Nat.Lab. Unclassified Report 2002/831

Date of issue: 06/02

Object based motion estimation

An introduction

J.C.Greiner

Authors' address data: J.C.Greiner; J.C.Greiner@stud.tue.nl

Unclassified Report: 2002/831

Title: Object based motion estimation
An introduction

Author(s): J.C.Greiner

Part of project: N/A

Customer: N/A

Keywords: motion-estimation, object based

Abstract: In this report an object based motion estimation (OME) algorithm is discussed. The goals for this assignment were:

- To understand the object based motion estimator (OME) in sufficient detail.
- To implement a working demo tool of this algorithm in order to visualize the effects of the algorithm.
- Obtain some figures on the quality of the vector fields generated by the algorithm and compare them to figures obtained from a 3DRS algorithm.
- A report, which documents the OME in sufficient detail (this report).

The report contains an entry level description of the OME algorithm and contains the implementation of a demo tool, which shows the effect of the algorithm on video sequences. Furthermore, some comparative figures of OME vs 3DRS are presented.

Conclusions: Overall the conclusion can be made that this report is an entry-level document on the OME algorithm, which was not previously available. Further the created program can serve as a demo tool, which illustrates the effect of the OME algorithm.
The performance numbers show that with given a set of limitations the OME algorithm on average produces vector fields which are of lower quality than vector fields created by the 3DRS algorithm. However the subjective quality of the OME vector fields are comparable to 3DRS. The current implementation of the OME algorithm is immature and needs to be improved upon. A new comparison will show more realistic performance numbers of the OME algorithm.

Preface

At the technical university of Eindhoven it is required to do several trainee-ships in order to get aquainted with the engineering profession before starting on a graduation project. One of these trainee-ships (the external trainee-ship) has to be carried out as an internship within a company that operates in the field of electrical engineering. The author has chosen an assignment which spans the external trainee ship and the graduation project.
The first (external trainee ship) part of this total project was carried out at the Philips research laboratories, located in Eindhoven, in the video processing and visual perception group (ViPs). The author was coached by Ir. R.B. Wittebrood and supervised by Prof. G. de Haan on this initial work.
The remainder of the project (the graduate work) shall be completed in the embedded systems and software (ESaS) group, also located at the Philips research laboratories (Eindhoven) under the supervision of Prof. J. van Meerbergen.

This report reflects the work done in the first part of the project. The goals for this part were:

- To understand the object based motion estimator (OME) in sufficient detail.
- To implement a demo tool of this algorithm in order to visualize the effects of the algorithm.
- Obtain some quality measures of the vector fields generated by the algorithm and compare them to figures obtained from a 3DRS algorithm.
- A report, which documents the OME (this report).

The goals of the entire project are:

- A comparison of hardware implementation of the 3DRS and OME estimators in silicon size, power dissipation and performance.
- An RT level design made in A/RT - builder.
- An answer to the following question: Does an algorithm designed (and optimized) for a DSP result in a better / worse ASIC implementation than an algorithm design, which is optimized for an ASIC implementation.

Contents

Chapter 1

Introduction

1.1 General introduction

Motion estimation is a powerful means to improve a variety of video processing systems. Many methods have been developed during roughly thirty years for different applications, such as motion-compensated(MC) filtering for noise reduction, MC prediction for coding and MC interpolation for video format conversion (MC-VFC). This last application is the most demanding as it requires estimation of true motion in a large range for picture rate conversion and with a sub-pixel accuracy for de-interlacing. In recent years several motion estimation algorithms have been developed at the Visual Processing and Visual Perception group of the Philips research laboratories to target the MC-VFC applications. The ultimate goal is to use these algorithms in future television sets to obtain a better image quality and thus a competitive edge over the competition.

One of the algorithms developed is the Object based Motion Estimator algorithm (henceforth dubbed OME). This algorithm was originally developed for use with the Trimedia DSP because other known algorithms like a full search block matcher or the three dimensional recursive search algorithm (henceforth referred to as 3DRS) required too much hardware resources to be able to run real time on a Trimedia producing usable results, and were therefor not suited for it. Because the software (Trimedia) implementation of the OME algorithm compares favorably with a software implementation of 3DRS, an hardware implementation of OME could prove useful in consumer electronics. Perhaps the hardware could lead to a better motion estimation than the 3DRS algorithm, which already has been successfully implemented in hardware for use in television sets.

This report marks a way-point in a project, which entails the conversion from the DSP oriented algorithm to an implementation in VHDL / Verilog. This VHDL / Verilog implementation can then be used as a building block for the implementation of MC-VFC in an ASIC. The results of the OME algorithm implementation shall be continuously compared to the 3DRS algorithm, as this algorithm serves as a suitable reference point.

Paragraph 1.2 will explain the principles of block-matching based motion estimation, which is currently the most commonly used form of motion estimation for commercial use. Paragraph 1.3 will briefly explain the 3DRS algorithm and in paragraph 1.4 an overview of the OME algorithm is given. Paragraph 1.5 will contain a comparison between these two algorithms. In chapter 2 a more in depth description of the OME algorithm is given, while the actual implementation is discussed in chapter 3. A performance comparison of OME vs 3DRS will be given in chapter 4, and the overall conclusions will be drawn in chapter 5. This chapter will further contain a description of what remains to be done in the remainder of the project.

1.2 Motion estimation

In the most commonly used (block-matching) methods, a displacement vector is assigned to the center $\vec{X}$ of a block of pixel positions $B(\vec{X})$ in the current frame $F(\vec{X}, n)$ by searching a similar block within a Search Area $SA(\vec{X})$, also centered at $\vec{X}$, but in the previous frame $F(\vec{X}, n-1)$. The similar block has a center, which is shifted with respect to $\vec{X}$ over the displacement vector $\vec{D}(\vec{X}, n)$. To find $\vec{D}(\vec{X}, n)$, a number of candidate vectors $\vec{C}$ are evaluated applying an error measure $\epsilon(\vec{C}, \vec{X}, n)$ to quantify block similarity. For an example of this procedure see figure 1.1.
More formally, CS^{max} is defined as the set of candidate vectors $\vec{C}$, describing all possible

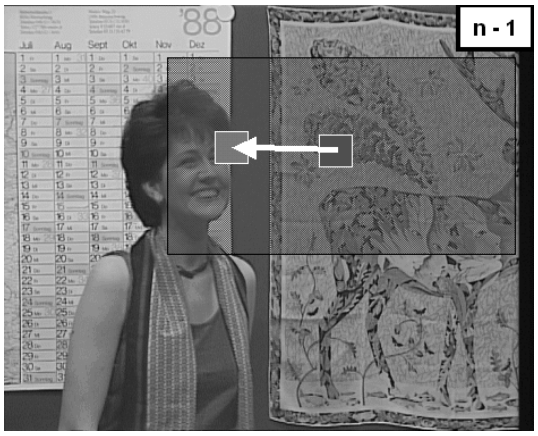

Figure 1.1: *The image shows a search area (the grey rectangle) in the previous frame n - 1. Within this frame a match for the currently processed block is sought after. The dark grey block indicates the position of the currently processed block (in frame n), the light grey block represents its match in the previous frame n - 1.*

displacements (integer on the pixel grid) with respect to $\vec{X}$ within the search area $SA(\vec{X})$ in the previous frame:

$$CS^{max} = \left\{\vec{C} | -N \leq C_x \leq N, -M \leq C_y \leq M\right\} \tag{1.1}$$

where N and M are constants limiting $SA(\vec{X})$. Furthermore, a block $B(\vec{X})$ centered at $\vec{X}$ and of size X by Y consisting of pixel positions $\vec{x}$ in the present frame n, is now

considered:

$$B(\vec{X}) = \Big\{\vec{x} | (X_x - 0.5X \leq x \leq X_x + 0.5X) \wedge (X_y - 0.5Y \leq y \leq X_y + 0.5Y)\Big\} \quad (1.2)$$

The displacement vector $\vec{D}(\vec{X}, n)$ resulting from the block-matching process, is a candidate vector $\vec{C}$ which yields the minimum value of an error function $\epsilon(\vec{C}, \vec{X}, n)$:

$$\vec{D}(\vec{X}, n) \in \Big\{\vec{C} \in CS^{max} | \epsilon(\vec{C}, \vec{X}, n) \leq \epsilon(\vec{F}, \vec{X}, n) \forall \vec{F} \in CS^{max}\Big\} \quad (1.3)$$

If, which is the common case, the vector $\vec{D}(\vec{X}, n)$with the smallest matching error is assigned to all pixel positions $\vec{x}$ in the block $B(\vec{X})$:

$$\forall \vec{x} \in B(\vec{X}) : \vec{D}(\vec{X}, n) \in \Big\{\vec{C} \in CS^{max} | \epsilon(\vec{C}, \vec{X}, n) \leq \epsilon(\vec{F}, \vec{X}, n) \forall \vec{F} \in CS^{max}\Big\} \quad (1.4)$$

rather than to the center pixel only, a large reduction of computations is achieved. As an implication, consecutive blocks $B(\vec{X})$ are not overlapping.
The error value for a given candidate vector $\vec{C}$ is a function, cost, of the luminance values of the pixels in the current block and those of the shifted block from a previous frame, summed over the block $B(\vec{X})$:

$$\epsilon(\vec{C}, \vec{X}, n) = \sum_{\vec{x} \in B(\vec{X})} cost\Big(F(\vec{x}, n), F(\vec{x} - \vec{C}, n - p)\Big) \quad (1.5)$$

A common choice for p is either 1 or 2, depending on whether the video signal is interlaced or not.

The downside of this method is that the the set of candidate vectors is very large, and thus creates a huge computational burden. In order to reduce this burden more efficient search strategies have been proposed and implemented. The next section will discuss an algorithm, which uses a more efficient strategy. Other search strategies can be found in [1].

1.3 Overview of the 3DRS algorithm

In order to reduce the computational burden of (full) block-matching based motion estimation, but still to be able to produce valid motion vectors for each block the 3DRS algorithm uses the following assumptions:

- Objects within a frame (such as persons) are larger than blocks.
- Objects have inertia.

