Motion-based segmentation of image sequences

Gunnar Farnebäck

May 6, 1996

Abstract

This Master's Thesis addresses the problem of segmenting an image sequence with respect to the motion in the sequence. As a basis for the motion estimation, 3D orientation tensors are used. The goal of the segmentation is to partition the images into regions, which are characterized by having a similar motion, where the motion model is affine with respect to the image coordinates. A method to estimate the parameters of the motion model from the orientation tensors in a region is presented. This method can also be generalized to a large class of motion models.

Two segmentation algorithms are presented together with a postprocessing algorithm. All these algorithms are based on the competitive algorithm, a general method for distributing points between a number of regions, without relying on arbitrary threshold values. The first segmentation algorithm segments each image independently, while the second algorithm recursively takes advantage of the previous segmentation. The postprocessing algorithm stabilizes the segmentations of a whole sequence by imposing continuity constraints.

The algorithms have been implemented and the results of applying them to a test sequence are presented. Interesting properties of the algorithms are that they are insensitive to the aperture problem and that they do not require a dense velocity field.

It is finally discussed how the algorithms can be developed and improved. It is straightforward to extend the algorithms to base the segmentations on alternative or additional features, under not too restrictive conditions on the features.

Contents

List of Figures

List of Tables

1 Introduction

Segmentation is an important step in many image processing applications. The idea is to make a partition of an image into a set of regions, often corresponding to objects in the image, based on some feature, such as motion or texture. In general the problem is very hard. Whether a segmentation is "correct" or not may be impossible to determine. The feature that the segmentation is based on may vary continuously between two different regions. This makes it hard to tell where to draw the line between the regions. It may even be possible that they are in fact so similar that they should be only one region.

In this Master's Thesis the segmentations are based solely on estimations of the motion in image sequences. The segmentations are however not limited to regions where the pixels have a common translation. The motion is modeled to be affine with respect to the image coordinates. This allows for objects that are e.g. rotating to be identified as single regions.

The basis for the motion estimation in this thesis is 3D orientation tensors, discussed in section 2.1. Using these it turns out to be rather easy to estimate the best affine motion model, given a region. It also turns out to be only moderately hard to estimate what part of an image is consistent with a given motion model. The problem is that at the beginning, neither regions nor motion models are available. This is another reason why the segmentation problem is hard.

An initial inspiration for this work was the papers [2, 3] by Adelson and Wang. They too discuss motion-based segmentation with affine motion models. A difference is that their motion estimation is based on optical flow. The first objective was to adapt their algorithm to use orientation tensors instead. This attempt was however never completed. In part because the description of their algorithm lacked details, in particular threshold values and other rather arbitrary constants, that were hard to reconstruct or find empirically.

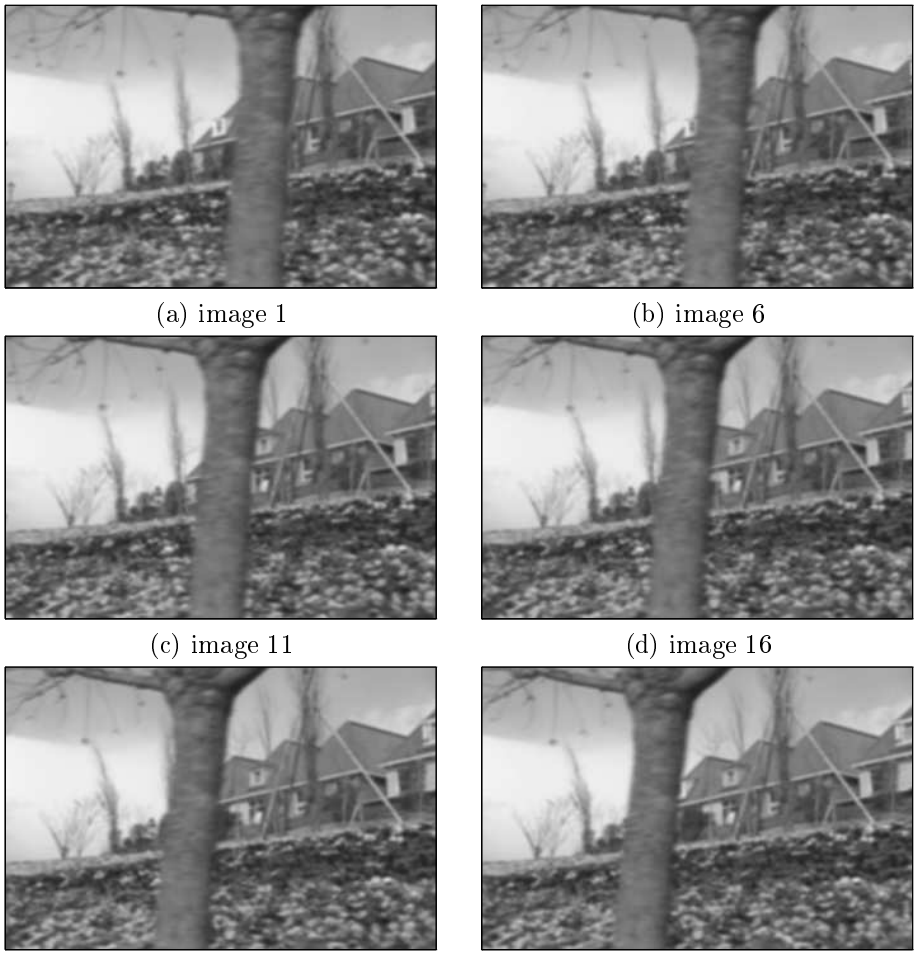
As a reaction to the problem of finding threshold values, the competitive algorithm, described in section 3.3, was developed. With that as a basis, two segmentation algorithms and a postprocessing algorithm were designed. These algorithms have in fact nothing in common with the algorithm of Adelson and Wang, except the use of affine motion models. The algorithms are described in section 4.

For illustrations of the algorithms and for testing them, the flower garden sequence, shown in figure 1, is used. This is in fact another connection to the papers by Adelson and Wang, since they use it too.

2 Motion model estimation

2.1 Orientation tensors

The velocity estimation is based on 3D orientation tensors, which are thoroughly described in [1]. In short, the orientation tensor $\mathbf{T}$ describes the distribution of the local signal energy in different directions, in a neighborhood of a point. $\mathbf{T}$ can be regarded as a positive semidefinite quadratic form, such that for a unit directional vector $\hat{\mathbf{e}}$, the signal energy in the direction $\hat{\mathbf{e}}$ is $\hat{\mathbf{e}}^T\mathbf{T}\hat{\mathbf{e}}$. This means that for a one-dimensional signal, i.e. a signal varying in only one direction, the



(a) image 1 (b) image 6

(c) image 11 (d) image 16

(e) image 21 (f) image 26

Figure 1: Selected images from the flower garden sequence.

energy is concentrated to the direction of the signal. The quadratic form can in general be decomposed by the spectral theorem as

$$\mathbf{T} = \lambda_1 \hat{\mathbf{e}}_1 \hat{\mathbf{e}}_1^T + \lambda_2 \hat{\mathbf{e}}_2 \hat{\mathbf{e}}_2^T + \lambda_3 \hat{\mathbf{e}}_3 \hat{\mathbf{e}}_3^T, \tag{1}$$

where λ_i are the eigenvalues, $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$, and $\hat{\mathbf{e}}_i$ is the corresponding set of orthogonal eigenvectors. For a one-dimensional signal, the orientation tensor has the form

$$\mathbf{T} = \lambda_1 \hat{\mathbf{e}}_1 \hat{\mathbf{e}}_1^T, \tag{2}$$

where $\hat{\mathbf{e}}_1$ is the direction of the signal.

An image sequence can be considered as a volume of stacked images. The significance of the 3D orientation tensor to the velocity in the image sequence follows from the following observations:

- A moving line locally gives rise to a one-dimensional signal in the volume. Then $\lambda_1 > 0$, $\lambda_2 = \lambda_3 = 0$ and $\hat{\mathbf{e}}_1$ is orthogonal to the 3D plane that is generated by the moving line.
- A moving point locally gives rise to a two-dimensional signal. Then $\lambda_1 = \lambda_2 > 0$, $\lambda_3 = 0$ and $\hat{\mathbf{e}}_3$ is aligned with the 3D line generated by the moving point.

In the moving line case the true velocity can not be estimated due to the aperture problem; only the velocity component normal to the line can be obtained. The velocity in the two cases is computed from the orientation tensor according to the following formulas:

$$\left\{ \begin{array}{lcl} \mathbf{v}_{normal} & = & -x_3(x_1\hat{\boldsymbol{\xi}}_1 + x_2\hat{\boldsymbol{\xi}}_2)/(x_1^2 + x_2^2) \\ x_1 & = & \hat{\mathbf{e}}_1 \cdot \hat{\boldsymbol{\xi}}_1 \\ x_2 & = & \hat{\mathbf{e}}_1 \cdot \hat{\boldsymbol{\xi}}_2 \\ x_3 & = & \hat{\mathbf{e}}_1 \cdot \hat{\mathbf{t}} \end{array} \right. \quad \text{moving line case,} \tag{3}$$

$$\left\{ \begin{array}{lcl} \mathbf{v} & = & (x_1\hat{\boldsymbol{\xi}}_1 + x_2\hat{\boldsymbol{\xi}}_2)/x_3 \\ x_1 & = & \hat{\mathbf{e}}_3 \cdot \hat{\boldsymbol{\xi}}_1 \\ x_2 & = & \hat{\mathbf{e}}_3 \cdot \hat{\boldsymbol{\xi}}_2 \\ x_3 & = & \hat{\mathbf{e}}_3 \cdot \hat{\mathbf{t}} \end{array} \right. \quad \text{moving point case,} \tag{4}$$

where $\hat{\boldsymbol{\xi}}_1$ and $\hat{\boldsymbol{\xi}}_2$ are orthogonal unit vectors defining the image plane and $\hat{\mathbf{t}}$ is a unit vector in the time direction.

