

# **Efficient Learning of Objects in Images**

THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER  
FOR THE DEGREE OF M.SC. IN THE FACULTY OF SCIENCE AND ENGINEERING

2004

**Gwenn Englebienne**  
Department of Computer Science

# Contents

<b>Abstract</b>	<b>6</b>
<b>Acknowledgements</b>	<b>9</b>
<b>1 Introduction</b>	<b>10</b>
1.1 Approaches to Image Segmentation . . . . .	11
1.1.1 Single Images . . . . .	11
1.1.2 Video Sequences . . . . .	12
1.2 Segmentation as an Optimisation . . . . .	13
1.3 Structure of this thesis . . . . .	15
<b>2 Previous Work</b>	<b>16</b>
2.1 Transformed Mixtures of Gaussians . . . . .	16
2.1.1 Introduction . . . . .	16
2.1.2 A More Theoretically Founded Look at TMG . . . . .	18
2.1.3 Discussion . . . . .	21
2.2 The GREEDY Algorithm . . . . .	22
2.2.1 Introduction . . . . .	22
2.2.2 The Algorithm . . . . .	24

---

<b>3</b>	<b>Implementation</b>	<b>30</b>
3.1	Transformations . . . . .	30
3.1.1	Cached transformations . . . . .	31
3.2	Machine Precision . . . . .	32
3.3	Parallel Execution . . . . .	33
3.3.1	Details of the Matlab Implementation . . . . .	34
3.3.2	Job Management System . . . . .	35
3.3.3	Helper Programs . . . . .	36
<b>4</b>	<b>Optimisations</b>	<b>37</b>
4.1	Transformations . . . . .	37
4.1.1	Objects of Limited Size . . . . .	39
4.2	Incremental resolution . . . . .	40
4.2.1	Chequerboard sampling . . . . .	42
4.3	Lazy updates . . . . .	43
4.4	Sparse and Incremental EM . . . . .	44
4.4.1	Sparse EM . . . . .	44
4.4.2	Incremental EM . . . . .	46
4.4.3	Tracking . . . . .	47
4.5	Colour . . . . .	49
4.6	Rotating Masks . . . . .	51
<b>5</b>	<b>Experiments</b>	<b>52</b>
5.1	Data Acquisition . . . . .	52
5.1.1	Existing Data . . . . .	52
5.1.2	New Video Data . . . . .	53

---

5.1.3	New Artificial Data . . . . .	54
5.2	Results . . . . .	54
5.2.1	Resolution Reduction . . . . .	55
5.2.2	Lazy Algorithm . . . . .	55
5.2.3	Incremental EM . . . . .	56
5.2.4	Colour . . . . .	58
5.2.5	Rotating Masks . . . . .	59
<b>6</b>	<b>Conclusions</b>	<b>61</b>
6.1	Key results . . . . .	61
6.1.1	Speed Issues . . . . .	61
6.1.2	Robustness of the Algorithm . . . . .	61
6.2	Future work . . . . .	62
6.3	Summary . . . . .	63
<b>A</b>	<b>The EM algorithm</b>	<b>68</b>
A.1	The Basic Algorithm . . . . .	68
A.2	Incremental EM . . . . .	69
A.3	Sparse EM . . . . .	70
<b>B</b>	<b>Implementation Details</b>	<b>72</b>
B.1	Learning the Background . . . . .	73
B.2	The EM algorithm . . . . .	73
B.3	Learning the Foreground Objects . . . . .	74
B.4	Learning Colour Objects . . . . .	75

# List of Figures

1.1	Aliasing of Border Pixels . . . . .	13
2.1	Masked latent object variables . . . . .	22
4.1	Transformation of the latent objects . . . . .	38
4.2	Translation of latent objects of limited size . . . . .	39
5.1	Some frames from the original sequence . . . . .	52
5.2	Some frames from the new sequences . . . . .	53
5.3	Artificial Colour Sequence . . . . .	54
5.4	Segmentation of the Original Sequence . . . . .	54
5.5	Likelihood vs. Time for Greedy and Sub-sampled Greedy . . . . .	55
5.6	Likelihood for ‘lazy’ and non-‘lazy’ greedy algorithm . . . . .	56
5.7	Comparison of GREEDY and incremental GREEDY . . . . .	57
5.8	Segmentation with Colour and Greyscale GREEDY . . . . .	58
5.9	Latent Objects for Rotating Masks . . . . .	60

## **Abstract**

The GREEDY algorithm was developed by Williams and Titsias as an efficient way of segmenting objects in images in a statistically justified way. In this paper we present our investigation of this algorithm. We tested our implementation of the algorithm on various datasets, and verified the results. We also explored different improvements on the basic algorithm, leading to a more efficient and hopefully more robust algorithm.

## **Declaration**

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

## Copyright

1. Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.
2. The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.
3. Further information on the conditions under which disclosures and exploitation may take place is available from the Head of the Department of Computer Science.

# Acknowledgements

It is traditional in this kind of work to include a little ‘*thank you*’ to about anybody — ranging from the author’s supervisor, colleagues, parents, to his girl or boyfriend and potentially a few pets.

I won’t stray from the well-throdden path in this matter, and I will thus start with a little word for my supervisor. But, if this is in any way possible, I would like this ‘thank you’ to be as unique as the help I received — for I was truly amazed by the assistance I got. Thank you, Dr. Rattray, for your persistent support and help for these last months.

Now unfortunately, I have neither girlfriend nor pet to thank, which is a pity as I’m sure a well written paragraph to that effect would have added to the literary value of this page . . . I would nevertheless like to thank Denise, for she has been a tremendous support, helping me to finish this work. Not only did she do a lot to keep the pressure up, to force me to focus on my work when I might have felt that 5 months would be plenty of time; she never failed me when I needed her moral support.

And finally, I’d like to thank Anna, as I’m sure I wouldn’t have been able to finish this work without her wonderfully prepared highly energetic culinary miracles!

— Manchester, September 2004

# Chapter 1

## Introduction

Image segmentation, the detection of objects in images, has long been a heavily researched topic of computer science. Image segmentation has many applications, ranging from simple monitoring applications (quality control in production processes, video surveillance applications in security systems, . . . ) to more complex and exciting areas like robotics (detection and manipulation or avoidance of objects) and artificial intelligence (object recognition, . . . )

Image segmentation is an important first step towards image interpretation and 3D image analysis. In the last decade and a half, there has been a trend towards a layered description of moving images. Such a description consists of three parts [1]:

1. A set of motion descriptors
2. Layers of support for those motion descriptors
3. A compact representation of the colour intensities for each layer

Such a decomposition of moving images in layers — sometimes called ‘*2.5D representation*’ — has many interesting applications. Its potential for very low bit-

rate yet high quality image encoding has been exposed in [22]; A static representation of the objects is transmitted, and each frame is encoded as the position and velocity of the objects that compose it.

Other, similar, applications include video editing, interesting special effects such as object removal and object introduction, but also plain video sequence generation. Moreover, if the object's position and velocity are known, it is trivial to interpolate its position and velocity between frames, opening up the possibility for frame rate conversion.

Also, the layered description of video sequences can be used as an effective way of storing video for automated video indexing software. Database systems allowing to query images by their content could use this representation to answer queries about both the static image characteristics (such as size, shape, texture, colour, transparency, . . . ) of the composing layers and their motion characteristics.

## 1.1 Approaches to Image Segmentation

### 1.1.1 Single Images

Traditional approaches to image segmentation have focused on single images. There are really four main approaches to the segmentation of single images: thresholding techniques, edge-based techniques, region based techniques and connectivity preserving relaxation techniques. More recently, the idea to use computational geometry to tackle segmentation was introduced [3].

Thresholding, the simplest and least robust technique simply classifies pixels based on their intensity. Pixels whose intensity lies in a certain range, are classified

as belonging to the object. This works if all pixels from the object are different from all those from the background. It is not very robust, since spatial information is ignored. Specifically, smooth transitions in intensity, such as with curved surfaces or pointed lighting are problematic.

Edge detection has long been the technique of choice. Basic edge detection is extremely simple; it's sufficient to threshold the first derivative of the image in the spatial domain. Such edge detection has limited usefulness, as it is very sensitive to noise in the image, and has problems handling curved surfaces.

Later algorithms, such as the Canny edge detector, improve matters somewhat by reducing the effects of noise drastically while still locating the edge very accurately. This makes edge detection a viable, indeed a very important part of machine vision. It still has its limitations, though. Most importantly, the edges of an object vary greatly depending on the background, lighting conditions, and position of the object.

Region-based techniques, such as “split and merge” techniques, and “region growing” techniques [10], as well as connectivity-preserving relaxation techniques do rely on spatial information and are therefore much more robust. They are strongly dependent on their initialisation parameters, however, and are prone to both over- and under-segmentation.

### 1.1.2 Video Sequences

The use of multiple images of the same scene rather than single images, such as we can find in video sequences, provides us with extra information to approach this problem[11]. Indeed, by assuming that objects remain rigid in a sequence, but move around in the image in a different manner than the background, connectivity

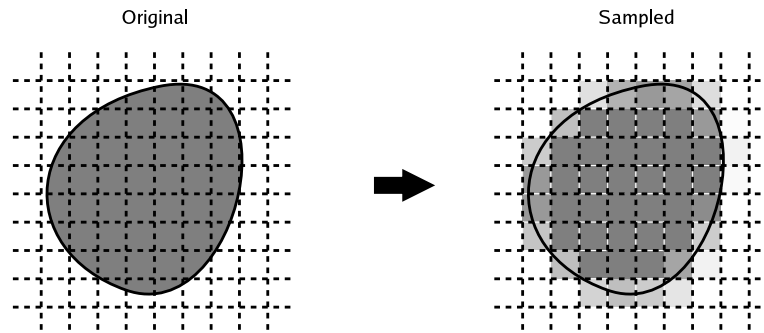


Figure 1.1: Aliasing of Border Pixels

between the features of an object can be inferred. Methods based on single images are limited to the detection of regions of similar intensity, while methods based on motion detect regions that move coherently, and are therefore probably connected. This is a more natural way of defining an object, since the object can have multiple features of different colours, but generally cannot fall to pieces.

The use of multiple images allows us to use machine learning techniques to assign classification probabilities to individual pixels. Different optimisation algorithms which have been designed to tackle the segmentation problem are described below.

## 1.2 Segmentation as an Optimisation

Although ad-hoc methods are quite successful in machine vision applications, a probabilistic approach is still justified — and ever more popular. Indeed, we want to assign each pixel to the object it belongs to. Unfortunately, since pixels are not infinitely small, the colour and the intensity of a pixel in a frame can be influenced by multiple objects. This is a form of aliasing, as is illustrated in figure 1.1: the sampling frequency is too low in the spatial domain, allowing high frequencies (rapid

changes in colour) in the image to be misrepresented. Fortunately for the observer, and unfortunately for the segmentation algorithms, the sheer size of the sensor works as a low-pass filter, therefore smoothing out the introduced artifacts.

It is therefore useful, and for some applications even crucial, to assign the pixels to the different objects with a certain probability. Ad-hoc methods allow us to do so as well, but a probabilistic approach provides a nice framework for it, and gives a sound theoretical basis on which to build.

In a more general setting, it is useful to note that the usefulness of a Bayesian framework is widely appreciated, as was brilliantly exposed by [12]. It is therefore little wonder that multiple maximum posteriori (MAP) approaches have been used to tackle the segmentation problem, such as [19, 21].

Jojic and Frey used a probabilistic approach to the segmentation problem [8, 6]. Their idea is that in a sequence of images (a video sequence), objects don't change shape very much; they just move around. Based on this assumption, they designed the system described in section 2.1. Their work inspired Williams and Titsias in their development of the GREEDY algorithm [23], as described in section 2.2. The greedy algorithm is a tractable method for segmentation of image sequences containing multiple objects using a probabilistically founded method.

This work is heavily based on the latter algorithm. In this thesis we present our own implementation (in `matlab`) of the GREEDY algorithm. We expose how we tried to make our implementation as efficient as possible, and how we tried to improve the robustness and convergence rate of the algorithm.

## 1.3 Structure of this thesis

**In the next chapter**, chapter 2, we provide an overview of those probabilistic approaches that have been used in the segmentation of image sequences. In particular, we describe the Transformed Mixtures of Gaussians and the GREEDY algorithm.

**In chapter 3** we describe how we implemented the algorithm in `matlab`, what difficulties we encountered and how we solved them.

**In chapter 4** we describe a few extensions and optimisations to the algorithm, showing how these can improve both its speed and robustness.

**In chapter 5** we explain how we acquired some new data sets, what results the algorithm gave with these, and how our optimisations improve the algorithm's performance.