The consequence of the first assumption is that the motion vector of the object in the current block can be found in at least one of the blocks in the direct vicinity of the current block. Consequently, it makes no sense to evaluate all possible vectors within the

search range; it should be sufficient to evaluate vectors taken from a candidate set CS^{max} consisting of the direct spatial neighbors:

$$CS(\vec{X},n) = \left\{\vec{C} \in CS^{max} | \vec{C} = \vec{D}(\vec{X} + \begin{pmatrix} iX \\ jY \end{pmatrix}, n)\right\}, i,j \in \{-1,0,1\} \qquad (1.6)$$

There are two problems with this candidate set:

1. Not all neighbors can be available yet (causality problem)

2. At initialization, all vectors are zero.

The first problem is solved with the second assumption; those vectors that have not yet been calculated in the current image are taken form the corresponding location in the previous vector field, profiting from object inertia.
In case the blocks are scanned from top left to bottom right, the candidate set is defined as:

$$\begin{aligned} CS(\vec{X},n) \quad = \quad & \left\{\vec{C} \in C^{max} | \vec{C} = \vec{D}(\vec{X} + \begin{pmatrix} kX \\ -Y \end{pmatrix}, n) \bigvee \right. \\ & \left. \vec{C} = \vec{D}(\vec{X} + \begin{pmatrix} iX \\ jY \end{pmatrix}, n-1) \right\}, \\ & k \in \{-1,0,1\} \qquad i \in \{-1,0,1\} \qquad j \in \{0,1\} \end{aligned} \qquad (1.7)$$

This candidate set implicity assumes spatial consistency (due to the first assumption) and/or temporal consistency (due to the second assumption) [1].
The second problem can be solved by adding an update vector, which results as the sum of either of the spatial candidates and a 'noise vector'. It is possible to further reduce the number of candidate vectors by simply omitting some of the spatio-temporal predictions from the candidate set. A graphical representation of such a sub set can be found in figure 1.2. The method realizes, due to the inherent smoothness constraint, very coherent and close to true-motion vector fields, which is very suitable for scanning format conversion. The 3DRS algorithm was the first motion estimator to be used in a consumer product. More information can be found in [2].

1.4 Overview of the OME algorithm

The idea of the OME algorithm is to divide a frame in several objects rather than blocks, each object describing movement in a certain direction. This way, a single object describes image regions with similar motion. Objects consist of a set of blocks and are subject to the following assumptions (notice similarity with the 3DRS algorithm):

[1] Even if the assumptions are false, the consistency in the vector field results anyway, as there are no other candidates available.

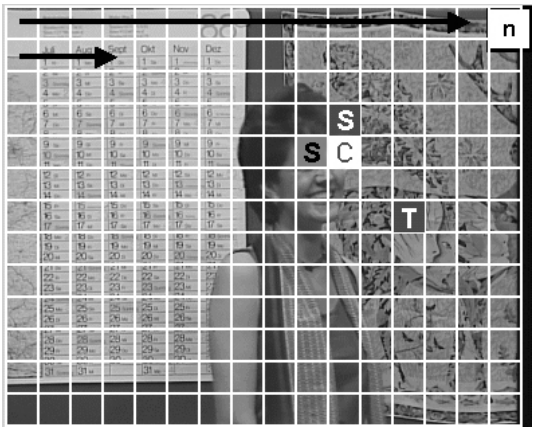

Figure 1.2: *A graphical representation of a candidate subset. The arrows indicate the scanning direction of the image. The block marked C is the currently processed block. The blocks marked S and T are spatio-temporally near blocks. To one of the spatial candidates a noise vector will be added to form the random candidate.*

- They have inertia, moving objects take time to speed up, slow down or change direction. A calculated motion vector will therefor change gradually. The result is that the obtained vector fields are temporally consistent.
- They are bigger than blocks. This means that its probably correct to assume that blocks close to the one currently evaluated have the same motion model. So the vector field is spatially consistent.

In order to perform motion estimation, either the movement of the objects must be known, so we can determine the location of the objects, or the location of the objects must be known so the motion of these objects can be determined. The process of dividing a frame (image) into objects is called motion segmentation. Since, neither the location of the objects, nor the movement of the objects is known a priori, this presents a chicken and the egg problem. See Figure 1.3.

Initially there are no objects defined and thus no motion model can be estimated for them. This problem needs to be overcome. The algorithm assumes that the objects in the image have no motion, ergo a zero motion model is assigned to them. The algorithm then tries to segment the motion models onto the image. Assuming there is motion in the image, the found motion models will not be correct. The algorithm records the image regions is which motion is present, i.e. which require a non-zero motion model. In the next recursion of the parameter estimator, these image regions are used to refine the motion models of the objects, which were found in the previous recursion. With the refined motion models a better segmentation of the image becomes possible. By recursively looping through these models, the algorithm should converge to a valid solution. Figure 1.4 shows a few iterations of the OME algorithm if there are 3 objects available for the OME algorithm. (a non moving object plus two moving objects).

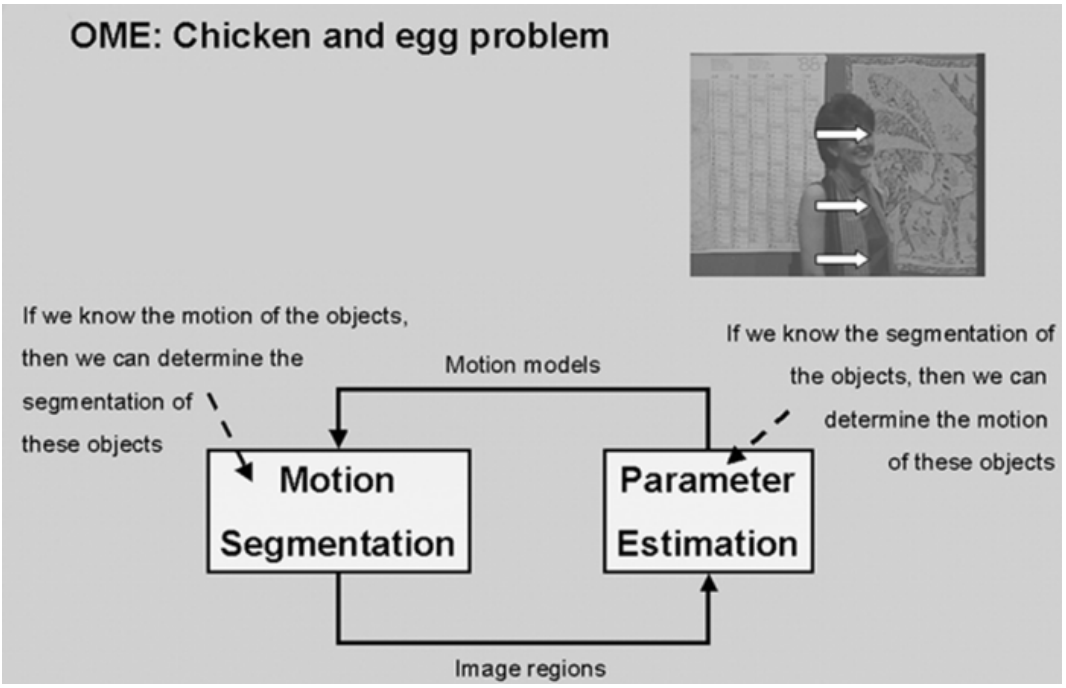

Figure 1.3: *The chicken and the egg problem*

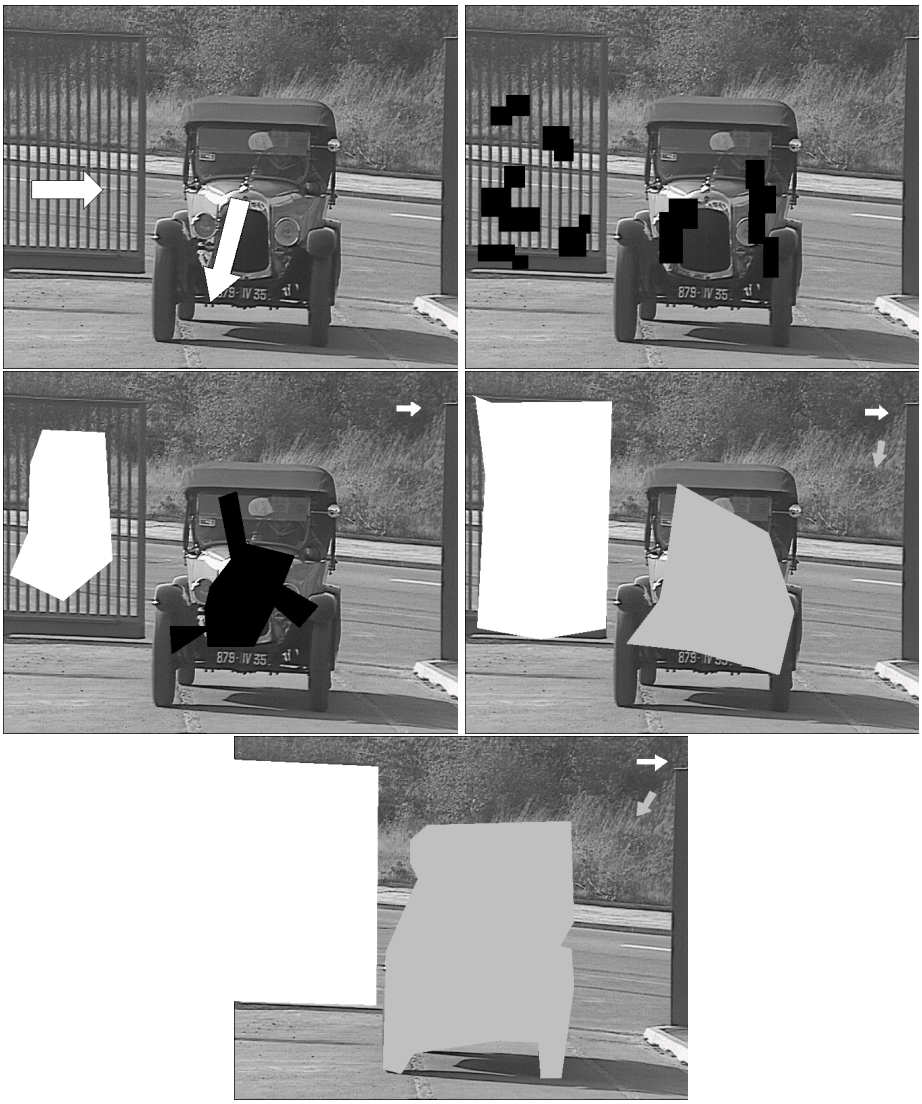

Figure 1.4: *Iteration of the OME algorithm. The upper left picture shows a snapshot of the car-gate sequence, in which the motion vector of the moving objects are drawn. Initially there are no motion models available. In the motion segmentation regions which require another motion model are recorded (the black regions in the upper right picture). In the next iteration of the OME algorithm the parameter estimator finds one motion model (depicted by the white arrow in the top right corner of the middle left picture). In the motion segmentation this model is mapped on to the white area in the same picture. However it also records some regions where the found (white) motion model does not fit. These regions are indicated by the black region. Proceeding with the next iteration the parameter estimator now finds two motion models: one is depicted with the white arrow in the top right corner of the middle right picture, the other motion model is depicted by a grey arrow in that same picture. In the motion segmentation these model are mapped on to the white and grey areas respectively. In the fourth iteration the motion models are refined (the models are again depicted by the grey and white arrow, now in the lower image) and a new, more accurate segmentation is found, as indicated by the grey and white areas. This refinement will continue, until a scene change occurs. After that this convergence process will start again.*

1.5 OME vs 3DRS

In 3DRS a motion vector is assigned to a block. For each block in the frame a new vector is obtained. In OME a block is assigned to an object. To each object a motion model is assigned, which has to be determined prior to the block-to-object assignment.
If each block within a frame is considered an object with its own motion model, OME reverts to an elaborate implementation of 3DRS. Elaborate in the sense that instead of using a random noise vector to ensure convergence, a logarithmic search towards a better vector is started which provides a new vector.