In a real image the orientation tensors usually fall into neither of these extreme cases. Then one have to decide which case to use. The line case gives less information and the point case is less stable. Fortunately this problem can be avoided by not computing the velocity field but instead using the orientation tensors directly to estimate a motion model.

The orientation tensors are typically computed as linear combinations of a number of basis tensors, where the coefficients of the linear combinations are the magnitudes of the outputs from certain quadrature filters. How the basis tensors and the quadrature filters may be chosen is explained in detail in [1].

2.2 Motion model

The motion model for each object is affine, i.e. the motion at each image point (x, y) of the object can be described as follows:

$$v_x(x, y) = ax + by + c, \tag{5}$$

$$v_y(x, y) = dx + ey + f, \tag{6}$$

where v_x and v_y are the x and y components of the velocity and a through f are the coefficients of the model. This model makes it possible to describe more than a common translation. Among the more complex transformations that can be described are the following:

- 3D translation of plane surfaces under perspective projection.
- 3D translation and rotation of plane surfaces under orthogonal projection.
- 2D translation, rotation, zoom and shear.

Often these assumptions are not perfectly satisfied. Then we have to settle for the best affine approximation. It should be noted that a perspective projection can be approximated with an orthogonal projection provided that the depth of all objects is small compared to the viewing distance, and that a general motion can be linearized into a translation plus a rotation.

2.3 Fitting the motion model to the orientation tensors

The motion model parameters for an object are estimated by minimization over a region representing the object, of a distance function between the motion model and the orientation tensors. To derive this function we need to discuss the properties of the orientation tensor in section 2.1 a little further.

2.3.1 Distance between an orientation tensor and a velocity hypothesis

A 2D velocity (v_x, v_y), measured in pixels/frame, can be extended to a 3D directional vector $\mathbf{v}$ and a unit directional vector $\hat{\mathbf{v}}$, in the following way:

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix}, \qquad \hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}. \tag{7}$$

In the moving line case it is clear that $\mathbf{v}$ lies in the plane that is generated by the moving line. Since $\hat{\mathbf{e}}_1$ is orthogonal to this plane it follows that $\hat{\mathbf{e}}_1$ and $\mathbf{v}$ are orthogonal too. In the moving point case $\mathbf{v}$ is parallel to $\hat{\mathbf{e}}_3$ and thus ortogonal to $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$. For the moving line it can not be required that $\mathbf{v}$ and $\hat{\mathbf{e}}_2$ be orthogonal but on the other hand is λ_2 (ideally) zero in that case.

This leads to a preliminary distance function between the velocity vector $\mathbf{v}$ and the tensor $\mathbf{T}$:

$$d(\mathbf{v}, \mathbf{T}) = \lambda_1 |\hat{\mathbf{v}} \cdot \hat{\mathbf{e}}_1| + \lambda_2 |\hat{\mathbf{v}} \cdot \hat{\mathbf{e}}_2|. \tag{8}$$

This distance function has two significant drawbacks that need to be remedied. First, it requires an eigenvalue decomposition. Second, it is not easy to minimize over a region.

The cure to the first problem is to note that

$$\begin{aligned}(9)\qquad \mathbf{T} &= \lambda_1\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \lambda_2\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \lambda_3\hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^T \\ &= (\lambda_1-\lambda_3)\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + (\lambda_2-\lambda_3)\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \lambda_3(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^T) \\ &= (\lambda_1-\lambda_3)\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + (\lambda_2-\lambda_3)\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T + \lambda_3\mathbf{I},\end{aligned}$$

hence

$$(10)\qquad \hat{\mathbf{v}}^T(\mathbf{T}-\lambda_3\mathbf{I})\hat{\mathbf{v}} = (\lambda_1-\lambda_3)(\hat{\mathbf{v}}\cdot\hat{\mathbf{e}}_1)^2 + (\lambda_2-\lambda_3)(\hat{\mathbf{v}}\cdot\hat{\mathbf{e}}_2)^2.$$

The subtraction of $\lambda_3\mathbf{I}$ from $\mathbf{T}$ means removing the isotropic part from the orientation tensor. This is a quite sensible thing to do since the isotropic energy can give no information about the orientation. It is also desirable to make the distance function invariant to the total energy in the neighborhood. This can be accomplished by dividing the tensor with its trace, *before removing the isotropic part*, normalizing the sum of the eigenvalues to one. These operations yield a modified orientation tensor $\tilde{\mathbf{T}}$ with the same eigenvectors as $\mathbf{T}$ but new eigenvalues $\tilde{\lambda}_i$:

$$(11)\qquad \tilde{\mathbf{T}} = (\mathbf{T}-\lambda_3\mathbf{I})/\operatorname{tr}(\mathbf{T}) = \tilde{\lambda}_1\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \tilde{\lambda}_2\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T,$$

where

$$(12)\qquad \tilde{\lambda}_1 = \frac{\lambda_1-\lambda_3}{\lambda_1+\lambda_2+\lambda_3}, \qquad \tilde{\lambda}_2 = \frac{\lambda_2-\lambda_3}{\lambda_1+\lambda_2+\lambda_3}.$$

Finally we have three useful distance functions:

$$(13)\qquad d_1(\mathbf{v},\mathbf{T}) = \mathbf{v}^T\tilde{\mathbf{T}}\mathbf{v} = \tilde{\lambda}_1(\mathbf{v}\cdot\hat{\mathbf{e}}_1)^2 + \tilde{\lambda}_2(\mathbf{v}\cdot\hat{\mathbf{e}}_2)^2,$$

$$(14)\qquad d_2(\mathbf{v},\mathbf{T}) = \hat{\mathbf{v}}^T\tilde{\mathbf{T}}\hat{\mathbf{v}} = \frac{d_1(\mathbf{v},\mathbf{T})}{|\mathbf{v}|^2},$$

$$(15)\qquad d_3(\mathbf{v},\mathbf{T}) = \frac{\hat{\mathbf{v}}^T\tilde{\mathbf{T}}\hat{\mathbf{v}}}{\tilde{\lambda}_1+\tilde{\lambda}_2} = \frac{d_1(\mathbf{v},\mathbf{T})}{|\mathbf{v}|^2\operatorname{tr}(\tilde{\mathbf{T}})}.$$

The strength of d_1 is that it is a quadratic form in $(v_x, v_y, 1)$ and is therefore easy to minimize over a region. A possible weakness is that it is more sensitive to large velocities than to small. To avoid that, d_2 has been normalized with respect to the norm of $\mathbf{v}$. To see the point of d_3 it should first be noted that by the construction of $\tilde{\mathbf{T}}$, pixels with mainly isotropic original tensors yield small d_1 distances. In this way these pixels make small impact on the minimization over a region, which is sound because the certainty of the 3D orientation is small at such points. Later it also turns out to be useful comparing how well two tensors fit to a given motion hypothesis. In that case it is not desirable that a high degree of isotropy is favored. Hence the normalization with respect to the eigenvalues of $\tilde{\mathbf{T}}$ in d_3.

Note that the eigenvector decomposition of the tensors is no longer necessary for any of these distance measures. We only need to compute the least eigenvalue, which is a faster and more stable operation.

For conveniency reasons the tildes will henceforth be dropped, effectively meaning that the original tensors are assumed to have been replaced by preprocessed tensors from the beginning. This preprocessing step also includes an initial averaging of the tensors over a 5×5 neighborhood, intended to stabilize the tensor estimates.

2.3.2 Distance between an orientation tensor and an affine motion model

The distance between an affine motion model, with parameters (a, b, c, d, e, f), and an orientation tensor $\mathbf{T}$ can be derived from the distance between a motion hypothesis and an orientation tensor in a straightforward manner. Combining (5), (6), and (7), we have

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{S}} \underbrace{\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ 1 \end{pmatrix}}_{\mathbf{p}}, \tag{16}$$

hence

$$d_1(\mathbf{v}, \mathbf{T}) = \mathbf{v}^T \mathbf{T} \mathbf{v} = \mathbf{p}^T \mathbf{S}^T \mathbf{T} \mathbf{S} \mathbf{p} = \mathbf{p}^T \mathbf{Q} \mathbf{p}, \tag{17}$$

where $\mathbf{Q} = \mathbf{S}^T \mathbf{T} \mathbf{S}$ is a positive semidefinite quadratic form. With some abuse of notation we have the three new distance functions

$$d_1(\mathbf{p}, \mathbf{T}) = \mathbf{p}^T \mathbf{Q} \mathbf{p}, \tag{18}$$

$$d_2(\mathbf{p}, \mathbf{T}) = \frac{\mathbf{p}^T \mathbf{Q} \mathbf{p}}{\mathbf{p}^T \mathbf{S}^T \mathbf{S} \mathbf{p}}, \tag{19}$$

$$d_3(\mathbf{p}, \mathbf{T}) = \frac{d_2(\mathbf{p}, \mathbf{T})}{\lambda_1 + \lambda_2}. \tag{20}$$

It should be noted that the two matrices $\mathbf{S}$ and $\mathbf{Q}$ are position dependent. Consequently the distance functions have hidden position dependencies.