**Finally, in chapter 6** we summarise our results, draw conclusions from the work, and expose a few area's in which future work would be worthwhile.

In the Appendices, we lay out the more technical parts:

**Appendix A** contains a short overview of the Expectation Maximisation algorithm. We give a brief description of the estimation and maximisation steps, and describe the merits of the sparse and incremental variants.

**Appendix B** exposes the details of the greedy algorithm. This appendix is heavily related to chapter 2, but contains the practical details of what is seen in that chapter.

# Chapter 2

## Previous Work

### 2.1 Transformed Mixtures of Gaussians

#### 2.1.1 Introduction

Transformed Mixtures of Gaussians [8] are probably best understood with a concrete example. Let's imagine we have a video sequence tracking a moving object, for example a close-up of a ball in mid-air. If the tracking were perfect, the ball would always be in the centre of the image. Imagine now, for argument's sake, that the ball was always showing the same face — rotating in a similar fashion as the moon does, never showing us its hidden face. In this ideal example, discriminating between the ball and the background would be trivial; the average of all images would show the ball, surrounded by a blurry, textureless area.

The variance on the pixels would give an indication of those pixels that are part of the ball: pixels with low variance would be part of the ball, while pixels with a high variance indicate the background. If we consider that each image is a  $P$ -sized

vector, where  $P = P_x \times P_y$ , the number of pixels in the image, then each image is a point in that  $P$ -dimensional space [2]. Again, in our ideal example, all points would be clustered around the mean image, and if we considered a diagonal covariance matrix, the variances on the diagonal would be high for pixels that are part of the background, and low for those depicting the object.

Let's now imagine that we have two sequences; one of a flying football, and one of a flying basketball, and that the frames are randomly intermixed. Both balls are easily discriminated due to their different texture. If we look at the position of these points in the  $P$ -dimensional space, we realise that they form two clusters; one for the football, and one for the basketball.

We could thus, in our ideal example, do some image segmentation and even object differentiation with a simple Gaussian Mixture Model. The problem with this is, of course, that in a real video sequence it is extremely improbable (i) to see exactly the same aspect of the object in all frames, and (ii) to have the object perfectly centred at all times.

The basic idea of Transformed Mixtures of Gaussians (TMG) is therefore to introduce a new hidden variable to our GMM: namely the transformation of the object in each frame. Each frame is transformed in order to centre the object, and the aspect of the object is simply computed as the mean of the cluster of transformed images. In this model, if the 2D sequence shows different aspects of the 3D object, each aspect will create a new cluster, and each cluster will be seen as a different object.

Obviously, since in a real video sequence the aspect of the object changes slowly, the position of the transformed images will be on a one-dimensional trajectory in the  $P$ -dimensional space. It is therefore hard to tell how many 'different' objects such

a tracking sequence contains and, in Jojic and Frey’s implementation, the number of objects is therefore a parameter.

### 2.1.2 A More Theoretically Founded Look at TMG

Segmenting an image sequence with TMG is thus an optimisation problem, where the mean and variance of each cluster are observed variables, while the assignment of each frame to the correct cluster and the transformation of each frame are hidden variables.

#### Transformed Gaussians

Any transformation can be implemented as a matrix multiplication. If we consider a frame  $\mathbf{x}$  with  $P = P_x P_y$  pixels as a column vector with  $P$  rows, the transformation matrix  $\mathbf{T}$  is a (sparse) square matrix with side  $P$ . We can thus write any transformation of an image  $\mathbf{x}$  as  $\mathbf{T}\mathbf{x}$ , while the inverse transformation is  $\mathbf{T}^\top \mathbf{x}$ , where “ $\top$ ” denotes the matrix transpose.

Let’s assume that the density of the observed image  $\mathbf{x}$  w.r.t. the non-transformed latent image  $\mathbf{z}$  and the transformation  $\mathbf{T}$  is given by

$$p(\mathbf{x}|\mathbf{T}, \mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{T}\mathbf{z}, \mathbf{\Psi}),$$

where  $\mathbf{\Psi}$  is a diagonal matrix of sensor noises. This allows the observed image to be slightly different from the transformed latent image. We assume that the choice of transformation  $\mathbf{T}$  is independent of the latent image  $\mathbf{z}$ , so  $p(\mathbf{z}|T) = p(\mathbf{z})$  and the joint distribution over the observed image  $\mathbf{x}$ , the transformation  $\mathbf{T}$ , and the latent

image  $\mathbf{z}$  is

$$p(\mathbf{x}, \mathbf{z}|\mathbf{T}) = p(\mathbf{x}|\mathbf{T}, \mathbf{z})p(\mathbf{z}|\mathbf{T}) \quad (2.1)$$

$$= \mathcal{N}(\mathbf{x}; \mathbf{T}\mathbf{z}, \mathbf{\Psi})p(\mathbf{z}) \quad (2.2)$$

As we discussed above, we assume the transformed images will be normally distributed around the latent image, or equivalently, that image has a normal density with respect to the transformed latent image:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \mathbf{\Phi}),$$

which, combined with eq. 2.2, results in the joint distribution:

$$p(\mathbf{x}, \mathbf{z}|\mathbf{T}) = \mathcal{N}(\mathbf{x}; \mathbf{T}\mathbf{z}, \mathbf{\Psi}) \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \mathbf{\Phi}),$$

from which we can derive the likelihood by integrating over  $\mathbf{z}$ :

$$\begin{aligned} p(\mathbf{x}|\mathbf{T}) &= \int_{\mathbf{z}} |d\mathbf{z}| p(\mathbf{x}, \mathbf{z}|\mathbf{T}) \\ &= \int_{\mathbf{z}} |d\mathbf{z}| \mathcal{N}(\mathbf{x}|\mathbf{T}\mathbf{z}, \mathbf{\Psi}) \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \mathbf{\Phi}) \end{aligned}$$

This integral is worked out in the appendix of [8], and its solution is:

$$p(\mathbf{x}|\mathbf{T}) = \mathcal{N}(\mathbf{x}; \mathbf{T}\boldsymbol{\mu}, \mathbf{T}\mathbf{\Phi}\mathbf{T}^\top + \mathbf{\Psi}).$$

The Covariance on the transformed latent image,  $\mathbf{\Psi}$ , models the sensor noise and may sometimes be assumed to be zero. As hinted in the introductory section,

the covariance on the distribution of the latent image,  $\Phi$  is often assumed to be a diagonal matrix for efficiency reasons, and when shown in raster–matrix format, the diagonal of the covariance indicates what pixels are part of the object (low variance), and what pixels are part of the background (high variance).

### Transformed Mixtures of Gaussians

If we think back to our example, we see that, in order to be able to discriminate frames with the football from frames with the basketball, we must now be able to differentiate between two clusters. We achieve this by adding a cluster index variable to model described above. The distribution over the observed and latent variables becomes

$$\begin{aligned} p(\mathbf{x}, \mathbf{z} | \mathbf{T}, c) &= p(\mathbf{x} | \mathbf{T}, \mathbf{z}) p(\mathbf{z} | c) \\ &= \mathcal{N}(\mathbf{x}; \mathbf{T}\mathbf{z}, \Psi) \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_c, \Phi_c), \end{aligned}$$

The *{cluster, transformation}*–likelihood then becomes:

$$p(\mathbf{x} | \mathbf{T}, c) = \mathcal{N}(\mathbf{x}; \mathbf{T}\boldsymbol{\mu}_c, \mathbf{T}\Phi_c\mathbf{T}^\top + \Psi).$$

We assume that  $\mathcal{T}$ , the set of possible transformations is finite. The probability of the observed frame  $\mathbf{x}$  can then be computed with a discrete summation:

$$p(x) = \sum_{c=1}^C \sum_{\mathbf{T} \in \mathcal{T}} \mathcal{N}(\mathbf{x}; \mathbf{T}\boldsymbol{\mu}_c, \mathbf{T}\Phi_c\mathbf{T}^\top + \Psi)$$

This can be optimised by the EM algorithm which is described in [8]. Trans-

formed Mixtures of Gaussians therefore try to optimise a Gaussian mixture model of transformed images, where we have two hidden variables (viz. which cluster a data point belongs to, and how it should be transformed) and two observed variables (mean and covariance).

### 2.1.3 Discussion

This method has two main limitations. First, the background needs to be changing a lot in order for the object to be segmented correctly. Indeed, since the diagonal in the covariance matrix indicates whether pixels belong to the object, if the background is too static, its pixels will be seen as part of the object. If the background does not contain enough structure, the colour of any pixel in the transformed background is independent on the transformation, resulting in a cluster in our  $P$ -dimensional space.

Second, this method only allows us to detect a single object in the frame. Multiple objects can be detected in a sequence, but only in successive images. If we think back about our example for a second, and imagine our two balls crossed in mid-air. The frames which contain both balls would be transformed such that the ball that generates the maximal likelihood be centred on the latent image, and the image is assigned to the cluster accordingly. The other ball becomes part of the background; it is just noise — and because the selected transformation is meaningless for the second object, the data point probably won't even be close to the mean of the other cluster.

Both limitations limit the usefulness of TMG in many practical applications, even though TMG are perfect in, for example, the optimisation of electron microscopy

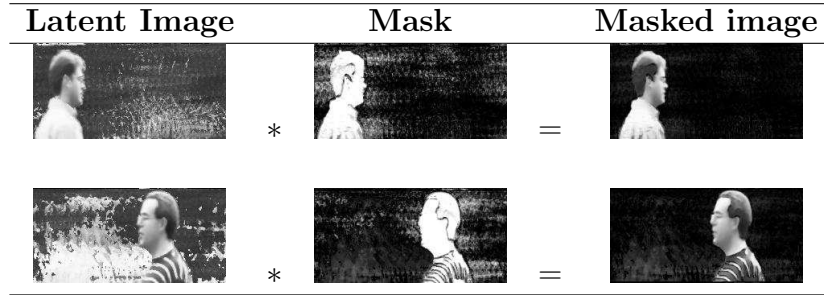


Figure 2.1: Masked latent object variables

images [8]. These two limitations are addressed in the GREEDY algorithm, which is described in the next section.

## 2.2 The greedy Algorithm

### 2.2.1 Introduction

The problem with the model described above is that Transformed Mixtures of Gaussians consider the whole frame as a single data point. Therefore, its position in the high-dimensional space can only be defined by a single object, and if two objects are present in the frame, the most likely transformation will be defined by one of them, while the other object will constitute noise.

The improvement that the GREEDY algorithm brings to the Transformed Mixtures of Gaussians lies in the fact that it introduces a new hidden variable to the optimisation, namely the probability for each pixel that the pixel belongs to an object. These probabilities are computed using robust statistics in order to allow for occlusions, as we'll explain below. The result is stored in a vector of the same size as the latent image, and can be seen as a mask we apply to the latent image, as illustrated in figure 2.1. The masked frame is then transformed, and a Gaussian

mixture model of the masked objects is optimised.

It is important to consider that this introduces an extra complexity to the TMG algorithm, which was already rather slow notwithstanding recent optimisations [7, 15]. The basic problem is that when we consider sequences with multiple objects, the number of instantiation parameters grows exponentially. Indeed, if we consider  $L$  objects in a sequence, and each object has  $J$  instantiation parameters, the total number of instantiations we need to consider is  $J^L$ . This becomes quickly intractable, even for small numbers of objects  $L$ .

The GREEDY algorithm avoids this combinatorial explosion by observing a simple fact: objects occlude each other.<sup>1</sup> Therefore a pixel whose intensity has been defined by an object is not affected by any of the other objects. This means that we can optimise the instantiation parameters of each object one at the time, effectively reducing the complexity from  $O(J^L)$  to  $O(LJ)$ .

This means that the GREEDY algorithm optimises one observed variable, the aspect of each object, and two hidden variables; the transformation of the objects in each frame and the classification of each pixel of that frame as to which object it belongs to. As with Transformed Mixtures of Gaussians, the transformation can be any affine transformation, but for efficiency reasons and due to time constraints on this project, we only looked at translations for now.

Below is a general description of the algorithm, where we also aim to point out its strengths and weaknesses. For a discussion of how the weaknesses might be addressed without compromising the strengths, please refer to the next chapters.

---

<sup>1</sup>Many combinations of occlusion are possible in a 3D world; an object can occlude another object, while also being occluded itself by that object, an object (as detected based on its motion) can occlude itself, . . . This does not affect the fact that a pixel is, in general, only defined by a single object

For a more technical description of the algorithm, and the mathematical description of the updates of the EM algorithm, please refer to Appendix B.