A previous research project was started to see if a software implementation of MC-VFC could be a viable solution. Since the 3DRS algorithm was already available at the time, the researchers tried to implement that algorithm. In order to be able to run on a software platform (chosen to be the Trimedia DSP), the number of operations in the algorithm had to be reduced. This reduction was accomplished by matching blocks on sub-sampled video data. However, a match based on sub-sampled data is more susceptible to aliasing. The software implementation of 3DRS showed an unacceptable loss in quality because of this. This trigger the development of the OME algorithm. Since the OME algorithm uses more data to obtain an object motion model, it produces more stable results if the motion estimation is done on sub-sampled video data. The results of OME vs 3DRS running on Trimedia can be seen in [3] and [4].

Since the OME algorithm compares favorable to 3DRS when run on Trimedia, a comparison of the two if the operations count doesn't need to be reduced (i.e. in an hardware solution) is interesting. If the outcome would be that OME still compares favorably then it could replace 3DRS in consumer electronics. A comparison of OME vs. 3DRS without sub-sampling will be given in chapter 4.

Chapter 2

The OME algorithm

2.1 Short introduction

In paragraph 1.4 the object based motion algorithm was summarized. As explained in that section, the algorithm uses motion models to describe motion. The motion model will be defined in paragraph 2.2. Paragraph 1.4 also showed that the algorithm consists of two subroutines, which are executed sequentially. These two different routines of the algorithm will be discussed in depth in the following sections. Paragraph 2.3 will describe the parameter estimation part of the algorithm (which determines the motion models of the objects), the motion segmentation of the frame (image) shall be discussed in paragraph 2.4.

2.2 Motion models

With every object O in a frame, a parameter vector $\vec{P}_o(n)$ is associated. This parameter vector describes the motion of object O at instance n. Motion models including scaling parameters (see [3], [5]) or rotation parameters ([6]), can be used for such a motion description. Although such elaborate motion models can be implemented in the discussed algorithm, they will not be discussed here. In this report a model, which can only describe translation is chosen. So the motion model simplifies to a motion vector. Therefor the parameter vector $\vec{P}_o(n)$ equals the displacement vector $\vec{D}_O(\vec{X},n)$,which is defined as:

$$\vec{D}_O(\vec{X},n) = \begin{pmatrix} S_x(O,n) \\ S_y(O,n) \end{pmatrix} \tag{2.1}$$

using $\vec{D}_O(\vec{X},n)$ for the displacement vector of object *O* at block location $\vec{X} = (X,Y)^T$ in the frame with index *n*. S_x and S_y are parameters which describe motion in the x and y direction respectively (expressed in a displacement of an integer amount of pixels).

2.3 The parameter estimator

The function of the parameter estimator is to determine a set of motion parameters (motion model) for each object O so a segmentation of the image into several image regions can be made in the motion segmentation unit. The set of parameters used in the algorithm can be chosen to be either one parameter S_x (only horizontal translation can be described) or to two parameters (S_x and S_y). A search is started for a set of parameters (or parameter), which describe the motion of the objects better than previously known. The algorithm does so by means of a logarithmic search, updating the previously estimated parameter as the search advances. At every stage of the search three candidates are tested: the previously estimated parameter set c and the previously estimated parameter set with an update parameter u_i from an update parameter set U_I added to or subtracted from it. This yields the candidate set $\{\overrightarrow{c-u_i}, \vec{c}, \overrightarrow{c+u_i}\}$, . Since only translation can be described the resulting parameter vector $\overrightarrow{Best}$ is a the same as a motion vector. In formula form:

$$\begin{aligned} \vec{c} &= \begin{pmatrix} S_x \\ S_y \end{pmatrix} \\ \overrightarrow{Best(i)} &= \min(cost(\overrightarrow{c-u_i}), cost(\vec{c}), cost(\overrightarrow{c+u_i})) \end{aligned} \tag{2.2}$$

with u_i an element in a set of update vectors U_I, with I the number of iterations in the logarithmic search, cost() some cost function and S_x and S_y the motion parameters of the previous frame. For an eight step implementation of the logarithmic search with two motion parameters S_x and S_y, U_I could be defined as:

$$u_1 = \begin{pmatrix} 8 \\ 0 \end{pmatrix}, u_2 = \begin{pmatrix} 0 \\ 8 \end{pmatrix}, u_3 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}, u_4 = \begin{pmatrix} 0 \\ 4 \end{pmatrix},$$

$$u_5 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, u_6 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, u_7 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, u_8 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

For an example of the logarithmic search see figure 2.1. To determine the best matching candidate use is made of a cost function that compares the current frame with a previous one using a set of $feature$ points, which represent objects. These $feature$ points are blocks (consisting of 8 x 8 pixels) on which the currently tested object in the previous frame and in the current one can be compared. They shall be discussed in section 2.4.4.
If motion is correctly described by the set of parameters then a cost function will be minimized. The cost function compares the feature points of the current frame and the feature points in the previous frame, which are obtained by displacing the feature points of the previous frame according to the motion parameters. The cost function uses Summed Absolute Differences (SAD), it is therefor named SAD. In formula form the SAD can be expressed as:

$$SAD = \sum_{\forall fp} W_{fp} \sum_{\forall x \in fp} \Big(F_p(\vec{x} + \vec{D}, n-1) - F_c(\vec{x}, n) \Big) \tag{2.3}$$

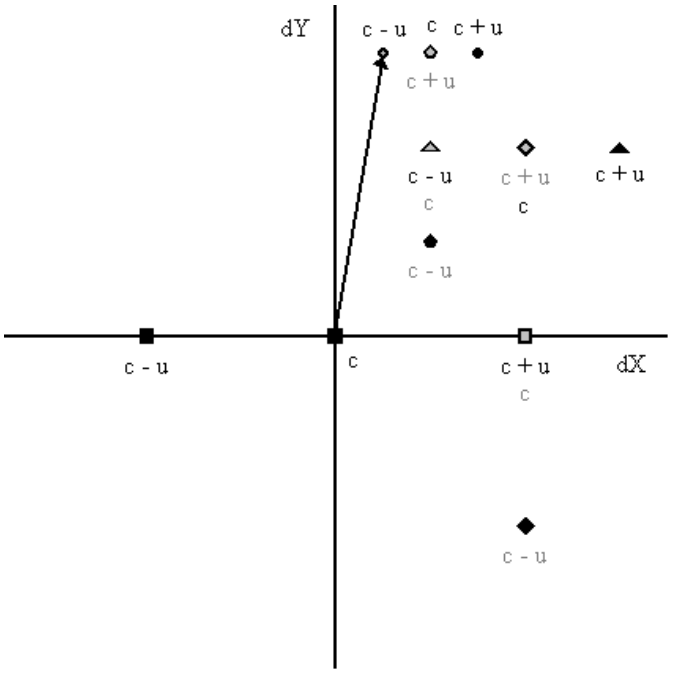

Figure 2.1: *The logarithmic search. The picture shows several iterations of the logarithmic search in the vector displacement domain (5 iterations are shown). The different geometrical shapes indicate the different iterations in the search. The search is done in horizontal (dX) and vertical (dY) direction, and as it advances the update vector u will become smaller. The candidates tested in the horizontal direction are marked with black letters, the candidates tested in the vertical direction are marked with grey letters. In the first iteration, c is initialized with the estimated parameter vector of the previous frame* $\vec{c} = (S_x, S_y)^T$*, then three horizontal candidates are tested (the squares marked c - u, c and c + u), of these the right most one is selected as the best (as indicated by its grey innards). This candidate is then used as the starting point in the second iteration. Now three candidates in the vertical direction are tested (the diamonds marked c - u, c and c + u). Of these the uppermost candidate is selected (marked by its grey innards) and it is used as starting point of the third iteration. The three triangle candidates (marked c - u, c and c + u) are tested, and the leftmost candidate is selected. The procedure of testing three candidates continues with the pentacles and the circles. After the last iteration in the search the vector pointing to the last selected candidate is returned. This vector will be the new motion model (vector) of the object.*

with F_p the luminance value of the previous frame, F_c the luminance value of the current frame, $\vec{x}$ a pixel position in a feature point block fp and $\vec{D}$ the estimated displacement vector. Further:

$$W_{fp} = \begin{cases} 1 & if \quad fp \neq accentuated \\ 4 & if \quad fp = accentuated \end{cases} \tag{2.4}$$

The set of parameters that leads to the lowest SAD is the set that matches best. It is on this set the search continues. Eventually the search terminates, and the result is a new set of parameters. To improve the robustness of the algorithm a low pass filtering can be applied to the newly obtained parameter vectors. For example, this could be done by averaging the currently found best parameter set with the previously found parameter set.

2.4 The motion segmentation unit

2.4.1 Recursive segmentation

A frame is divided in blocks. Consider the situation in which the motion models of all the objects are known. The task at hand then becomes the assignment of an object motion model to each block in the frame, the *motion segmentation*. The assignment of an object (and its motion model) to a block is done on a scan line, i.e. the blocks are considered one by one. To determine which object (and its motion model) best matches a block a number of predictions are taken. These predictions are motion models of objects which were assigned to spatio-temporally neighboring blocks. The set consist of the candidates C, S_H, S_V, $S_H + U$, T and R:

C A zero motion prediction.

S_H A horizontal spatial prediction. This candidate has the motion model of a horizontally neighboring block assigned to it. Because its motion is expected to have a good correlation with the currently processed block, this candidate ensures propagation of an object motion model in the horizontal part of the spatial domain.

S_V A vertical spatial prediction. This candidate has the motion model of a vertically neighboring block assigned to it. Because its motion is expected to have a good correlation with the currently processed block, this candidate ensures propagation of an object motion model in the vertical part of the spatial domain.

$S_H + U$ A prediction consisting of the spatial candidate with a bias vector update. This candidate ensures that even if the motion of a block is slightly different from the motion of the object its assigned to, its actual motion is present in the vector field of the frame. The bias field principle will be explained in section 2.4.3.

T A temporal prediction, which located somewhat further then the spatial prediction. This candidate has the motion model of a spatially near block in the previous frame. Through this candidate objects, which are moving towards the current location are detected.

R A random prediction. This is a motion model from a randomly chosen object. It is needed for convergence when the others predictions are wrong.