2.3.3 Determining the parameters for a region

For a given region the motion model parameters are determined by minimization of the sum of the distances from the parameters to the orientation tensors in the region.

$$d_{tot}(\mathbf{p}) = \sum_i d_1(\mathbf{p}, \mathbf{T}_i) = \mathbf{p}^T \left(\sum_i \mathbf{Q}_i \right) \mathbf{p} = \mathbf{p}^T \mathbf{Q}_{tot} \mathbf{p}, \tag{21}$$

where the sum is taken over the pixels in the region. Now the problem is to find the vector $\mathbf{p}$ that minimizes the quadratic form $\mathbf{p}^T \mathbf{Q}_{tot} \mathbf{p}$ with the restriction that the last element of $\mathbf{p}$ has to be 1. Make the partitions

$$\mathbf{p} = \begin{pmatrix} \bar{\mathbf{p}} \\ 1 \end{pmatrix}, \qquad \mathbf{Q}_{tot} = \begin{pmatrix} \bar{\mathbf{Q}} & \mathbf{q} \\ \mathbf{q}^T & a \end{pmatrix}, \tag{22}$$

where $\bar{\mathbf{p}} = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}$, $\bar{\mathbf{Q}}$ is a symmetric matrix, $\mathbf{q}$ a vector, and a a scalar. Then

$$d_{tot}(\mathbf{p}) = \bar{\mathbf{p}}^T \bar{\mathbf{Q}} \bar{\mathbf{p}} + \bar{\mathbf{p}}^T \mathbf{q} + \mathbf{q}^T \bar{\mathbf{p}} + a \tag{23}$$

which is minimized by

$$\hat{\bar{\mathbf{p}}} = -\bar{\mathbf{Q}}^{-1} \mathbf{q} \tag{24}$$

giving the minimum value

$$d_{tot}(\hat{\mathbf{p}}) = a - \mathbf{q}^T \bar{\mathbf{Q}}^{-1} \mathbf{q} = a - \mathbf{q}^T \hat{\bar{\mathbf{p}}}. \tag{25}$$

It is not guaranteed that $\bar{\mathbf{Q}}$ is invertible. In the case it is not, the equation $\bar{\mathbf{Q}} \hat{\bar{\mathbf{p}}} = -\mathbf{q}$ still can be solved; but with infinitely many solutions. Among these the one with minimal norm, corresponding to minimal velocity, should be chosen. This solution can in general be written as $\hat{\bar{\mathbf{p}}} = -\bar{\mathbf{Q}}^{+} \mathbf{q}$, where $\bar{\mathbf{Q}}^{+}$ is the pseudo inverse to $\bar{\mathbf{Q}}$.

The inverse of $\bar{\mathbf{Q}}$ is of course never computed explicitly. Instead $\hat{\bar{\mathbf{p}}}$ is obtained by standard methods for solving equation systems.

Usually the invertibility is not a problem in practice. More important though, is that robust estimation of the parameters requires a fairly large region. Based on experiments, less than about 200 pixels is not recommended.

How this technique can be applied to more general motion models is discussed in section 6.4.

3 Region construction

3.1 Region definition

The goal of the segmentation is to partition the image into a set of disjoint regions. Here a region R is defined to be a nonempty, *connected* set of pixels. Associated to the region R is a cost function $\mathbf{C}_{\mathsf{R}}(\mathbf{x})$, which is defined on all pixels in the image and which may vary with the pixels currently included in the region. It is understood that regions belonging to the same segmentation may not overlap.

3.2 Region growing

Regions are extended by a growing process, adding one pixel at a time. To preserve connectivity the new pixel must be adjacent to the region, and to preserve disjointedness it may not already be assigned to some other region. The new pixel is also chosen as cheap as possible. The details are as follows:

Let the border $\Delta\mathsf{R}$ of region R be the set of nonassigned pixels in the image which are adjacent to some pixel in R. For each region R, the possible candidate, $N(\mathsf{R})$, to be added to the region is the cheapest pixel bordering to R, i.e.

$$N(\mathsf{R}) = \arg\min_{\mathbf{x} \in \Delta\mathsf{R}} \mathbf{C}_{\mathsf{R}}(\mathbf{x}). \tag{26}$$

The corresponding minimal cost for adding the candidate to the region is denoted $C_{min}(\mathsf{R})$. In the case of an empty border, $N(\mathsf{R})$ is undefined and $C_{min}(\mathsf{R})$ is infinite. The process of adding $N(\mathsf{R})$ to R is called growing.

3.3 The competitive algorithm

Assuming that a number of regions $\{\mathsf{R}_n\}$ in some way has been obtained, the basic algorithm for partitioning the rest of the image is as follows:

1. Find the region R_i for which the cost to add a new pixel is least, i.e. $i = \arg\min_n C_{min}(\mathsf{R}_n)$.
2. Add the cheapest pixel $N(\mathsf{R}_i)$ to R_i.
3. Repeat the first two points until no pixels remain.

Note that it does not matter what the actual values of the cost functions are. It is only relevant which of them is lowest. Hence the algorithm is called competitive.

To use this algorithm for segmentation, it must of course be specified how to obtain the initial regions. It may also need to be adjusted in different ways. However, it is the theoretical basis of all the algorithms that are presented in the following section.

4 Segmentation algorithms

Two algorithms for segmentation of an image sequence are presented in this section. The first one treats one image at a time, assuming no previous knowledge of how the image should be segmented. The second algorithm uses the segmentation of the previous image in the sequence to get a faster and more stable segmentation of the next image. Following those two algorithms a post-processing algorithm is also presented. This algorithm refines the segmentation of a whole sequence by imposing certain continuity constraints.

4.1 Algorithm 1

The only input to this algorithm is an orientation tensor field for the image. It contains one orientation tensor for each pixel. Since there is no way to know in advance how many regions the image should be segmented into, or where they are located, the competitive algorithm must be significantly extended. The key to solving this problem is the introduction of a concept of preliminary regions, called candidate regions, in section 4.1.2. The cost function used, for all kinds of regions, is the d_3 distance function. This function depends on the motion model parameters $\mathbf{p}$, which for each region are computed using the d_1 distance function, as described in section 2.3.3.

The description of this algorithm starts with the principal ideas. Efficiency considerations and implementation details are discussed towards the end.

4.1.1 Finding the seeds

The first problem that must be solved is where to start growing regions. The suitability of a given position can be tested by putting an arbitrarily chosen

region at the point, determine optimal parameters, and see how well these parameters fit. An obvious choice of measure for this is to take the value of $d_{tot}(\hat{\mathbf{p}})$ from equation (25). Experiments have shown, however, that a better solution is to take the maximal cost (with respect to d_3 for the optimal parameters) among the points in the region. This is the measure that is used and it is referred to as the *maximal cost* for the region.

In the spirit of the competitive algorithm this procedure should be applied at every point of the image and the regions yielding the best fits should be used in the segmentation. Unfortunately this approach would not really work. First, it is very likely that the best fits would be obtained for a number of points close to each other. The corresponding regions would then overlap and could not be used together. Second, we still have no idea on how many regions the image should be segmented into.

4.1.2 Candidate regions

The solution is to make a distinction between the regions used in the previous section for testing where to start growing regions and the regions that will be part of the final segmentation. The former are called candidate regions while the latter are called real regions. The real regions must obey all conditions of the region definition in section 3.1. In particular they have to be disjoint. The candidate regions differ from the definition of regions in that they may overlap with each other, but not with the real regions. They also have an associated starting point, which always must be included in the region. If a real region comes to occupy a candidate's starting point, then this candidate has to be removed.

4.1.3 Iterative step

The main idea of the algorithm is to alternately grow the real regions and raise the currently best candidate region to the status of a real region. At the beginning of the algorithm there are no real regions and each pixel (ideally, not in practice) is the starting point for a candidate region. Iteratively, until all pixels have been claimed, the following steps are executed:

1. Each candidate region is (re)built in some way so that they do not overlap with the real regions. At the same time, the maximal cost, as defined in section 4.1.1, is computed for each region.

2. From the candidate regions the one with the least maximal cost is chosen as the *aspirant* for inclusion among the real regions.

3. As in the competitive algorithm, the cheapest pixel that may be added to one of the real regions is found.

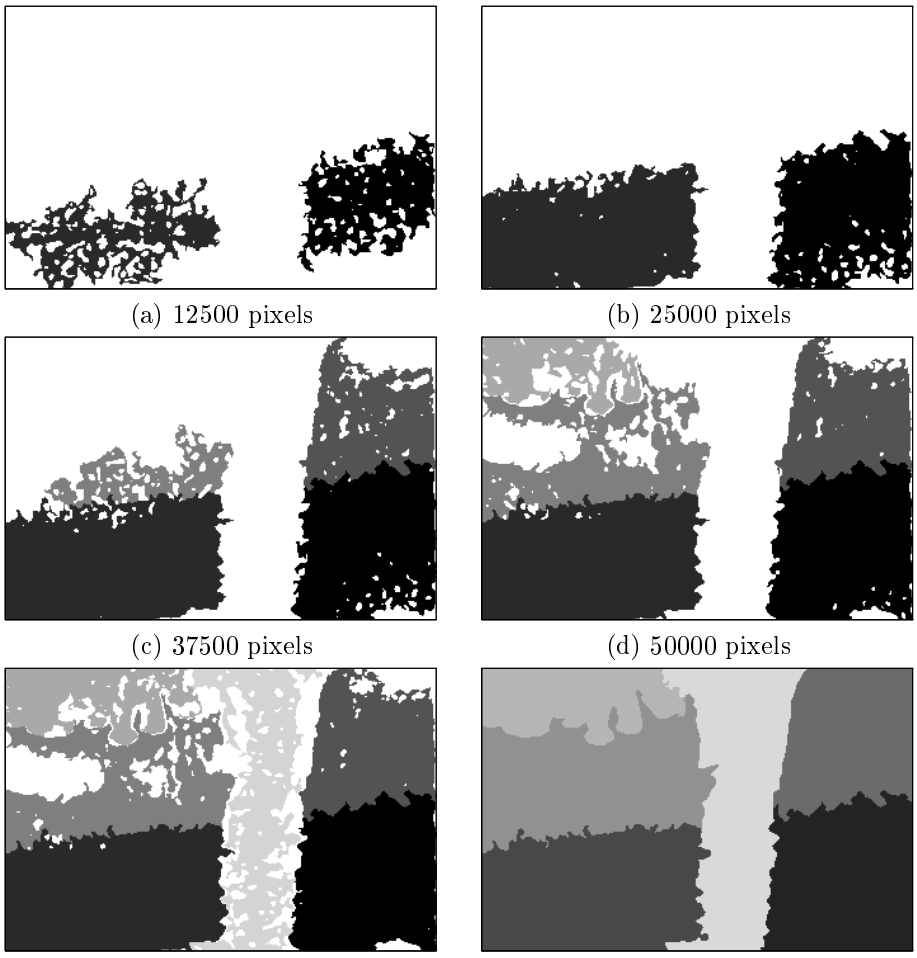
4. The least maximal cost from step 2 is compared to the cost of the cheapest pixel in step 3.