### **2.2.2 The Algorithm**

The algorithm first learns the aspect of the background, using robust statistics. The robust statistics allow us to mathematically account for the foreground objects in each frame, and as such to learn the background before we know the foreground objects. It then learns the aspect of each foreground object in turn, thereby avoiding the combinatorial explosion of instantiation parameters. It optimises the aspect and shape (mask) of the object by learning iteratively from all the frames, before moving on to the next object.

#### **Learning the Background**

We start with the background. The latent background is considered as a large image over which the camera panned to create the frames. Notice that, since we limit ourselves to translations in this simple form, deformation due to parallax is not taken into account, even though a moving background is almost always due to camera rotation. This makes our implementation mostly useful for static backgrounds — it accounts for sensor noise and jittery camera motion, but if the camera moves more than that, the background is simply seen as noise.

We reconstruct a frame by looking for the transformation of the latent background image that explains it best. Since the background is obstructed by the objects, we need to introduce a term that acknowledges this. We therefore model the probability that a pixel of the frame corresponds to that same pixel of the

transformed background using the following robust statistics:

$$p_b(\mathbf{x}_p; (\mathbf{T}_{j_b} \mathbf{b}_p)_p) = \alpha_b \mathcal{N}(\mathbf{x}_p; (\mathbf{T}_{j_b} \mathbf{b})_p, \sigma_b^2) + (1 - \alpha_b) U(\mathbf{x}_p) \quad (2.3)$$

Note that, as opposed to TMG, in this model we assume the variance to be the same for all pixels. This allows us to update the variance much more efficiently. As we'll see below, the usage of a mask to discriminate between pixels that are part of the object and pixels that aren't, allows us to do the same for the foreground objects. In the previous model, the variance of each pixel was used to discriminate between pixels that were part of the object (pixels with low variance), and pixels that were part of the background (pixels with high variance). In the current model, a separate mask is kept to indicate which pixels are part of the object, and the variance on all of those pixels is assumed to be identical.

When restricting ourselves to the background, an image  $\mathbf{x}$  is generated by instantiating the transformation variable  $j_b$  and generating  $\mathbf{x}$  accordingly:  $p(\mathbf{x}|j_b) = \prod_P p_b(\mathbf{x}_p; (\mathbf{T}_{j_b} \mathbf{b})_p)$ . Assuming an uniform prior on the transformation  $j_b$ , the log likelihood of the training images is  $L_b = \sum_N \log \sum_{j_b} P_{j_b} p(\mathbf{x}^n | j_b)$ , which can be maximised with the EM algorithm, as described in Appendix B.1.

### Learning one object

Let's now consider the case of a single object;  $L = 1$ . The object is represented internally as an image  $\mathbf{f}$ , of the same size as the frames of the observed sequence. However, since it is not the case that each pixel of the latent image is really part of the object, we also keep a corresponding mask,  $\boldsymbol{\pi}$ . The mask is a matrix of the same dimensions as the latent object, and probabilistically masks out the irrelevant

pixels:  $\pi_p \approx 1$  if  $\mathbf{f}_p$  is part of the object, or  $\pi_p \approx 0$  otherwise.

We can then express the likelihood of an image as:

$$p(\mathbf{x}|j_b, j_1) = \prod_{p=1}^P ((\mathbf{T}_{j_1} \boldsymbol{\pi}_1)_p p_{f_1}(\mathbf{x}_p; (\mathbf{T}_{j_1} \mathbf{f}_1)_p) + (1 - (\mathbf{T}_{j_1} \boldsymbol{\pi}_1)_p) p_b(\mathbf{x}_p; (\mathbf{T}_{j_b} \mathbf{b})_p)), \quad (2.4)$$

where  $p_{f_1}(\mathbf{x}_p; (\mathbf{T}_{j_1} \mathbf{f}_1)_p)$  is computed similarly to  $p_b$  using robust statistics. Indeed, as we'll see below, the foreground objects can also be occluded, which is allowed for by saying that, for object  $l$ :

$$p_{f_l}(\mathbf{x}_p; (\mathbf{T}_{j_f} \mathbf{f}_l)_p) = \alpha_f \mathcal{N}(\mathbf{x}_p; (\mathbf{T}_{j_f} \mathbf{f}_l)_p, \sigma_{f_l}^2) + (1 - \alpha_f) U(\mathbf{x}_p)$$

The resulting log likelihood over all training images is

$$L_1 = \sum_{n=1}^N \log \sum_{j_1, j_b} P_{j_1} P_{j_b} p(\mathbf{x}^n | j_1, j_b), \quad (2.5)$$

which could also be maximised with the EM algorithm. This would be quite demanding, however, since it involves a search over  $J_1 J_b$  possibilities, if the GREEDY algorithm didn't reduce this complexity to  $J_1$  by using a constrained, sparse, EM algorithm [18]. We explore the general form of the sparse variant of the EM algorithm in more detail in Appendix A.3. Using the fact that the background is already known, the distribution  $Q^n(j_1, j_b) = Q^n(j_1 | j_b) Q^n(j_b)$  is introduced, and a lower bound on  $L_1$  is found:

$$F_1 = \sum_{n=1}^N \sum_{j_b=1}^{J_b} \sum_{j_1=1}^{J_1} Q^n(j_1 | j_b) Q^n(j_b) [\log (P_{j_1} P_{j_b} \prod_P p(\mathbf{x}^n | j_b, j_1)) - \log(Q^n(j_1 | j_b) Q^n(j_b))]$$

This lower bound is maximised by choosing  $Q^n(j_b, j_1)$  to be the posterior  $P(j_b, j_1 | \mathbf{x}^n)$  for all images  $\mathbf{x}^n$ . We use the posterior probability of the background,  $P(j_b | \mathbf{x}^n)$  to find the best transformation that explains  $\mathbf{x}^n$ ,  $j_b^n$ , and approximate  $Q^n(j_b)$  as:

$$Q^n(j_b) = \begin{cases} 1 & \text{if } j_b = j_b^n \\ 0 & \text{otherwise} \end{cases}$$

This may seem like a coarse approximation, but in practice the value of  $P(j_b)$  is always close to 1 when  $j_b = j_b^n$ , and this reduces our lower bound to the expression:

$$F_1 = \sum_{n=1}^N \sum_{j_f=1}^{J_f} Q^n(j_1) (\sum_P \log(p(\mathbf{x}_p^n | j_b^n, j_1)) - \log(Q^n(j_1))) + \text{const},$$

which can be optimised with the EM algorithm in a much more tractable way than the likelihood given in eq. 2.5.

### Learning multiple objects

Learning the following objects is extremely similar to the learning of the first object, but with the additional optimisation that the pixels which have been explained by previous objects are removed from consideration.

We define the probability with which image  $\mathbf{x}$  is explained by object  $l$  as

$$P(\mathbf{x} | j_b, j_1, \dots, j_l) = \prod_{p=1}^P p(\mathbf{x}_p | j_b, j_1, \dots, j_l),$$

where  $p(\mathbf{x}_p | j_b, j_1, \dots, j_L)$  is defined similarly as in equation 2.4 as:

$$p(\mathbf{x}_p | j_b, j_1, \dots, j_L) = \sum_{l=1}^L \left( \prod_{k=1}^{l-1} (\mathbf{1} - \mathbf{T}_{j_k} \boldsymbol{\pi}_k)_p (\mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_{f_l}(\mathbf{x}_p; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) \right. \\ \left. + \prod_{k=l}^L (\mathbf{1} - \mathbf{T}_{j_k})_p p_b(\mathbf{x}_p; (\mathbf{T}_{j_b} \mathbf{b})_p) \right)$$

Similarly to the case of a single object, we maximise the lower bound on the log likelihood

$$F_l = \sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) \left( \sum_{p=1}^P \log p(\mathbf{x}_p^n | j_b^n, j_1^n, \dots, j_L^n) - \log Q^n(j_l) \right) + \text{const} \quad (2.6)$$

which can be tractably optimised over  $Q^n(j_l)$  and  $\mathbf{f}_l, \boldsymbol{\pi}_l, \sigma_l^2$ , but which does not ensure that objects we find are different. We can therefore make the algorithm much more efficient by removing the pixels that have been explained from consideration. We want to remove those pixels that correspond to the accordingly transformed known latent objects, and that are not occluded. For any object  $l$ , we consider the vector  $\boldsymbol{\rho}_l^n = (\mathbf{T}_{j_l^n} \boldsymbol{\pi}_l) * \mathbf{r}^{j_l^n}$ , where

$$r_p^{j_l^n} = \frac{\alpha_f \mathcal{N}(\mathbf{x}_p^n; (\mathbf{T}_{j_l^n} \mathbf{f}_l)_p, \sigma_l^2)}{\alpha_f \mathcal{N}(\mathbf{x}_p^n; (\mathbf{T}_{j_l^n} \mathbf{f}_l)_p, \sigma_l^2) + (1 - \alpha_f) U(\mathbf{x}_p^n)}$$

This vector  $\boldsymbol{\rho}_l^n$  will contain values close to 1 for pixels of image  $\mathbf{x}^n$  which correspond to object  $l$ , and close to 0 otherwise. In order to remove all pixels that have already been explained from consideration we introduce a vector  $\mathbf{z}_l^n$ , where  $\mathbf{z}_1^n = \mathbf{1}$ , and  $\mathbf{z}_l^n = \mathbf{z}_{l-1}^n * (\mathbf{1} - \boldsymbol{\rho}_l^n)$  for  $2 \leq l \leq L$ . This allows us to write the lower bound on

introduced in equation 2.6 as:

$$F_l = \sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) \left( \sum_{p=1}^P (\mathbf{z}_{l-1}^n)_p \log [(\mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_{f_l}(\mathbf{x}_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) + (\mathbf{1} - \mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_b(\mathbf{x}_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p)] - \log Q^n(j_l) \right) \quad (2.7)$$

It is noteworthy that the GREEDY algorithm treats background and foreground objects differently. Indeed, foreground objects are removed from consideration once they have been learnt, but the background is not. This is because experiments with a version where the background pixels were removed from consideration turned out to give worse results: too many pixels were considered good background candidates and were therefore wrongly removed from consideration. Experiments have shown that this resulted in noisy estimates for the foreground objects[23].

The details of the EM algorithm used to optimise the lower functions can be found in Appendix B.3. In the next chapter, we provide a description of our implementation.

# Chapter 3

## Implementation

### 3.1 Transformations

An important point in both TMG and the greedy algorithm is that the transformation matrices do not need to be explicitly instantiated. We can implement the transformations with more efficient code, while keeping the same functionality and upholding the validity of the algorithms.

It is thus useful in `matlab` to use the built-in image transformation functions `maketform` and `imtransform`, as these allow us to reduce the memory and CPU requirements of the algorithm. `maketform` creates a structure representing all aspects of the transformation. This structure is then passed on to `imtransform`, which applies the transformation to an image. The intermediate structure should thus be a valid candidate for caching, as we could reuse the same transformation description on the different images. In practice, this did not seem to yield much speed improvements, however, and this caching has been dropped in favour of caching the resulting transformed image as explained below.

Moreover, since we only experimented with translations in our implementation, we have an opportunity for an even better optimisation. Translations can be implemented in a much more efficient way in `matlab`; viz. as matrix operations. Indeed, although the `maketform-imtransform` pair is extremely powerful and flexible, it is rather slow on pure translations. We have therefore chosen to drop these in favour of basic matrix splitting and concatenation operations. These operations are extremely efficient in `matlab`, as matrix manipulations are the traditional strength of the language.

### 3.1.1 Cached transformations

In order to increase the speed of the algorithm even further, a practical trick was used. As we can see from the description of the algorithm, the latent foreground images and their corresponding mask are transformed quite often. It isn't necessary to calculate the transformation each time; the transformed image can be kept in memory.

Unfortunately, if we use the lazy updates explained in section 4.3, it is hard to know beforehand which transformations we'll need later on. To alleviate this problem, we wrote a function that uses a cache of those transformations which are already known. The function returns cached values if they are known, and fills in the cache when they aren't. This makes the use of a cache transparent to the rest of the algorithm. The cache is implemented as a structure of two matrices. Both matrices grow dynamically as new transformations are cached.

The first matrix is a column matrix that contains the transformed data. This matrix is grown dynamically in order to keep memory requirements realistic. The

other matrix keeps  $(j, index)$  pairs, keeping a log of the column in the cache matrix that contains transformation  $j$ . The latter matrix is kept sorted in ascending order of  $j$ , in order to allow us to use a binary search to access the cached results in  $O(\log n)$  time[16].