The location of the spatio-temporal candidates can be found in figure 2.2. The location of

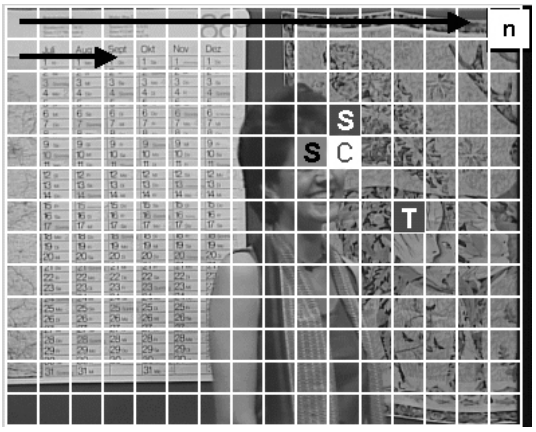

Figure 2.2: *location of the spatio-temporal candidates with respect to the scan direction, the scan direction is indicated by the arrows. The spatial candidates are depicted by the squares marked S, the temporal candidate is marked by a T and the current processed block is indicated by C.*

the temporal prediction is a trade-off between two issues. On the one hand, a large distance to the current block enables accurate detection of the object edges of rapidly moving objects, at the cost of a reduced ability to segment small objects. On the other hand, a very small distance to the current block enables the segmentation of small objects, but limits the accuracy with which objects edges can be estimated. Experiments show that the location of the temporal prediction as depicted in figure 2.2 is a good trade-off. The two spatial predictions enable propagation in two dimensions, this increases the convergence speed of the segmentation dramatically.
The downside of the locations of the predictions in figure 2.2 is that the motion models propagate towards the lower right corner. Since propagation in all directions is desirable, use is made of a rotating scan direction system, where the scanning directions and candidate configurations alter from recursion to recursion. The way this influences the propagation of the motion models can be seen in figure 2.3. The propagation direction is depicted as a grey cone. Additional to the spatial and temporal predictions, other candidates are needed, the random predictions. The random candidate is necessary in the situation that the spatial and temporal predictions are wrong (e.g. when new objects appear or a scene change occurs). Once the random prediction is assigned to a block it will propagate through the spatial and temporal predictions.

So from this set of predictive candidates (C, S_H, S_V, $S_H + U$, T, R) the best matching candidate is selected and its motion model is assigned to the current block. The best match is defined here as the candidate which has the lowest block match error $\epsilon(\vec{X},O,n)$, which is defined here as:

$$\epsilon(\vec{X}, O, n) = \sum_{\forall x \in X} |F_s(\vec{X} + \alpha\vec{D}, O, n - \alpha) - F_s(\vec{X} + (1 - \alpha)\vec{D}, O, n - \alpha)| \quad (2.5)$$

with F_s the luminance values of the sequence, x a pixel position in block X, $\vec{D}$ the displacement vector, and $n + \alpha$ the temporal position for which the current segmentation mask is valid ($0 \leq \alpha \leq 1$). Figure 2.4 shows how the temporal instance $n - \alpha$ is derived.

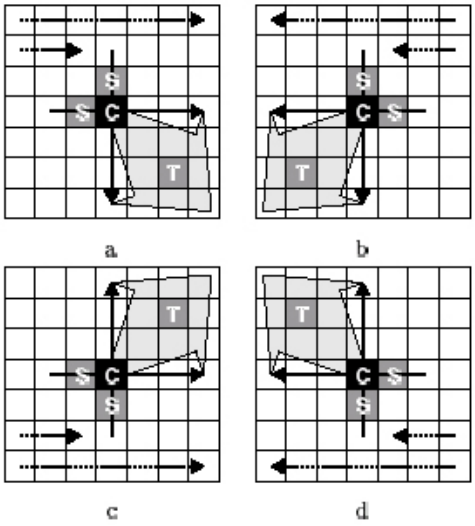

Figure 2.3: *Influence of scanning directions on propagation of motion models. Figures a,b,c and d show four different propagation directions (depicted by the grey cones) produced by four different prediction configurations and scanning directions (indicated by the dotted arrows). Propagation of motion models in every direction is ensured if the prediction configurations and scanning directions are alternated from recursion to recursion.*

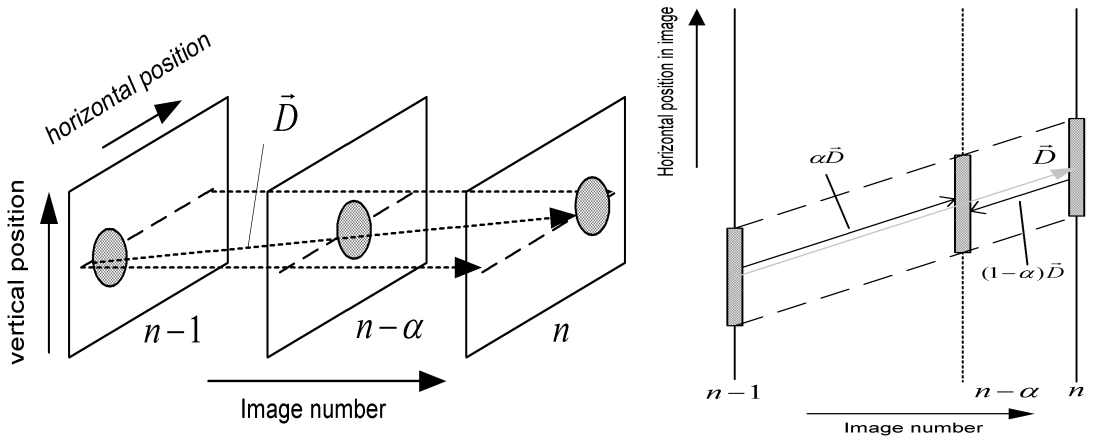

Figure 2.4: *Visualization of SAD calculation in the motion segmentation process. The left picture shows the motion of a ball from one frame to the next. The right picture shows a one-dimensional projection of the moving ball. $\vec{D}$ is the displacement vector of the block from the $(n-1)^{th}$ frame to the n^{th} frame. $\alpha\vec{D}$ and $(1-\alpha)\vec{D}$ are the vectors used to transfer the luminance data of the block from the $(n-1)^{th}$ and n^{th} frame to the $(n-\alpha)^{th}$ frame.*

2.4.2 Penalties

In order to reduce the risk of assigning erroneous models to blocks (which can occur due to noise and repeating image structures), temporal and random predictions are penalized:

$$\epsilon_c^p(\vec{P}, \vec{X}, n) = \epsilon_c(\vec{P}, \vec{X}, n) + p \tag{2.6}$$

The penalty p is used to suppress erroneous assignments. This means that predictions which have a high probability to be wrong should be penalized more. The random prediction has the lowest probability to give the correct motion model and will account for a large amount of errors in case of noise. The temporal prediction has a good correlation to the current block, but since it originates from the previous recursion, it will result in more errors than the spatial prediction. It follows that the penalty p is assigned one of three values, based on the type of prediction used in (2.6), $p \in \{p_s, p_t, p_r\}$. Since the spatial predictions are the most reliable, their penalty p_s is set to zero. The temporal penalty p_t is chosen small and the random penalty is chosen high, so $p_s < p_t < p_r$ holds. On edges, the assumptions as stated above are not true anymore and penalizing therefor might result in less accurate edges and merging of objects. However the increased consistency and accuracy within objects outweighs the decreased accuracy on edges.

2.4.3 The bias field

In order to accurately cover all possible object- and camera movements, the number of motion models need to be increased so camera movements such as zoom can be broken up in several objects, or support such motion with more complex motion models in the parameter estimator, which results in a slower convergence of the motion models. The alternative presented here is to define a bias field $\vec{b}(\vec{X}, n)$. The bias field defines a small adjustment vector for each block in the frame to take care of small deviations between the estimated object motion for a block (as provided by the motion model assigned to that block) and the actual motion in the video sequence of that block. The bias field principle is explained in Figure 2.5.

$$\vec{D}_O(\vec{X}, n) = \begin{pmatrix} S_x(O, n) \\ S_y(O, n) \end{pmatrix} + \vec{b}(\vec{X}, n) \tag{2.7}$$

So now the displacement vector at position $(\vec{X}, n)$ is described by eq. 2.7 instead of eq. 2.1. As mentioned in 2.4.1, the bias vector update of one of the spatial candidates will ensure that the small deviation between the estimated object motion field and the actual motion is corrected. The bias vector is only added to the spatio-temporal candidates mentioned above. The null object and the random object get a bias vector of zero assigned to them. To create the bias field, the segmentation step described in 2.4.1 has to be modified so that the bias field is updated along with the segmentation mask. To that end an extra motion vector $S_H + U$ is tested. This vector is created by adding a bias field update b_u to a spatial candidate (S_H). If this additional vector is chosen during the segmentation the bias field update vector b_u is added to the already predicted bias vector $\vec{b}(\vec{X}, n)$ to obtain

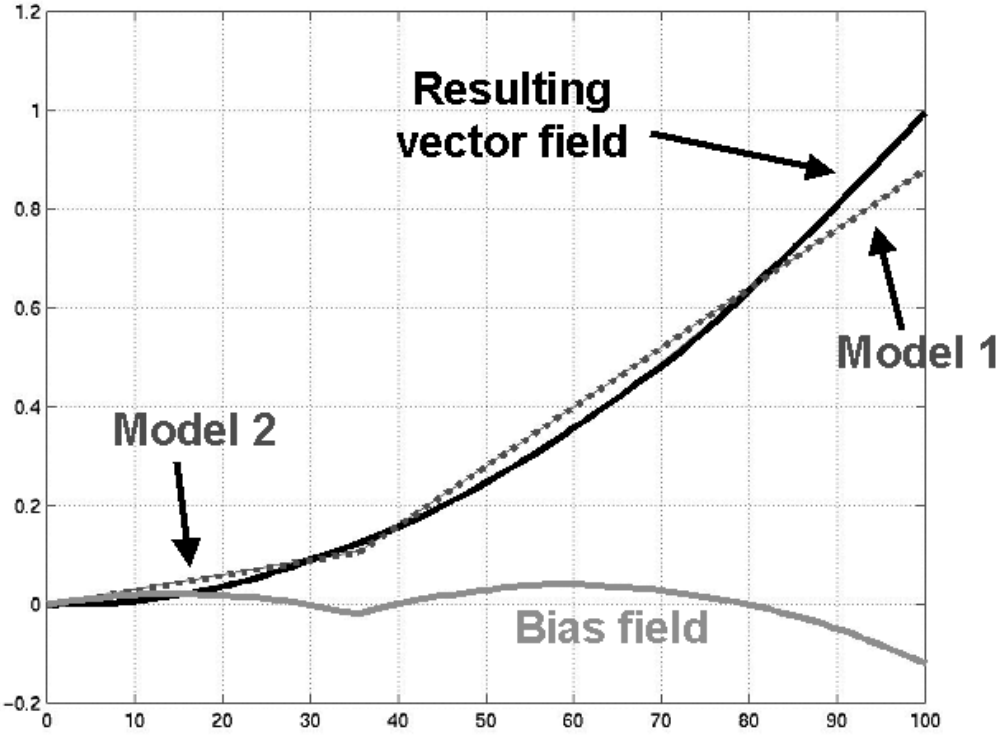

Figure 2.5: *The vector field resulting from the motion models 1 and 2 can only be a crude approximation of the actual motion in the sequence. By adjusting the crude approximation with a bias field, the resulting vector field will resemble the actual motion better.*

the new bias vector for that block:

$$\vec{b}(\vec{X}, n) = \vec{b}(\vec{X}, n) + b_u \tag{2.8}$$

The bias field vectors are clipped in order to ensure that only small deviations between the estimated and actual motion field can be adjusted. This bias vector update b_u is randomly chosen from a bias update set B_U:

$$B_U = \left\{ \begin{pmatrix} \pm 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \pm 4 \end{pmatrix}, \begin{pmatrix} \pm 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \pm 2 \end{pmatrix}, \begin{pmatrix} \pm 2 \\ 0 \end{pmatrix}, \begin{pmatrix} \pm 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \pm 1 \end{pmatrix} \right\}$$

2.4.4 Creating objects

Because the best block - object match (lowest SAD) found could perhaps be improved on, previous objects that didn't match to particularly well to the current block are kept track off. This information is then used to determine feature points for the current object. These feature points are used to represent the objects in the parameter estimator.

