REAL-TIME MARKERLESS 3-D HUMAN BODY TRACKING

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2006

By
Fabrice CAILLETTE
School of Computer Science

(EM2NN)

✓

TH 26547

Contents

List of Tables

List of Algorithms

List of Figures

Abstract

The ability to perform efficient human motion tracking is essential in a wide variety of applications such as human-computer interfaces, anthropological studies, entertainment, and surveillance. Markerless human body tracking involves recovering the parameters of a kinematic model from video sequences. This inference problem is made difficult by the noisy and ambiguous nature of camera images. The high dimensionality of the parameter space is also a major challenge, making human-body tracking a very active research area in the computer-vision community.

This thesis presents algorithms for real-time human body-tracking based on multiple camera views. A robust volumetric reconstruction technique is first presented, combining shape and colour from multiple views in an hierarchical scheme. Background segmentation and volumetric reconstruction are merged into a single process, with benefits in performance and robustness. The appearance model, used to relate the kinematic parameters to image observations, is composed of Gaussian blobs. This blob-based model is automatically acquired from the data, and updated from the reconstructed volume in an Expectation-Maximisation framework. Our first proposed tracking algorithm recovers the pose of the kinematic model directly from the blobs, using a two-steps inverse kinematics procedure. A second proposed method approaches tracking as a Bayesian estimation problem. To guide the propagation of samples in the parameter space, we propose a predictive model based on the combination of local dynamics and learnt variable length Markov models of behaviour. The evaluation of the likelihood of the candidate model configuration is critical for computational efficiency. We propose a novel evaluation procedure based on the relative entropy between mixtures of Gaussian blobs. The robustness and performance of our system are demonstrated on challenging video sequences exhibiting fast and diverse movements.

Declaration

Copyright

Acknowledgements

Of course I would like to thank my supervisor, Toby Howard, for his support and encouragements. I wish to acknowledge Aphrodite Galata for giving me the motivation to achieve more. Thanks are also due to my family, without whom nothing (or not much) would have been possible. Special mention to Juana for helping me going through all those years, and for so much more... Finally big thanks to all my friends in Manchester – Vivek, Gilles, Franck, Nikolay, Lilian, Jun, Gwenn, Bruno, and all the others – for making the whole experience enjoyable.

Publications

Some of the work described in this thesis also appeared in:

- F. Caillette and T. Howard. Real-Time Markerless Human Body Tracking with Multi-View 3-D Voxel Reconstruction. In A. Hoppe, S. Barman and T. Ellis, editors, *Proceedings of the* 15th *British Machine Vision Conference (BMVC)*, volume 2, pages 597–606, Kingston UK, September 2004.

- F. Caillette and T. Howard. Real-Time Markerless Human Body Tracking Using Colored Voxels and 3-D Blobs. In *Proceedings of the* 3rd *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 266–267, Arlington VA, November 2004

- F. Caillette, A. Galata and T. Howard. Real-Time 3-D Human Body Tracking using Variable Length Markov Models. In W. F. Clocksin, A. W. Fitzgibbon and P. H. S. Torr, editors, *Proceedings of the* 16th *British Machine Vision Conference (BMVC)*, volume 1, pages 469–478, Oxford UK, September 2005.

- F. Caillette, A. Galata and T. Howard. Real-Time 3-D Human Body Tracking using Learnt Models of Behaviour. Submitted to the journal of *Computer Vision and Image Understanding (CVIU)*.

List of Notations

t	:	index of the current frame (discrete timestep).	$(\mathbb{N})$
u	:	pixel measurement.	$(\mathbb{R}^3)$
N_u	:	number of pixel measurements.	$(\mathbb{N})$
μ	:	the true colour value of the pixel.	$(\mathbb{R}^3)$
w	:	noise associated with a pixel measurement.	$(\mathbb{R}^3)$
σ_w	:	standard deviation of the pixel noise.	$(\mathbb{R})$
Σ_w	:	covariance matrix of the noise.	$(\mathbb{R}^{3\times 3})$
σ_e	:	standard deviation of the error in the pixel model.	$(\mathbb{R})$
D_M	:	Mahalanobis Distance to the Gaussian model.	$(\mathbb{R}^3 \rightarrow \mathbb{R})$
D'_M	:	Mahalanobis Distance with shadows handling.	$(\mathbb{R}^3 \rightarrow \mathbb{R})$
T_d	:	classification threshold with d degrees of freedom.	$(\mathbb{R} \rightarrow \mathbb{R})$
s	:	pixel sample in the current frame.	$(\mathbb{R}^3)$
N_s	:	number of pixel samples.	$(\mathbb{N})$
$\mathcal{S}$	:	set of spatially distributed pixel samples $\{s_1, \ldots, s_{N_s}\}$.	$(\mathbb{R}^{3\times N_s})$
$\mathcal{V}$	:	a voxel.	
$s_\mathcal{V}$	:	size of the side of the voxel $\mathcal{V}$.	$(\mathbb{R})$
$X_\mathcal{V}$	:	position of the centre of the voxel $\mathcal{V}$.	$(\mathbb{R}^3)$
D	:	current recursive depth for the reconstruction.	$(\mathbb{N})$
D^+	:	maximal recursive depth for the reconstruction.	$(\mathbb{N})$
c_i	:	camera number i.	$(\mathbb{N})$
N_c	:	number of cameras.	$(\mathbb{N})$
$C^i_\mathcal{V}$	:	colour of the voxel, as seen from camera c_i.	$(\mathbb{R}^3)$
$\mathcal{C}_\mathcal{V}$	:	set of voxel colours, for all cameras $\{C^1_\mathcal{V}, \ldots, C^{N_c}_\mathcal{V}\}$.	$(\mathbb{R}^{3\times N_c})$
B	:	a single Gaussian blob.	
$\boldsymbol{\mu}$	:	full mean vector of a blob B.	$(\mathbb{R}^6)$
μ_X	:	mean position vector of a blob B.	$(\mathbb{R}^3)$
μ_C	:	mean colour vector of a blob B.	$(\mathbb{R}^3)$
$\boldsymbol{\Sigma}$	:	full covariance matrix of a blob B.	$(\mathbb{R}^{6\times 6})$
Σ_X	:	spatial covariance matrix of a blob B.	$(\mathbb{R}^{3\times 3})$
Σ_C	:	colour covariance matrix of a blob B.	$(\mathbb{R}^{3\times 3})$

Σ_{XC}	:	mixed colour-position covariance matrix of a blob B.	$(\mathbb{R}^{3\times 3})$
$\widehat{\alpha}$	:	offset of attachment along a bone segment.	$(\mathbb{R})$
$\widehat{\sigma_x}$	:	spatial standard deviation along the main axis.	$(\mathbb{R})$
$\widehat{\sigma_y}$	:	spatial standard deviation along the second axis.	$(\mathbb{R})$
$\widehat{\sigma_z}$	:	spatial standard deviation along the third axis.	$(\mathbb{R})$
N_b	:	number of blobs in the appearance model.	$(\mathbb{N})$
$\mathcal{B}$	:	set of blobs $\{B_1 \dots B_{N_b}\}$ in the appearance model.	
Jt_i	:	the i^{th} joint in the kinematic model.	
N_J	:	number of joints in the kinematic model.	$(\mathbb{N})$
l_i	:	length of the bone segment associated with joint Jt_i.	$(\mathbb{R})$
ω_i	:	axis of rotation for the joint Jt_i.	$(\mathbb{R}^3)$
θ_i	:	rotation angle of the joint Jt_i around ω_i.	$(\mathbb{R})$
θ_i^+	:	maximum rotation angle for the joint Jt_i.	$(\mathbb{R})$
θ_i^-	:	minimum rotation angle for the joint Jt_i.	$(\mathbb{R})$
$\dot{\theta}$	:	first derivative of the joint angle θ_i with respect to time.	$(\mathbb{R})$
Θ	:	set of all joint angles $\{\theta_1, \dots, \theta_{N_J}\}$.	$(\mathbb{R}^{N_J})$
$\dot{\Theta}$	:	set of all joint angles derivatives $\{\dot{\theta}_1, \dots, \dot{\theta}_{N_J}\}$.	$(\mathbb{R}^{N_J})$
$\boldsymbol{\Theta}$	:	set of all joint angles and their derivatives $\{\Theta, \dot{\Theta}\}$.	$(\mathbb{R}^{2\times N_J})$
P_0	:	global position of the root of the kinematic model.	$(\mathbb{R}^3)$
P_i	:	global position of a joint Jt_i.	$(\mathbb{R}^3)$
$\mathcal{P}$	:	vector of global positions of all joints.	$(\mathbb{R}^{3\times N_J})$
R_0	:	global orientation of the root of the kinematic model.	$(\mathbb{R}^{3\times 3})$
R_i	:	global orientation of a joint Jt_i.	$(\mathbb{R}^{3\times 3})$
G_i	:	goal position for the joint Jt_i.	$(\mathbb{R}^3)$
N_G	:	number of available goal positions.	$(\mathbb{N})$
$\mathcal{G}$	:	set of all goal positions.	$(\mathbb{R}^{3\times N_G})$
d_z	:	dimensionality of the observations.	$(\mathbb{N})$
$\mathbf{z}_t$	:	measurement or observation at frame t.	$(\mathbb{R}^{d_z})$
$\mathbf{Z}_t$	:	all available observations up to frame t.	$(\mathbb{R}^{t\times d_z})$
d	:	dimensionality of the parameter space.	$(\mathbb{N})$
$\mathbf{C}_t$	:	configuration of the model at frame t.	$(\mathbb{R}^d)$
$\mathbf{C}_t^i$	:	configuration of the particle i at frame t.	$(\mathbb{R}^d)$
$\mathbf{w}^i$	:	weight associated with the particle i.	$(\mathbb{R})$
N_p	:	number of particles.	$(\mathbb{N})$
k	:	index of a cluster of elementary movement.	$(\mathbb{N})$
$\mathcal{K}$	:	set of all the clusters of elementary movement.	
q	:	current state of the particle in the VLMM.	$(\mathbb{N})$
D_{KL}	:	Kullback-Leibler distance between distributions.	$(\mathbb{R}^{d_z}\times\mathbb{R}^{d_z}\rightarrow\mathbb{R})$

Chapter 1

Introduction

Markerless human-body tracking is a difficult problem which has been one of the important challenges of the Computer-Vision community for about 20 years. Since the earliest attempts [OB80, Hog83], progress have been made in all areas touched by human body tracking, but despite a great deal of attention in the recent years, the general problem remains unsolved. With this thesis, we intend to contribute to the state of the art in specific aspects of this complex problem. We shall particularly focus our efforts on the real-time tracking of structured motions, using multiple camera views. In this introductory chapter, we present the applications and motivations of our research, and then give an outline of the contributions and of the structure of the thesis.

1.1 Applications of Human Body Tracking

Full human-body tracking has a wide and promising range of applications. The film industry has been pioneering the need for motion capture since the emergence of realistic computer graphics. The capture and re-targeting of the movements of actors to animated characters is a very important application, used in films, but also in video games and in live broadcasts. Alternatively, computer-generated human figures necessitate realistic animations, which is very hard to achieve manually using modelling softwares. Using recent developments in statistical learning methods, it is possible to model and generate stylised motions [BH00]. However, these techniques still require large amounts of training data, usually acquired though motion capture.

The automatic tracking of human motions is important for surveillance and security. Computerised systems can detect suspicious patterns of behaviour [DH04] and trigger some alerts. Computers can also analyse movements for rehabilitative purposes [MPC$^+$05], such as assessing the recovery of patients. Motion analysis can help sportsmen locate their weaknesses, and improve their performances. For the general public, computers could become virtual teachers in activities such as dancing or sign-language, capable of both instructing students and correcting their errors.

Last but not least, motion capture finds exciting applications in smart offices or households [Coh98, Coh99, BMK$^+$00], where computers try to understand the intentions of humans. Movements and gestures are essential vehicles of communication, and recognising them is a milestone towards "human-aware" buildings. More generally, gestures can become an essential way to interact with computers, more natural and expressive than current computer-centred devices. The whole domain of computer interfaces could be reshaped by gesture-based interactions, which combined with speech, could allow users to interact freely with virtual objects. Video games are an obvious example of application that would greatly benefit from body tracking to enhance the immersion of the player. Likewise, tracking motions can be used to control realistic avatars in virtual environment. Social interactions would then be possible without the barrier of distance.

1.2 Motivation, Aims and Objectives

Commercial marker-based motion capture systems [Vic, Met] have been around for a few years. They can acquire movements at very high frequencies with good accuracy, which makes them an ideal tool for the film industry. However, these tracking systems are very invasive, typically requiring special clothing and a very controlled studio-like environment. Their high price also confines them to very specific applications, and prevents their wider adoption as a human-computer interaction technique.

By contrast, cameras are low-cost, flexible and non-invasive devices: their use for motion capture is a natural evolution towards ubiquitous computing. In the last few years, cameras have become commodity hardware and are increasingly integrated into various electronic equipment, from computers to mobile phones. Like the human eye, they can capture much more than the relative positions of the limbs. The information transmitted by cameras is very rich, and the main challenge in computer vision is to extract the small sub-set of information that is relevant for a particular application.

The non-invasiveness of cameras is probably the most important factor that could allow a wider adoption of full body-tracking setups. Ideally, the tracker should only rely on raw camera images, with no specific assumptions about the environment or the clothing of the subject. However, considering the state of the art, a solution to this general problem seems currently out of reach. Some constraints have therefore to be imposed onto the tracking environment. In our case, we assume a static environment and the availability of multiple camera views. In our opinion, these constraints are relatively easy to meet for indoor tracking, but restrict nonetheless the scope of possible applications.

Performance, although neglected by many researches, is a necessary condition for the system to be usable. Most applications are interactive, and demand a responsive tracking system. Even relying on the fast increase in computing power, trackers which currently run several orders of magnitude slower than real-time have little hope of wide adoption. An important aim of this research is to design a full-body tracker capable of running in real-time on commodity hardware. Many methodology choices are made with this efficiency constraint in mind. The notion of "real-time" is subject to interpretation, but throughout this thesis, we shall target a full system running at 10 Hertz on a single 2 GigaHertz computer. This targeted framerate represents a bare minimum for many applications, but would nonetheless allow interactive tracking of human motions.

Tracking people from camera images is difficult because of the high dimensionality of full body kinematics, the fast movements and frequent self-occlusions. Moreover, loose clothing, shadows, or camera noise may further complicate the inference problem. A robust tracker should cope with all these challenges, and provide ways of recovery whenever it fails. In this thesis, we attempt to develop original answers to all these issues. We finally demonstrate a full-body tracker running in real-time on a single computer, and robust enough to track challenging sequences of ballet dancing as acquired by low-quality webcams.

1.3 Summary of Contributions

Novel and original work presented in this thesis include:

- A fast hierarchical background segmentation technique, with robust classification of sets of pixel samples and handling of shadows.

- A volumetric reconstruction algorithm modelling uncertainty when combining the classifications from available camera views.
- The efficient inclusion of colour information in the voxel-based representation.
- The dynamic reorganisation of blobs to automatically learn the appearance model.
- An iterative tracking algorithm based on the positions and directions of blobs.
- In the context of Bayesian tracking, a prediction scheme based on variable length Markov models of behaviour.
- A fast evaluation framework for the particles, based on the cross-entropy between the blob models.

1.4 Thesis Outline

In this chapter, we briefly introduced the research field of human-body tracking and its applications.

Chapter 2 takes a statistical approach to the problem of background segmentation. We propose a model for robust classification of sets of pixels, and detail a hierarchical segmentation algorithm with significantly improved performance compared to per-pixel schemes. Chapter 2 also serves as an introduction to the volumetric reconstruction method, presented in Chapter 3. The volumetric reconstruction algorithm exploits multiple views to build hierarchically a voxel-based representation of the subject of interest, following the shape-from-silhouette paradigm. In Chapter 3, we detail the original aspects of our algorithm, and conclude with the inclusion of colour and some results.

Blobs are probabilistic descriptions of data-sets. In Chapter 4, we describe the use of 3-D Gaussian blobs as trackers for individual body-parts. The Expectation-Maximisation algorithm is chosen to relate the blobs to the voxels obtained from the volumetric reconstruction. In order to avoid manual initialisation of the appearance model, a scheme which automatically acquires the number of blobs and their repartition on the skeletal model is introduced.

Chapter 5 describes the parametrisation of the kinematic model, and presents a fast hierarchical tracking algorithm based on Inverse-Kinematics. The chapter concludes with some results highlighting both the benefits and the limitations of this method.

In an attempt to address some problems of the hierarchical approach, Chapter 6 introduces a global optimisation method based on particle filtering. The particles are propagated using the predictions of a variable length Markov Model, and evaluated using the blobs description from Chapter 4.

An overall evaluation of the tracking methods presented in this thesis is proposed in Chapter 7. Challenging sequences of ballet-dancing are used for quantitative and qualitative tests. Our method is also compared to other established tracking frameworks. Finally we conclude in Chapter 8 with a summary of the thesis and some suggestions of future work.

