimation.

Another possible improvement could deal with the full-automatic initialization of the tracking process; the initial contour for the first frame could be found by some kind of pre-processing (e.g. segmentation). The handling of occlusion from other objects is also an important matter. Finally, the proposed algorithm could also be slightly modified so that it could be applied on color and texture images. In this case, other energy terms, based on color and/or texture information, could be used.

References

[1] G. Hager, A Brief Reading Guide to Dynamic Vision. January 2001.

[2] Y. Wu, *Visual Tracking*. ECE510-Computer Vision Notes Series 7, 2001.

[3] D. M. Garvila, *The Visual Analysis of Human Movement: A Survey*. In Computer Vision and Image Understanding, Vol. 73, No. 1, pp. 82-98, January 1999.

[4] J.K. Aggarwall and Q.Cai, *Human Motion Analysis: A Review*. In Computer Vision and Image Understanding, Vol. 73, No. 3, pp. 428-440, March 1999.

[5] M. Turk, *Visual Interaction With Lifelike Characters*. Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, Killington, VT, pp. 368-373, October 1996.

[6] R. Kaucic, B. Dalton and A. Blake, *Real-Time Lip Tracking for Audio-Visual Speech Recognition Applications*. Proceedings of the European Conference on Computer Vision, Cambridge, UK, pp. 376-387, 1996.

[7] B. Dorner, *Hand shape identification and tracking for sign language interpretation*. In Looking at People, International Joint Conference on Artificial Intelligence, Chambery, 1993.

[8] M. Betke, J. Gips and P. Fleming, *The Camera Mouse: Visual Tracking of Body Features to Provide Computer Access for People With Severe Disabilities*. IEEE Transactions on Neural Networks and Rehabilitation Engineering, Vol. 10, No. 1, March 2002.

[9] W. Freeman, K. Tanaka, J. Ohta, and K. Kyuma, *Computer vision for computer games*. In Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition, Killington, pp. 100–105, 1996.

[10] N. Magnenat-Thalmann and D. Thalmann, *Human modeling and animation*. In Computer Animation, pp. 129–149, Springer-Verlag, Berlin/NewYork, 1990.

[11] C. E. Smith, C. A. Richards, S. A. Brandt and N. P. Papanikolopoulos, *Visual Tracking for Intelligent Vehicle-Highway Systems*. IEEE Transactions on Vehicular Technology, Vol. 45, No. 4, pp. 744-758, November 1996.

[12] M. Takatoo, T. Kitamura, Y. Okuyama, Y. Kobayashi, K Kikuchi, H. Nakanishi, and T. Shibata, *Traffic flow measuring system using image processing*. In Proceedings of SPIE, pp. 172-180, 1990.

[13] D. Rueckert, P. Burger, S. M. Forbat, R. D. Mohiaddin, and G. Z. Yang, *Automatic Tracking of the Aorta in Cardiovascular MR Images Using Deformable Models*. IEEE Transactions on Medical Imaging, Vol. 16, No. 5, pp. 581-590, October 1997.

[14] A. Davison, *Mobile Robot Navigation Using Active Vision*. PhD thesis, University of Oxford, 1998.

[15] D.C. Bentivegna, A. Ude, C.G. Atkeson, and G. Cheng, *Humanoid robot learning and game playing using PC-based vision*. In Proc. IEEE/RSJ 2002 Int. Conf. on Intelligent Robots and Systems, Vol. 3, pp. 2449–2454, 2002.

[16] S. Hutchinson, G.D. Hagar, and P.I. Corke, *A tutorial on visual servo control*. IEEE Transactions on Robotics and Automation, Vol. 12, No. 5, pp. 651–670, October 1996.

[17] K. Aizawa and T. Huang, Model-based image coding: Advanced video coding techniques for very low bit-rate applications. Proc. IEEE, Vol 83, No. 2, pp. 259–271, 1995.

[18] J. Park, S. Park, and J.K. Aggarwal, *Human Motion Tracking by Combining View-Based and Model-Based Methods for Monocular Video Sequences*. V. Kumar et al. (Eds.): ICCSA 2003, LNCS 2669, pp. 650–659, 2003.

[19] N. Paragios and R. Deriche, *Geodesic Active Contours and Level Sets for the Detection and Tracking of Moving Objects*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 3, pp.266-280, March 2000.

[20] G. Hager and P. N. Belhumeur, *Efficient Region Tracking With Parametric Models of Geometry and Illumination*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 10, pp. 1025-1039, 1998.

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

[22] K. Tabb and S. George, *Snakes and their influence on visual processing*. Technical Report No 309, Department of Computer Science, University of Hertfordshire, February 1998.

[23] A. Blake, R. Curwen, and A. Zisserman, *A framework for spatio-temporal control in the tracking of visual contours*. International Journal of Computer Vision, Vol. 11, No. 2, pp. 127–145, 1993.

[24] M. Isard and A. Blake, *Condensation – Conditional Density Propagation for Visual Tracking*. International Journal of Computer Vision, Vol. 29, No. 1, pp. 5-28, 1998.

[25] A. A. Amini, T. E. Weymouth, R. Jain, *Using Dynamic Programming for Solving Variational Problems in Vision*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 9, pp. 855-867, September 1990.

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

[27] C. Y. Xu and J.L. Prince, *Snakes, Shapes, and Gradient Vector Flow*. IEEE Transactions on Image Processing, Vol. 7, No. 3, pp. 359-369, March 1998.

[28] L. D. Cohen, *On Active Contour Models and Balloons*. CVGIP: Image Understanding, Vol. 53, No. 2, pp. 211-218, March 1991.

[29] L. D. Cohen and I. Cohen, *Finite Element Methods for Active Contour Models and Balloons for 2D and 3D Images*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, pp. 1131-1147, November 1993.

[30] L. Ji and H. Yan, *Attractable snakes based on the greedy algorithm for contour extraction*. Pattern Recognition, Vol. 35, pp. 791-806, April 2001.

[31] G. Xu, E. Segana, S. Tsuji, *Robust active contours with insensitive parameters*. Pattern Recognition, Vol. 27, No. 7, pp. 879-884, 1994.

[32] M.O. Berger, *Towards dynamic adaptation of snake contours*. World Scientific, Singapore, Proceedings of the Sixth International Conference on Image Analysis and Processing, Como, Italy, pp. 47-54, 1991.

[33] Y. Y. Wong, P.C. Yuen and C.S. Tong, *Segmented snake for contour detection*. Pattern Recognition, Vol. 31, No. 11, pp. 1669-1679, March 1998.

[34] E.T.Y. Lee, *Choosing nodes in parametric curve interpolation*. Computer-Aided Design, Vol. 21, No. 6, pp. 363-370, 1989.

[35] S. Lobregt and M. A. Viergever, *A Discrete Dynamic Contour Model*. IEEE Transactions on Medical Imaging, Vol. 14, No. 1, pp. 12-24, March 1995.

[36] A. Tsixlas, Combination of information from intensity and movement for the detection of objects, with applications on MPEG-4 and 7. Diploma Thesis, Technical University of Crete, Chania, December 2003.