## 3.2 Machine Precision

One of the issues that arose when applying probabilistic formulae to images was that the resulting likelihoods were below machine precision. Indeed, the probability of the data under the model is the product of the probability that each pixel is explained by that model,  $Pr(I|M) = \prod_{p=1}^P Pr(I_p|M)$ . Therefore, even for relatively low-resolution images,  $Pr(I|M)$  becomes too small to be represented.

This problem would be easily solved by using the log-likelihood instead of the likelihood — if we didn't need to compute any sums of probabilities. Unfortunately, it is very often necessary to compute normalised probabilities:  $P_i / \sum_{n=1}^N P_n$ , which we can't calculate by using the log-likelihood.

An easy solution to this problem is to compute the normalised likelihood as follows:

$$\frac{P_i}{\sum_N P_n} = \frac{\frac{P_i}{\max(P)}}{\sum_N \frac{P_n}{\max(P)}} \quad (3.1)$$

$$= \frac{\exp(\log(P_i) - \max(\log(P)))}{\sum_N \exp(\log(P_n) - \max(\log(P)))} \quad (3.2)$$

since the logarithm is monotonic and therefore  $\log(\max(x)) \equiv \max(\log(x))$ .

Now as the relevant probabilities are those which are close to the maximum, this effectively solves our problem. The normalisation of the maximum probability  $P_{max}$  boils down to the calculation of

$$\frac{1}{1 + \sum_{n \neq max} \frac{P_n}{P_{max}}}$$

and all relevant probabilities are well inside the machine's precision limits.

### 3.3 Parallel Execution

It is clear from the previous discussion that the performance of the algorithm is still an issue. Depending on the resolution of the images and the use of colour, it may take multiple hours or even days for the the algorithm to converge. In order to make our investigations possible, it was therefore necessary introduce some form of parallelism in our implementation. It is clear, even from a summary inspection, that the algorithm would be easily parallelisable. Indeed, the computation of  $Q^n(j_l)$  for the foreground objects and  $P(j_b)$  for the background objects is the most time-consuming part and must be done for all possible transformations, for all images independently. They can therefore be executed in parallel. As a side note, it is interesting to remark that it's even possible to let multiple processors work in parallel on the computation of a single value of  $Q^n(j_l)$  by splitting the image in parts. Each pixel can be treated independently; this is an "embarrassingly parallel" problem.

Unfortunately, `matlab` doesn't provide a very friendly environment for parallel execution. We therefore introduced some parallelism at another level. We kept a purely sequential implementation of the greedy algorithm, but made it very config-

urable. We could then run multiple instances of the algorithm in parallel on different machines, on different data sets, with different parameters. We achieved this easily by making use of the local organisation of the network, where all workstations can be running Linux and a consistent file system is provided through a Network File System (NFS).

### 3.3.1 Details of the Matlab Implementation

In order to make this manageable, it is of course necessary to be able to run the algorithm without any human oversight. We therefore wrote a shell function which reads all of its configuration parameters from a `matlab` file, and passes the correct parameters on to the greedy function. This function is invoked by the small job management system, fully written in shell scripts, that is described below.

The greedy algorithm can thus run without any user input. Since the results cannot be directly shown on screen, all results are output to file. One of the parameters to the greedy algorithm is the directory it should use for its output. This allows multiple `matlab` sessions to run concurrently, using the same code, running in the same directory (via NFS), on multiple machines.

The latent images and masks are written in JPEG format instead of `matlab` file format, in order to limit the disk space requirements. The likelihood is saved in a `matlab` file, in order to enable subsequent generation of graphics. The intermediate values of the latent variables, as well as a few bookkeeping variables, are stored to file after each iteration of the EM algorithm. This allows us to interrupt a run of the algorithm and to restart it later on without losing too much work.

### 3.3.2 Job Management System

The setting we just described allows us to run multiple `matlab` sessions on separate machines, and has the fringe benefit of putting idle workstations to good use. This is laudable, but has a downside; the primary function of the workstations is not to run as nodes in a parallel environment, but to be used as stand-alone machines. Although it is possible to check that the workstation is idle before starting a job, it is impossible to predict when somebody will log in. We should have a system that allows us to run our batch job as long as the workstation is free, and kill our job as soon as anyone logs in.

We solved this with a shell script, which was baptised with the extremely original name of `jobdaemon`. This little application runs in background on the workstations, and wakes approximately every second. When it wakes, it checks that the system load is low and that nobody is logged into the system. If the coast is clear, it polls a directory for new jobs. When a new job appears in the directory, the workstation attempts to ‘lock’ the job, in order to ensure that only one machine runs the job. The first machine to lock the job starts it in background and starts a monitoring thread.

The monitoring thread does not check for system load anymore — since its own job takes up 100% of the CPU time — but it checks for log-ins. If anyone but the owner of the job logs in to the machine, the job is immediately killed and unlocked. This frees up the workstation, and makes the job free for any other workstation to grab. If any other workstation is free, it will immediately try to lock the job, and resume the execution. Similarly, if the machine is rebooted, the last action of the `jobdaemon` will be to free the job.

It is important to note here that this system does not save the state of the application when it kills it. It is therefore crucial to make the application save its state regularly in order to be able to recover from sudden death without need for a complete recalculation of the previously obtained results.

### 3.3.3 Helper Programs

As a last note on this section, we wrote a few helper programs that allow us to monitor the status of the jobs and nodes, and to facilitate the submission of new jobs. We have already seen `jobdaemon`, which starts and monitors the jobs on the nodes. The other scripts are:

**runbatch:** This program is run on a central, controlling node. It starts the `jobdaemon` on the worker nodes, and monitors that the node is alive. If the node is dead, any job it might have been running is freed.

**jobstatus:** is a simple tool giving a listing of all the jobs in the system, together with their status: whether the job is running, queued and waiting for a free node, or done.

**schedulejob:** is a helper tool making it easier to add a `matlab` job to the queue. Indeed, the job system simply executes the jobs in its queue, and is completely unaware of the job's need for other programs — such as `matlab`. `Schedulejob` therefore creates a shell script that invokes `matlab` in the correct directory, with the right parameters for the execution of our `matlab` job.

# Chapter 4

## Optimisations

In this chapter we present some improvements that can be made to the GREEDY algorithm. This does not concern any changes to the model or the algorithm used to optimise it, but rather additions that improve the general quality of the results and the speed with which the algorithm converges.

### 4.1 Transformations

The translation of the latent image to match the location of the object in the frame can be implemented in different ways. The original algorithm is ‘lossless’: the inverse transformation of the transformed image yields the original image, without any error. This is achieved by wrapping the image around, as is shown in fig. 4.1

It is however possible to use ‘lossy’ translations, where we transform the image in a straightforward way, and discard the areas that fall outside the resulting picture. Both methods are depicted in figure 4.1. Using lossy translations makes sense, because in a real video sequence, the objects don’t wrap around the edges. As

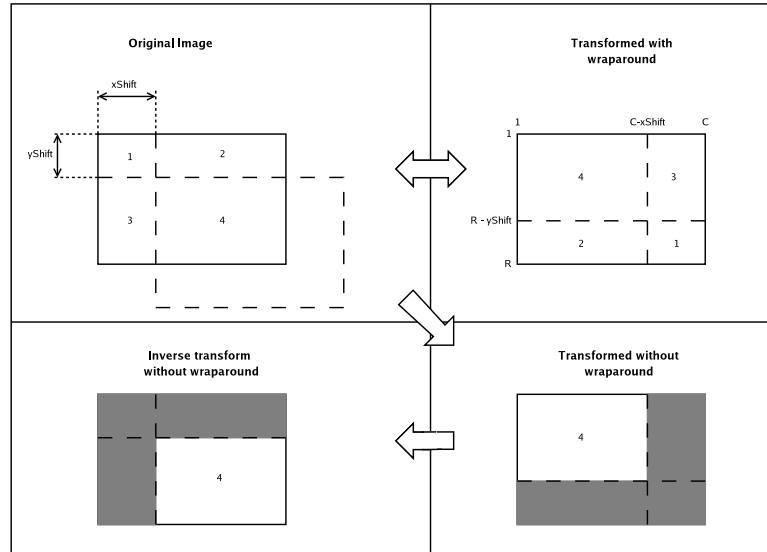


Figure 4.1: Transformation of the latent objects

a consequence, all pixels that have been wrapped generate probabilities which are known to be irrelevant, therefore distracting the algorithm from the really interesting regions.

Unfortunately, simply discarding the parts of the image that end up outside the view area is not good enough, as we can see from equation 2.7 that the likelihood depends on the number of pixels — and therefore on the considered area. It is therefore crucial that the relevant area of the transformed image always be of the same size, which is not the case if we simply discard the parts that fall outside the transformed image. Moreover, the use of wraparound does have another advantage. Since no information is lost in the translation, shifting the image over the whole width of the image in the X direction, or over the whole height of the image in the Y direction, results in a null operation. Any pixel  $(x, y)$  can be translated to any coordinates  $(x', y')$  by some shift in between. When using lossy translation, however, we need to consider both negative and positive shifts to cover all possible



Figure 4.2: Translation of latent objects of limited size

translations. This effectively doubles the number of translations we need to consider.

### 4.1.1 Objects of Limited Size

It is interesting to realise that objects are generally smaller than the frames. If we should know the maximal size of an object, it would be possible to limit the size of the latent object. This would also reduce the number of valid transformations and reduce the effects of transformed noise, while still ensuring that the area of the transformed image is the same for every transformation. Figure 4.2 illustrates this.

Tests with a manually chosen latent object size have shown that when the object size is reasonable, the algorithm converges much faster. Not only are fewer iterations needed, each iteration is also performed faster because of the limited number of pixels that need to be considered.

It is important to point out that this type of lossy translations doesn't affect the validity of the algorithm. Indeed, since both latent object and latent object mask are translated in the same way, those pixels whose intensity is unknown have a zero mask value. This means that they are never considered, not when computing the likelihood of the translation, nor when updating the latent image.

### Discussion

There are two main drawbacks to this method. First, it is difficult for the algorithm to know beforehand what the size of each object will be. The algorithm will only know the size of the objects after convergence, which is of course a bit late.

Second, it is hard to initialise the latent variables with limited object size. In the case of a full size latent image, as is the case of the original GREEDY algorithm, we are sure that the object is in the image. We can therefore initialise the latent variables with a random frame, convergence mostly removes irrelevant pixels. With latent objects of smaller dimensions, the problem is harder, since we don't know anything about the location of the objects yet. The worst case, is to initialise the latent object with a part of the image such that a part of the object is in the initial latent variable. The algorithm will then converge to that partial object, and the rest of the object remains outside the latent image.

The best solution in an interactive system would probably be to ask the user to draw a box around objects he wants to segment. This would solve both the size and initialisation problem, and is probably a very intuitive thing to do. This is only a solution in interactive applications, however, and may be seen as a superfluous step.

## 4.2 Incremental resolution

Another trick we tried to reduce the runtime of our algorithm was to reduce the search space by reducing the resolution of the images. When our algorithm is not yet converging, a huge number of transformations need be evaluated, and those evaluations take a long time. By reducing the resolution of the images at that stage, the number of transformations that need be evaluated is reduced, since it isn't

necessary to try translations of less than one pixel. Moreover, the sheer reduction in matrix size also affects the computation time quadratically.

For  $n$  resolution steps, the adapted algorithm is as follows:

**Step 1:** reduce the resolution of the observed images by sub-sampling it  $n - 1$  times

**Step 2:** Initialise the latent images with the reduced resolution observed images.

**Step 3:** Apply the greedy algorithm to the reduced-resolution frames.

**Step 4:** Decrease  $n$  by one. Increase the resolution of the latent images and masks by interpolation, and re-initialise the input images by sub-sampling the observed images  $n - 1$  times.

**Step 5:** Loop from step 3 until  $n = 1$

Once the lower-resolution object images have converged, we increase their resolution by simple interpolation and use these as initialisation for the EM algorithm with full resolution images. The next EM step starts with a very good initialisation, and converges very fast.

This new algorithm seems to work quite well for objects that contain little fine structure, or when the objects in the sequence are very dissimilar. When the objects are too similar (and their dissimilarities are lost in the resolution reduction), the first EM step may fail to converge correctly. The initialisation to the next step is then very bad, which often causes the following EM step to fail. This may cause a snowball effect, which the algorithms can recover from when the resolution gets good enough, but which slows the algorithm down enormously instead of speeding it up.