Chapter 2

Background Segmentation

When the cameras are fixed and the environment is relatively static, silhouettes are appealing visual cues, both robust and fast to extract. This chapter presents a novel hierarchical scheme for background segmentation, based on statistical analysis of the camera noise and robust classification of sets of samples. After a general introduction to background segmentation and a presentation of the state of the art, a Gaussian model of background pixels is described. Segmentation is then possible on individual pixels, but also on sets of pixel samples, leading to a hierarchical scheme. Shadows and changes in lighting are handled in a computationally efficient way. The chapter concludes with qualitative results and a performance evaluation.

2.1 Introduction

Segmenting an image means labelling each of its pixels as belonging to zones or classes of the image. Background segmentation is a sub-case of general segmentation, where one is only interested in a binary classification of the pixels: either they belong to the object of interest, or they do not. Different synonymous names can be found in the literature for "background segmentation", like "background subtraction", "foreground segmentation" or more generally "image segmentation". The term "background" refers to every pixel in the image that does not belong to the object of interest.

Background segmentation is used in numerous tracking algorithms (see [MG01] for a survey) because it discards the background region and isolates the object of interest. Unaffected by irrelevant features, tracking is then simpler and more robust. An important advantage of background segmentation over most other image cues is its capacity to handle low-quality and blurry images. Motion-blur is indeed omnipresent in motion-capture, giving a hard time to edge or texture-based detectors. Background segmentation, however, is almost unaffected by blurry regions. The resulting silhouettes encode the shape and the pose of the subject in a simple way, and constitute reliable cues for tracking.

A single silhouette is often too ambiguous to infer the pose of a human subject, especially considering the frequent self-occlusions characterising human motions. Multiple views are then frequently used, either through projection and evaluation of the fitness of the model [CTMS03, GSD03b, DF99], or 3-D reconstruction [MTHC03, LSL01, CKBH00]. The need for real-time multiple silhouettes extractions imposes strong performance constraints on background segmentation techniques. These techniques are usually kept simple enough to run in real-time while leaving enough spare processing time for subsequent algorithms.

The main inconvenience of background segmentation is the need to build a model of the background. In most cases this implies capturing the empty scene prior to the tracking and keeping cameras immobile during the tracking. These constraints are not too restrictive for an indoor office-like environment but applications such as tracking from archive video footage are ruled out. Another challenging problem is that backgrounds do not usually remain totally static during long capture durations because of changes in lighting or moving objects. A way to update dynamically the background model during the actual segmentation is needed.

In the literature, classification is often performed independently on each pixel by comparing the current colour of a pixel with a model of the background. A standard framework for background segmentation is presented in Algorithm 2.1. In this algorithm, a model for each background pixel is first built from an empty scene, and the actual segmentation is then performed by comparing current pixel values with the background model. This scheme is followed by most of the real-time image segmentation methods, despite using more or less complex models of the background. The choice of the model must be a compromise between an over-simplified one, leaving many pixels misclassified, and a too complex one where the evaluation of the distance between the current pixels and the background model would be too complex for real-time execution.

Image segmentation is often performed on a per-pixel basis, ignoring all higher

Algorithm 2.1: Basic framework for pixel-based background segmentation.

```
▷ Background acquisition with an empty scene;
foreach training image of the background do
    foreach pixel do
        build and update the background model (Section 2.3);
    end
end

▷ Online background Subtraction;
foreach new input frame do
    foreach pixel do
        compute the distance between the current pixel colour and the
        background model (Section 2.4);
        if distance > threshold then
            pixel is labelled foreground;
        else
            pixel is labelled background;
            update the model with the current pixel;
        end
    end
end
```

level structure. A justification is that image segmentation is performed as a pre-processing step, and classification errors can be recovered from at later stages. Nevertheless, foreground regions are often contiguous, and spatial coherency can be exploited to improve performance and robustness of background segmentation. When introducing a spatial prior, one must be careful not to bias the estimation. In this chapter, we shall introduce a novel hierarchical scheme, performing classification on sets of samples in an unbiased way.

If cameras were perfect *noiseless* devices, background segmentation would be trivial: any kind of distance between a single image of the background and the current frame would exhibit the object of interest, and thresholding values above zero would finish the algorithm. Unfortunately cameras have inherent noise which cannot be ignored. The theoretical way to handle noise is first to measure its distribution, then to model it and finally to design robust statistical methods dealing with it.

After a presentation of related work on background segmentation, this chapter starts with the elaboration of a statistical model for the pixels of the background (Section 2.3). The statistical tools to perform segmentation on individual pixels and on sets of pixels are then introduced in Section 2.4, leading to a hierarchical segmentation scheme. Section 2.5 then presents a simple and efficient shadows removal algorithm. Finally, a comparative evaluation of our method followed by a discussion concludes the chapter.

2.2 Related Work

The most basic methods reported in the literature (such as [BL01b, TMSS02, Sze90, CKBH00]) do not attempt to model noise. A simple subtraction is performed between the current pixel's intensity and the corresponding pixel in a reference frame of the background. The result is then thresholded with an arbitrary chosen value. While this approach is very fast, it only gives good results if the contrast between the foreground object and the background is high. The segmentation of human body with standard clothing in a cluttered environment using this simple scheme is poor.

Wren *et al.* [WADP97] use a more advanced model of the background, where the YUV colour of each pixel is modelled by a trivariate Gaussian distribution. The number of training background samples is arbitrary, but the method is robust and runs in real-time. A single Gaussian distribution is enough to model static backgrounds, but applications like road traffic control need to account for more variability in the background model. A logical extension is then to use more than one component per pixel: Friedman *et al.* [FR97] and Stauffer *et al.* [SG99] both model each background pixel by a mixture of Gaussians. The Gaussian models are learnt using an Expectation-Maximisation framework. While this approach is theoretically appealing and gives reliable results in difficult cases, it is too general (and hence computationally expensive) for a simple, mostly static, office environment. The algorithm of Stauffer *et al.* [SG99] will be evaluated in Section 2.6.

Another interesting extension was proposed by Javed *et al.* [JSS02] with a framework for hierarchical segmentation. The segmentation is done successively at 3 levels: at the pixel level, a standard mixture of Gaussians is used in conjunction with gradients to classify each pixel as belonging to background or foreground. At a "region" level, individual misclassifications are recovered from, as a foreground region is assumed to be continuous and bounded by high-gradient values. Finally, the frame level accounts for global illumination changes. The whole framework seems particularly well adapted to outdoor monitoring, but unfortunately no performance evaluation is provided.

When multiple views are available, some researchers thought of including depth information in the model of the background. Harville *et al.* [HGW01b, HGW01a] use a standard mixture of Gaussians in YUV space for colour information, and add depth collected from disparity in the Gaussian models. The method is targeted at pedestrian monitoring applications, and the learning rate of the model is modulated by the detected "activity" at the corresponding pixels. Similarly Taycher and Darell [TD02] perform background segmentation simultaneously on multiple views using a 3-D model of the background previously acquired using range scanning lasers. An object is then classified as foreground if it occludes the background. These methods are very interesting, especially in our case where multiple views are available, but are either too computationally demanding (computing disparity) or not flexible enough (the need to acquire a 3-D model of the background).

Using prior knowledge about foreground objects, Zhao and Nevatia [ZN02] propose a Bayesian framework where each pixel is segmented using both a colour model and the prior probability that it belongs to an object-level model. This scheme is applied to human monitoring where the heads of people are first detected, and some assumptions are then made about the expected position of the body. The facts that humans stand mostly vertical, and that their size can be bounded by perspective projection of the head position are exploited. Unfortunately, the same kind of scheme cannot easily be applied to full human body tracking because the positions of the limbs are difficult to predict. Using the tracked position from the previous frame as a model for Bayesian segmentation could be an idea, but since we are in turn using the segmentation for estimating the body pose, this would lead to a biased estimation.

2.3 A Model for Background Pixels

In this section, we propose a statistical model for pixels belonging to the background. A formal definition of the noise, and a measurement protocol are first introduced. The pixels' colour representations are also discussed before starting the actual experimental noise measurements. From the obtained results, a model is proposed, along with a quantitative error estimation based on the number of training images.

2.3.1 Hypotheses and Formulation

Understanding camera noise should be the very first step for designing vision algorithms. In this section, measurements of the camera noise will be used to propose a statistical model of pixels' variations. While not demonstrated here, we will assume that pixels are statistically independent from each other. Indeed, the hardware camera sensors being independent for each pixel, it is reasonable to assume the same independence in the output image. Moreover, the noise associated with a given pixel does not depend on the previous values of this pixel (in other words, this is *uncorrelated noise*). We will also assume that noise properties are not correlated with the position of the pixel on the image. Some of these assumptions might prove false in certain specific circumstances, but they constitute a reasonable basis of work for our purpose.

Because each pixel is constantly affected by noise, we need to measure its colour values over multiple frames to get an estimate of its true (noiseless) value, μ. Let $\{u_1 \dots u_{N_u}\}$ be some measured colour values of the same pixel over N_u time steps, and $\{w_1 \dots w_{N_u}\}$ the noise vectors associated with each measurement.

$$\forall i \in [1 \dots N_u], u_i = \mu + w_i \tag{2.1}$$

We assumed that the noise distribution has a zero-mean, but since we have no ground truth for the pixel's value, there is no way to measure a possible bias in the noise. Actually, even if the noise was biased, we would simply consider the real value plus the bias as the correct value for the pixel. So, if N_u is a statistically significant number of samples we write:

$$E[w] = E[\{w_1, \dots, w_{N_u}\}] = 0 \tag{2.2}$$

where $E[\bullet]$ denotes the expected value of a statistical variable.

2.3.2 Measuring the Noise

Regardless of the underlying processes generating the camera noise at each pixel, the sum of pixel samples over a sufficient number of frames closely follows a Gaussian (or Normal) distribution. This powerful result comes from the Central Limit Theorem [Fel45] which states that the sum of a large number of independent, identically distributed random variables follows asymptotically a Gaussian distribution. In our case, each sample is a 3-Dimensional vector (dimensionality of the colour-space) so that the sum of sample values follows a trivariate Gaussian distribution:

$$\sum_{i=1}^{N_u} u_i \sim \mathcal{N}_3(N_u.\mu, N_u.\Sigma_w) \tag{2.3}$$

where Σ_w is the covariance matrix of the noise, which is not known yet. In particular, it follows from Equation 2.3 that the true value of the pixel, μ, is the expectation of the mean of the samples:

$$\mu = E[\frac{1}{N_u}\sum_{i=1}^{N_u} u_i] \tag{2.4}$$

Actually, we chose the properties of the noise (null expectation) for this exact purpose. More interestingly, we are now capable of evaluating the uncertainty, or variance, associated with the measurement:

$$Cov[\frac{1}{N_u}\sum_{i=1}^{N_u} u_i] = \frac{\Sigma_w}{N_u} \tag{2.5}$$

It is worth noticing that the variance of the measurement decreases linearly with the number of samples. Note that this same result can also be obtained with the Cramer-Rao lower bound. This result is interesting because it will allow us to obtain a good estimate of μ with an affordable number of samples. Nevertheless, in order to evaluate this "sufficient" number of samples, we need to measure the noise covariance (Σ_w). This is not a direct and easy measurement, since the true value of each pixel (μ) is needed to determine the quantity of noise in the samples (Equation 2.1).

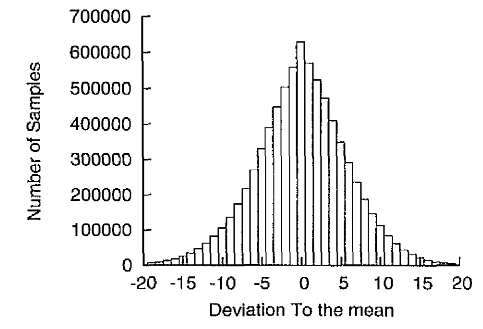
As seen in Equation 2.4, the maximum-likelihood estimate of μ over the samples $\{u_1 \ldots u_{N_u}\}$ is the mean of the samples, and the uncertainty associated with this estimator is modelled by the covariance matrix from Equation 2.5. Another way to put it is to see the mean of the samples as a random vector following a trivariate Gaussian distribution of mean μ and covariance matrix Σ_w/N_u. The proposed experimental approximation consists in taking a statistically large number of samples ($N_u \simeq 1000$), so that the uncertainty in the measurement is minimal (Equation 2.5), and thus having:

$$\mu \simeq \frac{1}{N_u}\sum_{i=1}^{N_u} u_i \tag{2.6}$$

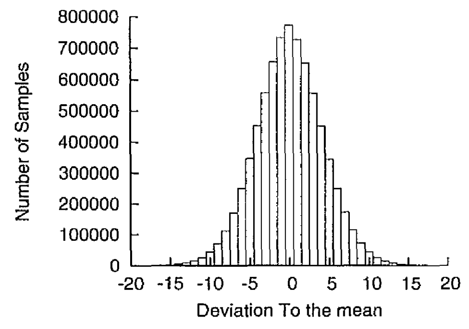
From this and Equation 2.1, obtaining the noise vector for each sample is straightforward:

$$w_i = u_i - \mu \simeq u_i - \frac{1}{N_u}\sum_{i=1}^{N_u} u_i \tag{2.7}$$

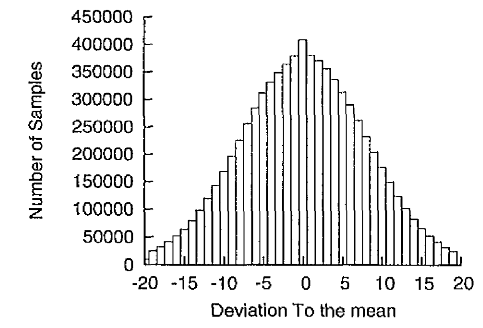
The next sections exploit this equation to study the properties of the noise, and we propose a model accounting for pixels' variations.



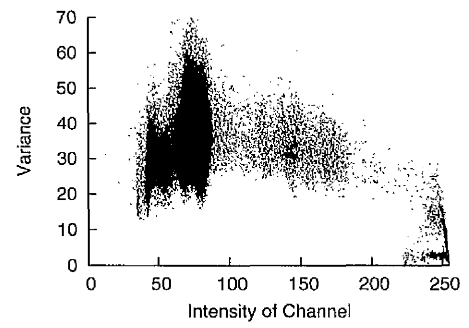
(a) Noise for the Red Channel. Measured standard deviation $\sigma_{w_R} = 4.89$.



(b) Noise for the Green Channel. Measured standard deviation $\sigma_{w_G} = 3.69$.



(c) Noise for the Blue Channel. Measured standard deviation $\sigma_{w_B} = 7.91$.



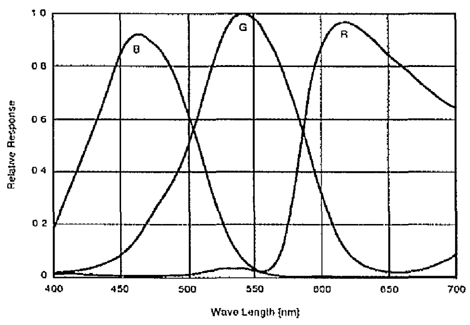
(d) Correlation between noise variance and colour intensity for the Red channel

Figure 2.1: *Noise distribution in RGB colour-space.*

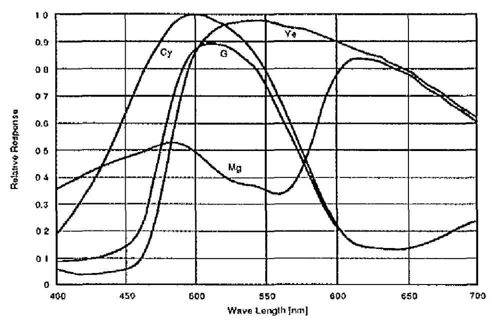
2.3.3 Measurements and Interpretation

The first measurements presented in Figure 2.1 show the characteristics of the noise in RGB colour-space (see Appendix A for an overview of colour-spaces). The noise on each colour channel was measured independently as in Equation 2.7. The first graphs (Figure 2.1(a)–(c)) are cumulative noise values over all the pixels of the image, for 100 image samples ($N_u = 100$). The resolution of the camera was set to 320×240 for this test, which generated over 7.5 million measurements of the pixel values. The distribution of the noise can then be deduced for each colour channel.

It can be noticed that the distributions for each colour component have roughly a Gaussian shape, even if closer inspection tends to reject this assertion. Actually, these distributions are, at best, sums of Gaussian distributions, all zero-centred but with different variances: Figure 2.1(d) exhibits the relation between the variance of the noise and the colour intensity for the Red colour channel. There is a correlation between colour intensity and noise, darker pixels tending to suffer from more noise



(a) Sony Wfine CCD$^{\mathrm{TM}}$.



(b) Sony Super HAD CCD$^{\mathrm{TM}}$.