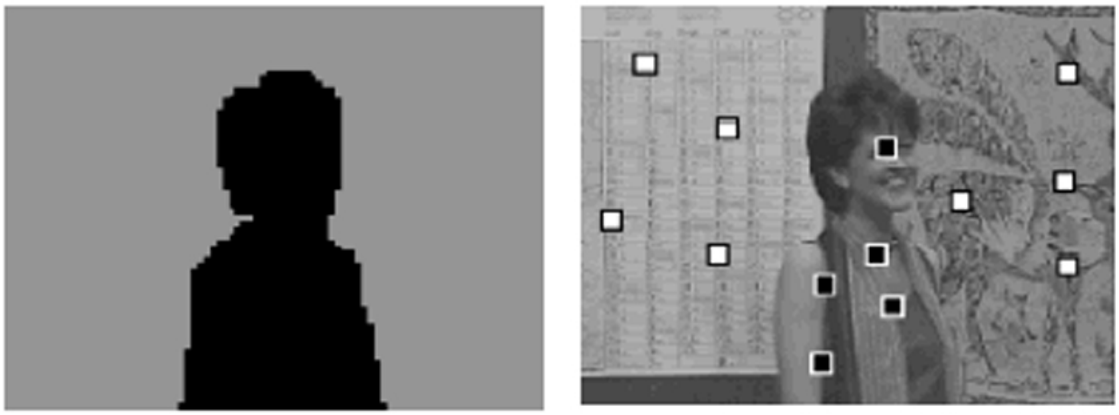

Figure 2.6: *Location of the feature points and the resulting segmentation.*

If the currently processed block didn't match well with other objects and it is assigned to the current object, then this block is probably a good point on which to estimate motion of the current object. So an accent is given to this block to indicate it should weigh more heavily when determining a new set of motion parameters for this object (see section 2.3). Moreover, this weighing also accomplishes a focussing of object parameter estimators. If a feature point is assigned to a certain object (by its weight), it makes no sense for other parameter estimators to use this feature point to improve the motion model of their object. Therefor the parameter estimators will each focus on a particular image region, thereby enhancing the motion model of that region.

In order to estimate motion in the entire image it is necessary that the feature points are distributed evenly over the image, so steps are taken to insure this, the implementation of which can be found in section 3.3. Figure 2.6 shows an example of a segmentation obtained through the use of the OME algorithm. It shows the location of the feature points as well as the resulting segmentation. The white feature points focus on the background, which is almost stationary. The black feature points focus on the lady that is moving from the left to the right. Figure 2.7 shows how the parameter estimators focus on a subset of the feature points, and how the algorithm converges towards a better motion model.

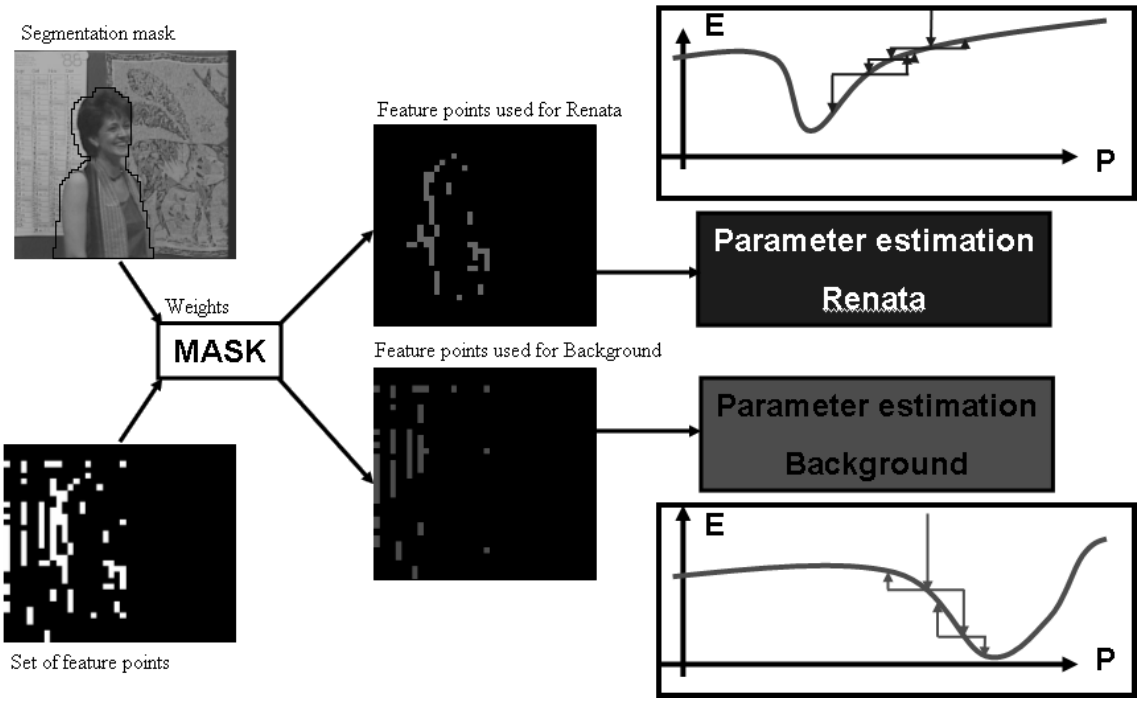

Figure 2.7: *Convergence of the OME algorithm. The figure shows a derived segmentation of a frame in the Renata sequence, when there are two objects available. Below the segmentation the location of the set of feature points is given. These two sources are used to determine which feature points are suitable to estimate motion models of a particular object. This is done by weighing the feature points per object. In this case the distinction is made between Renata and the background. Now the parameter estimator of both Renata and the background use their assigned set of feature points to come up with a better motion model. To determine which motion model is better, a cost function is minimized. With each iteration of the algorithm the parameter estimator converge towards the minimum of this cost function.*

Chapter 3

The implementation of the OME algorithm

3.1 Introduction

In this chapter the implementation details of the OME algorithm shall be discussed. Section 3.2 will describe some general implementation details such as the tracking of variables throughout the algorithm. Section 3.3 will discuss the implementation of the motion parameter estimator, and section 3.4 will discuss the motion segmentation unit, which are the key elements of the OME algorithm. Finally, the implementation of the visual aid is discussed in section 3.5. To aid the reader in understanding the implementation, tables with the variables used are given in the section were they are mentioned.

3.2 General implementation details

In order to be able to track all variables that need to be interchanged between the parameter estimator and the motion segmentation unit a context was created. Other than those variables the context also contains frame and block size information and information on variables, which are only used in the motion segmentation unit or the parameter estimator, although that is strictly not necessary. The algorithm is executed by repeated execution of the motion parameter estimator followed by the execution of the motion segmentation procedure. Since a visualization of the vectors is desirable, the vector information is encoded in the color information, see pseudo code below.

```
for (i = first frame; i <= last frame; i++) {
  Parameter_Estimation();
  Motion_Segmentation();
  Create_Vector_Overlay();
}
```

3.3 Implementation of the parameter estimator

The parameter estimation procedure starts with making a backup of the previous parameters. They are needed for post processing (see section 3.4.4. Then the main loop of the procedure starts, which loops over all objects. Within this loop first the number of available parameters (degree of freedom) is determined, since the number of iterations in the logarithmic is dependant of those parameters. If only a single parameter (only horizontal motion can be described) is available the parameter updates used in the logarithmic search are limited to four (which is sufficient for a search in the horizontal domain). If there are two parameters available to describe motion the number of candidates is eight. The motion parameters, which were found in the previous recursion of the parameter estimator (to determine the parameters of the previous frame) are used as a starting point in the logarithmic search. The search is started by iterating over the update parameters. For each update parameter three SAD calculations are done. The first calculates the SAD if the parameters are not altered, the second and third calculates the SAD if the candidate vector is added to, or subtracted from the known parameters. The SAD values are stored in an array named SAD. Once all three SAD values in the current iteration are calculated, the lowest is determined by hailing the subroutine *best_candidate_index()* . The candidate that is returned is used as a starting point in the next iteration of the search. Then the search procedure is repeated. When the search terminates it has resulted in a new parameter set. To prevent the motion parameters from changing drastically (objects do not change drastically but gradually due to inertia as explained in section 2), the newly found parameters are low pass filtered by averaging them with the previous parameters found for the object. For more clarity a pseudo code description and a flow graph of the parameter estimator are given below:

```
void Estimator ()
{
  Backup_previous_motion_vectors(best_parameter, best_prev_parameter);

  for( all objects) {
    for ( do all iterations in logarithmic search ) {
      Select_Candidates(c-u,c,c+u);
      for(candidates c = 0; c ¡ number of candidates; c++) {
        estimator.SAD[c] = Calculate_SAD ();
      }
    }

    /* store the update which has the lowest sad */
    best_parameter = Best_Candidate_Index();
    best_parameter = ClipParameter(); /* boundary check of the parameters*/
    estimated_parameter[layer]= FilterParameter(best_parameter, best_prev_parameter);
  }
}
```