   (a) If the least maximal cost is best, the corresponding candidate region is raised to the status of real region.

(b) Otherwise the cheapest pixel is added to the corresponding region. The aspirant region returns being one candidate region among the others.

To avoid excessive fragmentation of the image into small regions, the comparison cannot be made directly between the least maximal cost and the cost of the cheapest pixel. Instead the first value is multiplied by a penalty factor λ before the comparison is made.

The process is clearly monotone because pixels are only added to the segmentation, never removed. In each iteration at least one pixel (many in case 4a, one in case 4b) is added to the segmentation. Consequently the algorithm needs no more iterations than there are pixels in the image, to finish.



(a) 12500 pixels (b) 25000 pixels

(c) 37500 pixels (d) 50000 pixels

(e) 62500 pixels (f) 75820 pixels

Figure 2: The set of real regions, when a varying number of pixels have been distributed.

In figure 2 the development over time for an image in the test sequence is shown. Only the real regions are included, for natural reasons, and the darker the region, the earlier it was included.

4.1.4 Construction of the candidate regions

The most important omitted detail is how exactly the candidate regions are built. In the current implementation the image is first divided into a large number of overlapping squares, each of size 21×21 pixels (except close to the borders where there is a number of somewhat smaller rectangles). The distance between the centerpoints of these squares has been chosen to be 4 or 6. This gives preliminary candidate regions, and the centerpoints of the squares are their associated starting points.

Given a square it is possible to compute optimal motion model parameters, which also is done. However, there are some problems with using square candidate regions. First, it is a rather strong requirement that an object must contain a square of a given size to have a fair chance of being detected. Second, if a square does indeed fit well to a model but there happens to be a pixel with an off tensor, this also destroys the chances for the region, at least if the maximal cost measure is used, as recommended. Third, it is not obvious how to later adapt the squares to the presence of real regions. A real region may e.g. split the square into two disjoint parts.

The chosen solution is to give up the squares as soon as the first set of optimal parameters have been calculated. The candidate regions are then regrown from their starting points, the way they would have been grown using the competitive algorithm without competitors, until they reach a predetermined size of 400 pixels. This procedure is also used when the candidate regions are rebuilt in later iterations. Then they may of course not be grown into the real regions, and if they cannot obtain a size larger than 200 pixels, they are eliminated.

These limits are rather arbitrarily chosen. The lower limit is mainly intended to avoid uncertain estimates of motion model parameters, which are caused by too small regions. It also prevents very small regions to arise, for good or for bad. For the upper limit, the following considerations must be taken into account:

1. The higher it is, the more robust is the motion model parameter estimation.

2. Objects smaller than the upper limit are not prevented from forming a region but it is harder than for larger objects. The reason for this is that pixels outside the object must be added to the region, increasing the maximal cost. This effect remains until all the pixels adjacent to the object has been occupied by real regions.

3. The higher the limit, the more time does it take to rebuild the region.

The rebuilding of the squares will tend to give regions which are better adapted to the objects in the image. With better regions it should be possible to obtain better motion model parameters. And then a new rebuilding might yield still better regions. This process of alternatingly regrowing the regions and computing new optimal parameters can be repeated arbitrarily many times.

It is not clear how how many times the initial squares should be regrown in this manner. One possibility is to continue until the regions have converged.

Unfortunately this would not work in practice because the regions may enter a cyclic pattern as well as converge. More important though, is that numerous iterations probably are just a waste of time. In the current implementation it is done twice. It is possible that it would be better to do it only once or maybe to only regrow the region once but not compute new optimal parameters.

The size of the initial squares is not critical. The choice is a trade-off between robust parameter estimation and avoiding having multiple objects in one region.

The distance between the centerpoints is a question of efficiency. Candidates with centerpoints very close tend to yield nearly identical regions after regrowing. But the centerpoints can not be spread too sparsely either. Then smaller objects may be missed by the candidate regions.

4.1.5 Choosing the penalty factor

The choice of the penalty factor λ, discussed in section 4.1.3, depends on how important it is to avoid having objects unnecessarily split into more than one region. A too small value of λ leads to excessive fragmentation while a too large value causes multiple objects to be fused into one region. With the objective to get fairly large regions, values in the range from 10 to 40 have successfully been used. It should be noted though, that excessive fragmentation is a far less serious problem than incorrect fusion of multiple objects. In the first case it is relatively easy to detect that two regions have nearly the same parameters while in the latter case it is close to impossible to tell that a region had better be split into several.

In fact, a reasonable alternative to trying to get a good segmentation immediately, would be to intentionally allow excessive fragmentation. As a final step, regions with close motion models could then be combined.

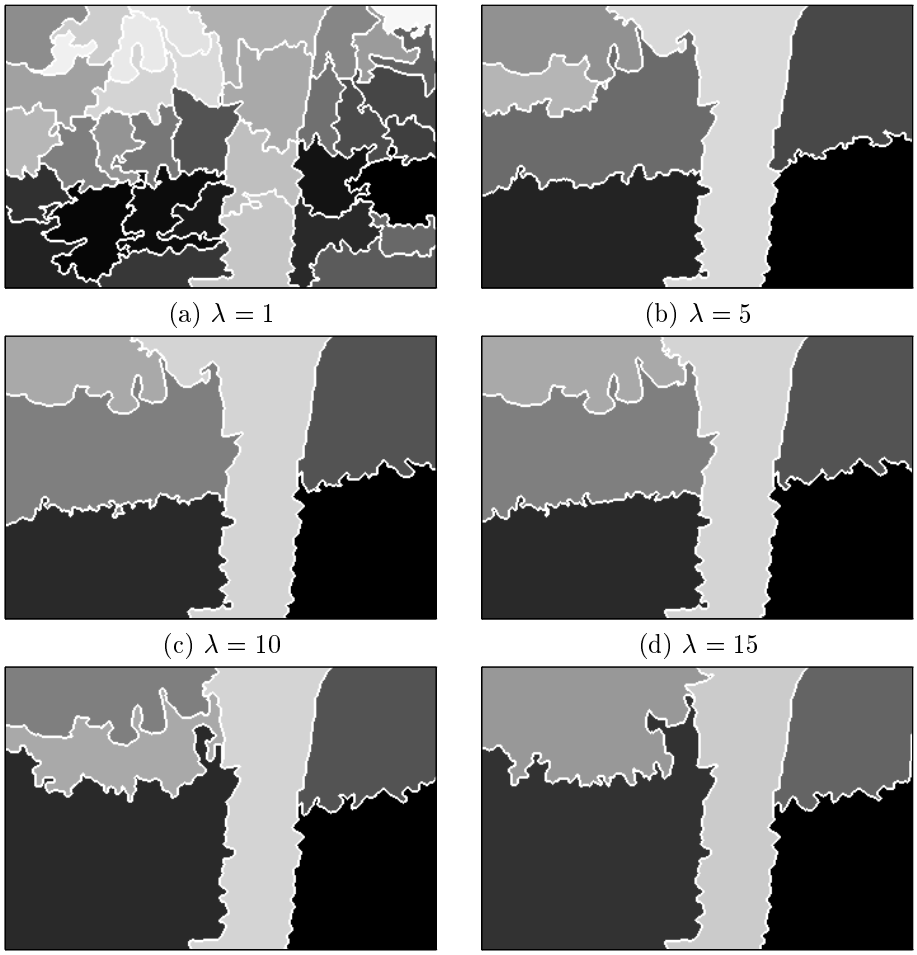
Another problem with a too large penalty factor is that given two adjacent regions in the final segmentation, the first of them to appear gets an unfair advantage over the other. What may happen is that the first region takes pixels that would be cheaper for the other region, only because that has not yet entered the competition.

Segmentation results for varying λ are shown in figure 3.

4.1.6 Efficiency considerations

Unfortunately the time complexity of this algorithm is not very good. For a straight-forward implementation, most of the time is spent rebuilding the candidate regions. The number of candidate regions is approximately proportional to the number of remaining pixels. Summing over all iterations yields the result that the candidate regions are rebuilt $\mathrm{O}(N^2)$ times, where N is the number of pixels in the image. The time for these rebuildings is also at least proportional to the maximum size to which the candidate regions are grown.

Clearly it is important to avoid rebuilding regions unnecessarily. One possibility is to note that in each iteration usually only one, or at most relatively few, pixels are included among the real regions. Hence only the closest candidate regions can be affected, and no others need to be rebuilt. However, there are still quite a few candidate regions that may be affected and if one has to search through all candidate regions to find them, the time complexity remains $\mathrm{O}(N^2)$.



(a) $\lambda = 1$ (b) $\lambda = 5$

(c) $\lambda = 10$ (d) $\lambda = 15$

(e) $\lambda = 20$ (f) $\lambda = 25$

Figure 3: Segmentation results for varying penalty factors.

It is of course possible to avoid an exhaustive search by keeping better track of the regions but that may lead to a time/memory trade-off.