Figure 2.2: *Spectral sensitivity characteristics of some common camera sensors (reproduced from Sony camera specifications). In order to widen the recovered spectrum, some sensors use more than 3 input filters, which are combined to form the output.*

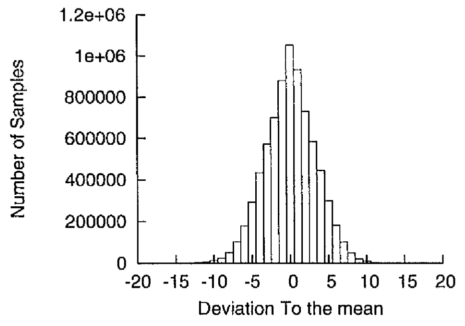
than brighter ones.

The motivation for these measurements was to quantify the variance of the noise, ${\sigma_w}^2$, and this has been done for each colour channel independently. The covariance matrix linking the noise variations for each colour channel was measured over 1000 image samples:

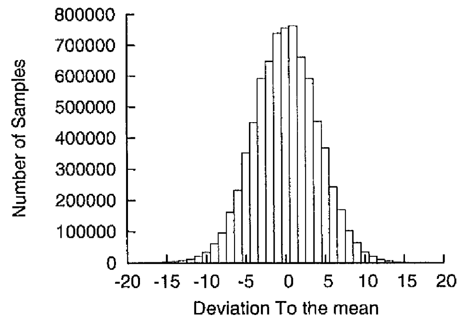
$$Cov(w_{RGB}) = \begin{pmatrix} 23.87 & 1.93 & 10.30 \\ 1.93 & 13.63 & 2.91 \\ 10.30 & 2.91 & 62.51 \end{pmatrix}$$

The diagonal elements are the variances for Red, Green and Blue channels. The other values represent the dependencies between variations due to the noise on different channels. We can notice, for example, that the noises on the Red and Blue channels are strongly interdependent. This is explained by the fact that RGB components are computed as a transformation of an internal hardware colour representation which matches the camera sensors. For every model of camera, the range of the sensors never matches exactly the output colour space (Figure 2.2). A correction is then needed, where output colour components are computed as a non-linear combination of sensor measurements. Some noise on a camera sensor is then reported to more than one output colour component, hence the observed dependencies.

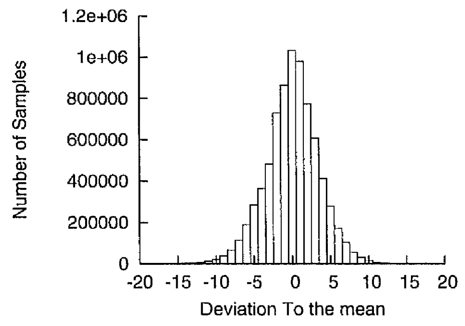
The exact same measurements in YUV colour space are reported in Figure 2.3. It is very noticeable that noise distributions on YUV channels are narrower than on RGB channels. The standard deviations are consistently smaller, meaning that the noise is weaker in YUV colour space. These results should however be taken carefully: an



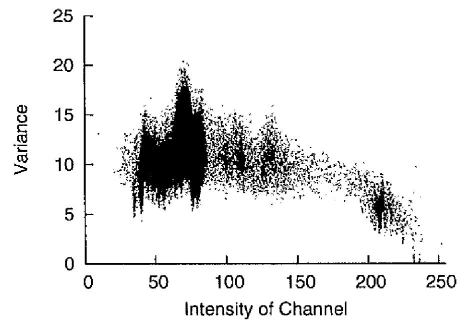
(a) Noise distribution for the Y channel. Measured standard deviation $\sigma_{wY} = 3.07$.



(b) Noise distribution for the U channel. Measured standard deviation $\sigma_{wU} = 3.89$.



(c) Noise distribution for the V channel. Measured standard deviation $\sigma_{wV} = 2.73$.



(d) Correlation between noise variance and colour intensity for the Y channel.

Figure 2.3: *Noise distribution in YUV colour-space.*

important point to consider is that the YUV space is "smaller" than the RGB one. For example, the value $(255, 255, 255)$ is possible in RGB, but not in YUV. This alone could explain the narrower noise variances observed in YUV space. The measured covariance matrix is:

$$Cov(w_{YUV}) = \begin{pmatrix} 9.43 & 1.42 & 0.04 \\ 1.42 & 15.15 & -0.68 \\ 0.04 & -0.68 & 7.48 \end{pmatrix}$$

Dependencies are minimal, which gives an advantage to the YUV colour space with respect to noise (at least with our model of cameras). There is a small dependency between noise on Y and U channels (correlation coefficient equal to $\frac{\sigma_{wYU}}{\sqrt{\sigma_{wYY} \cdot \sigma_{wUU}}} = 0.12$), but still weaker than the dependencies observed in RGB colour space. From these results, we choose the YUV colour-space, both for of its good performance toward noise and for the separation of luminance which will prove useful to handle shadows.

2.3.4 Statistical Model for the Background Pixels

The noise distributions measured in Figure 2.3 represent the cumulative noise for all the pixels in the image. However the properties of the noise depend on the colour value of the pixel, and also on other parameters such as its position in the image. So, even if it can provide an overall evaluation of the noise properties, the cumulative noise distribution for all pixels is inadequate to find a model for the noise for individual pixels.

Measuring the noise distribution for each pixel, in every condition would be an impossible task. A Gaussian model for noise has been widely and successfully used in the literature. Within this research, we therefore assume a Gaussian model for the camera-noise, and justify this choice *a posteriori* by the correspondence of the measurements to the model. A pixel sample s belonging to the background is then modelled by a trivariate Gaussian distribution of mean vector μ and covariance matrix Σ_w:

$$s \sim \mathcal{N}_3(\mu, \Sigma_w) \tag{2.8}$$

where, as defined in Equation 2.6:

$$\mu \simeq \frac{1}{N_u}\sum_{i=1}^{N_u} u_i \qquad \text{and} \qquad \Sigma_w = \frac{1}{N_u}\sum_{i=1}^{N_u}(u_i-\mu)(u_i-\mu)^T \tag{2.9}$$

The probability density function of the trivariate Gaussian distribution is then defined as:

$$f_{\mathcal{N}_3}(s,\mu,\Sigma_w) = \frac{1}{(2\pi)^{\frac{3}{2}}\sqrt{|\Sigma_w|}}e^{-\frac{1}{2}(s-\mu)\cdot\Sigma_w{}^{-1}\cdot(s-\mu)^T} \tag{2.10}$$

2.3.5 How Many Training Frames?

Having found an appropriate model for the pixels of the background, we still need to know how this model should be built. Estimating the mean and covariance matrix of a Gaussian distribution is straightforward, but these estimations have to be trusted for subsequent analysis to be meaningful. In this section, we estimate the number of training frames required in order to derive an accurate model of the background. This question is not insignificant, since, as we shall see, an inadequate number of training frames can result in significant errors.

Using the noise covariance matrix previously measured in Section 2.3.3, we can

Confidence level (α):	50%	68%	80%	90%	95%	98%	99%	99.7%
Constant z_α:	0.67	1.00	1.28	1.64	1.96	2.33	2.58	3.00

Table 2.1: *Values of the constant z_α, in number of standard deviations, for two-sided α confidence intervals with Gaussian distributions [Mit97].*

compute its standard deviation:

$$\sigma_w = \sqrt{|Cov(w)|} \tag{2.11}$$

In order to get the maximal error, we are interested in an upper bound of the standard deviation, and thus of the variance. By definition, the maximal variance of a distribution lies along the first eigenvector of the covariance matrix. Put another way, the maximal variance is the greatest eigenvalue of the corresponding covariance matrix. We can then estimate the standard deviation of the noise for our camera in YUV colour space: $\sigma_w = \sqrt{15.54} = 3.94$. Following Equation 2.5, the standard deviation σ_e of the measurement over N_u samples can now be estimated:

$$\sigma_e = \sqrt{var[\frac{1}{N_u}\sum_{i=1}^{N_u} u_i]} = \frac{\sigma_w}{\sqrt{N_u}} \tag{2.12}$$

One of the interesting properties of the Gaussian distribution is that confidence intervals are well known (Table 2.1) and allow straightforward error estimation. In particular, if we consider the error between the actual measurement and the expected colour of a pixel:

$$error = \left|\frac{1}{N_u}\sum_{i=1}^{N_u} u_i - E[\frac{1}{N_u}\sum_{i=1}^{N_u} u_i]\right| = \left|\frac{1}{N_u}\sum_{i=1}^{N_u} u_i - \mu\right| \tag{2.13}$$

then, this error follows a normal distribution of standard deviation σ_e. For a particular background pixel, this error has α percent chances of being lower than $z_\alpha.\sigma_e$ where z_α is the constant defined in Table 2.1. Table 2.2 reports the maximal errors ($z_\alpha.\sigma_e$) for different confidence levels α, and number of background samples N_u.

To analyse Table 2.2, we should keep in mind that the measured standard deviation of the noise as given by Equation 2.11 was $\sigma_w = 3.94$. Even if the noise itself does not necessarily follow a Gaussian distribution, it means that measured pixel values are typically that distant from their expected value, μ. It is very important to minimise the

α	Samples N_u / Std dev. σ_e	1 / 3.94	5 / 1.76	10 / 1.25	25 / 0.79	50 / 0.56	100 / 0.39	1000 / 0.12
90%	Max. error	6.62	2.96	2.09	1.32	0.94	0.66	0.21
	percentage	2.59%	1.16%	0.82%	0.52%	0.37%	0.26%	0.08%
95%	Max. error	7.73	3.45	2.44	1.55	1.09	0.77	0.24
	percentage	3.02%	1.35%	0.95%	0.60%	0.43%	0.30%	0.10%
98%	Max. error	9.18	4.11	2.90	1.84	1.30	0.92	0.29
	percentage	3.59%	1.60%	1.13%	0.72%	0.51%	0.36%	0.11%
99.7%	Max. error	11.82	5.29	3.74	2.36	1.67	1.18	0.37
	percentage	4.62%	2.07%	1.46%	0.92%	0.65%	0.46%	0.15%

Table 2.2: *For a given number of image samples, N_u, this table gives the maximal error which is expected not to be exceeded α percent of the time. The percentage rows show how this maximal error relates to the range of possible values for the colour channel.*

error between actual measurements and expected values of the pixels because this error is additive with the noise itself and introduces bias for each measurement. As a rule of thumb, considering that the accuracy on each channel is hardly greater than $1/256^{th}$ of the intensity range (colours are often encoded with integer values), choosing a maximal error between 1 and $\sigma_w/2$ is sensible. This maximum of $\sigma_w/2$ is purely indicative, but reflects the fact that the uncertainty of the mean of the model should be significantly lower that its variance. The value of α has to be chosen carefully too, considering the very high number of pixels in an image. Taking $\alpha = 90\%$ for example means that there will statistically be 10% of outliers. This could be alright if the subsequent algorithms are very robust to outliers, but in general, confidence values greater than 98% should be considered.

Best compromises between accuracy and practical capture duration constraints can be found for $50 \leq N_u \leq 100$, where the error in the expected colour of the pixels is relatively small compared to the noise itself. Such numbers of samples can be reached quickly in most systems with capture rates being usually greater than 15 frames per second. Fewer samples can be used if the capture time is very limited, but $N_u = 10$ sample frames seems a strict minimum for tackling the noise.

2.4 A Metric for Segmentation

In Section 2.3, we have defined a Gaussian model of the background for each pixel of the image. Our aim is to segment the image into a background and a foreground region.

For this purpose, we need a metric, or "distance" to evaluate how close a given pixel is to the background model. This section defines such a distance between pixels and the background models, and shows how thresholds on this distance can be derived. Segmentation of individual pixels is extended to a set of pixel samples, eventually leading to a novel hierarchical segmentation scheme.

2.4.1 Mahalanobis Distance

We saw in Section 2.3.4 that a background pixel s is modelled by a trivariate Gaussian distribution of mean vector μ and covariance matrix Σ_w. The exponent of the exponential is a quantity that characterises the distance between the current sample and the mean of the distribution. It is called the Mahalanobis distance, and is denoted as $D_M()$:

$$D_M(s, \mu, \Sigma_w) = (s - \mu) \cdot {\Sigma_w}^{-1} \cdot (s - \mu)^T \tag{2.14}$$

This formulation is relatively similar to a standard Euclidean distance with an additional normalisation using the covariance matrix of the distribution. In particular, when the covariance matrix Σ_w is equal to the identity, the Mahalanobis distance is equal to the squared Euclidean distance. In practice, it means that the square root of the Mahalanobis distance measures the number of standard deviations between a sample and the mean of the Gaussian distribution. This normalisation is very useful because all the pixels on the image can now be compared with a similar metric. A common threshold value can therefore be found based on the Mahalanobis distance. Note that the mahalanobis distance can also be defined between two arbitrary points by replacing the mean of the distribution in Equation 2.14 by the second point of interest.

2.4.2 Segmentation of Individual Pixels

Since each pixel sample is composed of 3 channels (YUV), the distribution of Mahalanobis distances naturally accounts for these 3 degrees of freedom. We also use the fact that the Mahalanobis distances between a Gaussian model and the samples generated from this model follow a Chi-Square distribution. The Mahalanobis distance between a background pixel and its model of the background is a random variable following a Chi-Square distribution with $d = 3$ degrees of freedom (χ_3^2).

$$D_M() \sim \chi_3^2 \tag{2.15}$$

The Chi-Square distribution can also be seen as a model for the normalised "strength" of the noise. The probability density function of a Chi-Square distribution with d degrees of freedom is given below for $x > 0$:

$$f_{\chi^2_d}(x) = \frac{1}{2^{\frac{d}{2}}\Gamma(\frac{d}{2})} x^{\frac{d}{2}-1} e^{-\frac{x}{2}} \tag{2.16}$$

where

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt \qquad \text{and in particular, for } d{=}3: \qquad \Gamma(\tfrac{3}{2}) = \frac{\sqrt{\pi}}{2}$$

A comparison between the χ^2_3 distribution, and the actual measured Mahalanobis distances for background pixels over 20 frames (over 1.5 million samples) is presented in Figure 2.4. As we can see, measurements coincide very closely with our theoretical assumptions, which constitutes the *a posteriori* justification for using Gaussian distributions to model background pixels.

We can also notice from Figure 2.4 that the probability of measuring a small Mahalanobis distance (less than 1) is comparatively small: it is indeed very unlikely to observe minimal noise on the three channels simultaneously. However, this particularity of the distribution can be ignored for classification purpose because even if samples with very low Mahalanobis distances are unlikely, they should still be considered as belonging to the background model.

In order to find the most appropriate threshold, some values of the probability density function of the Chi-Square distribution are presented in Table 2.3. The last line shows the values of the cumulative distribution function of χ^2_3, highlighting the relatively good "compactness" of the Mahalanobis distance. For example, choosing a threshold of $D_M() < 15$ means that 99.82% of background pixels should be correctly classified. Likewise, for any desired confidence level α, we can choose a threshold $T_d(\alpha)$ on the Mahalanobis distance using the cumulative density function of χ^2_d:

$$\int_0^{T_d(\alpha)} f_{\chi^2_d}(t)dt = \alpha \quad \Rightarrow \quad P(D_M(s) < T_d(\alpha)) = \alpha \tag{2.17}$$

Of course, this level of statistical misclassifications assumes a perfect (or at least very good) model of the background, which, as we saw in Section 2.3.5, is hard to achieve. In practice, using between 50 and 100 training frames increases the rate of misclassifications by only a few percent.