### 4.2.1 Chequerboard sampling

$P(j_b|\mathbf{x}^n)$  and  $Q(j_l|\mathbf{x}^n)$  are the normalised probabilities of the transformation of the latent variable over the data. They are computed as described in Appendix B, equations B.1 and B.2 respectively. Let's have a closer look at  $Q^n(j_l)$ , since it is computed for each foreground object, and there are often multiple foreground objects while there is, by definition, only one background. Improving the speed of computation of  $Q^n(j_l)$  therefore is more relevant than that of  $P(j_b|\mathbf{x}^n)$ . Here's how we calculate  $Q^n(j_l)$ :

$$Q^n(j_l) \propto \exp \left[ \sum_{p=1}^P (\mathbf{z}_{l-1}^n)_p \log \left( (\mathbf{T}_{j_l} \pi_l)_p p_{f_l}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) \right. \right. \\ \left. \left. + (\mathbf{1} - \mathbf{T}_{j_l} \pi_l)_p p_b(x_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p) \right) \right]$$

As we can see, we need to compute the sum over all pixels of the probability that each pixel of the frame corresponds to that pixel of the transformed latent image. This result is then normalised over all transformations, which discards any impact of the size of the images. So the only thing that affects the value of  $Q^n(j_l)$ , is whether the latent object is transformed to the right position in the observed image.

We can see that it is not compulsory to compute the sum over all pixels; if we consider a sufficient amount of relevant pixels — i.e. pixels belonging to the object — we can determine whether the transformation was correct or not. The computation of the normalised  $Q^n(j_l)$  over a carefully selected range of pixels gives the same results as the normalised  $Q^n(j_l)$  over all pixels, but at a lower computational cost.

We have investigated multiple different methods to reduce the dimensionality of the images. The method that seemed to give the best results, as defined by the least

expensive computationally and most accurate seemed to be to take bands out of the image, both in the X and Y direction.

### 4.3 Lazy updates

Let's look back for a second at the lower bound on the likelihood which we introduced in section 2.2.2 as equation 2.7. We reproduce the equation here for convenience:

$$F_l = \sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) \left( \sum_{p=1}^P (\mathbf{z}_{l-1}^n)_p \log[(\mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_{f_l}(\mathbf{x}_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) + (\mathbf{1} - \mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_b(\mathbf{x}_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p)] - \log Q^n(j_l) \right)$$

The likelihood is only affected by those transformations of the latent images for which  $Q^n(j_l) \neq 0$ . In practice, there is a limited number of transformations for which that is the case, viz the transformations that approximately centre the latent object on the position of the object in the frame. In all other cases,  $Q^n(j_l) \approx 0$ , and those transformations don't affect the likelihood. In the M step of the greedy algorithm, the updates of the latent foreground images  $\mathbf{f}_l$  and masks  $\boldsymbol{\pi}_l$  are therefore dependent on the probability of the hidden variables  $Q^n(j_l)$ , the probability that the latent image  $\mathbf{f}_l$  must be transformed by transformation  $j_l$  to generate the  $n$ th image,  $\mathbf{x}^n$ .

It is therefore pointless to compute all terms for which the transformation is very improbable, as these do not affect the result. The "lazy" algorithm therefore only computes the term  $\sum_{p=1}^P (\mathbf{z}_{l-1}^n)_p \log[(\mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_{f_l}(\mathbf{x}_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) + (\mathbf{1} - \mathbf{T}_{j_l} \boldsymbol{\pi}_l)_p p_b(\mathbf{x}_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p)] - \log Q^n(j_l)$  when  $Q^n(j_l)$  is big enough to make this affect the likelihood.

This “lazy” implementation of the algorithm reduces the computational cost of the algorithm dramatically. Indeed, if we take for example a 200 by 100 pixel image, such an image allows for 20 000 possible translations, all of which have a normalised probability of approximately 0 except for one. The computational cost of the M-step is reduced by 5 orders of magnitude. The speed of convergence is not affected by this either, as shown in chapter 5.

Of course, although this is a worthwhile improvement, we have to remember that the M-step is not the only computationally expensive step in the algorithm; the computation of  $Q^n(J_l)$  itself is rather expensive. We address this below.

## 4.4 Sparse and Incremental EM

### 4.4.1 Sparse EM

The traditional sparse EM algorithm [18] assumes that the result of an iteration of the E-step will yield very similar results to the results of the previous iteration. This is mostly the case in our algorithm, but there are exceptions to this: if two objects in the sequence are somewhat similar, the first estimations for those objects will not be very good. The most probable transformation will therefore oscillate between both objects, and a traditional sparse EM reduces the convergence speed of the algorithm considerably.

When segmenting images from a video sequence, however, it is improbable that an object has moved very much from one image to the next. It is therefore possible to speed up the algorithm by constraining the transformations we consider for each

image. Indeed, since

$$\arg \max_{j_l} (Q^n(j_l)) \approx \arg \max_{j_l} (Q^{n+1}(j_l)),$$

we may choose to evaluate  $Q^n(j_l^n)$  only for values of  $j_l^n$  which are in the “neighbourhood” of  $j_l^{n-1}$ , the value of  $j_l$  that yielded  $\max(Q^{n-1}(j_l))$ . By “neighbourhood of  $j_l$ ”, we refer to the set of those transformations that yield spatially similar results to  $\mathbf{T}_{j_l}$ .

A small improvement to this, is that the first derivative of the location — the speed — of the object gives us more information than the mere location. We can therefore restrict the set of considered transformations even more. Indeed, since:

$$\begin{aligned} \arg \max_{j_l} (Q^n(j_l)) - \arg \max_{j_l} (Q^{n-1}(j_l)) \\ \approx \arg \max_{j_l} (Q^{n+1}(j_l)) - \arg \max_{j_l} (Q^n(j_l)) \end{aligned}$$

we may reduce our search space for frame  $n$  to those values of  $j_l$  which are in the neighbourhood of<sup>1</sup>  $2j_l^{(n-1)} - j_l^{(n-2)}$ . This allows us to search a smaller neighbourhood without degrading robustness of the algorithm.

## Discussion

Although this has a spectacular impact on the speed of the algorithm, it is important to consider that occlusions can completely break this system. Let’s imagine for a moment that the first object the algorithm tracks is not the foremost object, but

---

<sup>1</sup>In reality, we need to separate the speed in the X and Y directions of the object, and compute the value of  $j_l$  accordingly. This depends on the specifics of the indexing of the transformation matrices, however, and is therefore of little general interest.

rather the largest. This happens often enough, and is normally accounted for by the robust statistics we use to compute our probabilities. We track our object, and when our object starts disappearing, we find the most likely transformation of our latent image in the neighbourhood of our last transformation. If the object stays occluded for a few frames, the real object will have moved on quite a bit while our most likely transform has done a little “Brownian motion” around the last known location of the object.

When the real object reappears, it is often outside the range of considered transformations, and the algorithm loses track of it. This results in failure to converge, albeit very *fast* failure to converge. We can avoid this by making the range of transformations we evaluate big enough, but of course this reduces the usefulness of the improvement.

It makes more sense to reduce the range of transformations we evaluate once the algorithm starts to converge. We can then constrain the search to the transformations around the transformation that generated the maximal  $Q^n(j_i)$  at the previous iteration. This is a real form of sparse EM, but where the choice of iterations where we consider the full space of transformations is more founded than in the general case (see Appendix A.3).

#### 4.4.2 Incremental EM

In the incremental EM algorithm (see Appendix A), the visible statistics are maximised each time the expectation step has been applied to a single data point. We can see intuitively that this will converge faster than a pure EM algorithm if we see the estimation step as optimising the likelihood function w.r.t. the hidden variables,

based on the observed variables. Since the estimation step is based on the results of the maximisation step, the more accurate those results are, the more precise the results of the E-step will be. The improved convergence of the incremental version of the EM algorithm, as opposed to the traditional EM algorithm, was shown empirically in [18].

The problem with an incremental EM algorithm is that the gain in the number of iterations is offset by the fact that each step takes much longer to compute. It is possible to improve the execution time of the algorithm, however, by implementing an intermediate version of the algorithm: instead of performing the M-step after each E-step, we only perform the M-step after  $n$  E-steps, where  $n$  is a small integer number.

In our case, this means we update the latent object images, the latent image mask, and the variance on the latent image each time we've seen a frame. This improves the number of steps needed for convergence dramatically, as the statistics used for the estimation of the hidden variables incorporate all the knowledge we have at each step. A comparison of the convergence rate for different values of  $n$  is shown in chapter 5.

### 4.4.3 Tracking

Another important bottleneck in the optimisation of the lower bound on the likelihood, as given in eq 2.7, is the calculation of  $Q^n(j_l)$ . Indeed,  $Q^n(j_l)$  is the normalised  $p(j_l|\mathbf{x})$  over all transformations. We've shown before that the result was  $\approx 1$  for  $j_l = j_l^n$ , and that we could reduce the computational cost of equation 2.7 by computing the term which is multiplied by  $Q^n(j_l)$  only if  $Q^n(j_l)$  is big enough to

make it worthwhile. Unfortunately, the factor itself is rather expensive to compute. It would therefore be interesting to reduce the search space to likely values of  $j_l$  only.

Titsias and Williams have separated the sparseness out of the basic EM algorithm, and combined this with an incremental EM algorithm where the search space over the transformations is reduced as discussed before [20]. Their new algorithm starts by 'tracking' an object during a single pass over the sequence. During the tracking pass, the latent variables are updated after each iteration, the range of transformation indices  $j_l$  is limited and the neighbourhood of the most likely transformation for the last frame, and most likely transforms are stored.

Both the background and the foreground are learnt by tracking an object frame after frame. The resulting data — the latent object images, the masks, and the most likely transformation — are then used to initialise the EM algorithm, dramatically reducing the search space and therefore speeding up the algorithm.

## Discussion

The huge advantage of the Tracking algorithm is, of course, speed. The big reduction in the search space of the algorithm, combined with the incremental EM makes for speedy convergence. This improvement can be quite dramatic, when the algorithm succeeds.

The big drawback of this method, however, is that, since nothing guarantees that the algorithm will start learning the foremost object first, it might lose track of an object due to occlusions. If this happens, the reduced search over the transformations ensures that the object is not found when it reappears, which will result in failure to converge. This is not really a limitation of the algorithm, since an increase

of the size of the explored “transformation neighbourhood” solves this problem. Unfortunately, an increased search space does have a negative impact on the performance, which diminishes the usefulness of the method.

We have experimented a little with a variant of this algorithm, where an ever increasing range of transformations was explored around the most likely transformation of the previous frame. The idea was that this would solve the problems of occlusion, and thus make for a very robust yet fast algorithm. We have not been able to find a satisfying criterion to indicate when the most likely transformation was found. We still feel that this is worthy of some more investigation, however.

## 4.5 Colour

It is obvious that the data can be classified more precisely if pixels are represented with colour rather than just intensity. Indeed, different colours can result in the same greyscale intensity which results in reduced discrimination. In colour images, each pixel is defined by three separate intensities — corresponding to the amount of red, green and blue in the perceived colour.

Little adaptation is needed to our algorithm in order to learn objects in colour images. The original GREEDY algorithm uses a Gaussian probability density function on the pixel intensity of each pixel to model the probability that a pixel from the observed image corresponds to that same pixel of the transformed latent image (see section 2.2). Now a greyscale image is represented by a matrix of size  $P_x \times P_y$ , where each element represents the intensity of a pixel. A colour image, on the other hand, is represented by a matrix of  $P_x \times P_y \times 3$  pixels, where each  $1 \times 1 \times 3$  sub matrix represents the colour intensities in the red, green, and blue colour channel

respectively.

In order to model the probability of a pixel with a normal probability density function, the one-dimensional normal distribution  $\mathcal{N}(\mathbf{x}_p; \boldsymbol{\mu}_p, \sigma^2)$  thus becomes a three-dimensional normal  $\mathcal{N}(\mathbf{x}_p; \boldsymbol{\mu}_p, \mathbf{C})$  [17]:

$$p(\mathbf{x}_p) = \frac{1}{\sqrt{(2\pi)^3 |\mathbf{C}|}} \exp\left(-\frac{1}{2}(\mathbf{x}_p - \boldsymbol{\mu}_p)^\top \mathbf{C}^{-1}(\mathbf{x}_p - \boldsymbol{\mu}_p)\right)$$

where  $\mathbf{x}_p$  is a vector with the 3 colour components of the  $p$ th pixel of  $\mathbf{x}$ ,  $\boldsymbol{\mu}_p$  is the vector with the colour components of average colour for that same pixel  $p$  over all images.