Another possibility, which has been implemented, is to rebuild only the candidates being close to inclusion among the real regions. This is accomplished by storing the set of candidate regions in a list that is sorted with respect to their maximal costs. In subsequent iterations, steps 1 and 2 of section 4.1.3 are replaced by the following steps:

1. Rebuild the first candidate in the list.
2. Resort the list, i.e. move the rebuilt candidate to its new position.
3. If the currently first candidate in the list already has been rebuilt, it is chosen as aspirant for inclusion among the real regions. Otherwise repeat from step 1.

This algorithm is based on the assumption that the maximal costs increase when additional pixels get occupied. The assumption is not always correct but it is so often enough for the algorithm to work satisfyingly. One case where the assumption fails is when a candidate region is completely surrounded by real regions and the worst pixel gets occupied. This makes it even harder for small objects to form regions, cf. point 2 of section 4.1.4.

4.1.7 Implementation details

As mentioned in the definition of regions, section 3.1, the cost function $\mathbf{C}_R(\mathbf{x})$ may vary with the pixels currently included in the region. In the current implementation this is reflected in the fact that optimal parameters are recomputed on a regular basis. The parameters for the candidate regions are recomputed every time they are regrown, not only while forming the initial candidates. For the real regions, the parameters are recomputed for every fiftieth pixel that is added to a region.

It has not been studied what effects these recomputations have. It seems however probable that it is done more frequently than it is worth.

Another implementation detail that may be worth noting, is that the candidate rebuilding process from the previous section is implemented somewhat differently. Instead of testing if the currently first candidate in the list already has been rebuilt it is tested if a rebuilt region *remains* first. Considering that the parameters are recomputed each time, this may lead to additional rebuildings of some candidates or in the worst case to an infinite loop, where two or more candidates get rebuilt alternately. The latter case has this far never occured, but this part of the algorithm should be reimplemented if it is going to be used in applications.

4.2 Algorithm 2

This algorithm takes advantage of the segmentation of the previous image when it segments a new image. One reason for that is that by using old information, the new segmentation can be done faster and more robustly. Another important reason is to get a segmentation that is consistent with the previous one, and where the objects are labeled in the same way. This is important in some applications, such as tracking and video coding. It also enables the postprocessing

algorithm, section 4.3, to be applied. A drawback, however, is that it does not take into account the possibility of objects suddenly appearing or disappearing.

The inputs to this algorithm are an orientation tensor field for the current image and a segmentation of the previous image. The only extension to the competitive algorithm that is needed here is the construction of the initial regions. Like the previous algorithm, the d_3 distance function is used as cost function, and the d_1 distance function is used for recomputing the parameters for a region.

4.2.1 Initial regions

It is a basic assumption of this algorithm that the new image contains the same objects as the previous. From the segmentation of the previous image we also know the motion model parameters for each region. A straightforward way to find the segmentation of the new image would be to simply relocate each region according to its motion model. There are a number of problems with this approach, however, e.g.

1. The pixels would not move exactly to new pixel positions so the regions would have to be resampled.

2. Regions may overlap after relocation, which is of course not acceptable in the final segmentation. This problem is nontrivial to solve, especially the connectivity restriction requires carefulness.

3. There may appear holes in the segmentation. This is, however, easy to deal with, using the competitive algorithm.

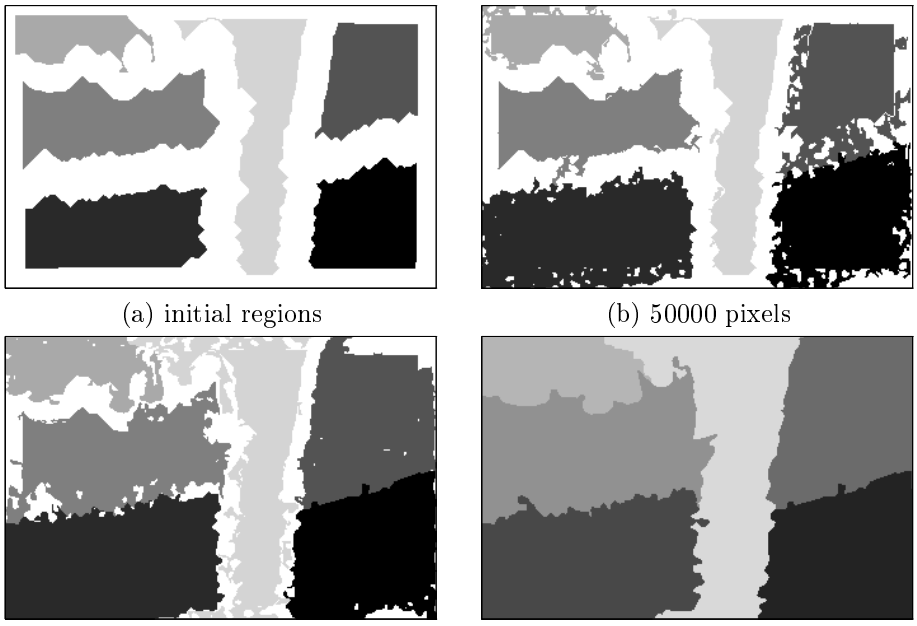
The chosen solution is simpler but less sofisticated, mainly because it ignores the motion information. Assuming that no region has moved very far, there will be a significant overlap between the previous and the new positions. Especially is it likely that the "center" of a region is in the intersection of the old and the new positions. Therefore, to find initial regions that can be grown, each region is simply shrunk to half its size in a connectivity preserving manner. In this way it is guaranteed that the initial regions will each be connected, and disjoint from each other. The assumption that no region moves far is reasonable given the fact that large motions not can be handled well anyway, see section 6.2.1.

The choice of shrinking the regions to *half* their sizes is rather arbitrary and it has not been studied how well other fractions would work. It is however clear that removing a large fraction would be slower and may yield small regions, which is bad for parameter estimation, while it would decrease the risk that an initial region accidentally contains pixels that should belong to another region.

4.2.2 Iterative step

For the initial regions, new optimal parameters are computed. Then the rest of the pixels are distributed strictly according to the competitive algorithm.

The development of the regions for an image in the test sequence is shown in figure 4.



(a) initial regions (b) 50000 pixels

(c) 62500 pixels (d) 75820 pixels

Figure 4: (a) The initial regions in algorithm 2. (b) – (d) The regions when a varying number of pixels have been distributed.

4.3 Postprocessing algorithm

Application of the two aforementioned segmentation algorithms on a whole sequence yields segmentations that are more or less stable over time. The first algorithm treats each image independently and can therefore not guarantee that the segmentations consist of corresponding regions, or even the same number of regions. Should the segmentations happen to have corresponding regions, it is still nontrivial to match those between the images.

If algorithm 1 is used only on the first image, and algorithm 2 is used for subsequent images, it is guaranteed that all segmentations have the same number of regions and also that corrresponding regions are consistently labeled. Still the regions may be unstable, the borders may float around, even after compensation for the motion of the objects. The postprocessing algorithm in this section is intended to stabilize the regions by imposing continuity over time on the segmentations.

4.3.1 Principal ideas

The basic idea is to trace how each region moves through the three-dimensional space-time volume, where the movement is defined by the estimated motion models. Each region traces some kind of cylinder, possibly skewed or deformed in some other way. This volume is said to be *influenced* by the region in question. The influences from all regions are summed and then all images are resegmented. In the resegmentation phase, each pixel is assigned to the region with the most influence over the pixel. There is one exception to this rule; the

spatial connectivity restriction of the regions takes precedence.

4.3.2 Algorithm

As input this algorithm takes the segmentations of all images in an image sequence. The segmentations must be consistently labeled. The algorithm consists of the following steps:

1. Start with a cleared influence volume.
2. For each pixel in each image:
   (a) Note what region it belongs to.
   (b) Follow the path according to the motion models for that region at each image, forward and backward in time. Add influence to each pixel the path passes through, or rather to the four pixels (per image) closest to the path, weighted with respect to their distances to the path.
3. For each image, resegment according to the following steps:
   (a) Shrink each region to half its size so that the pixels with the least support are removed first, while preserving connectivity.
   (b) Apply the competitive algorithm with the negative of the influences as cost function.

The choice to shrink the regions to *half* their sizes is arbitrary and should not be critical. They could probably be shrunk less.

It should be noted that the running time for the first part of the algorithm is quadratic in the number of images. For long sequences this behaviour must be avoided. A simple modification to get rid of the problem is to make the pathfollowing in step 2b restricted to a fixed number of images, instead of continuing to the start and the end of the sequence. Another problem for long sequences is that the memory requirement increases linearly with the length of the sequence. The proposed modification would however solve that problem too.

5 Implementation and results

5.1 Implementation

The algorithms described in section 4 have been implemented as Matlab mex-files, written in C++. The computation of the tensor fields is carried out by software available at the Computer Vision Laboratory, and is here regarded as a black box. The tensor preprocessing has been implemented as a Matlab script. Auxiliary programs for converting between different data formats, presenting results, etc., have been implemented as C programs, UNIX shellscripts and Matlab scripts. The code for the segmentation algorithms is included in appendix A.

5.2 Performance

The algorithms have been tested on 26 frames from the flower garden sequence, shown in figure 1. The dimensions are 340×223 pixels. The programs have been run on a SUN Sparcstation 20. The computation of the original orientation tensor field takes 1.5 minutes per image and the preprocessing of the tensors takes 12 minutes per image. The running time of algorithm 1 is 45 minutes per image, of which about 75 percent is spent on the construction of 4346 initial candidate regions (section 4.1.4) and about 25 percent on the iterative step (section 4.1.3). Algorithm 2 requires 2 minutes per image, distributed approximately equal between the construction of the initial regions and the iterative step. The time to run the postprocessing algorithm on the 26 images in the sequence is 90 minutes which averages to 3.5 minutes per image. Note that the running time for a part of this algorithm is quadratic, however (cf. section 4.3.2). For this sequence about 10 percent of the time is spent on that part.