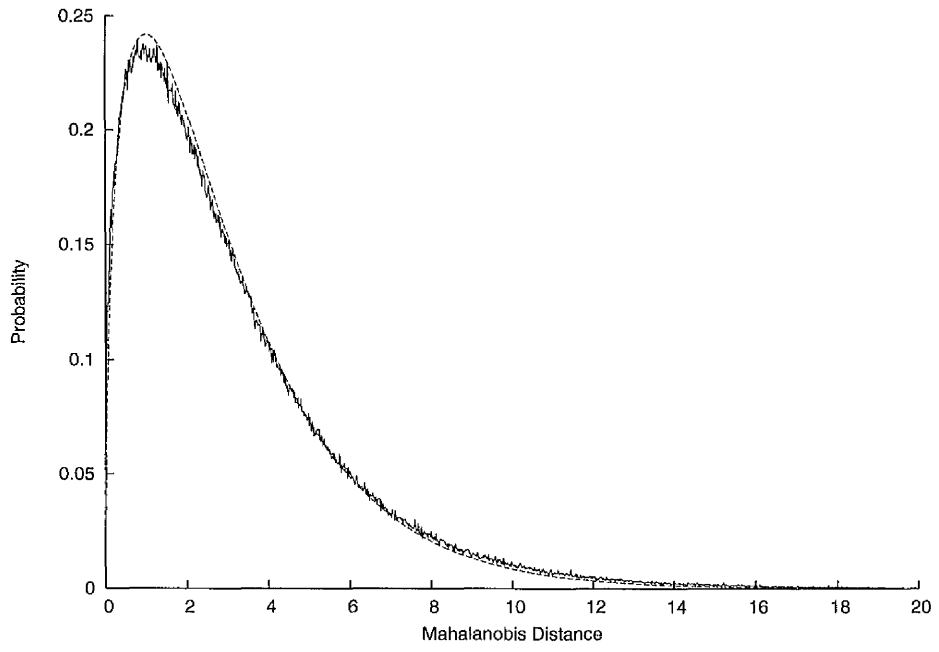


Figure 2.4: *The dashed line is the probability density function of the χ^2 distribution with* 3 *degrees of freedom. The solid line represents the measured cumulative Mahalanobis distances for around* 1.5 *millions of background pixels.*

x	1	3	9	12	15	18
$f_{\chi_3^2}(x)$	0.242	0.154	0.013	0.0034	0.00085	0.00021
$\int_0^x f_{\chi_3^2}(t)dt$	0.199	0.608	0.971	0.9926	0.9982	0.9996

Table 2.3: *Some values of the probability density function and cumulative distribution function for the Chi-Square distribution with 3 degrees of freedom.*

An example of background segmentation with a threshold of $D_M() < 18$ is presented in Figure 2.5. Note that the environment is cluttered and the clothing of the subject is very similar to some elements of the scene. Shadows are obviously the main difficulty: some methods to reduce their effect will be presented in Section 2.5. The rest of the segmentation is relatively good considering the quality of the cameras and the absence of post-processing. A few samples are still misclassified (given our model and Table 2.3, an average of 31 pixels should be misclassified for a 320×240 image and a threshold of 18) but this is not really a concern, for two reasons. Firstly, we are using multiple cameras for 3-D reconstruction, and there is very little chance to observe these random misclassified samples at coherent positions across the different views. The misclassifications due to noise will then naturally be discarded during the reconstruction process (as described in Section 3.3.2). The second reason is that pixels

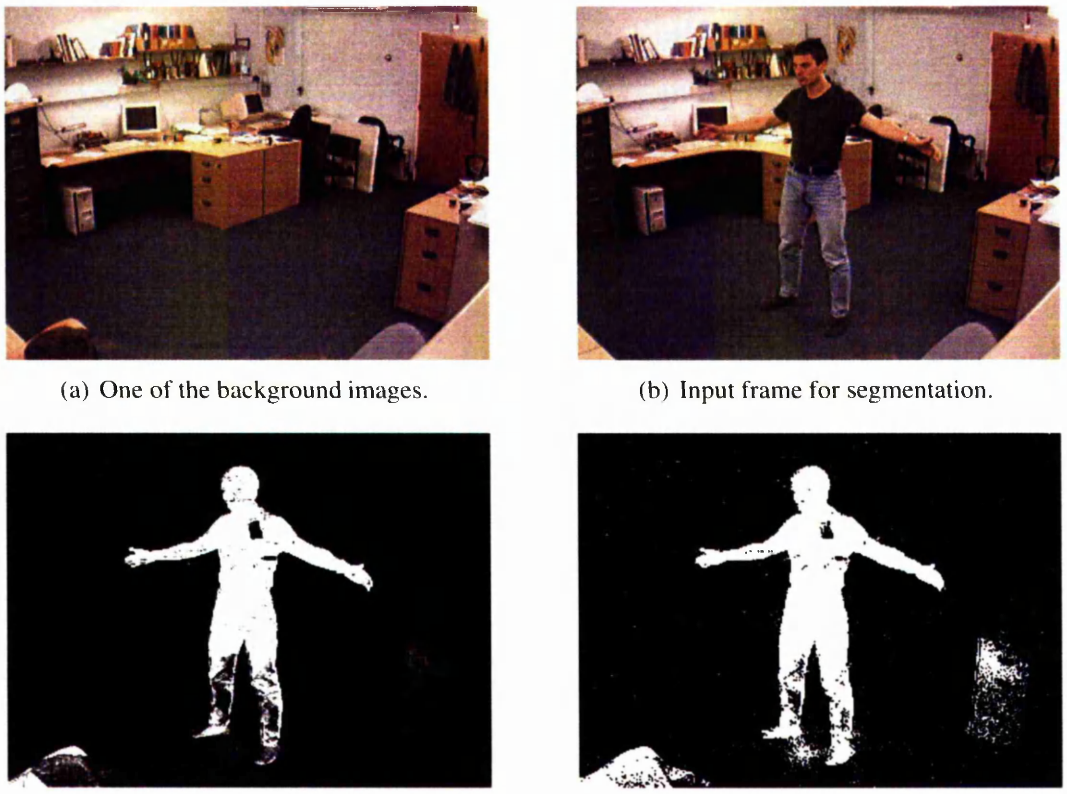

(a) One of the background images. (b) Input frame for segmentation. (c) Mahalanobis distances as grey-levels. (d) Binary segmentation at $D_M() < 18$.

Figure 2.5: *Segmentation with* 100 *background "training" images. Note that the background is not empty during training, which has consequences on the segmentation. Shadows are the main cause of misclassifications, but using a threshold of* 18 *still leaves a few random samples misclassified.*

will not be segmented individually, but a decision will be taken about a set of uniformly distributed samples instead. This scheme is detailed in the next section, but intuitively speaking, the probability of misclassifying a set of samples should be lower than with a single sample because of the spatial incoherence of noise.

2.4.3 Classification of a Set of Samples

Let us now consider a set of pixel samples $\mathcal{S} = \{s_1 \dots s_{N_s}\}$ spatially distributed in the current image frame. Our goal is to decide with reasonable confidence whether the whole set of samples $\mathcal{S}$ belongs to the background, the foreground, or an edge. It is said to belong to an edge if some of its pixels belong to the background while others belong to the foreground. The classification of a set of samples is an important problem because it will allow us to take decisions about whole areas of the image without segmenting each individual pixel in a binary way. We claim that classifying sets of samples is statistically more robust and more efficient than standard "per-pixel" classification techniques. This new scheme will be the basis of voxel classification presented in Section 3.3.2.

The classification has to be done for the whole set of samples, and not independently on each individual sample. Indeed, the likelihood that $\mathcal{S}$ has been generated by the background model is the product of the probabilities that each sample s_i belongs to the background model. As a simple example of this, let us consider a set of 8 samples, each having a 60% probability of belonging to the background. Segmenting the samples independently from each other leads to classify all of them as background, and consequently to classify the whole set as background. However, the real probability that the whole set of samples belongs to the background is only $(0.6)^8 \simeq 1.7\%$, which does hardly constitute enough evidence to discard the set of samples straight away.

In order to derive probabilities for $\mathcal{S}$, we need to to be able to compare the distributions of individual samples on a similar basis. The Mahalanobis distance (Equation 2.14) can be used as a point of comparison, since it has the interesting property of being normalised with respect to the initial Gaussian distribution. As we saw in Section 2.4 the Mahalanobis distance $D_M(s_i)$ between a pixel sample s_i and a trivariate Gaussian model follows a Chi-Square distribution with 3 degrees of freedom.

We could threshold these individual Mahalanobis distances, but our goal is to classify the whole set of samples $\mathcal{S}$. The likelihood that the whole set $\mathcal{S}$ has been generated by the background model is the product of the individual probabilities for each sample. Looking back at Equation 2.10, we can see that because of the exponential, a product of Gaussian probabilities corresponds to a sum of Mahalanobis distances. The total "distance" of $\mathcal{S}$ to the background model is then the sum of the individual Mahalanobis distances at each sample:

$$D_M(\mathcal{S}) = \sum_{i=1}^{N_s} D_M(s_i) \tag{2.18}$$

The Chi-Square distribution has the interesting property that the sum of independent variables, each following a Chi-Square distribution, follows itself a Chi-Square distribution with a number of degrees of freedom equal to the total degrees of freedom in the independent variables. Practically, this means that $D_M(\mathcal{S})$ follows a Chi-Square distribution with $3 \times N_s$ degrees of freedom, where N_s is the number of pixel samples

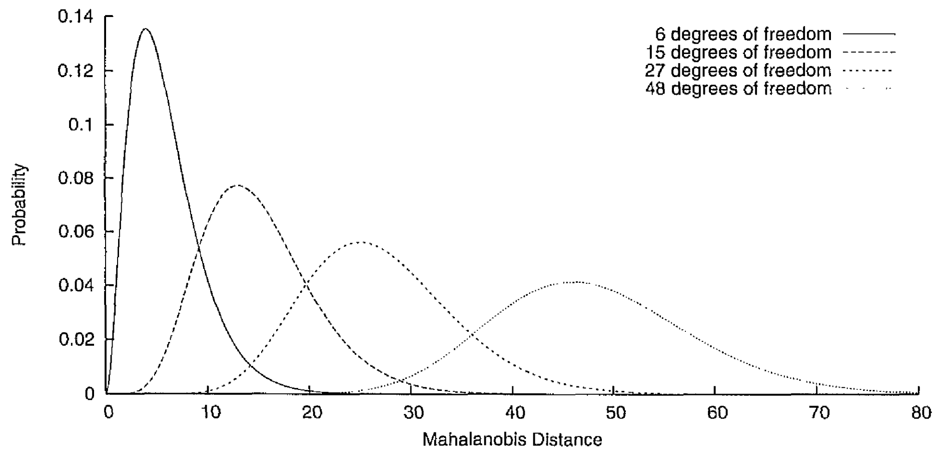


Figure 2.6: *Chi-Square distribution for 2 trivariate samples (6 degrees of freedom), 5 samples (15 dofs), 9 samples (27 dofs) and finally 16 samples (48 dofs).*

in the set $\mathcal{S}$:

$$D_M(\mathcal{S}) \sim \chi^2_{3.N_s} \tag{2.19}$$

The probability density function of a general order Chi-Square distribution was given in Equation 2.16, and a graphical representation of the distribution for various number of samples is shown in Figure 2.6. The graph shows that the uncertainty in the measurement (broadness of the distribution) increases with the number of samples. Nevertheless, by considering the set of samples as a whole, this uncertainty increases slower than when segmenting each sample independently. In practice, we saw in the previous section that a threshold of $D_M() < 18$ was adapted to threshold single pixels. One might naively combine these individual thresholds to classify the set, which would amount to a "virtual threshold" of $18 \times 9 = 162$ for a set of 9 samples. By comparison, Figure 2.6 reveals that the sum of Mahalanobis distances for $N_s = 9$ samples (27 degrees of freedom) should not exceed a threshold between 60 and 70, which is 2.5 times lower than the previous one. At the same time, considering the set of samples as a whole is more tolerant to individual outliers (samples with individual Mahalanobis distance greater than normal) as long as the total distance is statistically plausible. By thresholding the sum of distances on samples instead of individual distances, the condition of belonging to the background model is then both tighter and more robust.

The appropriate threshold for the sum of an arbitrary number of samples has now to be determined. In a very similar way to that in which the threshold on individual pixels was defined in Section 2.4.2, the choice of a threshold for a set of samples is based on the Cumulative Density Function (*cdf*) of the Chi-Squared distribution. We might want to find the threshold T that will classify correctly a set of N_s samples $\alpha\%$ of the time, which is equivalent to finding the solution of the equation $cdf_{\chi^2_{3.N_s}}(T) = \alpha$. We shall denote as $T_{3.N_s}(\alpha)$ the threshold classifying a set of k samples with a confidence level α, and which is the solution of:

$$\int_0^{T_{3.N_s}(\alpha)} f_{\chi^2_{3.N_s}}(t)dt = \alpha \tag{2.20}$$

These threshold functions are plotted in Figure 2.7 for confidence levels $\alpha = \{95\%, 99\%, 99.9\%\}$ and the number of degrees of freedom d in the range $[1 \ldots 100]$. When the number of degrees of freedom is fixed, the threshold $T_d(\alpha)$ can simply be pre-computed. However the number of pixel samples can be dynamically adjusted as we shall see in Section 3.3.2. The values of the thresholds can then either be tabulated and used as such, or approximated by a simpler function evaluated in real-time. Second-order polynomials are appropriate for approximating the threshold functions, because of their simplicity and the relative smoothness of the functions to approximate. A simple least-square fitting gives the desired polynomial coefficients:

$$\begin{aligned}
T_d(0.95) &\simeq -0.0013.d^2 + 1.3193.d + 4.987 \\
T_d(0.99) &\simeq -0.0019.d^2 + 1.4491.d + 8.5205 \\
T_d(0.999) &\simeq -0.0019.d^2 + 1.5094.d + 13.0247
\end{aligned}$$

These fitted polynomials are shown as error bars on the curves of Figure 2.7, exhibiting the goodness of the fit in the interval $d = 3.N_s \in [1, 100]$. For higher degrees of freedom, the polynomial approximations can become inadequate, but for our purpose, this interval is sufficiently wide.

The result of this new segmentation scheme based on sets of 9 samples is presented in Figure 2.8. Each pixel is segmented in a set including its 8 neighbours, and the whole set is thresholded at a distance of 70. As expected, the final segmentation is less sensitive to random noise, and more of the silhouette is recovered. This is particularly visible on the trousers of the subject, which colour is very close to the floor's. On the performance side, there is a slight overhead when adding all the distances together (Equation 2.18), but without particular optimisation we found the drop in speed negligible.

With this new segmentation scheme, sets of samples belonging to the background

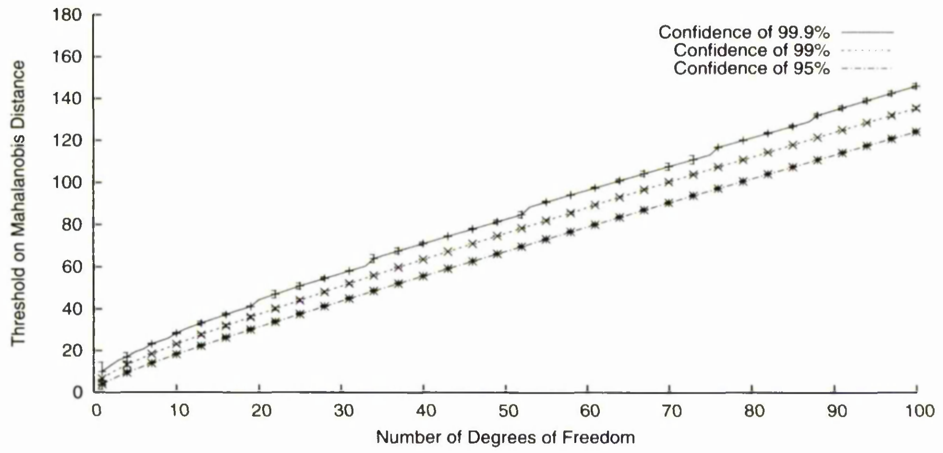
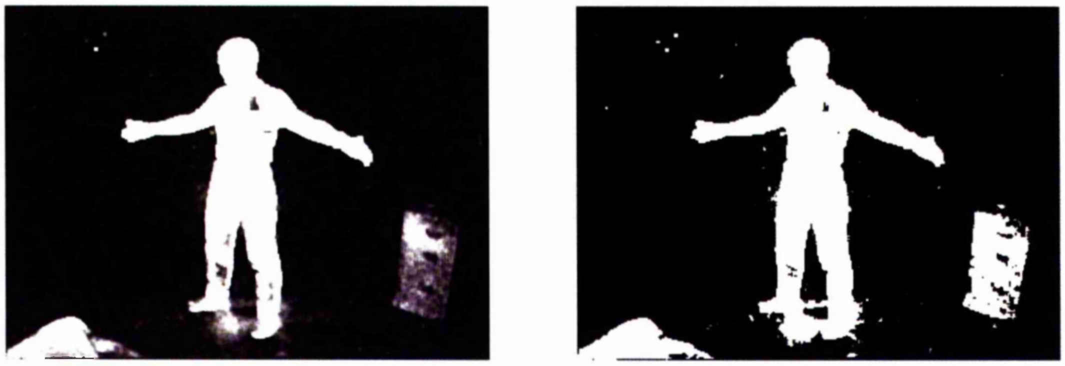


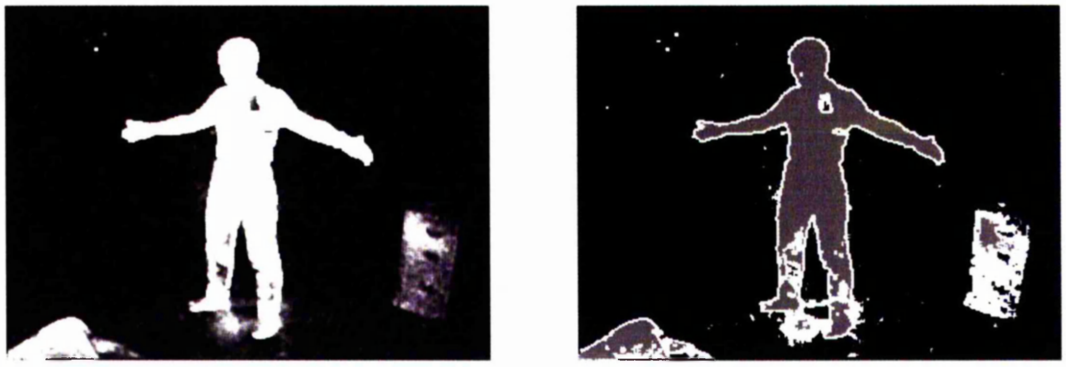
Figure 2.7: *Values of the Mahalanobis distance giving a particular level of confidence (95%, 99% and 99.9%), with relation to the number of degrees of freedom. Second-order polynomial fittings are displayed as error bars (very small on the graph).*



(a) Grey-level distances. (b) Thresholding at $D_M(S) < 70$.