The covariance matrix  $\mathbf{C}$  is a  $3 \times 3$  matrix whose elements are updated as follows, for  $1 \leq i, j \leq 3$ :

$$C_{ij} = \frac{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{z}_{l-1}^n * \bar{\mathbf{s}}_{j_l}^n * \mathbf{r}_{j_l}^n)^\top (\mathbf{x}_i^n - (\mathbf{T}_{j_l} \mathbf{f}_l)_i) (\mathbf{x}_j^n - (\mathbf{T}_{j_l} \mathbf{f}_l)_j)}{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{z}_{l-1}^n)^\top (\bar{\mathbf{s}}_{j_l}^n * \mathbf{r}_{j_l}^n)}.$$

The rest of the ‘colour’ algorithm stays mostly identical to what was described in section 2.2. Indeed, where the pixels of the observed images and the latent object images have become vectors of length 3, all the resulting probabilities are still scalars. Therefore, the masks are still two-dimensional matrices. The update functions for the mask  $\boldsymbol{\pi}_l$  remains unchanged, and the update of each of the colour channels of the latent image objects is updated based on the corresponding colour channel of the observed image using the same update rules as for the intensity. The resulting updates are shown in Appendix B.4.

## 4.6 Rotating Masks

The GREEDY algorithm learns each object, one at the time, and avoids the combinatorial explosion of parameters by removing the explained pixels from consideration. In the original algorithm, this is done by recursively updating a mask,  $\mathbf{z}$ , for each observed image and thus removing those pixels that (i) belong to the object spatially and (ii) have the correct intensity to be part of the object — i.e. are not occluded:

$$z_l^n = \prod_{i=1}^n (\mathbf{1} - (\mathbf{T}_{j_l^i} \boldsymbol{\pi}_l) * \mathbf{r}_{j_l^i}^i)$$

In some cases, the algorithm fails to converge, or converges very slowly when the data is noisy. In those cases, it may help to alternate the learning of the objects, and to update the mask as follows:

$$z_l^n = \prod_{i \neq n} (\mathbf{1} - (\mathbf{T}_{j_l^i} \boldsymbol{\pi}_l) * \mathbf{r}_{j_l^i}^i)$$

This helps the algorithm to focus on the pixels that are relevant. This method is not useful in all situations; it is mostly interesting when the sequence contains two objects that have a similar  $p(\mathbf{x}|j_l)$ . The most likely transform will then oscillate between both objects and the resulting latent object will contain a representation of both objects. It may take an enormous lot of iterations before a single object gets the ‘upper hand’, and the algorithm starts converging. With ‘rotating masks’ and alternating updates of the objects, distracting objects get masked out very soon, helping the objects to converge in the next iterations.

# Chapter 5

## Experiments

### 5.1 Data Acquisition

#### 5.1.1 Existing Data

Our first tests were done with the same sequence Jojic and Frey used [13, 14], and which was also used by Titsias and Williams in [23, 20]. This sequence, some frames of which are shown in figure 5.1, is useful as a comparison with previous implementations.

However, this is an ‘easy’ sequence, in the sense that the background has lots of texture, the foreground objects are dissimilar, and the overall contrast is very high. Moreover, although this sequence is available in (colour) AVI format, the



Figure 5.1: Some frames from the original sequence

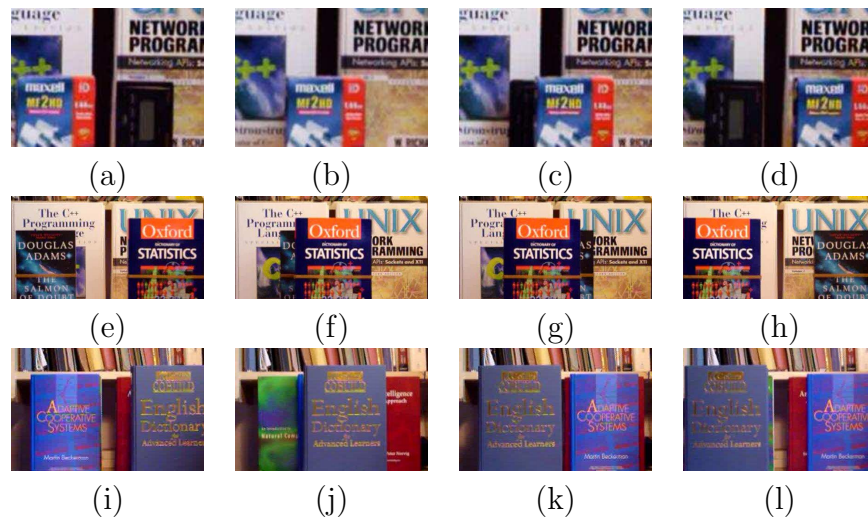


Figure 5.2: Some frames from the new sequences

frames are saved in greyscale. Since we implemented a version of the algorithm that discriminates pixels according to all three colour channels, we needed a colour dataset.

### 5.1.2 New Video Data

Generating video sequences for this algorithm is not a trivial task. Indeed, in our implementation, translations are the only transformations. It is therefore crucial that the angle at which objects are seen does not change (as this would introduce shearing), and that the distance between the objects and the camera remains fixed (as a variable distance would introduce scaling). At the same time, we wanted to generate different sequences of different degrees of difficulty.

After a few trials, we decided to generate sequences of books sliding on a table-top. The sequences, shown in figure 5.2, have varying levels of contrast, levels of illumination, structure, and size. They were used both on the ‘greyscale’ and ‘colour’



Figure 5.3: Artificial Colour Sequence

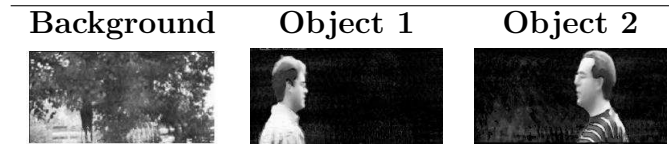


Figure 5.4: Segmentation of the Original Sequence

implementation, in order to show the differences between both algorithms.

### 5.1.3 New Artificial Data

In order to test the validity of our colour-based segmentation algorithm, we needed a data set in which colour was crucial to the discrimination. We generated the images with an image processing program, and added normally distributed noise with mean = 0 and variance = 0.2 to each pixel of the images. This was done in order to mimic the noise that can be found in real-world images.

## 5.2 Results

The experiments related below were all done with the sequence of Frey and Jovic, except when explicitly shown. The resulting segmentation is shown in figure 5.4 and is not reproduced for each experiment. Indeed, the differences are minute — if present at all — and it is much more relevant to look at the likelihood instead.

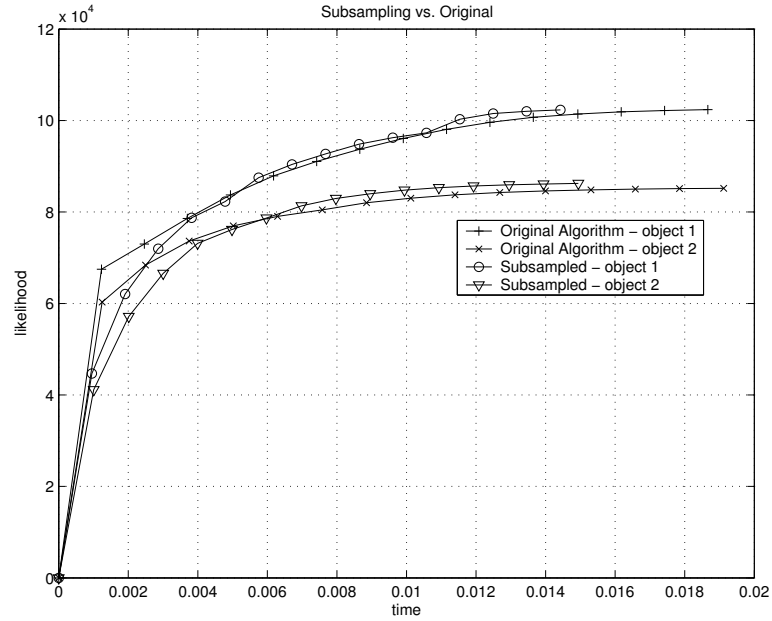


Figure 5.5: Likelihood vs. Time for Greedy and Sub-sampled Greedy

### 5.2.1 Resolution Reduction

Figure 5.5 shows the evolution of the likelihood in function of time. We can see that the reduced precision of the evaluation of  $Q^n(j_l)$  ensures that the algorithm converges a little slower in terms of the number of iterations. The reduced time required for each iteration, however, makes up for this. The algorithm in which we subsample the images converges sooner than the original algorithm.

### 5.2.2 Lazy Algorithm

The lazy algorithm results in an important speed improvement when compared to the normal, non-lazy greedy algorithm. From fig. 5.6(a), we can see that the likelihood is not affected in any way by the ‘laziness’. It converges exactly as if the term were computed in full. In fig. 5.6(b), however, we can see that the computational

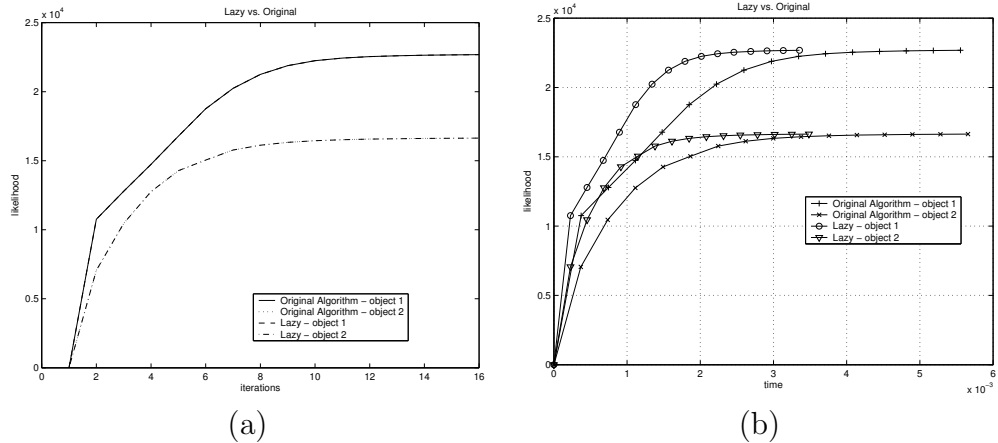


Figure 5.6: Likelihood for ‘lazy’ and non-‘lazy’ greedy algorithm

cost has been spectacularly reduced.

### 5.2.3 Incremental EM

We compare the convergence speed of the original GREEDY algorithm with the incremental version of the algorithm in figure 5.7. Figure (a) shows the difference in convergence between simple greedy, and incremental greedy with  $n = 1$ : the latent variables are updated after each data point. Figure 5.7(b) compares the original algorithm with the incremental variant, but where the incremental updates are applied only during the first pass over the data. During the next iterations, the regular GREEDY algorithm is used.

In figures 5.7 (c) and (d) we can see the results of the same runs, but where the likelihood is plotted as a function of time rather than the number of iterations. We can observe that each iteration of the incremental algorithm takes longer, and that the advantage of using it is only clear when it is limited to the first iteration. This is what happens in the tracking algorithm [20], where the most likely transformations