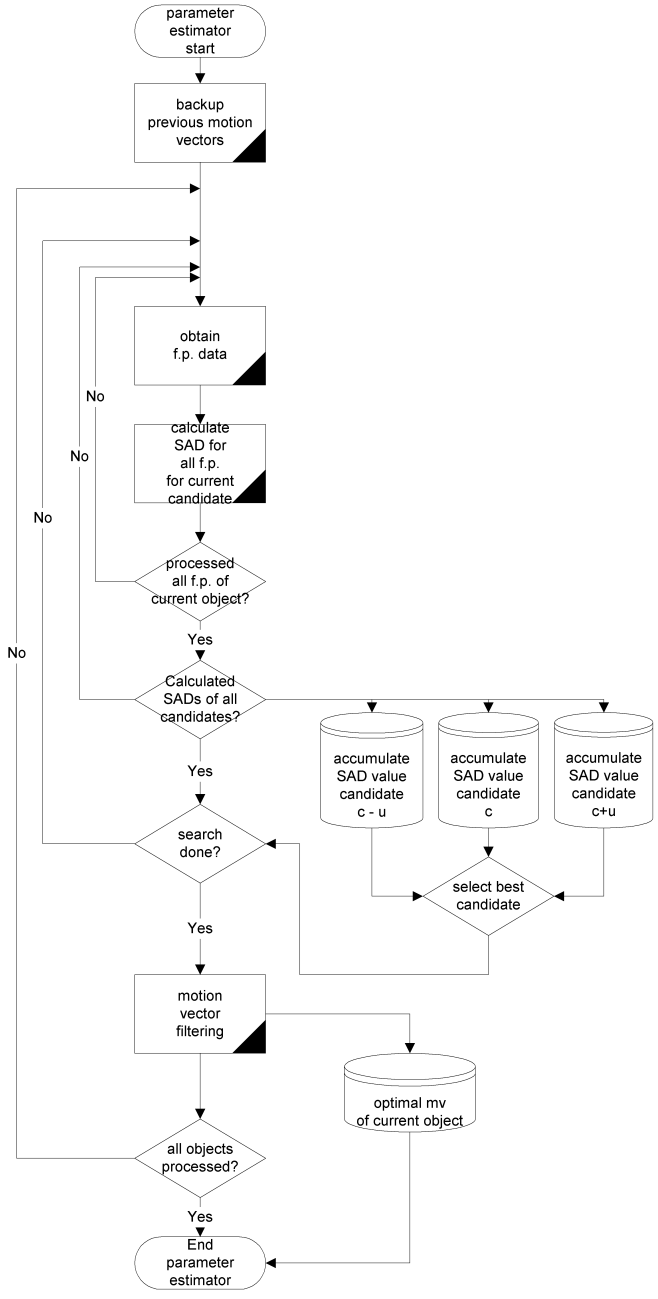

Figure 3.1: *A flow-graph of the parameter estimator.*

3.4 Implementation of the motion segmentation unit

3.4.1 Pre-processing

variable name	small description
sad_min_thr[]	This array holds the minimum SAD value (threshold value) for an object to be considered as a ”no match” for an object.
sad_max_thr[]	This array holds the maximum SAD value (threshold value) for an object to be considered as a ”no match” for an object.

Table 3.1: Variables used in the preprocessing.

The segmentation procedure starts with a pre-processing, which is needed to be able to perform the motion segmentation . The variables used in the preprocessing are given in table 3.1. First a backup of the previously used feature points is made so that if too few new feature points are found in this segmentation iteration, the previous set of feature points can be used. Then the *Get_Thresholds* procedure is invoked. To determine whether or not a block is suitable as a feature point for the current object, the thresholds of the *sad_min_thr* and *sad_max_thr* arrays are used.

$$sad\_min\_thr(object) < SAD(object) < sad\_max\_thr(object) \tag{3.1}$$

These thresholds are set in such a way that if eq. 3.1 holds, there is a good probability that the current object will fit for this block. The contents of the *sad_min_thr* and *sad_max_thr* arrays are set in the *Get_Threshold* procedure. The thresholds are determined as follows:

$$sad\_min\_thr(object) = MAX(sad\_min\_abs\_thr(object), sad\_min\_rel\_thr(object)) \tag{3.2}$$

with $sad\_min\_rel\_thr(object)$ defined as:

$$sad\_min\_rel\_thr(object) = C(object) * SAD\_avg(object) \tag{3.3}$$

with C a constant that is defined per block (chosen to be initially 6 for each object). $sad\_max\_thr(object)$ is defined as

$$\begin{aligned} sad\_max\_thr(object) \quad = \quad & MEDIAN(sad\_thr\_max\_abs\_low(object), \\ & sad\_thr\_max\_rel(object), \\ & sad\_thr\_max\_abs\_high(object))) \end{aligned} \tag{3.4}$$

with $sad\_thr\_max\_rel(object)$ defined as:

$$sad\_thr\_max\_rel(object) = C(object) * SAD\_avg(object) \tag{3.5}$$

with C a constant that is defined per block (chosen to be initially 20 for each object). At the moment these thresholds are not optimized due to time constraints. The last item of pre-processing is the setting of the scan direction (the reason for this was explained in

section 2.4.1). A phase variable determines the current scan direction. It alters between top-to-bottom and bottom-to-top scanning. In pseudo code the preprocessing part looks like this:

/textrm/* backup of previously estimated positions */

Save_Estimate_Positions(&*context);

Get_Thresholds(&*context);

Set_Scan_Direction();

3.4.2 The main loop - first inner loop

variable name	small description
$lay\_array[]$	This array holds the object number to which a candidate is assigned.
$pen\_array[]$	This array holds the penalty which is assigned to a candidate.
$vec\_array[]$	This array holds the vector which was defined in the bias field and / or the bias update vector.
$abs\_dif\_array[]$	Holds the calculated SAD value per candidate.
$abs\_dif\_layer[]$	Holds the assigned SAD value of each layer.
$\overrightarrow{mv}$	The displacement vector. corresponds to $\vec{D}$ of section 2.4.1.
$\overrightarrow{mv\_l}$	Down-scaled version of $\overrightarrow{mv}$, corresponds to $\alpha\vec{D}$ of section 2.4.1.
$\overrightarrow{mv\_r}$	Difference vector of $\overrightarrow{mv}$ and $\overrightarrow{mv\_r}$. Corresponds to $(1-\alpha)\vec{D}$ of section 2.4.1.
$tpos$	Integer variable ($tpos \in \{0...32\}$, which indicates the Temporal Position Of Segmentation.
$TEMP\_RES\_SHIFT$	denumerator of tpos, α (of section 2.4.1) = (tpos ¿¿ TEMP_RES_SHIFT).
$norm\_vec\_sub_x$	Scale factor in needed to obtain sub-pixel accuracy.
$norm\_vec\_sub_y$	Scale factor in needed to obtain sub-pixel accuracy.
x_l, y_l	Left hand side variables.
x_r, y_r	Right hand side variables.
x_l_int, y_l_int	integer pixel grid variables.
x_r_int, y_r_int	integer pixel grid variables.
x_l_frac, y_l_frac	fractional pixel variables.
x_r_frac, y_r_frac	fractional pixel variables.
$sample\_l$, $sample\_r$	luminance pixel values obtained from the previous and current frame respectively.
$high\_sad[]$	Array which records which objects did not match.

Table 3.2: Variables used in the first loop.

The main loop iterates over all block lines and per block line two loops are executed: the first assigns an object to each block (the actual motion segmentation), the second deter-

mines the feature points on which a parameter estimation is done in the estimator unit. These loops shall be discussed sequentially. The variables of the first loop are given in table 3.2.

Fetching the prediction candidates
In the first loop the first order of business is to fetch the set of candidates. These candidates consist of a zero motion candidate, a spatial candidate, a spatial candidate with a bias vector update, a temporal candidate and a randomly selected candidate. To prevent candidates with low correlation to the current block of being easily selected, penalties are assigned per block as described in section 2.4.2. The zero motion candidate, the spatial candidate with a bias field update and the randomly selected candidate are penalized with the random penalty, which is set at a value of 512 which corresponds to an additional difference of 2 per pixel in a block. The temporal penalty for the temporal candidate is set at 128, or 0.5 per pixel extra difference. The penalty of spatial candidate is chosen to be zero. These penalty values are known to give a good result. The information on the several candidates is stored in three arrays: the *lay_array*, the *pen_array* and the *vec_array*. The *lay_array* stores the object number of the candidate, the *pen_array* stores the penalty that belongs that candidate and the *vec_array* stores the bias field vector of the candidate and the bias update (only applies to the spatial candidate, which receives this bias update).

Calculating SAD values - the zero motion candidate
Once the candidates are known, their SAD is calculated and stored in the *abs_dif_array* for future use. In case the tested candidate belonged to the zero motion object, the calculation becomes a simple subtraction of all the pixels of a block in the current frame and those in the previous frame. However, the SAD found has to be corrected by multiplying it with a constant. This correction is necessary because for a candidate belonging to a non-zero motion object a bilinear interpolated value is calculated to obtain quarter pixel accuracy. This quarter pixel accuracy is used in determining the best matching object for the current block. Since only integer operations are permitted (easier to implement), the required accuracy is obtained by up scaling the original pixel data by a factor of 4 in both the horizontal and vertical domain. The motion vectors as supplied by the parameter estimator are already scaled. By subtracting the motion vectors of the current pixel position and performing a bilinear interpolation on the result, the quarter pixel accuracy is achieved.

Calculating SAD values - non zero motion candidates
If the tested candidate belongs to a non-zero object the SAD calculation becomes a little more difficult. First the motion parameter set of the object is converted into a vector. Because the current motion model only covers translation, the conversion is reduced to a copy of the parameters into vectors. The bias field vector is then added to that vector to derive the total displacement vector This vector is then split in two vectors: one is a down scaled version of the original vector ($\overrightarrow{mv\_l}$), the other is the difference vector ($\overrightarrow{mv\_r}$) between the original vector and the down-scaled one:

$$\overrightarrow{mv\_l} = \alpha * \overrightarrow{mv}$$

$$\overrightarrow{mv\_r} = (1 - \alpha) * \overrightarrow{mv} \tag{3.6}$$

α is defined as:

$$\alpha = tpos\ \ TEMP\_RES\_SHIFT tpos \in int(0...32), TEMP\_RES\_SHIFT = 5 \tag{3.7}$$

The $\overrightarrow{mv}$ vector is the equivalent of the displacement vector $\vec{D}$ as mentioned in section 2.4.1. The $\overrightarrow{mv\_l}$ and $\overrightarrow{mv\_r}$ vectors are the equivalent of the $\alpha\vec{D}$ and (1 - α)$\vec{D}$ vector described in that same section.
These vectors are used to obtain left and right index variables. The position of the current block is scaled up by a factor of *norm_vec_sub_x* in the x direction and a factor of *norm_vec_sub_y* in the y direction, thereby creating new positions between the pixels on the integer grid (needed to obtain sub pixel accuracy). Then the $\vec{mv\_l}$ vector is added to these up-scaled values to obtain the left variables (*x_l*, *y_l*), the $\vec{mv\_r}$ vector is subtracted from the up-scaled values to obtain the right variables (*x_r*, *y_r*). In formula form:

$$\begin{aligned} x\_l &= blk.bx * block\_width * norm\_vec\_sub\_x + mv\_l.vx \\ y\_l &= blk.by * block\_height * norm\_vec\_sub\_y + mv\_l.vy \\ x\_r &= blk.bx * block\_width * norm\_vec\_sub\_x - mv\_r.vx \\ y\_r &= blk.by * block\_height * norm\_vec\_sub\_y - mv\_r.vy \end{aligned} \tag{3.8}$$