Figure 2.8: *Segmentation using the centre pixel and its eight neighbours as the sample set. A few details are blurred compared to the per-pixel segmentation (Figure 2.5), but the tradeoff is that less noise remains and some parts of the silhouette that were badly segmented are now recovered. One can notice that shadows are stronger, but a method to reduce their effect will soon be introduced.*

can be detected and discarded with a better confidence level than previously possible. The remaining sets of samples are then either part of the foreground, or an edge. Detecting edges is not necessary for the segmentation itself, but it is done at a very low cost, and it will prove very useful for the hierarchical scheme presented in Section 2.4.4.



(a) Grey-level distances. (b) Edges and foreground sample sets.

Figure 2.9: *Segmentation with foreground and edges classification. The sets of samples classified as foreground are shown in grey and the edges in white. The threshold used for discarding background sets of samples was* 70 *and the individual samples threshold was* 18.

To differentiate foreground sets from edges, we fall back to per-sample thresholds: a set of samples is then classified as foreground if first it is not background, and second a high percentage α_f of its samples are individually classified as foreground. The coefficient α_f accounts for possible outliers in the per-sample segmentation. It was found that a coefficient of $\alpha_f = 0.9$ gave robust results without misclassifying edge sets into foreground. All the sample sets that are neither background nor foreground are then classified as edges. Looking at the result of this classification (Figure 2.9), edges and foreground regions are mostly detected correctly. Extra edges appear in regions of uncertainty like the trousers or the shadows, but this is not particularly a problem since edges are eventually integrated into the foreground. Moreover, we shall devise a scheme to handle shadows in Section 2.5.

2.4.4 Hierarchical Silhouette Extraction

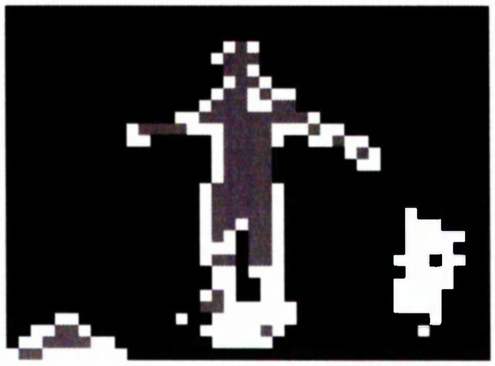
This section is a logical extension of the ideas presented in the previous section: if a set of pixel samples can be classified with reasonable confidence as background, foreground or edge, then a simple hierarchical scheme can be devised where edge regions are recursively subdivided. This makes sense because edges are the regions where details are needed, while foreground and background areas are advantageously modelled by large uniform regions. Additionally, this section serves as introduction to the hierarchical 3-D reconstruction scheme presented in the next chapter.

The only difficulty at this point is to find the right sampling for an area of arbitrary size. The number of samples, N_s, has to be sufficient to cover the area as uniformly as possible, and at the same time it should be kept as low as possible for efficiency reasons. The whole justification of the method is the ability to classify an area with a sub-linear number of samples with respect to the number of pixels in the area. For this reason, we experimented with a number of samples equal to the square-root of the number of pixels in the area. For an area of $16 \times 16 = 256$ pixels, the number of samples will then be 16. For small areas (less than 3×3 pixels), however, this scheme gives poor results and we fall back to an exhaustive sampling of the area. Uniform sampling inside rectangular areas is relatively straightforward, and there is no need to consider more complex areas for simple image segmentation. A generalisation to polyhedral areas will however be necessary for volumetric reconstruction: the reader is referred to Appendix B for additional details.

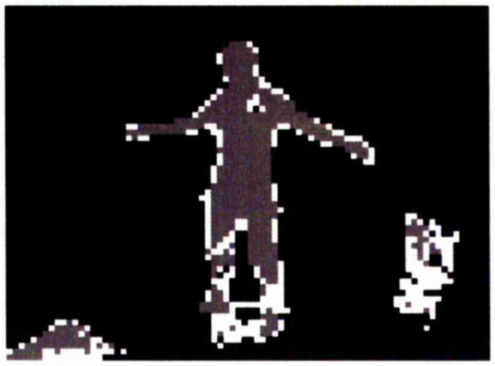
Hierarchical silhouette extraction is demonstrated in Figure 2.10. The total number of samples used for full classification is $11,885$ which, compared to the $76,800$ pixels of the full image, represents a factor 6.5 of improvement in direct performance. Figure 2.10(d) shows that samples are mostly distributed on the edges, maintaining a good overall level of accuracy. The final silhouette is slightly less detailed than in Figure 2.8, but one can notice that there are also fewer outliers in the background. This is explained by the fact that classifying large regions with higher number of samples makes the whole classification less sensitive to individual outliers. More results of hierarchical silhouette extraction including shadow handling are presented later in this chapter.

2.5 Handling Shadows

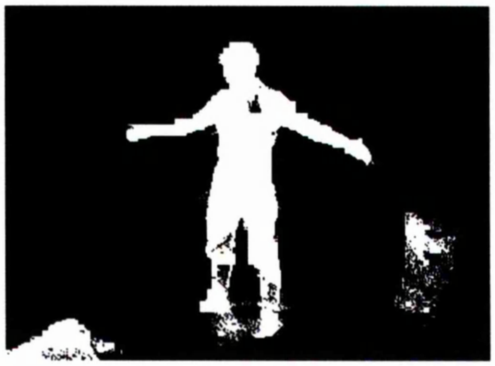
Shadows are omnipresent in real-life setups. Detecting and removing them automatically is crucial for the quality of the segmentation. A point of the scene is shadowed if part of the light it receives in normal circumstances is occluded. The corresponding pixels on the camera images are therefore still representing the same object, but under different lighting conditions. Depending on the type of lighting and the physical properties of the object, the colour of the pixels can be modified in a number of ways. For example, shadowing an object exhibiting some specular reflection can change drastically the colour of the corresponding pixels. Coloured light sources, when occluded, can also change the apparent colour of an object.



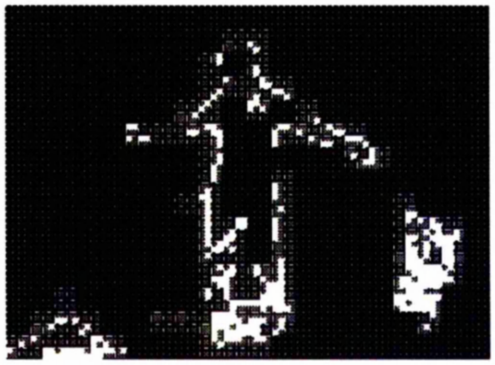
(a) Intermediate subdivision stopped with 8×8 blocks. Grey areas are already classified as foreground, and white blocks are edges to be subdivided.



(b) Next subdivision step with 4×4 blocks. More details appear in edge regions.



(c) Final classification result with resolution of 1 pixel.



(d) Each white dot represents a sample used to classify image regions. As expected, they concentrate on edges.

Figure 2.10: *Hierarchical silhouette extraction with initial subdivision into* 16×16 *areas. Two intermediate steps are shown on top with the final result on the bottom left. The input frame is the same as for Figures 2.5, 2.8 and 2.9.*

2.5.1 Related Work

Most of the recent approaches to background segmentation propose original solutions for shadows removal. In PFinder [WADP97], the chrominance channels are normalised by the overall luminance in YUV space. This is claimed to produce a stable illumination independent colour information. Cheung *et al.* [CKBH00] compute a colour "angle" in RGB space for pixels with lower luminosity than the model, and use a different manually-set threshold for the floor than for the rest of the background. Mixtures of Gaussians are used by Stauffer *et al.* [SG99] and Friedman *et al.* [FR97] with the advantage of having shadows automatically modelled by one of the Gaussians, as long as the shadows appear in the training frames. Unfortunately the generality of the framework leads to computationally expensive algorithms. Lo and Yang [LY02] propose a specialised shadow filter based on the combination of four low-level filters into a neural network classifier. The results are encouraging, even if the method is mostly aimed at offline full image processing.

2.5.2 A Gaussian Model for Shadows

In order to keep the system simple and fast enough, we only try to tackle the most usual type of shadows which are characterised by a loss in luminosity. Using the YUV colour-space, variations of luminosity almost entirely appear on the luminance (Y) channel only. We carried out measurements under various lighting conditions and found that the two chrominance channels (UV) are almost unaffected by a moderate change of luminosity. This observation breaks for important lighting changes, but it is relatively safe to assume that the shadows cast in a normal office environment belong to the category of moderate lighting changes: with multiple light sources as well as a strong diffuse lighting (radiosity), a person can only occlude the total illumination very partially.

We measured the variations of intensity on the luminance channel when background pixels got shadowed, and found these variations surprisingly constant for pixels of different colours and initial luminosities. In the office environment of Figure 2.5, the average luminosity loss accounted for $\alpha_L = 5.5\%$. Of course this coefficient is bound to differ for other environments and lighting conditions, and could be measured automatically. However, for the sake of simplicity, we treat α_L as a manually-set constant of the system. The idea is to build a Gaussian model of a shadowed pixel by "shifting" the original Gaussian model by α_L on the Y channel. For each pixel, the Mahalanobis distance $D_M()$ is then selectively computed from the original model or from this new model for shadows.

For a given background pixel, with a Gaussian model of mean $\mu = (\mu_Y, \mu_U, \mu_V)^T$ and covariance Σ_w, the new mean μ' and covariance $\Sigma_w{}'$ of the shadowed Gaussian model are:

$$\mu' = \begin{pmatrix} (1-\alpha_L).\mu_Y \\ \mu_U \\ \mu_V \end{pmatrix} \qquad \Sigma_w{}' = \Sigma_w \tag{2.21}$$

Note that the covariance matrix is kept the same as in the non-shadowed model, even if the camera noise can have different properties for different luminosities. This approximation is mostly aimed at saving memory by storing only one covariance matrix for each pixel, and did not affect the quality of the model for low values of α_L. Stronger shadows, like the ones cast by a single light source, might require a re-evaluation of $\Sigma_w{}'$, but are beyond the scope of this thesis.

This simple model of shadowed pixel can then be computed for all pixels of an empty scene, without needing to observe the pixels under shadow. Our scheme can then be used to detect and remove previously unseen shadows.

2.5.3 Efficient Distance Computation

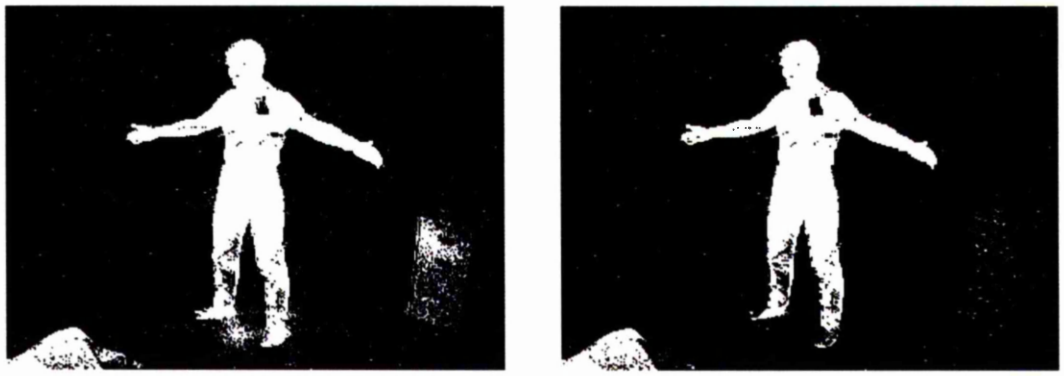
A decisive advantage of our model over a standard Mixture of Gaussians is its efficiency. Indeed, with a standard mixture of Gaussian, the distance to all individual Gaussian models needs to be computed before selecting the one giving the smallest distance. In our approach, however, a simple test on the value of the luminance (Y) component of a pixel reveals the appropriate model. In practice, we can then discard shadows without any noticeable performance loss.

The statistical advantages of a Mixture of Gaussians are nevertheless retained, and the theoretical analysis from Section 2.4 remains valid. Whether the pixel is shadowed or not, we select at runtime the appropriate Gaussian model. Using the "shifted" Gaussian distribution from Equation 2.21, the new distance, D'_M, defined in the following way:

$$D'_M(s,\alpha_L) = \begin{cases} D_M(s,\mathcal{N}_3(\mu',\Sigma_w{}')) & if \quad s_Y < (1-\frac{\alpha_L}{2}).\mu_Y \\ D_M(s,\mathcal{N}_3(\mu,\Sigma_w)) & otherwise \end{cases} \tag{2.22}$$

still follows a Chi-Square distribution with 3 degrees of freedom.

As shown with Figure 2.11, replacing $D_M()$ with $D'_M()$ on the data from Figure 2.5 allows us to discard most of the shadows without affecting the rest of the silhouette: the shadow on the floor has been totally removed, and very few shadowed pixels remain misclassified on the drawers. Unlike most other shadow removal schemes, this result is achieved without extra memory usage or noticeable performance deterioration. Moreover, the distance $D'_M()$ still has the same properties as the original Mahalanobis distance, and can be used in the recursive sampling algorithm described in Section 2.4.4.


(a) Without shadows treatment.

(b) Shadows as a 5.5% loss in luminosity.

Figure 2.11: *Results obtained with the new distance $D'_M()$. Most of the shadows are now included in the background model and discarded by thresholding individual pixels at $D'_M() < 18$.*

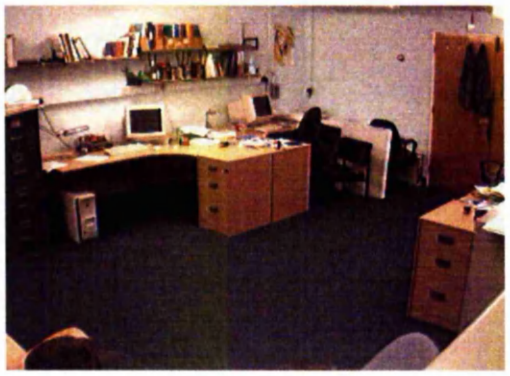
2.6 Evaluation and Conclusion

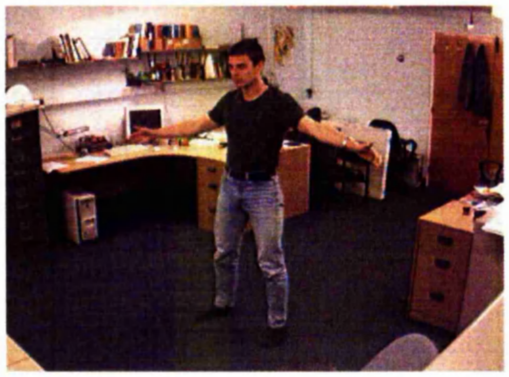
In this final section, we present the final results of our segmentation algorithm and compare them to another standard method. We conclude with some performance comparisons and a summary of the chapter.

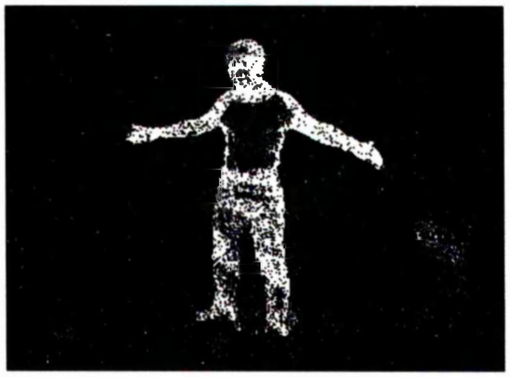
2.6.1 Qualitative Results

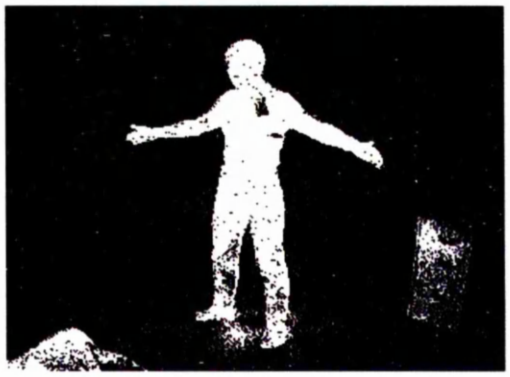
No ground truth data was available to quantify the robustness of our algorithm. We therefore had to fall back to visual inspection and limit ourselves to qualitative results. In Figure 2.12, we compare side by side our hierarchical method, including shadows handling, and the mixture of Gaussians model proposed by Stauffer and Grimson [SG99]. This model (which we shall refer to as Grimson's model) is primarily aimed at monitoring applications in which the background can change rapidly, such as tracking pedestrians in a street. Nevertheless, Grimson's model has also been increasingly popular for various other types of applications, including indoor tracking.