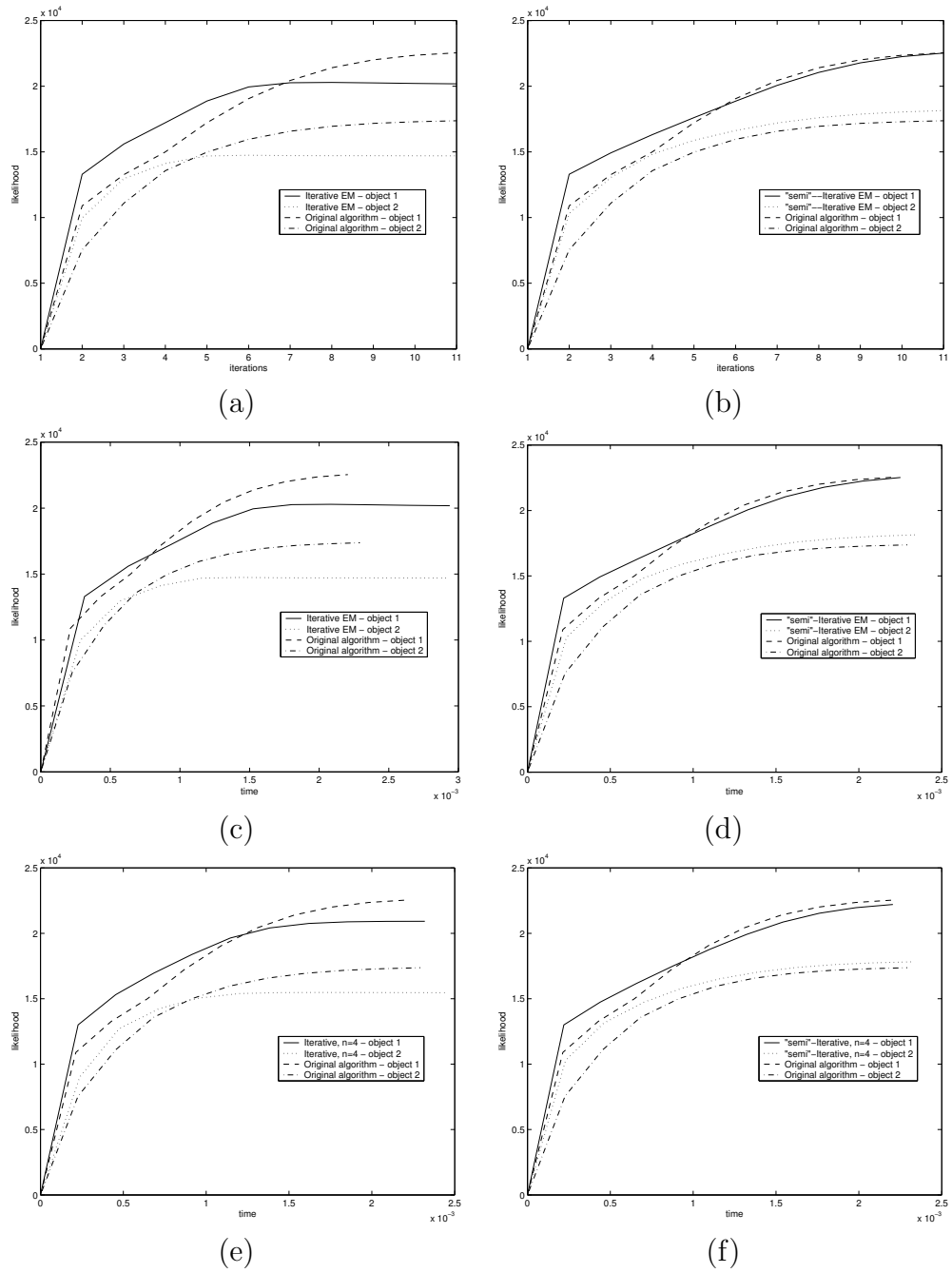


Figure 5.7: Comparison of GREEDY and incremental GREEDY

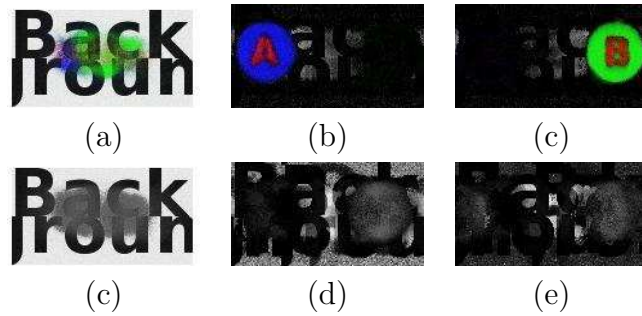


Figure 5.8: Segmentation with Colour and Greyscale GREEDY

are found during a first, incremental pass, and later refined in the regular GREEDY algorithm.

In figures 5.7 (e) and (f), we compare the performance of the ‘intermediate’ incremental algorithm, where the latent objects are updated after every 4 data points. Figure (e) shows the pure incremental version, while figure (f) shows the version where incremental updates are only performed in the first pass. It is clear that, although this ‘intermediate’ version has very similar convergence properties as the pure incremental EM and has very comparable run-times as the original algorithm, the best results are still obtained by applying incremental EM during the first pass only.

#### 5.2.4 Colour

In order to demonstrate the ‘colour’ version of our algorithm, we created the artificial video sequence, of which we show some frames in figure 5.3. This figure has a rather low contrast in greyscale, but its contrast in colour is very high. This sequence is interesting for multiple reasons. First, it is a colour image, which allows us to give an example of a sequence that is successfully segmented with our ‘colour’ algorithm,

while it fails with the greyscale version. Figure 5.8 (a), (b), and (c) show the results of segmentation with the colour algorithm, while (c), (d), and (e) show the corresponding results of the greyscale algorithm.

The other reason for which this sequence is interesting, is that both objects are identical in size. Therefore, the probabilities  $p(\mathbf{x}^n | \mathbf{f}_l, \sigma_l^2)$  are very similar for both objects. This causes the GREEDY algorithm to fail since, depending on the noise present in the frame, both objects are equally likely to generate the most likely transformation. Rotating masks bring a possible solution to this problem.

### 5.2.5 Rotating Masks

The ‘rotating’ mask allow us to remove the pixels that have been explained by any object, except the current one, from consideration. In some cases, this makes the difference between success and failure of the algorithm. Figure 5.9 shows the results of a test on artificial data. Images (a) and (b) show the resulting latent objects with the original algorithm. As we can observe, the algorithm cannot converge to a single object — or does so very slowly. Images (c) and (d) show the results obtained with the ‘rotating’ mask. As we can see from the likelihood, shown in figure (e), the algorithm has trouble finding an object on the first iteration, but once the masks are initialised, it converges very quickly.

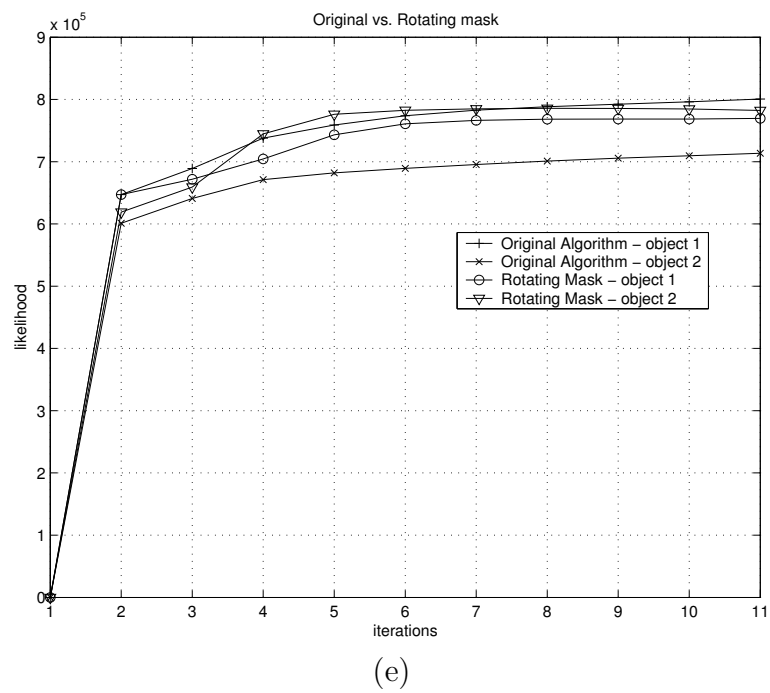
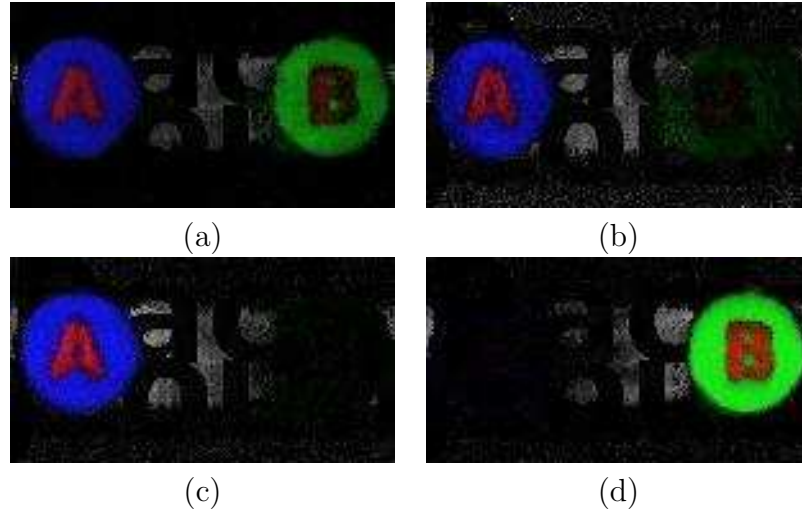


Figure 5.9: Latent Objects for Rotating Masks

# Chapter 6

## Conclusions

### 6.1 Key results

#### 6.1.1 Speed Issues

We have seen that the GREEDY algorithm makes the learning of multiple objects in image sequences tractable. The biggest remaining bottleneck is the number of possible transformations for each latent object that need be evaluated for each image. If we allow the image to rotate and change in shape, we get an exponential explosion of the number of transformations. We believe our implementation is quite efficient, and is a good basis for future investigations of the algorithm.

#### 6.1.2 Robustness of the Algorithm

In its general form, the algorithm is quite robust. Occlusions are handled beautifully, and the algorithm converges even when initialised badly. This original algorithm is sometimes quite slow, however, and improvements were needed to make it usable.

Some of those improvements tend to make the algorithm less robust, by reducing the search space for the hidden variables. We have investigated how we could improve the robustness for different datasets, and investigated what speed improvements could be made without jeopardising the convergence.

## 6.2 Future work

Probably the most obvious candidate for future work would probably be a reduction of the transformation space that needs to be searched. “Tracking” is an excellent step in that direction, but we feel it can still be improved upon.

We lacked time to investigate different types of transformations, and could not investigate the capabilities of the model w.r.t. articulated objects. Moreover, the algorithm does not learn the number of objects present in a sequence. In the future, we would like to investigate possible ways to infer those, as well as as effective ways to learn the transformations undergone by the latent object. It may be possible to use some kind of ‘breadth first’ search through the range of transformations (first finding the most likely translation based on the rotation and shearing of the object in the previous frame; then, for that translation, the most likely rotation, and so on).

Another interesting topic would be to improve the results with small latent object images, by learning the size of each latent object and initialising it correctly. Finally, we would like to allow objects to enter and leave the scene.

## 6.3 Summary

In this thesis we have presented our own implementation of the GREEDY algorithm. We have tested the algorithm on different data sets, both real-world video sequences and artificial data sets. We have investigated what were the key properties of the algorithm, investigated possible improvements to the original algorithm, and evaluated the merits of those changes.

# Bibliography

- [1] Serge Ayer and Harpreet S. Sawhney. Layered representation of motion video using robust maximum-likelihood estimation of mixture models and MDL encoding. In *Fifth International Conference on Computer Vision*, pages 777–785, 1995.
- [2] Dana Harry Ballard. *An Introduction to Natural Computation*, chapter 4. The MIT Press, 1997.
- [3] Bernard Chazelle et al. Application challenges to computational geometry. In Bernard Chazelle, Jacob E. Goodman, and Richard Pollack, editors, *Advances in Discrete and Computational Geometry – Proc. 1996 AMS-IMS-SIAM Joint Summer Research Conf. Discrete and Computational Geometry: Ten Years Later*, number 223 in Contemporary Mathematics, pages 407–423. Amer. Math. Soc., 1999.
- [4] CIE. Commission internationale de l'éclairage proceedings, 1931. Cambridge University Press, 1932.
- [5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*,

- 39(1):1–38, 1977.
- [6] Brendan J. Frey and Nebojsa Jojic. Transformed component analysis: Joint estimation of spatial transformations and image components. In *IEEE International Conference on Computer Vision (2)*, pages 1190–1196, 1999.
- [7] Brendan J. Frey and Nebojsa Jojic. Fast, large-scale transformation-invariant clustering. In *Advances in Neural Information Processing Systems 14*, pages 721–727, 2001.
- [8] Brendan J. Frey and Nebojsa Jojic. Transformation-invariant clustering using the em algorithm. *IEEE transactions on pattern analysis and Machine Intelligence*, 25(1), January 2003.
- [9] H.O. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14(2):174–194, June 1958.
- [10] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunk. *Machine Vision*, chapter 3. McGraw-Hill International Editions, 1995.
- [11] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunk. *Machine Vision*, chapter 14.2. McGraw-Hill International Editions, 1995.
- [12] E. T. Jaynes and G. Larry Bretthorst. *Probability theory: The Logic of Science*, chapter 1. Cambridge University Press, 2003.
- [13] Nebojsa Jojic and Brendan J. Frey. Learning flexible sprites in video layers. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition, Maui, HI*, 2001.