5.3 Results

The test sequence has been segmented in the following ways:

1. Using only algorithm 1.
2. Using algorithm 1 for the first image followed by algorithm 2 for the remaining images.
3. Improving segmentation number 2 by invoking the postprocessing algorithm.

Selected images from the segmented sequences are shown in figure 6 (algorithm 1), figure 6 (algorithm 2), and figure 7 (postprocessing algorithm). In algorithm 1 the distance between the starting points for the candidate regions was 4 and the penalty factor λ was 10. Other constants had the values that are proposed in the descriptions of the algorithms, and also listed in table 1.

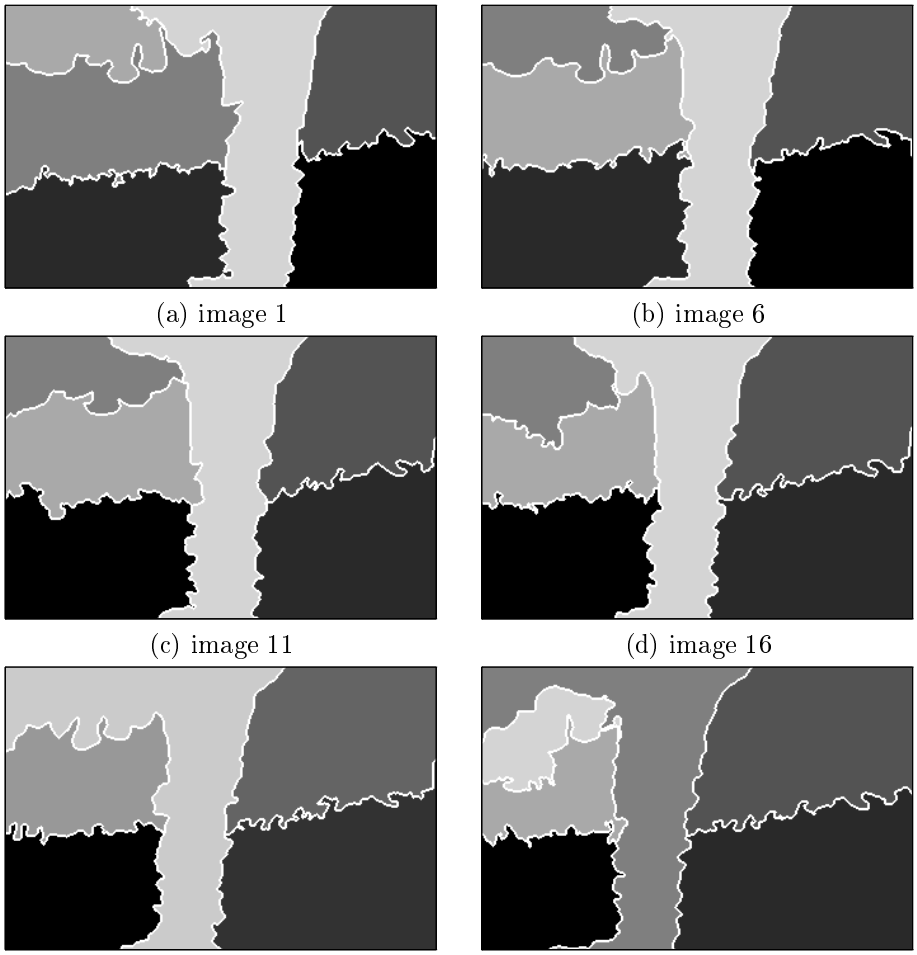
6 Discussion

6.1 Strengths of the algorithms

6.1.1 Insensitivity to the aperture problem

It is briefly mentioned in section 2.1 that for a moving linear structure, only the velocity component normal to the structure can be measured. This problem is fundamental when estimating the velocity in an image sequence and is commonly called the aperture problem.

The aperture problem is a serious problem for algorithms that estimate a local motion model, based solely on pointwise velocity estimates in some neighborhood. At points where the aperture problem is present, the velocity estimation algorithm may either report the normal component or give up. In the former case the mix between estimates of the true velocity and normal components yields incorrect motion model estimates and in the latter case useful information is wasted. A minimum requirement to deal with this problem is that the

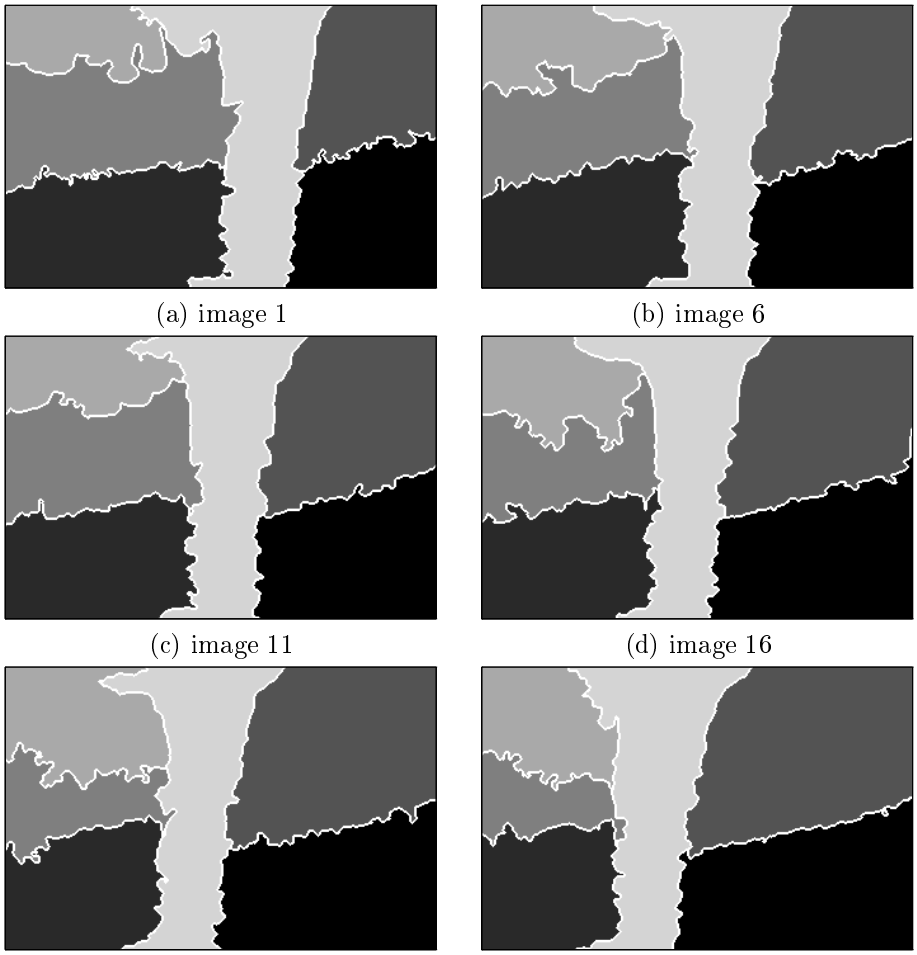


(a) image 1 (b) image 6

(c) image 11 (d) image 16

(e) image 21 (f) image 26

Figure 5: Segmentation results for algorithm 1.

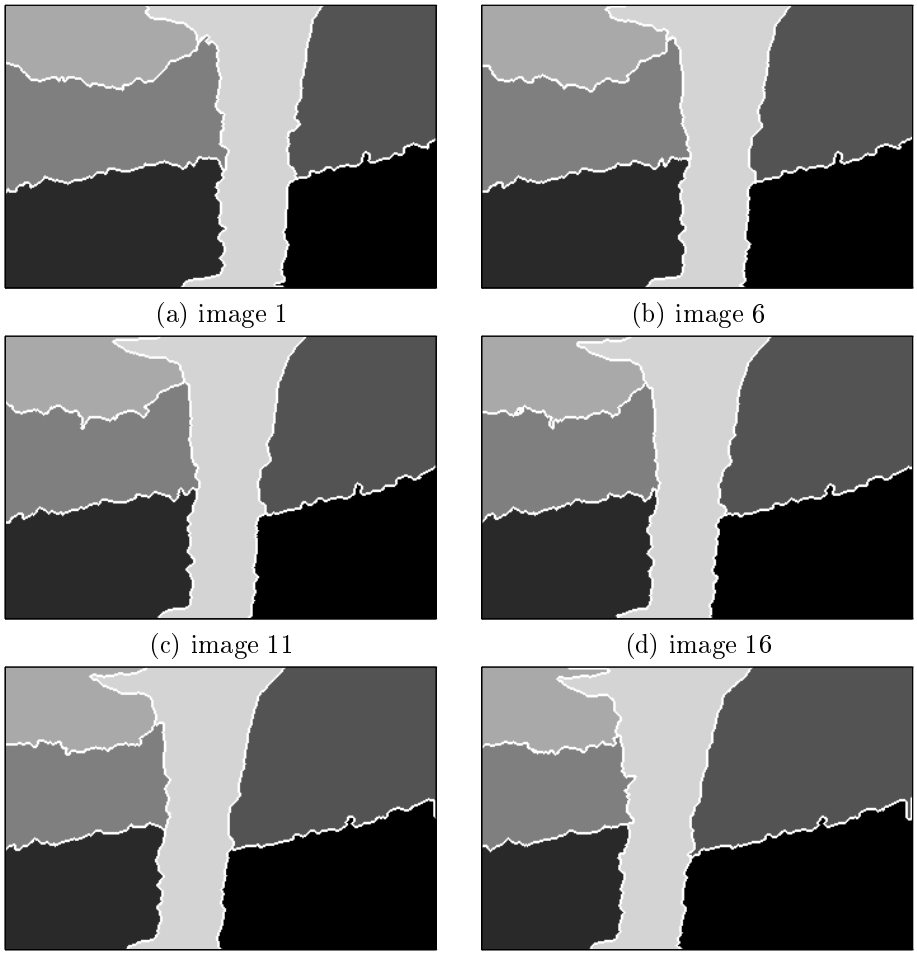


(a) image 1 (b) image 6

(c) image 11 (d) image 16

(e) image 21 (f) image 26

Figure 6: Segmentation results for algorithm 2.