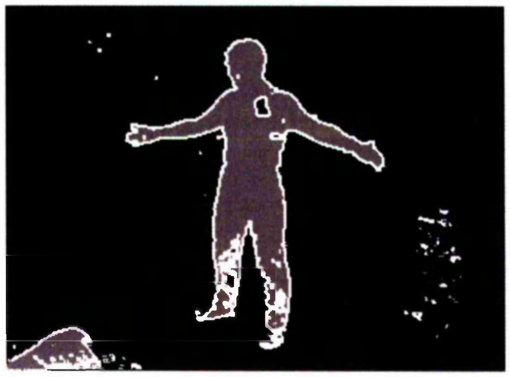
For both algorithms, we used 100 background training frames. Grimson's algorithm has a mechanism for learning the background model incrementally. In Figure 2.12(c), we activated this feature with an update coefficient $\alpha = 0.003$. The part of the image that was not empty during the acquisition of the model is then correctly segmented, and shadows are incorporated into the model and discarded. However,


(a) Background (still not empty).


(b) Current input image.


(c) Grimson, 5 Gaussians per pixel and incremental update with coefficient $\alpha = 0.003$


(d) Grimson, 10 Gaussians per pixel. No incremental update.


(e) Our algorithm classifying pixels with a 3×3 window and handling of shadows.

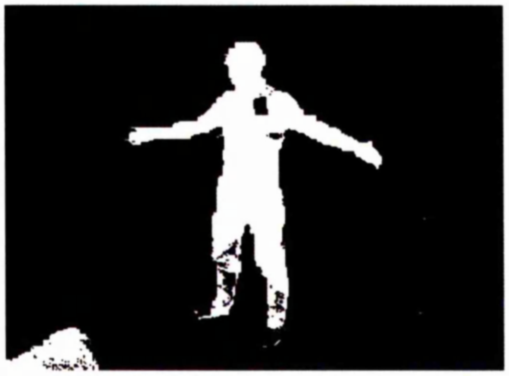

(f) Hierarchical segmentation with handling of shadows, starting from 16×16 blocs.

Figure 2.12: *Comparison between our segmentation algorithm, and the mixture of Gaussians model proposed by Grimson et al. [SG99].*

	Gaussians per Pixel	Handling of Shadows	Time ms/frame	Speed fps
Pixel-based Mahalanobis	1	No	16.2	61.6
Pixel-based Mahalanobis	1	Yes	16.8	59.4
Classification with neighbours	1	Yes	18.7	53.4
Hierarchical Seg. Init. 8×8	1	Yes	12.5	79.6
Hierarchical Seg. Init. 16×16	1	Yes	10.1	99.2
Grimson	2	No	35.8	27.9
Grimson	5	No	66.2	15.1
Grimson	10	No	106.3	9.4

Table 2.4: *Performance comparison on* 320×240 *input images.*

since the subject stands relatively static, he is also partially incorporated into the background model, leading to an overall poor segmentation. In Figure 2.12(d), we disable the incremental update of the model, and increase the number of components to 10 Gaussians per pixel. The result is then very similar to the segmentation obtained with a single Gaussian per pixel (shown in Figure 2.5), which is not surprising since we experimentally showed in Section 2.3 that the camera noise is Gaussian.

In Figure 2.12(e), we classify each pixel using its 9 neighbours, as described in Section 2.4.3. Shadows are also handled by using the new distance $D'_M()$ from Section 2.5. By combining information from a set of pixels, the overall segmentation looks substantially better than using individual pixel classification. Finally, in Figure 2.12(f) we use the hierarchical scheme described in Section 2.4.4, with initial blocs of 16×16 pixels. Even if some fine details are lost because of the initial coarse sampling, the subject is almost fully segmented from the background, while the noise and the shadows have a minimal effect.

2.6.2 Performance Considerations

The comparative performance of the various segmentation schemes described in this section are gathered in Table 2.4. All performance results were measured on a 2 GHz Pentium computer. Because of its higher number of Gaussian components, Grimson's model is always significantly slower than all the methods presented in this chapter. With an average processing time of about 10ms, our hierarchical segmentation algorithm is the only one allowing the simultaneous segmentation of more than 3 camera views in real-time.

2.6.3 Conclusion

In this chapter, we justified experimentally a Gaussian model for background segmentation of static scenes. We then improved the robustness of the standard segmentation by classifying sets of samples as opposed to individual pixels. A scheme to handle shadows at very low extra cost was also introduced. Real-time performance for multiple input streams was finally achieved with hierarchical segmentation.

This chapter, while self-contained, can be seen as an introduction to Chapter 3 where multiple camera views will be combined for volumetric reconstruction, using a hierarchical scheme very similar to the present one. The robustness of the classification will be improved by modelling uncertainty, and using other camera views for disambiguation.

The main weakness of our model is the absence of background update scheme, where changes in the background could be incorporated incrementally into the model. While this is not a problem for static backgrounds (frequent in human-body tracking), other applications may require a more complex (multimodal) model: a model such as Grimson's [SG99] could then prove more adapted.

Chapter 3

Volumetric Reconstruction

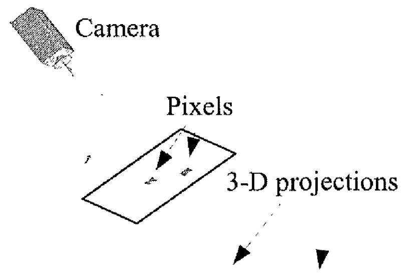
Having defined a way to segment input images, our next step is the volumetric reconstruction of the object of interest. Such a 3-D reconstruction is possible because the cameras are fully calibrated, providing a mapping between 3-D object points and their 2-D projections on the camera views. This chapter introduces a fast and robust algorithm for 3-D reconstruction based on Shape-From-Silhouette methods, which, as their name suggests, are concerned with the reconstruction of an object from a set of its silhouettes. The novelties of the proposed algorithm include a hierarchical statistical framework and the inclusion of colour.

3.1 Background and Basic Principle

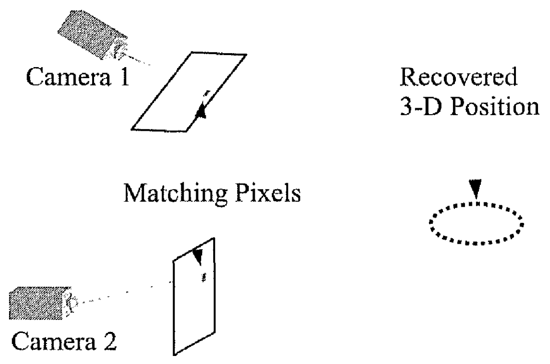
In this section, we start by introducing the basic concepts of Shape-From-Silhouette, and define the term "Visual-Hull". The historical context of these methods and some current research goals are then briefly presented, before discussing a standard algorithm and its limitations.

3.1.1 Shape-From-Silhouette

Since our cameras are fixed and calibrated, the projection of any 3-D point is uniquely known through the projection matrix and the non-linear distortions. The inverse mapping – from a pixel to 3-D – is more complex and ambiguous since every pixel corresponds to an infinity of 3-D points, as illustrated by Figure 3.1(a). However, using



(a) Each pixel of the image plane corresponds to a generalised cone in 3-D space.

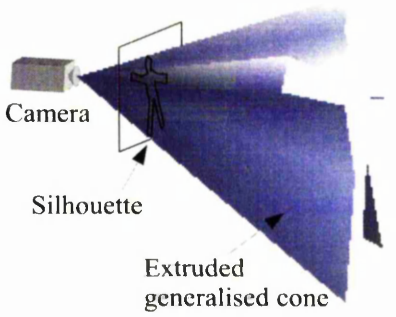


(b) The intersection of the extruded cones from matching pixels defines the corresponding 3-D location.

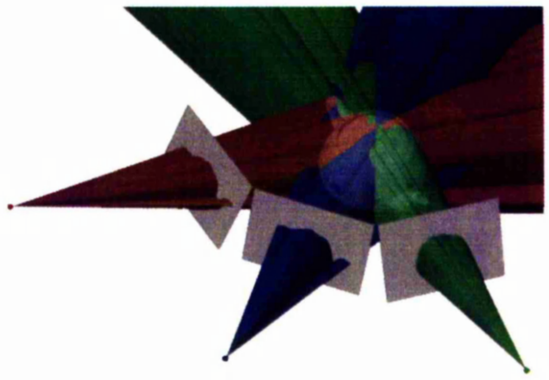
Figure 3.1: *Correspondence between individual pixels and 3-D locations.*

a minimum of 2 camera views, and assuming a way to find matching pixels in each view, we can find a finite 3-D location by intersecting the projected volumes, as shown in Figure 3.1(b).

Following a similar idea, a silhouette is the projection of an object onto an image plane. The object of interest then lies totally inside the generalised cone extruded from its silhouette and passing through the camera centre (see Figure 3.2(a)). Intersecting these generalised cones from multiple camera views is a technique called Shape-From-Silhouette: using a sufficient number of cameras placed in a way as to cover the widest possible range of angles, the intersection of these generalised cones can produce a relatively good approximation of the 3-D shape of the object. This approximate volume is called the *Visual-Hull* of the object (Figure 3.2(b)). The Visual-Hull is also commonly defined as the largest possible volume which exactly explains a set of consistent silhouette images. This last definition is more formal, but equivalent to the intersection of generalised cones. In the literature, the Visual-Hull is sometimes referred to as the optimal convex approximation of an object, therefore assuming an infinite number of views. Note that for a convex object, the Visual-Hull would then be equal to the object itself. For practical reasons, however, our definition of the Visual-Hull in this chapter will be limited to using a finite number of camera views (three to five in practice). We can then immediately see that the Visual-Hull can become a rough approximation of the real shape of the object. Keeping in mind that our goal is not photo-realism but tracking, we will show in the following chapters that the Visual-Hull is sufficient for our purpose.



(a) Projection of the silhouette into 3-D space as a generalised cone originating from the camera centre.



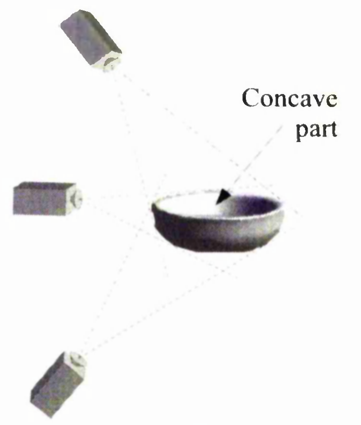
(b) The Visual-Hull of the object is the intersection of the generalised cones extruded from its silhouettes (illustration reproduced from [MBR⁺00]).

Figure 3.2: *Shape-From-Silhouette as intersection of projected silhouettes.*

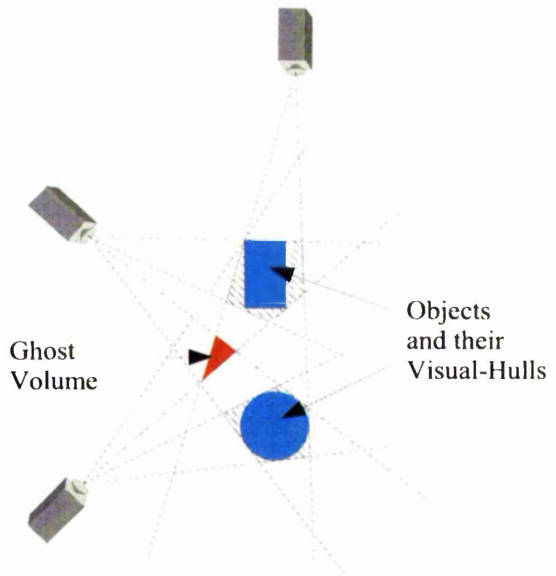
A fundamental flaw of the Shape-From-Silhouette methods is that concave parts are not reconstructed (for the simple reason that they do not appear on silhouettes). A surface point of the object is considered to be part of a concave patch if there is not a single tangent line from this point that do not re-intersect the object. More intuitively, if a point is not reachable by sweeping a long ruler on the surface of the object, then it belongs to a concave patch. Figure 3.3(a) presents a simple example of object for which the Shape-From-Silhouette method is inefficient. This limitation is actually a minor problem when dealing with the human body which can be seen as a set of locally convex parts. Self-occlusions for some body poses can produce temporary concavities, but the use of a model can then compensate for the weaknesses of the reconstruction.

Another problem can appear when at least two distinct objects are present in the field of view of the cameras. Depending on the position and the number of cameras, some parts of the 3-D space can appear to belong to the Visual-Hulls of the objects, although they are only artifacts due to occlusions. These wrongly reconstructed parts are called *Ghost Volumes*. Figure 3.3(b) illustrates this phenomenon in 2-D. Ghost volumes are normally a manageable problem if there are not too many occlusions between objects or if the number of viewpoints is sufficiently high, but it is also easy to foresee how problematic the reconstruction of multiple people can become. Ghost volumes can also appear with a single subject because of self-occlusions. Once again, the use of a model will prove necessary to disambiguate these situations.

Other kinds of limitations will be discussed in the rest of this chapter, but the principal advantage of the Visual-Hull over other reconstruction methods remains its overall



(a) Concavities cannot be reconstructed with the Shape-From-Silhouette method.



(b) 2-D illustration of "ghost volumes" appearing with multiple objects and a limited number of views.

Figure 3.3: *Some weaknesses of the Shape-From-Silhouette method: Concavities and Ghost-Volumes.*

simplicity and efficiency. Artifacts can be observed in almost every existing reconstruction method, but silhouette extraction is sufficiently robust to make Shape-From-Silhouette method a competitive choice on noisy and degraded images.

3.1.2 The Standard Algorithm and its Limitations

In this section we present the standard voxel-based algorithm for Visual-Hull reconstruction. This algorithm is the basis for the novel reconstruction method introduced later in the present chapter.

The space of interest is first subdivided into discrete voxels. The idea is to consider voxels independently from each other, and project them successively onto the image planes of the available camera views. If a given voxel projects outside the silhouette of the object in at least one camera view, then it is necessarily outside the intersection of the visual cones, and can therefore be discarded. The full algorithm is given below.

Despite its simplicity, this algorithm performs well and is still used in recent publications, for example in [LSL01, TMSS02, MTHC03, KBG05]. Its complexity is linear

Algorithm 3.1: Standard Visual Hull Algorithm

```
▷ The space of interest is divided into discrete voxels;
▷ Initialise all voxels as inside voxels;
foreach voxel do
    foreach camera view do
        ▷ project the current voxel on the current camera view;
        if the projection of the voxel lies outside the object's silhouette then
            ▷ Classify the current voxel as outside voxel;
            ▷ Skip other views, and consider the next voxel;
        end
    end
end
▷ The Visual-Hull is the set of all inside voxels;
```

with the number of voxels, limiting strongly the resolution of the voxel-space. In practice, the maximal resolution manageable in real-time on a 2 GHz CPU is 128×128×128. Fortunately, the number of cameras does not have a linear impact on the complexity since most of the voxels are discarded before testing all the views. The robustness of the reconstruction depends directly on the quality of the silhouettes.

3.1.3 Background on Shape-From-Silhouette Methods.

Origins and History

The idea of reconstructing the shape of an object from its silhouettes is not new. With his PhD thesis, back in 1974, Baumgart [Bau74] was the first to use silhouettes to estimate the shape of a miniature horse, using four camera views. Actually, he used external contours instead of silhouettes to compute a polyhedral reconstruction of the object, but the basic idea would still be retained in much subsequent research. Almost 10 years later, Martin and Aggarwal [MA83] proposed a volumetric representation as a collection of "volume segments". In subsequent papers, Chien and Aggarwal *et al.* [CA86, CA89] adopted an octree-based representation to improve speed and memory usage, but the reconstruction was still limited to 3 orthographic projections. This approach was extended to 13 standard orthographic views by Ahura and Veenstra [AV89]. Shape-From-Silhouette was then known as "volume intersection".

Potmesil [Pot87] allowed arbitrary viewpoints and perspective projections, while retaining the octree data-structure. Using a single camera, Szelinski [Sze90] reconstructed static objects on a turntable, achieving real-time performance thanks to optimisations such as half-distance transforms and sequential octree refinement. Noborio *et al.* [NFA88] proposed a novel octree reconstruction method in 3-space, where visual cones are computed from a polygonal approximation of the silhouettes, thus eliminating the need for perspective projection.