- 
- [14] Nebojsa Jojic and Brendan J. Frey. A generative model for 2.5d vision: Estimating appearance, transformation, illumination, transparency and occlusion. *International Journal on Computer Vision*, 2002. submitted.
- [15] Anitha Kannan, Nebojsa Jojic, and Brendan Frey. Fast transformation-invariant factor analysis. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1263–1270. MIT Press, Cambridge, MA, 2003.
- [16] Donald Ervin Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching, chapter 6. Addison Wesley, 3 edition, 1997.
- [17] Tom Mitchell. *Machine Learning*. McGraw–Hill, 1997.
- [18] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.
- [19] Hai Tao, Harpreet S. Sawhney, and Rakesh Kumar. Dynamic layer representation with applications to tracking. *Proc. IEEE conf. on Computer vision and Pattern Recognition*, pages 134–141, 2000.
- [20] Michalis Titsias and Christopher K. Williams. Fast unsupervised greedy learning of multiple objects and parts from video. 2004. To appear in: Generative-Model Based Vision Workshop 2004.
- [21] Philip H.S. Torr, Richard Szeliski, and P. Anandan. An integrated bayesian approach to layer extraction from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):297–303, March 2001.

- [22] John Y. A. Wang and Edward H. Adelson. Representing Moving Images with Layers. *The IEEE Transactions on Image Processing Special Issue: Image Sequence Compression*, 3(5):625–638, September 1994.
- [23] Christopher K.I. Williams and Michalis Titsias. Greedy learning of multiple objects in images using robust statistics and factorial learning. *Neural Computation*, 16(5):1039–1062, May 2004.

# Appendix A

## The EM algorithm

In this appendix, we give a description of the basic Expectation Maximisation (EM) [5, 9] algorithm. Both the greedy algorithm and the optimisation of TMG are based on this algorithm, which is very widely used and comes in multiple varieties. We outline the basic principles, and give a short overview of the most interesting variants.

### A.1 The Basic Algorithm

The Expectation Maximisation (EM) algorithm is an iterative algorithm that maximises the likelihood of a system in which some of the model variables are not observed. Suppose we observe some variables  $Z$  of the data, but not the variables  $Y$ . Assume further that we model the data with a system that has parameters  $\theta$  as  $P(y, z|\theta)$ . We want to find the value of  $\theta$  which maximises the log likelihood  $L(\theta) = \log P(z|\theta)$  given the observed data. We start with some initial guess of the maximum likelihood parameters,  $\theta^0$ . We then apply the following two steps

iteratively, generating estimates  $\theta^1, \theta^2, \dots$  for  $t = 1, 2, \dots$ :

**In the E-Step:** we estimate a probability distribution  $\tilde{P}^t$  over the range of  $Y$  such that  $\tilde{P}^t(y) = P(p|z, \theta^{t-1})$ .

**In the M-Step:** we set

$$\theta^t \leftarrow \arg \max_{\theta} [\log P(y, z|\theta)].$$

Each step is guaranteed to improve the likelihood or, if a maximum has been reached, to keep it unchanged. We thus avoid the need to try all combinations of the hidden and the observed variables.

## A.2 Incremental EM

During the E-step of the traditional EM algorithm, the sufficient statistics of the hidden variables are computed. Once the hidden variables have been estimated over all the data points, the observed statistics are maximised in the M step.

We can feel intuitively that the algorithm would converge faster if both the hidden and the observed variables were updated in fast succession; this would allow the algorithm to ‘interpret’ the data in the light of all the knowledge it has yet learnt.

This intuition is confirmed empirically [18]; the EM algorithm converges much faster — i.e. in fewer iterations — if we alternate the E and the M step of the algorithm after each data point instead of after the whole data set.

Unfortunately, the the M-step is often rather computationally intensive compared to the E-step. As a consequence, the savings in the number of iterations are offset by the increased computation time of each step. This problem is easily

solved, however, by running the M-step every  $n$  data points, with  $n$  small. This combines the advantages of speedy convergence with lower computational cost and substantially improves the run time of the algorithm.

### $\gamma$ -decay EM

A variant of incremental EM updates the statistics in the E-step with an exponentially decaying average of the recently visited statistics. One can see intuitively that this will result in faster convergence, since the most recently visited datapoints were evaluated with the most recent estimation for the latent variables of the model, which are therefore supposed to be the most accurate, and are given the preponderance.

We evaluated the merits of this method with the GREEDY algorithm, but this did not seem to improve our results. This is probably due to the fact that this method is mostly efficient for a large number of data points, which is not the case in our (short) video sequences.

## A.3 Sparse EM

The ‘sparse’ variant of the EM algorithm is useful when our model contains hidden variables which can take a great range of values, but of which only a few are ‘plausible’. It is often the case that one value has a probability of one, and that all other values consequently have a probability of 0. In that case it may be useful to likely value of iteration  $t - 1$ . All other values of the variable keep estimate the variable at iteration  $t$  in the neighbourhood of the most their probability of 0, and are re-evaluated at infrequent intervals.

When the plausible values are stable, this may save considerable computing time. In some cases, however, it may prove that the most likely value is rather unstable, or oscillates between a small number of values. This is the case in the GREEDY algorithm, where the most likely transformation is wont to jump from one object in the observed image to the other. In that case, a ‘traditional’ sparse algorithm is bound to fail, or at least degrade the performance of the algorithm dramatically, and other, similar, techniques can be used.

# Appendix B

## Implementation Details for the GREEDY algorithm

Below is a condensed description of the GREEDY algorithm. The algorithm as it is represented here is taken in its integrity form [23]. We also use the same notation as in their paper, notably that:

- $\mathbf{x} * \mathbf{y}$  represents the element-wise product of two matrices of equal size.
- $\mathbf{y} * \mathbf{y}$  is written  $\mathbf{y}^2$  for compactness.
- Similarly,  $\mathbf{x} ./ \mathbf{y}$  denotes the element-wise division of two matrices.
- $\mathbf{1}$  denotes a vector of only ones.

## B.1 Learning the Background

Using this equation, we learn the background by optimising:

$$L_b = \sum_{n=1}^N \log \sum_{j_b=1}^{J_b} P_{j_b} \prod_{p=1}^P (\alpha_b \mathcal{N}(x_p^n; (T_{j_b} \mathbf{b})_p, \sigma_b^2) + (1 - \alpha_b) U(x_p^n))$$

## B.2 The EM algorithm

The EM algorithm optimises the log-likelihood described in eq. B.1 by performing the following steps:

**In the E step** we estimate the posterior probability for the transformation hidden variable  $P(j_b | \mathbf{x}^n)$  based on the frame  $\mathbf{x}^n$ , and for  $\mathbf{r}_{j_b}^n$ , a vector whose  $p$ th element stores the probability that pixel  $p$  is part of the non-occluded background. These are computed as follows:

$$P(j_b | \mathbf{x}^n) = \frac{p(\mathbf{x}^n | j_b)}{\sum_{i=1}^{J_b} p(\mathbf{x}^n | i)} \quad (\text{B.1})$$

$$(\mathbf{r}_{j_b}^n)_p = \frac{\alpha_b \mathcal{N}(\mathbf{x}_p^n; (T_{j_b} \mathbf{b})_p, \sigma_b^2)}{\alpha_b \mathcal{N}(\mathbf{x}_p^n; (T_{j_b} \mathbf{b})_p, \sigma_b^2) + (1 - \alpha_b) U(x_p^n)}$$

**In the M-step** of the algorithm, we maximise the parameters  $\mathbf{b}$  and  $\sigma_b^2$  with respect to those with the following:

$$\mathbf{b}_p \leftarrow \frac{\sum_{n=1}^N \sum_{j_b=1}^{J_b} P(j_b | \mathbf{x}^n) (\mathbf{T}_{j_b}^\top (\mathbf{r}_{j_b}^n * \mathbf{x}_n))_p}{\sum_{n=1}^N \sum_{j_b=1}^{J_b} j_b = 1 P(j_b | \mathbf{x}^n) (T_{j_b}^\top \mathbf{r}_{j_b}^n)_p}$$

$$\sigma_b^2 \leftarrow \frac{\sum_{n=1}^N \sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n) ((\mathbf{r}_{j_b}^n)^\top (\mathbf{x}^n - T_{j_b} \mathbf{b}))^2}{\sum_{n=1}^N \sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n) ((\mathbf{r}_{j_b}^n)^\top \mathbf{1})}$$

### B.3 Learning the Foreground Objects

As we described in chapter 2, the foreground objects are learned each in turn by optimising the lower bound on  $L_l$ :

$$F_l = \sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) \left( \sum_{p=1}^P (\mathbf{z}_{l-1}^n)_p \log[(\mathbf{T}_{j_l} \pi_l)_p p_{f_l}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) + (\mathbf{1} - \mathbf{T}_{j_l} \pi_l)_p p_b(x_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p)] - \log Q^n(j_l) \right).$$

This function is optimised using the following EM algorithm:

**In the E-step,** the following sufficient statistics are gathered about the data, based on the current estimates for  $\mathbf{f}_l, \pi_l, \sigma_l$ :

$$\bar{\mathbf{s}}_{j_l}^n = \frac{(\mathbf{T}_{j_l} \pi_l)_p p_{f_l}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p)}{(\mathbf{T}_{j_l} \pi_l)_p p_{f_l}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) + (\mathbf{1} - (\mathbf{T}_{j_l} \pi_l)_p) p_b(x_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p)}$$

$$r_p^{j_l^n} = \frac{\alpha_f \mathcal{N}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p, \sigma_l^2)}{\alpha_f \mathcal{N}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p, \sigma_l^2) + (1 - \alpha_f) U(x_p^n)},$$

and, based on those statistics,  $F_l$  is maximised with respect to  $Q^n(j_l)$ :

$$Q^n(j_l) \propto \exp \left[ \sum_{p=1}^P (\mathbf{z}_{l-1}^n)_p \log \left( (\mathbf{T}_{j_l} \pi_l)_p p_{f_l}(x_p^n; (\mathbf{T}_{j_l} \mathbf{f}_l)_p) + (\mathbf{1} - \mathbf{T}_{j_l} \pi_l)_p p_b(x_p^n; (\mathbf{T}_{j_l} \mathbf{b})_p) \right) \right] \quad (\text{B.2})$$

**In the M–step,** both those statistics and the computed  $Q^n(j_l)$  are used to optimise the measured statistics using the following updates:

$$\begin{aligned}
(\mathbf{f}_l)_p &\leftarrow \frac{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{T}_{j_l}^\top (\mathbf{z}_{l-1}^n * \bar{\mathbf{s}}_{j_l}^n * \mathbf{r}_{j_l}^n * \mathbf{x}^n))_p}{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{T}_{j_l}^\top (\mathbf{z}_{l-1}^n * \bar{\mathbf{s}}_{j_l}^n * \mathbf{r}_{j_l}^n))_p} \\
(\pi_l)_p &\leftarrow \frac{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{T}_{j_l}^\top (\mathbf{z}_{l-1}^n * \bar{\mathbf{s}}_{j_l}^n))_p}{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{T}_{j_l}^\top (\mathbf{z}_{l-1}^n))_p} \\
\sigma_l^2 &\leftarrow \frac{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{z}_{l-1}^n)^\top [\bar{\mathbf{s}}_{j_l}^n * \mathbf{r}_{j_l}^n * (\mathbf{x}^n - \mathbf{T}_{j_l} \mathbf{f}_l)^2]}{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{z}_{l-1}^n)^\top [\bar{\mathbf{s}}_{j_l}^n * \mathbf{r}_{j_l}^n]}
\end{aligned}$$

## B.4 Learning Colour Objects

Intensity maps or greyscale images are represented by a single value for each pixel. The way we perceive colour has defined the way colour images are saved: we see any colour as a combination of three primary colours: red, green, and blue[4]. Colour images therefore contain three values per pixel, namely the amount of red, green, and blue light required to reproduce the original colour.

In order to learn from colour images, we model the probability of identity of pixels according to a Normal probability distribution. The definition of  $p()$ ,  $\bar{\mathbf{s}}_{j_l}^n$ , and  $\mathbf{r}_{j_l}^n$  does not change, and the update of the latent mask,  $\boldsymbol{\pi}_l$ , is therefore identical in both the ‘colour’ and the ‘greyscale’ algorithm.

Each pixel of the latent object is updated in the same way for colour images as it was for greyscale images. The difference is that the three colour channels are

updated separately:

$$(\mathbf{f}_l^c)_p \leftarrow \frac{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{T}_{j_l}^\top (\mathbf{z}_{l-1}^n * \bar{\mathbf{S}}_{j_l}^n * \mathbf{r}_{j_l}^n * \mathbf{x}_n^c))_p}{\sum_{n=1}^N \sum_{j_l=1}^{J_f} Q^n(j_l) (\mathbf{T}_{j_l}^\top (\mathbf{z}_{l-1}^n * \bar{\mathbf{S}}_{j_l}^n * \mathbf{r}_{j_l}^n))_p}, \text{ for } 1 \leq c \leq 3$$