(a) image 1 (b) image 6

(c) image 11 (d) image 16

(e) image 21 (f) image 26

Figure 7: Segmentation results after invoking the postprocessesing algorithm.

velocity estimation algorithm, in addition to the velocity estimate, tells whether the estimate is a normal component or a true velocity, or more generally gives some kind of certainty measure. It is of course also necessary to find a good way to incorporate the certainty information into the motion model estimation.

The orientation tensors contain all the necessary information and the theory in sections 2.3.1 and 2.3.2 shows how it can be used. The idea of the algorithm is to minimize a distance (d_1) between a motion model, or rather the velocity field it represents, and the orientation tensors in a region. For points where the aperture problem is present, the distance between the tensor and a velocity hypothesis is zero for all velocities with the correct normal component. For other velocities the distance is only dependent on the error in the normal component. At points where the true velocity can be estimated (i.e. the moving point case), the distance increases with the error in any direction. For the majority of points that fall between these two extreme cases, the distance function takes on an intermediate behaviour.

In this way the effect of the aperture problem can be nearly eliminated. If a region contains linear structures in different orientations, the information from these is combined in the process of minimizing the total distance. Only when the whole region consists of one linear structure, there is a problem of insufficient information. In this case however, the algorithm still yields a result. Among the motion models that conforms to the normal velocity components for the structure, the one with smallest parameters is chosen. This is about the best that can be done.

The related problem of areas where no velocity information is available at all is addressed in the following section.

6.1.2 Dense velocity field unnecessary

It is not always possible to estimate the velocity at a point, not even the normal component. The most obvious case is a nontextured area, where any motion, or absence of motion, is impossible to detect. The orientation tensor is in this case zero. Another important case is areas with only uncorrelated noise. Velocity is undetectable here too. The orientation tensor is not zero in this case but still isotropic, i.e. all eigenvalues are equal. Isotropic or close to isotropic tensors may also appear in areas with certain conflicting structures.

All isotropic tensors are mapped into null-tensors in the preprocessing (section 2.3.2). Since the distance function d_1 is identically zero in this case, they have no effect at all on the estimation of the motion model for a region that have some points of this kind. If the region consists solely of points with null-tensors, it is of course impossible to say that one motion model is better than another, without additional constraints. The constraint that is used is to take the model with the smallest parameters, in this case the zero motion model.

What is important to note is that the motion model estimation does not require a dense velocity field, i.e. there is allowed to be some points where no velocity can be estimated.

It is also interesting to see what the segmentation algorithms do with such points. As no velocity can be estimated there, there is no reason to prefer one motion model over another, so the points may be assigned to any adjacent region. Another alternative is to treat those points as an outlayer, i.e. not assign them to any region. The choice between these two alternatives is dependent on

the application. Some applications, such as video coding, usually require a complete partition, while other applications do better if uncertain points are left out. The two segmentation algorithms in this thesis follow the first way. It is however easy to adapt them to the other strategy. The idea is essentially to stop the competitive algorithm when it runs out of good points instead of when it runs out of all points.

For a more thorough analysis it should be noted that exactly isotropic tensors are somewhat anomalous. They cannot completely be avoided but usually the presence of noise yields tensors that are only close to isotropic. The above discussion does of course apply to these too, but to a lesser extent as they become more anisotropic.

For algorithm 1 it is important that a group of points with nearly isotropic tensors does not manage to form a region on their own, because the motion model estimation would be highly uncertain, or in extreme cases more or less random. Supposing that a candidate region consists of these points, the estimated motion model would fit badly to a lot of the tensors. That the tensors have small eigenvalues after the preprocessing would not matter, because the distance function d_3, which is used as cost function, is normalized with respect to the eigenvalues. Hence the maximal cost would be very high and the chances to be included among the real regions would be very low.

For the same reason uncertain points tend to be far from the motion models of adjacent regions. This means that the cost for these points will be very high and that most of the other points will be occupied before these are, at the very end. The alternative strategy to leave uncertain points as an outlayer, can be implemented by stopping the algorithm when the cost for adding another point becomes too high.

6.1.3 Few threshold values and other arbitrary constants

One of the main objectives in the development of the algorithms, has been to avoid introducing threshold values and other "magic" constants that must be carefully and empirically chosen and possibly changed for every sequence as well. The competitive algorithm is a result of this philosophy; values are only compared among each others, never to external thresholds.

A complete exclusion of arbitrary values has, however, not been possible to obtain for the complete segmentation algorithms. This is not only for bad but may in fact be to advantage since it can be useful to have some design parameters to play with; at least if their effects are reasonably possible to anticipate.

The arbitrary constants are listed in table 1. Their exact effects are described in the presentation of the algorithms. The general effects listed in the table are the main factors that are influenced by the choice of value. "Accuracy" refers to the quality of the segmentation and "estimation" refers to the possibility of robustly estimating the motion model.

Many of the constants only give a trade-off between speed and accuracy. The most interesting constants are probably the lower and upper size limits for the candidate regions, together with the penalty factor λ.

short description	general effects	typical value	section
initial size of candidates	speed, accuracy, estimation	21×21	4.1.4
distance between starting points	speed, accuracy	$4 - 6$	4.1.4
lower limit for candidate size	accuracy, estimation	200	4.1.4
upper limit for candidate size	speed, accuracy	400	4.1.4
penalty factor λ	accuracy	$10 - 40$	4.1.3 4.1.5
shrinking factor algorithm 2	speed, accuracy, estimation	50%	4.2.1
shrinking factor postprocessing	speed, accuracy	50%	4.3.2

Table 1: Arbitrary constants

6.1.4 Independent motion model estimation and segmentation

It is clear from the description of the algorithms that the motion model estimation algorithm and the segmentation algorithms are independent of each other. The motion model estimation obviously works for any regions, however obtained. The segmentation algorithms need the motion estimation only to get cost functions for the different variants of the competitive algorithm. This independence is not an advantage in itself but it allows for flexibility.

First, the motion model estimation is interesting in itself. How the motion information can be used is discussed in sections 6.5.2 and 6.5.3.

Second, the segmentation algorithms can be used with other cost functions. This makes it possible to modify them to base the segmentation on other criteria than motion, or additional criteria together with motion. Naturally, the cost function cannot be chosen too carelessly if it is supposed to work well; it has to behave in roughly the same way as the described one. The case of additional criteria is further discussed in section 6.3.2.

The postprocessing algorithm on the other hand does not even use the motion model estimation and can in fact be applied to any consistently labeled segmentation of an image sequence.

6.2 Weaknesses of the algorithms

6.2.1 Fast moving objects

An important drawback with the motion model estimation algorithm is that it is unable to detect large motions. This weakness is inherited from the orientation tensor computation. As briefly mentioned in section 2.1, the tensor computation involves the output from some quadrature filters. These filters must necessarily be of limited size, to get a reasonable computing time. Clearly, if an object moves so fast that it is within the borders of the filter at only one image, there

is no possibility to estimate its motion.

The filters used in the evaluation of the segmentation algorithms have been of the size $9 \times 9 \times 9$. Movements faster than 5 pixels per image have not been possible to estimate. In fact, the highest velocities that the algorithms can handle seems to be about 2 to 3 pixels per image.

A simple, but certainly not satisfying method to avoid this problem, is to subsample the image sequence in the spatial directions. Another simple method, if it is feasible, is to increase the sampling rate.

A more sofisticated approach would be to use a (spatial) scale pyramid. At a very small scale, any velocity should be small enough to allow estimation. At increasing scales the estimation should get better as long as the velocity is small enough. There are, however, a number of problems that must be solved. First, it is not obvious how the final tensor should be constructed from the tensors at different scales. Second, one wants a *local* velocity estimation, but at small scales a rather large neighborhood is involved.

If the problem of estimating large motions is solved, one must remember to improve algorithm 2 too. A part of it is dependent on the assumption that the objects do not move too fast.

6.2.2 Efficiency

Another drawback with the algorithms is their time efficiency. They are clearly too slow for real-time applications. Even for some off-line applications they may be too slow, especially algorithm 1. It should be noted, however, that time efficiency has not been a main objective in the development of the algorithms, and that it may be possible to speed up the code significantly. Some ideas on how this can be done are discussed in section 6.3.4.

In the current implementation, the memory requirements are rather large too. This is mostly because the quadratic forms $\mathbf{Q}$ from section 2.3.2 are precalculated and stored in memory.

6.2.3 Appearing and disappearing objects

Image sequences where the set of visible objects varies are especially cumbersome. Algorithm 2 cannot handle this case well at all. A suddenly appearing object will not get a corresponding region and must instead be incorporated among the existing regions. The region corresponding to a disappearing object will remain in the partition, and probably start competing with another region for some object.

For algorithm 1 this situation is of no consequence since it segments each image independently from the others. But using algorithm 1 to segment an entire sequence is not very useful since it is in general hard to tell how the regions in the different images relate to each other. Without this information the segmentation is considerably less useful.

A possible solution to this problem is to combine the two algorithms, which is discussed in section 6.3.1.

6.2.4 Small objects

The problem with small objects is that robust estimation of motion models requires large enough regions. Hence neither of the algorithms can detect very

small objects. This problem can be avoided by decreasing the lower size limits of the candidate regions, at the risk of getting bad motion model estimates. A more robust solution requires modification of the algorithms, preferably combined with additional information about the small objects, such as known motion or texture. A different approach to improve the situation is given by the proposed three-dimensional algorithm in section 6.3.3.

6.3 Future work

There is a lot more work that can be done to develop and improve the algorithms. Some ideas are addressed in this section. In addition it would be valuable to evaluate the usefulness of the algorithms by applying the segmentations to some application.