These left and right values are then down-scaled (in *x_l_int*, *y_l_int*, *x_r_int*, *y_r_int*) to obtain integer pixel grid variables, which can be used in the addressing of the current and previous frame. The difference between the up-scaled values and re-up-scaled integer grid values are stored (in *x_l_frac*, *y_l_frac*, *x_r_frac*, *y_r_frac*) and are used in the bilinear interpolation. In section 2.4.1 the cost function $\epsilon(\vec{X}, O, n)$ calculation was already explained, the actual implementation is as follows:

$$SAD = \sum_{\forall \vec{x} \in B(X)} |sample\_l - sample\_r| \tag{3.9}$$

where the $sample\_l$ indicates the pixels obtained from the previous frame and the $sample\_r$ indicates the pixels taken from the current frame. The sum is calculated over all the pixels in a block. $sample\_l$ is defined as:

$$\begin{aligned} sample\_l \;=\; & (norm\_vec\_sub\_y - y\_l\_frac) * \\ & \Big((norm\_vec\_sub\_x - x\_l\_frac)P_1 + x\_l\_frac * P_2\Big) + \\ & y\_l\_frac * \\ & \Big((norm\_vec\_sub\_x - x\_l\_frac)P_3 + x\_l\_frac * P_4\Big) \end{aligned} \tag{3.10}$$

Where P_1 - P_4 indicate the four pixels taken of the frame (see figure 3.2). The *sample_l* variable is equal to the value of $F_s(\vec{x} + \alpha\vec{D})$ as described in section 2.4.1 and is determined as follows:

$$F_s(\vec{x} + \alpha\vec{D}) = (4 - f_y)\Big((4 - f_x)P_1 + f_x P_2\Big) + f_y\Big((4 - f_x)P_3 + f_x P_4\Big) \tag{3.11}$$

with f_y and f_x fractional parts, defined as the distance to the nearest pixel on the integer pixel data grid and P_1, P_2, P_3 and P_4 the nearest pixels on the integer pixel grid. This is the same formula as written above for the *sample_l* variable, only easier to read. The $F_s(\vec{x}$+(1 - $\alpha)\vec{D})$ or *sample_r* value is obtained similarly. Figure 3.2 shows an example of bi-linear interpolation. The SAD is stored in the *abs_dif_array*.

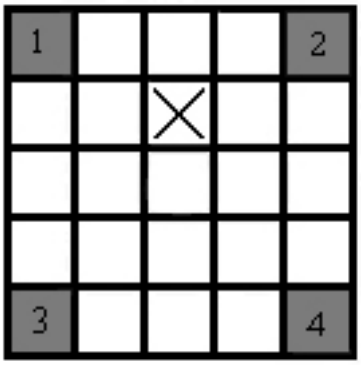

Figure 3.2: *Bilinear interpolation. In the example the interpolated value becomes: (4 - 1)((4 - 2)P1 + 2P2) + 1((4 - 2)P3 + 2P4) = 6P1 + 6P2 + 2P3 + 2P4*

Determining the best candidate and update of segment mask and bias field
Once the SAD of all candidates is calculated, the best candidate is determined. The object to which this best candidate belongs to is stored in a temporary variable. The tested block is then assigned to this object in the segmentation mask. The bias field of the currently processed field / frame is updated as well. A block counter is incremented and the SAD of the best candidate is added to a variable. These variables are used in the calculation of the SAD block-average in the post processing. The SAD of the candidates (which was stored in *abs_dif_array)* is copied to an array named *abs_dif_layer* at the object entry to which they belong.

Preprocessing for feature point selection
An array named *high_sad* consisting of two block lines is then filled for each block. This array will later be used to determine suitable feature points. The filling of an *high_sad* array block line entry (*high_sad[0][x]*) is an iterative process. The entry contains an object number, which is initially zero. Now if the calculated SAD of the block as stored in *abs_dif_layer* is deemed too high to be considered a reasonable match for the currently evaluated object, the *high_sad* array entry is incremented to indicate that previous objects apparently did not match. The filling of the *high_sad* array marks the end of the first loop. The first loop in pseudo code:

```
for( all block lines) { /* first block line loop (1/2) */
  for (all blocks on line ) {
    Set_Prediction_Candidates();
    for (all predictions candidates) {
      Calculate_SAD();
```



```
    }

    Determine_best_candidate();
    Update_Segment_Mask_and_Bias_field();
    Fill_high_SAD_Array();
  }
}
```

3.4.3 The main loop - second inner loop

variable name	small description
$high\_sad[]$	Array which records which objects did not match.
sum_high_sad	Variable used to determine if a block is suited to be a feature point.
cl_cur	Variable used to determine if a feature point needs to be accentuated.
$index\_fill\_est[]$	Index variables per object.
$increment\_fill\_est[]$	Address increment variables per object.
$stop\_fill\_est[]$	Termination of feature point assignment procedure per object.
$cnt\_fill\_est[]$.	Holds number of feature points assigned per object.
MAX_BLOCKS	maximum number of feature points per object.
MIN_BLOCKS	minimum number of feature points per object.

Table 3.3: Variables used in the second loop.

The second loop is started directly after the first loop has terminated. The variables used in this loop are given in table 3.3. The *high_sad* array is used to determine two variables: *sum_high_sad* and *cl_cur*. The first is used to determine the feature points for each object, the second is used to determine which of those feature points should be accentuated.

Obtaining an evenly distributed set of feature points
To ensure that the feature points are distributed evenly over a frame four arrays named *index_fill_est*, *increment_fill_est*, *stop_fill_est* and *cnt_fill_est* are used. The set of feature points is limited to *MAX_BLOCKS* for each object. Since there are potentially more blocks that can serve as feature points, some of the previously found feature points need to discarded to create room for newly found feature points. Therefor the *ep* (feature point) array is filled in multiple iterations. In the first iteration, an entry, as indicated by the *index_fill_est* entry of the current object, of the *ep* array is filled if a candidate feature point is encountered. This continues until the *index_fill_est* equals *MAX_BLOCKS* (so a *MAX_BLOCKS* amount of feature point candidates are entered in the *ep* array). At this time the *increment_fill_est* entry for the current object is incremented by one. In the second iteration, the *ep* array is filled again with the exception that of every two entries int the array only one is updated with a newly found candidate feature point (i.e. the *index_fill_est* entry increments with 2 each time). In this way only *MAX_BLOCKS* / 2 are overwritten of the *MAX_BLOCKS* candidates found in the first iteration so that a distributed set of feature points still remains. Once those *MAX_BLOCKS* / 2 entries of the second iteration

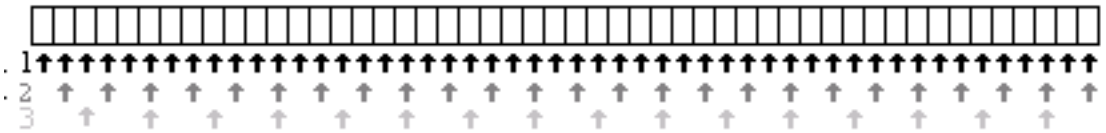

Figure 3.3: *Filling the feature point array procedure. In the first pass each entry of the feature array is updated (as indicated by the black arrows, in the second only one of every two entries is updated as indicated by the dark grey arrows, in the third pass only every third entry(indicated by the light grey arrows), and so on and so forth. This method ensures an spatially even distributed set of feature points.*

are filled, the *increment_fill_est* entry is increment again and a new iteration is started in which *MAX_BLOCKS* / 3 positions are filled. This procedure continues until all blocks are processed or the *increment_fill_est* entry is equal to *MAX_BLOCKS* / 2. If the latter occurs, the *stop_fill_est* entry is set to 1 and the filling of the feature point array stops for the current object. In figure 3.3 the filling procedure is shown for 3 iterations and a *MAX_BLOCKS* of 50.

Adjusting thresholds

Every time a new candidate feature point was written in the *ep* array the entry for the current object of the *cnt_fill_est* array was incremented. This value can be used to determine whether or not finding feature points was easy or difficult. If the *cnt_fill_est* entry was below *MIN_BLOCKS* then finding feature points was too difficult. In this case the confines of the qualifying terms should be expanded. These qualifying terms are expressed in the thresholds which were obtained in the pre-processing. If the *cnt_fill_est* entry was greater than *MAX_BLOCKS*, finding feature points was too easy and the confines of the search need to be tightened.

At the end of the second loop, the pointers to the block lines of the *high_sad* array are swapped so the information of the previously processed block line remains in the second block line. The pseudo code description of the second loop is given below.

```
for(macroblock line) {
  for ( all objects) {
    for (macroblock) {
      /* count blocks with high sad in previous layers */
      if (high_sad[macroblock] == (object_no))
        set_macroblock_as_feature_point();
      /* count blocks of current layer */
      if (segment_mask[(macroblock)] == object_no)
        accentuate_feature_point();
      make_feature_point_distribution_even();
    }
  }
Swap_pointers();
```


3.4.4 Post-processing

At this time the main loop of the segmentation process ends and some post processing is done. In this post processing the update of the set of feature points for each object is done. The update procedure also accomplishes the updates of the thresholds (if required). Then the edges of the frame are processed. This edge processing simply consists of copying the segment mask and bias field values of the nearest blocks, which were processed. Finally the SAD block-average is calculated and stored and the phase is flipped. The pseudo code is given here:

/* Final processing at end of field. */
Update_Feature_Points ();
Process_Edges ();
Update_Stats();
Flip_Phase();

In order to make the concept of the motion segmentation easier to grasp, a flow graph is drawn in figure 3.4.

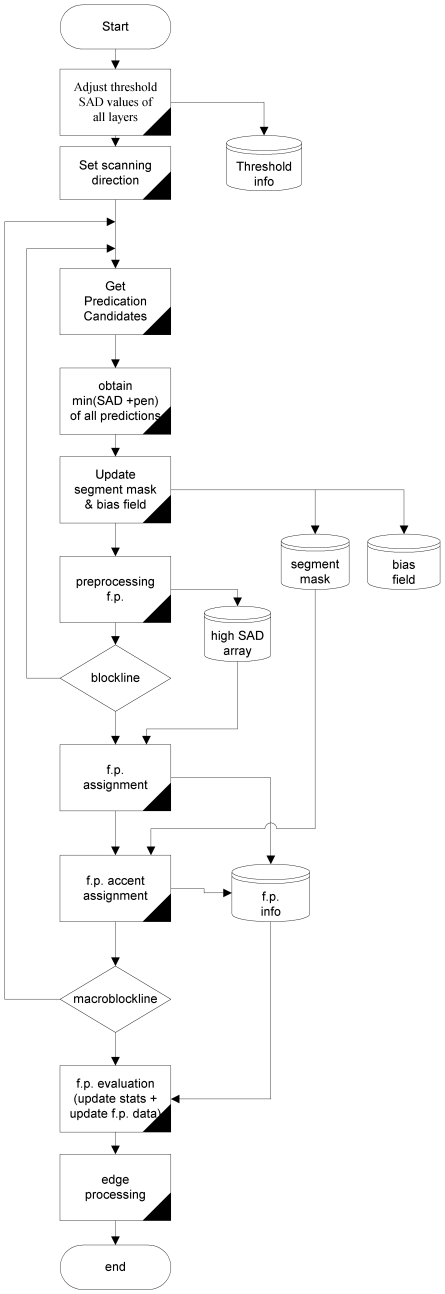

Figure 3.4: *A flow graph of the motion segmentation part of the OME algorithm implementation.*

3.5 Displaying information in the output file

3.5.1 The segmentation and vector fields

In order to verify if the motion segmentation matches the actual objects in the frame, a visual aid is very helpful. The easiest way to implement this aid is to replace the color information of the original file with the vector information. This way the segmentation is made clear through the different colors for the objects. So this is exactly what is implemented. Each block has its own parameter model and bias field assigned to it. The display procedure obtains this information for every block in the frame and then assigns to each pixel within the block the colors matching the motion vectors. The x coordinate information is coded in the u information of a yuv signal, the y coordinate information is coded in the v information. The luminance (y) is left intact.