In an attempt to give some formalism to the Shape-From-Silhouette methods, Laurentini introduced the term Visual-Hull in a series of articles [Lau94, Lau95]. This term is now commonly used to refer to the broad result of the Shape-From-Silhouette algorithms, even if Laurentini originally defined it more restrictively as "the optimal volume that can be reconstructed from all possible silhouettes". In addition to enunciating the properties of Visual-Hulls, Laurentini's work highlights the limitations of Visual-Hulls for object recognition.

Polyhedral Representation

More recently, real-time polyhedral Visual-Hull methods were introduced by Matusik *et al.* [MBM01], Lazebnik *et al.* [LBP01] and Franco *et al.* [FB03]. In contrast to approximate volumetric representations, polyhedral Visual-Hulls have a greater potential of accuracy, and their rendering benefits from hardware acceleration. Their main limitations are the overall fragility of the method which relies on a perfect silhouette segmentation, and the fact that triangle meshes are difficult to exploit for shape recognition or tracking.

Following a more theoretical approach, Brand *et al.* [BKC04] proposed a way to compute algebraically the polyhedral Visual-Hull of an object in dual space, resulting in an algorithm with linear complexity with respect to the number of observed contour points. As a logical extension to polyhedral Visual-Hulls, Bottino and Laurentini [BL04] recently proposed a smooth representation using curved patches.

Volumetric Reconstruction using Voxels

Volumetric representations have also benefited from a renewed interest in the last few years (to date 2005). This is mostly due to advances in computer hardware, opening the door to the real-time use of Visual-Hulls. Standard voxel-based reconstruction techniques have been used by Luck *et al.* [LSL01], Theobalt *et al.* [TMSS02] and Mikic *et al.* [MTHC03] as a basis for human-body tracking. All these approaches use a full silhouette segmentation, followed by a standard voxel projection technique (Algorithm 3.1), and some are reported to work in real-time on a cluster of PCs with CPUs up to 2 GHz.

Bottino and Laurentini experimented with voxel-based volumetric reconstruction in [BL01a] and [BL01b], with the interesting detail that boundary voxels are the only ones kept and used. Cheung *et al.* [CKBH00] proposed a "Sparse Pixel Occupancy Test" to classify voxels based on pixel samples. Silhouettes are still fully segmented on a cluster of PCs, but the reconstruction now runs in real-time on a single machine. Using the parallel processing capabilities of Graphic Processing Units (GPUs), some fast reconstruction methods have also been proposed [HLGB03, HLS04]. Unfortunately, the resolution of the reconstructed volume is strongly limited by the memory of the video card. A more general drawback is the lack of flexibility and robustness of these techniques, which have to comply with a static programming pipeline.

Hierarchical Approach and Octrees

Even if they are not as widely used as simple voxel spaces, octrees still present compelling advantages in allowing adaptive level of detail with a lower memory consumption. A possible reason for their slow adoption is their relative complexity, especially regarding the interpretation of the generated volume. Nevertheless, following Szelinski's reconstruction method [Sze90], Davis *et al.* [DBC$^+$99] describe a system for movement acquisition. In subsequent publications [BD02, BSD03] by the same authors, the octree volume is used for coarse-to-fine body pose estimation. In [BSD03], the reconstruction is performed on a cluster of PCs, and real-time performance is achieved through various caching techniques. The method itself is very standard, and only the optimisations and the distribution on a cluster differentiate it from Szelinski's early algorithm.

Space Carving and Colour Consistency

Space Carving is a volumetric reconstruction method that uses colour consistency as well as contours to reconstruct an object. A very broad survey of reconstruction techniques is available from Dyer [Dye01]. The idea is to keep only the voxels that are photo-consistent across all camera views from which they are visible. Of course the visibility criterion is a big issue: it is usually solved by making multiple plane-sweep passes, using each time only the cameras in front of the plane, and iterating until convergence. Unfortunately, the complexity of this method disqualifies Space Carving for real-time algorithms, even using hardware acceleration [SBS02a]. The reconstructed volume contains only the surface voxels of the object, and is commonly called the *Photo-Hull.*

Cheung *et al.* [Che03, CBK03b, CBK05] proposed a very interesting mixed approach between Visual-Hull and photo-consistency. Using the property that the bounding edge of the Visual-Hull touches the real object at at least one point, a photo-consistency test along the "bounding edges" of the Visual-Hull allows to refine the reconstruction at moderate cost. Unfortunately, the reconstruction is then very sparse and to be practical needs a lot of input data. Other encouraging reconstruction results have been reported by De Bonnet and Viola [BV99] and Broadhurst *et al.* [BDC01] using iterative optimisation techniques for space carving.

Other Extensions

A strong constraint of shape-from-silhouette methods is that the cameras have to be fully calibrated. Accurate calibration is not an easy task, and some authors have started to search for alternatives. Bottino *et al.* [BL03] pose the problem of Visual-Hull reconstruction when the relative position of the cameras is unknown. In their paper, they derive a number of inequalities to decide whether silhouettes are compatible with one another. However, their analysis is only a very first step towards automatic calibration from silhouettes, and is mainly focused on orthographic projections. Another potential limitation is that the object must exhibit some particular features, such as identifiable corners, for the silhouettes to be differentiable.

Trying to tackle the problem of robustness in silhouette extraction, Grauman *et al.* [GSD03a] propose a Bayesian framework where silhouettes are recognised and corrected using a training dataset. A generalised Principal Component Analysis is performed on multi-view silhouettes training data to generate a searchable feature space. When presented with a new set of silhouettes, the closest correspondence in the feature space is used to correct the silhouette. In [GSD03b], the same authors extend this approach to tracking, where the feature space is augmented with the parameters of a kinematic model. The main drawback of the method is that the training dataset has to be sufficiently large to include all possible body configurations and is dependent on the placement of the cameras. This scheme is then mostly adapted to tracking on very structured data, like pedestrians in a street.

In another attempt to improve on robustness, Snow *et al.* [SVZ00] formulate the Visual-Hull reconstruction as the minimisation of an energy function. This function has a classical data term based on silhouettes data, and a new smoothness term supposed to limit the noise in the generated volume. The voxel-based volume is then computed using a Graph-Cut algorithm [BVZ01]. The formulation of the reconstruction process as an energy minimisation problem is very interesting and can probably be further developed. However, the smoothness term does not appear to be an adequate answer because it tends to aggregate together nearby body parts and discard finer features. Furthermore, the computational cost of the method is reported to be very high.

In summary, Visual-Hulls remain an active area of research because of their efficiency, robustness and overall simplicity. Shape-From-Silhouette methods have also benefited from a recent (to date 2005) renewal of interest, thanks to advances in computer hardware making them practical in real-time scenarios.

3.2 A Novel Hierarchical Reconstruction Approach

A novel method for real-time 3-D reconstruction is introduced in this section. After describing the method itself and the associated statistical framework, we discuss some extensions derived from the use of a model.

3.2.1 Aims and Constraints

Our aim is to present an algorithm for 3-D reconstruction that is best adapted to human-body tracking. Even though an algorithm using a polyhedral reconstruction and surface normals for tracking was recently presented by Niskanen *et al.* [NBH05], polyhedral meshes are mainly aimed at rendering and remain ill-suited for tracking. Voxel and octree based techniques, on the contrary, have already proved adaptable to human-body tracking [CKBH00, MTHC03, TMSS02, BD02]. A volumetric representation is thus adopted, without yet being bound to a particular data-structure. Design concerns for the proposed algorithm include:

- *Real-time on a single machine:* All the previously cited methods are either too slow for real-time, or require a cluster of PCs. In order to be really usable, our method should be able to perform the image capture, silhouette segmentation and 3-D reconstruction in real-time ($\geq$ 10fps) on a single machine. Moreover, enough resources should be left available for the tracking process. This is a challenge, especially considering that full silhouette segmentation alone has not yet been achieved in real-time for multiple high resolution views (see Section 2.6). However, using optimisations like per-sample segmentation in a hierarchical framework, we will show that high framerates are achievable without compromising on accuracy.

- *Good maximal accuracy with acceptable memory footprint:* When using voxel-space, the maximal accuracy has a strong impact on the memory footprint of the data-structure, making it very hard to achieve high resolutions. Octrees, on the other hand, suffer from a relatively high overhead inherent to the hierarchical structure. Our aim is to be able to reach an accuracy of the order of one centimetre for a human-size tracking area. An optimal subdivision level of $1/256^{th}$ of the tracking space should then be achieved, but only where it is needed. Indeed, generating too much data would be detrimental to the performance of the tracking step. The general idea is then to retain the best features of both octrees and voxel-spaces, in the form of a hierarchical voxel reconstruction scheme that does not require a static tree structure.

- *Robustness to errors and noisy data:* Robustness is too often overlooked in reconstruction algorithms which tend to rely on a perfect image segmentation. Unfortunately, camera noise and artifacts like shadows make segmentation unreliable in real-life environments. Binary silhouette extraction is then bound to contain errors, and post-processing algorithms tend to work without knowledge about the underlying data. A statistical approach, with "soft thresholds" is then desirable to cope with local errors.

- *Flexibility and adaptation to various tracking space configurations:* A very useful feature for human body tracking is the possibility to follow the subject in relatively large environments. Restricting the tracking zone to the size of the subject is however needed to keep accuracy maximal. The solution is to dynamically move the region of interest in order to follow the movements of the subject. This approach is all the more appealing because the position of the subject is known through tracking. Of course, keeping the space of interest static allows optimisations like lookup tables, but we shall show that the advantages of a dynamic approach are more desirable.

- *Inclusion of extra-information such as colour:* A minimum amount of information should be lost during the reconstruction process, and what is lost anyway should be of minimal significance for the tracking process. Having colour camera images as input, and assuming that colour information must be valuable for the tracking, we will describe a way to include colour in the reconstructed voxels. Once again, this is done at a minimal cost (no occlusion/visibility test possible), while keeping in mind the needs of tracking.

3.2.2 Algorithm Overview

The main concept of our proposed algorithm is its hierarchical nature. Just like for an octree, the idea is to start with a space coarsely subdivided in a few voxels and recursively refine those voxels which need further subdivision. Typically, voxels needing subdivision are those on the edge of the object of interest. The logic behind this is that large background regions can be discarded quickly, and foreground regions do not need subdivision if they belong entirely to the object of interest with reasonable confidence. A voxel is classified as edge if its projection is classified as edge on at least one view, while the projections on the remaining views are either classified as edge or foreground. Methods for projecting and classifying voxels will be detailed in the rest of this section. Classification is mostly based on the ideas discussed in Chapter 2.

The term "voxel" is defined as "the smallest distinguishable cube-shaped part of a three-dimensional space", meaning that it could not theoretically be subdivided. However, in the rest of this thesis, voxels will denote a cube-shaped part of the three-dimensional space with a discrete size. This liberty is taken for the sake of simplicity, and also because no other term is totally adequate. The input of the algorithm is then a voxel $\mathcal{V}$ (position and size). The output classification of the voxel is expected to fall into four categories:

- *Background:* $\mathcal{V}$ is not part of the object of interest. It is then immediately discarded.
- *Foreground:* $\mathcal{V}$ is entirely enclosed inside the visual-hull of the object of interest. It is then kept and passed to the subsequent tracking process. Note that the voxel is not necessarily inside the object of interest, since the visual-hull is only an approximation of the real object.
- *Edge:* $\mathcal{V}$ is on the edge of the Visual-Hull of the object of interest. Because the voxel is too big to describe the disparities of the underlying data, it is therefore subdivided into 8 sub-voxels (octants). The classification algorithm is called recursively on each of the sub-voxels until either they fall into the two first categories or a maximal recursion level is reached. The maximal recursive depth is a tradeoff between accuracy and speed.

- *Unknown:* This category is temporarily used if there is not enough evidence to classify $\mathcal{V}$ into either *background, foreground* or *edge*. Spatial continuity is then used to disambiguate *unknown* voxels.

3.2.3 Flexible Recursive Approach

The reconstruction algorithm starts with a root voxel which has the size of the subject (typically 2 metres wide), and is centred on her expected position. At initialisation, where no tracked position is yet available, the root voxel is simply positioned at the centre of the tracking area. This root voxel is then recursively subdivided. At each recursive call, the current voxel is classified according to Algorithm 3.4 (page 75), and voxels classified as *edge* are in turn subdivided until a maximal depth D^+ is reached.

The general reconstruction algorithm is summarised in Algorithm 3.2, and detailed in the rest of this section.

Initial Subdivision Depth

Looking at Algorithm 3.2 in more detail, the function `MinDepth` (line 3.2.1) first gives the minimal recursive subdivision depth depending on the position of the current voxel and the model from the last frame. The idea behind this method is that a finer initial subdivision should be beneficial to regions where the subject is expected to be. On the contrary, other regions that are relatively far from the expected position of the subject can be inspected more coarsely. The initial subdivision is important because the sampling of pixels inside the projected area of the voxels is relatively sparse: if the initial subdivision is too coarse, it is then possible to miss a small feature like a finger. By contrast, a coarse subdivision of the empty regions has a beneficial impact on speed.

A minimal subdivision level is enforced, ensuring that all regions of the tracking area are decently inspected. Indeed, since the tracking area is only 2 meters wide, a fast movement can place a body part in an unexpected region within a few frames. The policy used for initial subdivision is that the closer the voxels are to the model (expected position of the subject), the finer the initial subdivision becomes. The distance between a voxel and the model will be formulated in Chapter 4. We denote as $D_M(X_{\mathcal{V}}, \text{model})$ the generalised Mahalanobis distance between the position of a voxel

Algorithm 3.2: Framework for the novel reconstruction method. The recursive method is first called on a root voxel and then refines on the regions needing it, depending on the classification of voxels at each level.

```
      ▷ Start with a root voxel;
      begin
3.2.1     if recursiveDepth $<$ MinDepth( $X_\mathcal{V}$ ,model) then
              TempClass $\leftarrow$ edge;
          else
3.2.2         TempClass $\leftarrow$ Classification of the current voxel according to
              Algorithm 3.4;
          end
          if TempClass = edge then
              if recursiveDepth $\geq D^+$ then
3.2.3             return foreground;
              else
3.2.4             ▷ recursive call of the method on the octants;
3.2.5             ▷ unknown octants take the classification of the majority;
3.2.6             if All octants have the same classification then
3.2.7                 return the common classification of the octants;
                  else
3.2.8                 ▷ create the foreground octants as voxels;
                  end
              end
          end
          return TempClass ;
      end
```

$\mathcal{V}$ and the appearance model. The function MinDepth is then defined as follows:

$$\texttt{MinDepth}(X_\mathcal{V}, \text{model}) = \begin{cases} 3 & \text{if} \quad D_M(X_\mathcal{V}, \text{model}) > 5 \\ 4 & \text{if} \quad D_M(X_\mathcal{V}, \text{model}) \in [2,5] \\ 5 & \text{if} \quad D_M(X_\mathcal{V}, \text{model}) \in [0.7, 2[\\ 4 & \text{if} \quad D_M(X_\mathcal{V}, \text{model}) \in [0, 0.7[\end{cases} \tag{3.1}$$

The chosen thresholds are arbitrary, but have proved to work well in practice. Actually, since the distance $D_M()$ is normalised with respect to the size of the blobs, the thresholds are defined in terms of standard deviations to the subject. These thresholds therefore remain constant when the size of the subject or the size of the tracking space vary. It can be seen that a voxel is less finely subdivided if it is very close to the model: such a voxel may indeed be entirely enclosed in the subject and be wholly classified

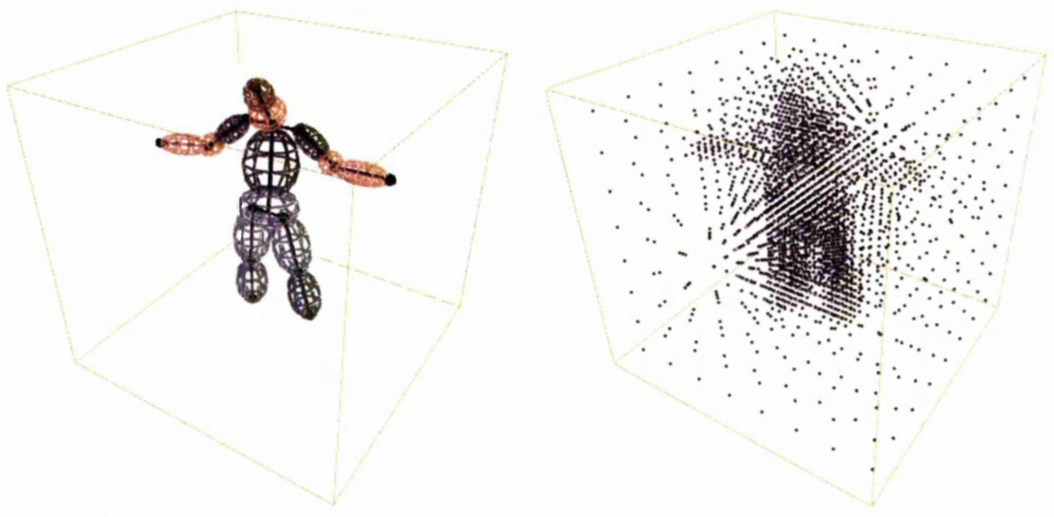

(a) Model and blobs tracked from the last frame (b) Initial subdivision where each black dot represents the centre of a voxel.