6.3.1 Combination of algorithm 1 and algorithm 2

A natural development of the presented algorithms is to combine algorithm 1 and algorithm 2. The idea is to take advantage of the previous segmentation, like algorithm 2, but still be open for inclusion of new regions, like algorithm 1. This approach would counter the problem of suddenly appearing or disappearing objects, discussed in section 6.2.3.

A rough sketch of how it can be done follows:

1. First the regions from the previous segmentation have to be incorporated. This must be done in a more sophisticated way than in algorithm 2, to allow for objects to appear or disappear. A possibility is to project the regions according to their motion models and then shrink or threshold them based on how well they fit the new tensor field. Regions that fit too badly are removed. Some care must be taken to assure that the regions remain connected and disjunct from each other. An important problem to deal with is the possibility of regions getting split or merged.

2. With the regions from the first step included as real regions in the segmentation, candidate regions can be constructed essentially in the same way as in algorithm 1, section 4.1.4.

   (a) Distribute starting points over the nonoccupied pixels. Easiest is to distribute them regularly but a randomized distribution might be an interesting alternative, given that the area to distribute over now is irregular.

   (b) Obtain a first motion model estimate for each candidate, from arbitrary neighborhoods of the starting points.

   (c) Grow the candidate regions from their starting points, until they reach some given size or run out of points. Too small regions are eliminated.

3. Distribute the pixels among the real regions and raise candidate regions to the status of real regions when appropriate, exactly as in algorithm 1, section 4.1.3.

If this new algorithm is used instead of algorithm 2 to segment a whole sequence, it should still be possible to improve the result using a postprocessing algorithm. The presented algorithm would probably need some development though.

6.3.2 Using additional information

In many applications it is unlikely that it is sufficient to base the segmentation on motion only. From the results in section 5.3 one can see that although the region containing the main part of the tree has obtained a correct general shape, the boundaries of the region do not exactly match the boundaries of the tree. To get a better segmentation it is necessary to use additional information, such as texture, or in general one or more local features.

The natural way to extend the segmentation algorithms is to modify the cost function. This can easily be done if the following two conditions are satisfied:

1. The feature can be estimated robustly over a region.
2. It is possible to measure how well a value of the feature fits at a given point or at some neighborhood of the point. This gives a distance function that is analogous to the previously used ones.

Now the new cost function can be constructed as a combination between the old cost function and the distance function for the additional feature. The combination may be a weighted sum but there are other possibilities as well.

A straight-forward method to obtain the distance function in condition 2 is to estimate the feature locally at the point and then compare this estimate to the region estimate from condition 1. A problem here is that it is not necessarily possible to obtain a robust local estimate for the feature.

It should however be pointed out that condition 2 does not require that the feature itself be estimated at the point. It is sufficient if a hypothesis can be tested. This is in fact what is done for the distance functions in section 2.3.2. There the affine motion model is the feature that is estimated over a region. At a given point, the model can be tested against a tensor, which is computed from a neighborhood of the point. But the tensor alone is not sufficient to estimate a motion model.

6.3.3 Three-dimensional region growing

The presented segmentation algorithms work with one image at a time. The continuity over time that can be expected in an image sequence is only weakly exploited. Algorithm 1 ignores it completely and algorithm 2 uses only one previous segmentation. The whole sequence is not involved until the postprocessing algorithm is invoked.

A stronger impact from the continuity over time would result if the segmentation were made directly in the three-dimensional volume of stacked images. One algorithm of this kind can be obtained simply by extending algorithm 1 to three dimensions. The following modifications are needed:

1. The motion model must be extended to include the time coordinate t. Otherwise the allowed object motions would be significantly restricted. How motion models can be extended is discussed in section 6.4.

2. The regions and candidate regions are extended to three dimensions but with the same properties as before. The region growing is done in three dimensions.

3. Motion model estimation is done over three-dimensional regions, but it still works just as before.

Note that the problem with appearing and disappearing objects becomes nonexistent. The problem with small objects is not solved, in a strict sense, but with another dimension to play with the required number of points can be obtained for spatially much smaller objects. The consistent labeling of the regions over time now comes for free.

There are, however, a number of problems that may or may not be solved:

1. The memory requirements are greatly increased, roughly proportionally to the number of images.

2. The time efficiency probably gets worse than before.

3. The shapes of the regions should be restrained to be consistent with their motions.

4. Likewise the connectivity constraint in the time direction should reflect the motions of the regions. A small but fast moving object may be disconnected in the classical sense.

6.3.4 More efficient implementations

The time efficiency is a weakness of the implemented algorithms. Possible improvements include the following list.

1. The tensor preprocessing step could be implemented as a Matlab mex-file rather than as a Matlab script. Also the tensor estimation as well as the tensor preprocessing are local operations that could efficiently be parallelized.

2. There are a number of modifications that may or may not affect the quality of the segmentations. Those have been discussed in the description of the algorithms.

   (a) The number of times that the initial candidate regions are regrown could be decreased, section 4.1.4.

   (b) The optimal motion model parameters could be recomputed less often, section 4.1.7.

   (c) The initial regions for algorithm 2 could be made larger, i.e. the old regions could be shrunk less, section 4.2.1.

   (d) The same is true for the postprocessing algorithm, section sec:algorithm postprocessing.

   (e) The quadratic running time of the postprocessing algorithm could be avoided, section sec:algorithm postprocessing.

3. A sophisticated approach would be to use a scale pyramid. This idea is discussed in section 6.2.1 as a possible method to avoid the problem of too large motions. But the idea could be taken further by running the algorithms at multiple scales. With some luck large regions as well as fast moving regions could be found at a small scale, while small regions and the details of the borders could be found at a larger scale.

   Make the algorithms stand-alone programs instead of Matlab mex-files. This would require that the

6.4 Generalization of the motion model

The affine motion model utilized by the segmentation algorithms is by no means the only possible choice of motion model. In fact, the distance functions in section 2.3.2 and the theory for distance minimization in section 2.3.3 only depend on the property that the motion model be linear in its *parameters*. This fact is reflected in equation (16) for the affine case.

To give an example, suppose that we have a quadratic motion model, such that

$$v_x(x,y) = ax^2 + bxy + cy^2 + dx + ey + f, \tag{27}$$

$$v_y(x,y) = gx^2 + hxy + iy^2 + jx + ky + l, \tag{28}$$

for some parameters a through l. Then the matrices $\mathbf{S}$ and $\mathbf{P}$ of equation (16) would be replaced by

$$\mathbf{S} = \begin{pmatrix} x^2 & xy & y^2 & x & y & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x^2 & xy & y^2 & x & y & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{29}$$

and

$$\mathbf{P} = \begin{pmatrix} a & b & c & d & e & f & g & h & i & j & k & l & 1 \end{pmatrix}^T. \tag{30}$$

The remaining theory would work out as before.

An interesting special case is when the motion model is the simplest possible; a pure translation. Then

$$v_x(x,y) = a, \tag{31}$$

$$v_y(x,y) = b, \tag{32}$$

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{33}$$

and

$$\mathbf{P} = \begin{pmatrix} a \\ b \\ 1 \end{pmatrix}. \tag{34}$$

Compared to using equations (3) and (4) at a single point to estimate the motion, this approach makes it possible to combine the motion information in

a neighborhood in order to both get a more robust estimation and avoid the aperture problem (as discussed in section 6.1.1). This method is related to, but not identical to, averaging the tensors over a neighborhood before estimating the velocity using equation (3) or (4). The main difference is that the tensors are not preprocessed, as described in section 2.3.1, in the latter case.

6.5 Applications

Segmentation is an important step in a number of applications. The presented algorithms or the extensions proposed in section 6.3 should be useful in any of these. Additionally the motion model estimation may be useful on its own, especially after the generalization in section 6.4.

Three possible applications are briefly discussed in this chapter.

6.5.1 Segmentation based video coding

In segmentation based video coding methods, motion compensated prediction is an important step. Hence the estimated motion models come to use and the segmentation into regions characterized by a common motion makes sense. It is important to have consistently labeled segmentations and also desirable that the segmentations are stable. These conditions are fulfilled by algorithm 2 together with the postprocessing algorithm.

Unfortunately the weaknesses listed in section 6.2 cannot be ignored. In particular efficiency is an important issue in video coding. Without significant speed improvements, these algorithms can at best be used for off-line coding. To counter the problem of appearing and disappearing objects, the improvements proposed in sections 6.3.1 and 6.3.3 should be useful. The latter would also be able to deal with small objects.

The extension proposed in section 6.3.2 may also come to use since it may be valuable to have segmentations that are better adapted to the texture in the images.

6.5.2 Motion estimation

Motion estimation may in itself be an application. One may e.g. wish to estimate how some material is deformed when exposed to stress. The strength of the presented algorithm is twofold. First, it is able to estimate a wide range of motion models (cf. section 6.4). Second, it does not require special preparation of the material or sophisticated measuring devices. It is only needed that the material is sufficiently textured and that a video sequence of the experiment can be obtained. A weakness of the algorithm is of course the limitation to small motions. The size of the motion may however be possible to adjust with the sampling rate of the camera.

6.5.3 Depth estimation

If the motion of the camera is known it may be possible to estimate the depth and the slope of stationary objects in the scene, from the affine motion models corresponding to their apparent motions. To give an example, the flower garden sequence, figure 1, has the property that objects seem to move slower the further

away they are. For a sloping object, different parts of it seem to have different velocities, which is reflected by the affine motion model.

A Code

References

[1] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1.

[2] J. Y. A. Wang and E. H. Adelson. Layered representation for motion analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 361–366, June 1993.

[3] J. Y. A. Wang and E. H. Adelson. Spatio-temporal segmentation of video data. In *Proceedings of the SPIE Conference: Image and Video Processing II*, February 1994.