3.5.2 Visualization of the location of the feature points.

To verify whether the feature points assigned per layer matched the location of the object, a loop was created to display the location of said feature points. First every object was assigned its own color of feature points. Then the locations of the feature points, which are stored in the *ep* array, were retrieved. At what remains then is to overwrite the yuv values to the color of the object.

Chapter 4

OME vs 3DRS

4.1 Introduction

In the previous section an implementation of the OME algorithm was discussed. In this section a comparison will be made between that implementation and a 3DRS implementation made by J.Theelen (see [7]). Section 4.2 will discuss the criterium used for this comparison, and section 4.3 will show the results obtained from the comparison.

4.2 Criterium for evaluating motion vector fields

The criterium used for the comparison is the *modified mean square error (MMSE)* method. The modification is defined is such a way that the resulting figure to some extent reflects the quality of the vector – true motion relation. The quintessence of the modification is that the validity of the vectors is extrapolated outside the temporal interval on which they are calculated. The extrapolation, because of object inertia, is expected to be more legitimate if the vectors represent true velocity than if they only indicate a good match between blocks of pixels.
In both the implementation of the OME and 3DRS the best matching candidate displacement vector $\vec{D}$ is assigned to all pixel positions $\vec{x}$ in a block B($\vec{X}$).
With the vectors so defined, output sequences are created by interpolating each output field as the motion compensated average from two successive input frames and using the displacement vectors created by either the OME or 3DRS algorithms. Interpolated fields are thus found as:

$$F_{mc}(\vec{x},n) = \frac{1}{2}\Big(F(\vec{x}-\vec{D}(\vec{x},n),n-1) + F(\vec{x}+\vec{D}(\vec{x},n),n+1)\Big) \qquad (4.1)$$

To calculate the MMSE, the squared pixel differences between the interpolated output and the original input frame are summed over a frame excluding a boundary area and normalized with respect to the number of pixels in this measurement window MW. So

the MMSE performance criterium can be written as:

$$MMSE = \frac{1}{\vec{x} \in MW} \sum_{\vec{x} \in MW} (F_s(\vec{x}, n) - F_{mc}(\vec{x}, n))^2 \tag{4.2}$$

where $F_s(\vec{x}, n)$ is frame n of the input sequence and $F_{mc}(\vec{x}, n)$ is frame n, estimated from its neighboring frame $n-1$ and $n+1$ according to eq. 4.1.

4.3 OME vs 3DRS - the results of the comparison

The comparison was done using 6 input sequences of which snapshots are shown in figure 4.1. These sequences were selected to provide critical test material for the OME and 3DRS algorithms and include several categories of difficulties. In order to create a fair comparison between the two algorithms, both were subjected to the same limitations:

- limited to integer pixel accurate motion vectors.
- same spatio-temporal location of the predictions candidates (see figure 2.2).
- same set of penalty values p_s, p_t and p_r per block.

The results of the comparison are plotted in figure 4.2. From the plots the conclusion can be drawn that the implementation of the 3DRS algorithm performs up to a factor of three better than the OME algorithm. This difference can be explained by the immature implementation of the OME algorithm. Since the motion segmentation part of the OME algorithm and the 3DRS algorithm are very similar, the results of both algorithms should be closer to each other (the results should comparable measured in percents, not in factors).

Figure 4.1: *Snapshots of the sequences used for comparison.*

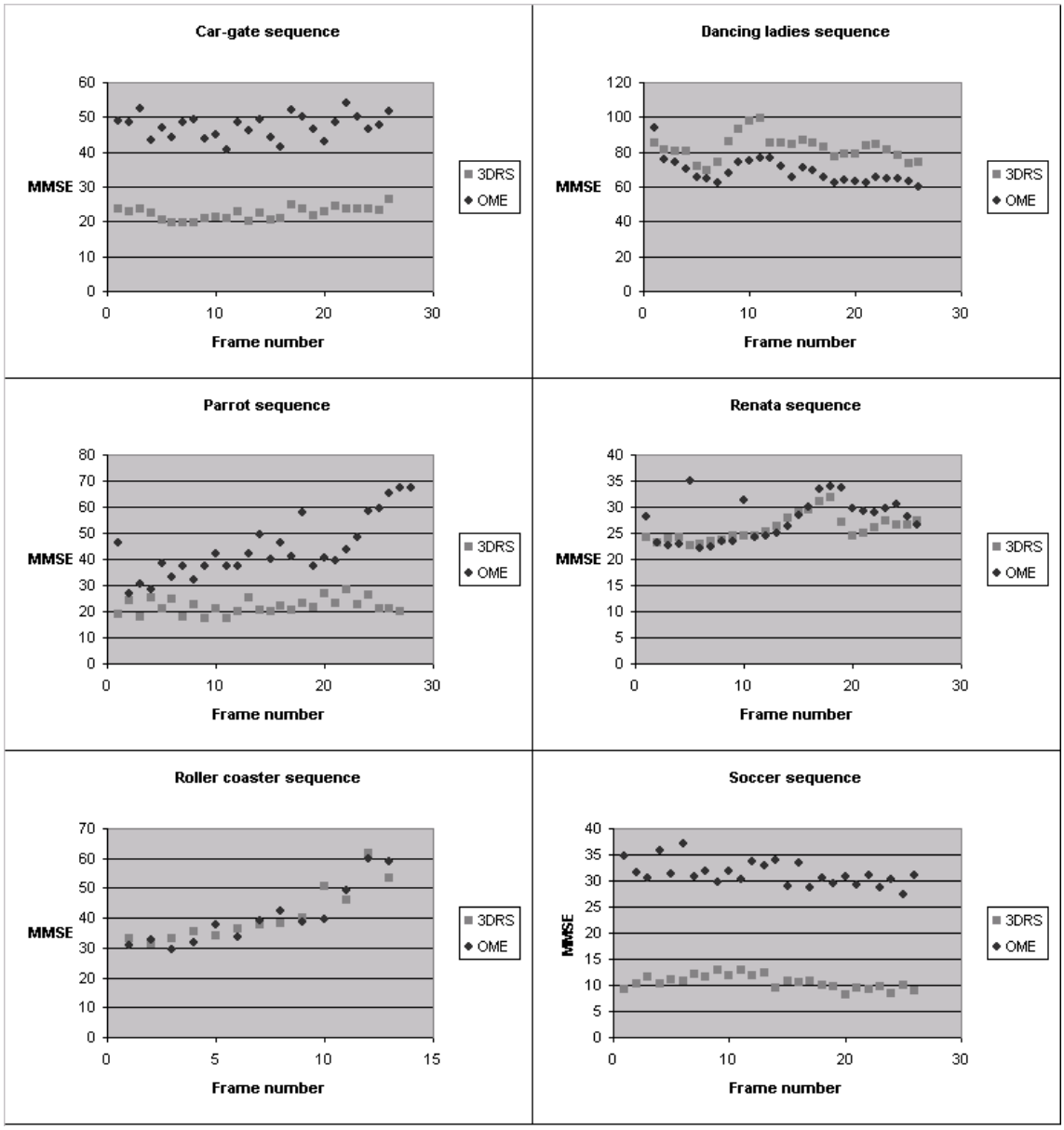

Figure 4.2: OME vs 3DRS plots, the plots show the MMSE values of the OME and 3DRS algorithms for 6 sequences

Chapter 5

Conclusions

The goals for this report were:

- To understand the object based motion estimator in sufficient detail.
- A report which documents the OME in sufficient detail.
- To implement a demo tool of this algorithm in order to visualize the effects of the algorithm.
- Obtain some figures on the quality of the vector fields generated by the algorithm and compare them to figures obtained from a 3DRS algorithm.

All goals were met; this report is the document which is mentioned in item number two. A working demo tool has been created, which works quite nicely at pixel accuracy, and some comparative figures on motion vector quality were found.

However work remains to be done. At the moment the motion vectors used in the created program are pixel accurate. The implementation of the 3DRS algorithm in hardware to which the implementation of the OME algorithm, as described in this report, will be competing has a quarter pixel accuracy. Therefore the created program needs to be extended to incorporate this accuracy as well. Further, the several thresholds used in the program are optimized for a version of the algorithm which operates on sub-sampled video data, and not optimized for this implementation which does not use sub-sampling. These various thresholds need to be optimized for the current implementation for they could result in a better segmentation.

The figures on the quality of the vector fields of the OME and 3DRS algorithms show that the 3DRS performs better than OME. A better implementation of OME is required in order to create a fair comparison. The questions which remain are:

- When a better implementation of OME is available, will the OME algorithm compare favorably to 3DRS?

- Will the vector field obtained by use of the OME algorithm improve significantly if the algorithms accuracy is improved to quarter pixel accuracy?

The questions will be answered in the successor of this report.

Overall the conclusion can be made that this report adds an entry level document on the OME algorithm, which was not previously there. Further the created program can serve as a demo tool which illustrates the effect of the OME algorithm if sub sampling is not used. The program will be developed further as already mentioned above.

Acknowledgements

The author would like to thank Rimmert Wittebrood and Gerard de Haan for their contribution to the implementation of the demo tool, as well as their insight into the OME algorithm. Furthermore, the author would like to thank Gerard de Haan for his permission to use parts of his book ([1]) in this text.

Bibliography

[1] de Haan, G., *Video processing for multimedia systems*, 1^{st} edition 2000, pp 187-197, 90-9014015-8

[2] de Haan, G., *Video processing for multimedia systems*, 1^{st} edition 2000, pp 198-200, 90-9014015-8

[3] Wittebrood, R.B., de Haan, G., *Real-time recursive motion segmentation of video data on a programmable device*, IEEE Trans. on Consumer Electronics, Aug. 2001, pp559–567

[4] Wittebrood, R.B., de Haan, G., *Second generation video format conversion software for a digital signal processor*, IEEE Trans. on Consumer Electronics, Vol. 46, No. 3, pp857–865, Aug. 2000

[5] Schutten,R.J., Pelagotti, A., de Haan, G., *Layered motion estimation*, Philips J. Res 1998, pp 253–267

[6] Tekalp, A.M., *Digital video processing*, Prentice Hall Signal Processing Series, 1995, pp200–203, ISBN 0-13-190075-7

[7] Theelen,J.,*A motion estimation demonstration tool*, Eindhoven University of Technology, department of electrotechnical engineering, internship report ICS/ES 798, 2002