Figure 3.4: *Initial subdivision of the tracking area using a blob model as a prior, and the function* `MinDepth` *from Equation 3.1.*

as *foreground*. The result of this initial subdivision is shown in Figure 3.4. The cost of testing each voxel against the model is non-negligible, but since this is done only at low recursive depths, the number of voxels to test is low. This cost is still far lower than the one of having a finer but uniform initial subdivision, where far more voxels would have to be classified against the camera images.

Voxel Classification and Recursive Subdivision

Referring to Algorithm 3.2, the actual classification of a voxel occurs at line 3.2.2. This classification involves projecting the voxel onto the image planes, sampling from the corresponding areas and finally classifying the whole voxel. These steps will be detailed in the next section, and summarised in Algorithm 3.4. If the voxel is classified as either *foreground*, *background* or *unknown*, then this classification result is passed directly to the higher level voxel, from which the current one was recursively subdivided. The reason why the actual processing is not done at the recursive level of the current voxel is that additional post-processing is possible at a higher level, where knowledge about neighbouring sub-voxels is available.

Processing happens at the current level of recursion only if the current voxel has been classified as *edge*. At first (line 3.2.3), if the recursion level is higher than the maximal depth D^+, then the voxel is re-classified as *foreground*. This is a very simple way of limiting the complexity of the reconstruction. These voxels could also be re-classified as *background* to limit even more the total number of voxels, but a complete reconstructed volume is preferred.

If the current voxel is classified as *edge* and the recursion depth is still below the maximum D^+, then the voxel is divided into 8 sub-voxels. The subdivision is straight-forward because the size and the centre of the current voxel are known: if the current voxel has a size $s_\mathcal{V}$ and position $X_\mathcal{V} = (x_\mathcal{V}, y_\mathcal{V}, z_\mathcal{V})^T$, then the octants have a size equal to $s_\mathcal{V}/2$ and are centred on $(x_\mathcal{V} \pm \frac{s_\mathcal{V}}{4}, y_\mathcal{V} \pm \frac{s_\mathcal{V}}{4}, z_\mathcal{V} \pm \frac{s_\mathcal{V}}{4})^T$. The algorithm is then called recursively on each of these sub-voxels (line 3.2.4), returning the corresponding classification results. Once again, the current recursion depth is one less than the one of the sub-voxels which are going to be processed. This allows neighbouring sub-voxels to be taken into account for disambiguation. Of course, only 8 sub-voxels are available and not the rest of the neighbours, but this is sufficient for the basic post-processing presented here. More complex schemes would necessitate storing the octree structure and performing multiple passes.

Re-Classification of Unknown Voxels Using Spatial Continuity

The first post-processing task (line 3.2.5) is to decide on a better classification for the sub-voxels currently classified as *unknown*. The *unknown* class is temporary and should be disambiguated. The approach taken here is to look at the neighbour voxels and to re-classify an *unknown* voxel with the class of the majority. If an *unknown* voxel is surrounded by a majority of *foreground* voxels, there is a high probability that it should have been classified as *foreground* in the first place: it is then re-classified as such. This process is illustrated in Figure 3.5. The same is true for *unknown* voxels surrounded by *background* or *edge* neighbours. In the rare cases when an *unknown* voxel is surrounded by a majority of other *unknown* voxels, then the second most important representation is considered. An *unknown* voxel is allowed to keep its classification only if all its neighbours are also *unknown*, in which case a later scheme will pass the information for processing at a lower recursion level.

This approach is justified by the spatial continuity (or smoothness) of the reconstructed volume. In most other shape-from-silhouette algorithms, smoothness is enforced as a 2-D post-processing step on the extracted silhouettes. This has the disadvantage of ignoring other camera views, which could disambiguate a voxel in a more obvious way. Our approach waits until no more image evidence can be exploited to use the smoothness argument in 3-D.

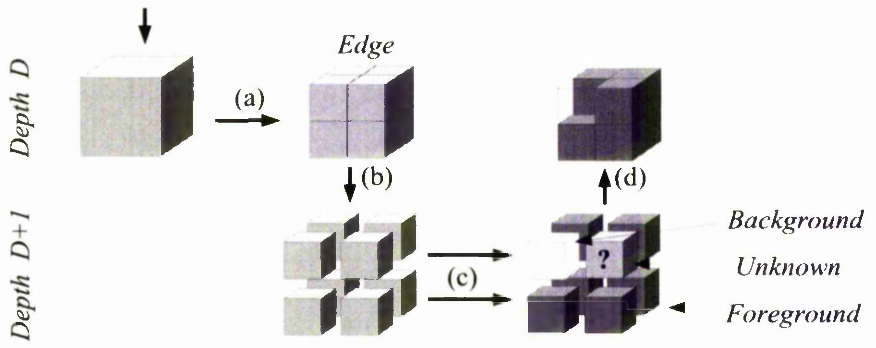


Figure 3.5: *At the current depth D of the algorithm, a voxel is classified as edge (a). It is then subdivided (b), and each sub-voxel is independently classified (c) at the next recursion level. When returning from the recursive calls (d), the sub-voxel classified as unknown takes the classification of the majority.*

Merging Groups of Voxels with Similar Classification

Still following the process flow of Algorithm 3.2, the second post-processing task consists in merging sub-voxels with similar classification into a higher-level voxel. This is simply done by checking whether all the sub-voxels have the same classification (line 3.2.6) and, if it is the case, passing the common classification to the previous recursion level (line 3.2.7). It can seem illogical to expect all sub-voxels to have the same classification, because the common classification should have been detected at the previous recursion level. However, the classification of a voxel is based on randomly sampled pixels, making it non-deterministic when voxels are close to a class boundary. Furthermore, sub-voxels that were first classified as *unknown* can now contribute to make all the sub-voxels alike. This merging process (illustrated in Figure 3.6) is quite simple, and still highly beneficial for compactness of the final reconstruction.

Finally, in the last line of Algorithm 3.2 (line 3.2.8), sub-voxels classified as *foreground* are "physically" created. This only involves passing their position, colour and recursion level to the next stage. When reaching line 3.2.8, the only possible classifications of the sub-voxels are *foreground*, *background* and *edge*. The sub-voxels classified as *edge* can safely be ignored because they have been dealt with at a lower recursion level. Only *foreground* sub-voxels are therefore created, and in all cases, the classification of the current voxel (necessarily *edge*) is passed to the previous recursion level.

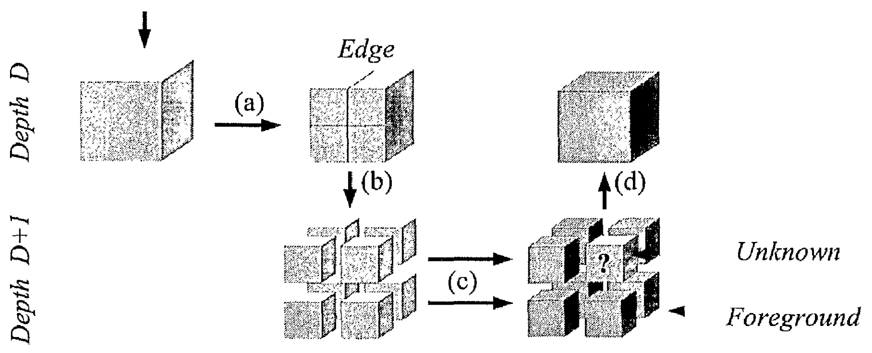


Figure 3.6: *The first steps (a-b) are the same as in Figure 3.5. All sub-voxels are now classified (c) with a compatible type. When returning from the recursive calls (d), unknown sub-voxels still take the classification of the majority, and additionally, if all sub-voxels have the same classification, the voxel at depth D is re-classified from edge to the common class.*

Flexibility of the Reconstruction Algorithm

The recursive nature of the algorithm leads to flexibility. Some basic post-processing can be done at very small extra cost, and since there is no static data-structure, every single parameter of the reconstruction process can be adjusted dynamically. For example, each of the voxels gets its position and size from its parent, which means that the initial position and size of the tracking space can be adjusted dynamically at no extra cost. This allows to track a subject within a relatively large space, without the need to reconstruct the whole tracking space.

In the same way, the maximal reconstruction depth, D^+, is easily adjustable. It can then be adapted depending on the resources currently available or the need for accuracy. For a global search (for example at initialisation), the maximal depth of subdivision can be relatively low (typically 5 or 6). On the contrary, when more accuracy is needed D^+ can be increased to up to 8, which would correspond to a static voxel-space of $256\times256\times256$. Of course, the performance of the reconstruction highly depends on the choice of D^+, but not as much as with standard voxel-based reconstruction techniques: only the edge voxels are here subdivided until a depth of D^+, which represents a low percentage of the total number of voxels.

3.3 Voxel Classification

This section presents the steps leading to the classification of an individual voxel from image evidence. These steps include the projection of the voxel onto the camera image planes, followed by the sampling of the projected area. An efficient sampling method using pre-computed patterns is introduced. Finally, the voxel classification scheme is detailed, largely based on the classification method introduced in Chapter 2.

3.3.1 Projection of Voxels onto Image Planes and Uniform Sampling

As seen in the previous section, the classification of a voxel depends on the nature of the data it refers to. Projecting a voxel onto an image plane is then equivalent to finding the image data associated with this voxel. A 3-D point can be projected onto the image plane of a camera in a straightforward and unique manner. A voxel, however, is a volumetric 3-D entity and its projection onto the image plane is a polyhedral 2-D area. The data associated with this voxel are all the pixels lying inside this projected area.

A standard way to project a voxel would be to project its eight corners (or vertices) and find the area delimited by the corresponding eight 2-D points. Since the projection of a convex object is itself convex, the projected area of a voxel is the convex hull of its projected vertices (Figure 3.7), which can be computed very effectively using for example the Quick-Hull Algorithm [BDH96]. Note that to be completely accurate, the camera lens distortions should be applied on the contour points of the convex-hull, instead of only the eight projected vertices.

Among the two steps (projection of the vertices and extraction of the convex-hull), the projection of voxel vertices can be implemented in an efficient manner: since adjacent voxels share the same vertices, a caching scheme can drastically reduce the number of projections. Lookup tables are also commonly used to speed up vertex projection but they imply a static zone of interest and a consequent memory usage. Moreover, vertex projection can now be achieved very efficiently exploiting the Single Instruction Multiple Data (SIMD) instruction sets featured in most CPUs. Unfortunately, convex-hull estimation is more computationally demanding.

We describe in Appendix B a novel caching method for real-time pixels sampling inside the projected area of the voxels. The image plane of each camera is first subdivided into small regions onto which voxels project with similar shapes of areas. For each of these regions, and for each desired number of pixel samples, we pre-compute offline a pool of patterns. To ensure a good repartition of the samples when

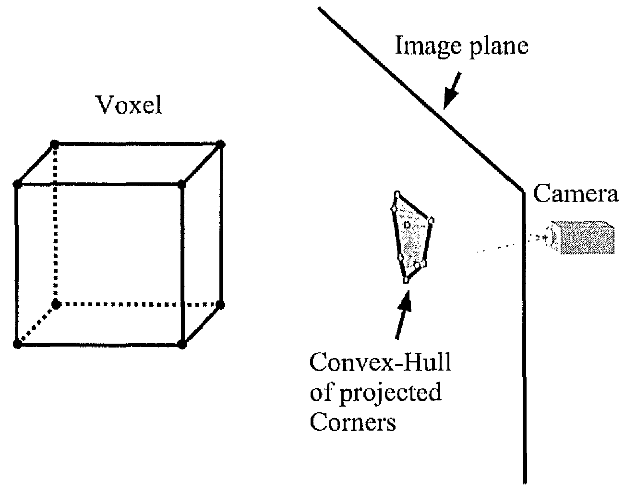


Figure 3.7: *Projection of a voxel onto an image plane: It is computed as the convex-hull of the eight projected vertices of the voxel.*

pre-computing the patterns, we use clustered random sampling with a heuristic maximising the spacing between samples. The online computation is then reduced to the projection of the centre of the voxel, the random selection of a pattern of samples, and the placement of this pattern onto the image plane.

3.3.2 Voxel Classification

We are now able to obtain sets of pixel-samples uniformly distributed inside the projected areas of any given voxel. For a given voxel $\mathcal{V}$, and a camera view c_i, let us denote these samples as $\mathcal{S}^i = \{s^i_1 \dots s^i_{N_s}\}$. Following Section 2.4.3 on image segmentation, $\mathcal{S}^i$ is classified into either *background*, *foreground* or *edge* categories. As we have seen, this classification is performed on all samples at once, improving robustness at no extra cost. Shadows are also handled (Section 2.5) in the classification process.

Re-visiting the Classification of a Set of Samples by Modelling Uncertainty

The classification of $\mathcal{S}^i$ into 3 categories (*background*, *foreground* or *edge*) is relatively reliable. However, due to the complexity of the backgrounds, the choice between one category and another can become uncertain. To keep background segmentation simple, this uncertainty was not previously taken into account, but since the classification of a voxel involves multiple views, a more robust scheme is now proposed. An extra category is then added to the previous set of possible classifications. The new *unknown* category is used to denote a set of samples for which there is no strong evidence justifying any of the 3 previous ones. It is then used as a fallback category, used in the last resort in the hope that other camera views will be able to disambiguate the situation. Indeed, each view brings additional information, leading to a final classification of the voxel. The *unknown* category is then mostly used as a temporary classification denoting uncertainty. However, if all the available camera views are not sufficient to disambiguate the situation, a voxel can be entirely classified as *unknown*, using its neighbours for disambiguation during the recursive reconstruction (see Section 3.2.3).

The new classification process of S^i including the *unknown* category is now detailed. The Mahalanobis distance between S^i and the model of the background associated with the camera c_i is $D_M(S^i)$. As seen in Section 2.4.3, the threshold on the Mahalanobis distance $D_M()$ is a function of the desired level of confidence α, and the number of degrees of freedoms $(3.N_s)$ associated with S^i. The threshold, $T_{3.N_s}(\alpha)$, can be approximated with good accuracy by a second order polynomial, allowing a fast evaluation for any number of samples.

Algorithm 3.3: Classification of a set of samples, including an *unknown* category. This is the partial classification of a voxel from a single camera view.

```
if $D_M(S^i) < T_{3.N_s}(\alpha_1)$ then
    return background;
else if $D_M(S^i) > T_{3.N_s}(\alpha_2)$ then
    if $\forall s_j^i \in S^i, D_M(s_j^i) > T_3(\alpha_2)$ then
        return foreground;
    else
        return edge;
    end
else return unknown;
```

Given two confidence levels α_1 and α_2, such as $\alpha_1 < \alpha_2$, the partial classification of a voxel from the camera view c_i is performed according to Algorithm 3.3. The choice of the confidence levels is still arbitrary, but it is independent of the number of samples and of the statistical properties of the model of the background. As a reminder, a confidence level represents the proportion of sample-sets that are truly part of the foreground when their distance to the model of the background is greater that the corresponding threshold. In practice, we chose a relatively low confidence level for the first threshold ($\alpha_1 \simeq 0.98$), which has the effect of letting through more sample-sets than usual, even if they could actually be part of the background. Indeed, since voxels classified as background in a single view are immediately discarded, this classification should be used only for cases exhibiting strong evidence. Similarly, the second confidence level used to decide whether a set of samples or an individual sample are part of the foreground, is chosen high ($\alpha_2 \simeq 0.999$) in order to minimise misclassifications. The large gap between the two thresholds is designed to encourage the use of the *unknown* category, which delegates the classification of uncertain cases to other camera views.

Classification of a Voxel Combining Information from all Views

The full classification of a voxel using all the available camera views is presented in Algorithm 3.4 and illustrated by Figure 3.8. The algorithm is essentially a combination of the ideas exposed earlier in this section. Voxels are projected consecutively onto the available image planes, producing for each view a set of pixel samples. This set of samples is then classified using Algorithm 3.3, and the partial classifications are combined to infer the final classification of the voxel using the following rules:

- *background* classification is chosen if the partial classification on at least one view is *background*, independently of the partial classification of other views.
- *foreground* classification needs at least one partial classifications to be *foreground* and the others to be either *foreground* or *unknown*.
- *edge* classification requires at least one partial *edge* classification with the rest classified as either *edge*, *foreground* or *unknown*.
- *unknown* classification is only used when all the available partial classifications are also *unknown*.

The position and the size of the zone of tracking are dynamically adjusted to the subject, which means that some voxels might not be visible at all time from all the cameras. While voxels are visible from at least one camera view, the classification occurs normally, and the only possible problem is a poorer classification as the number of views decreases. In the worst case, when a voxel is not visible from any of the camera views, it is classified by default as *background*. This choice is made to avoid those voxels having an influence on the later tracking process, as it happens relatively often that corners of the tracking space fall out of view using cameras with narrow focal.

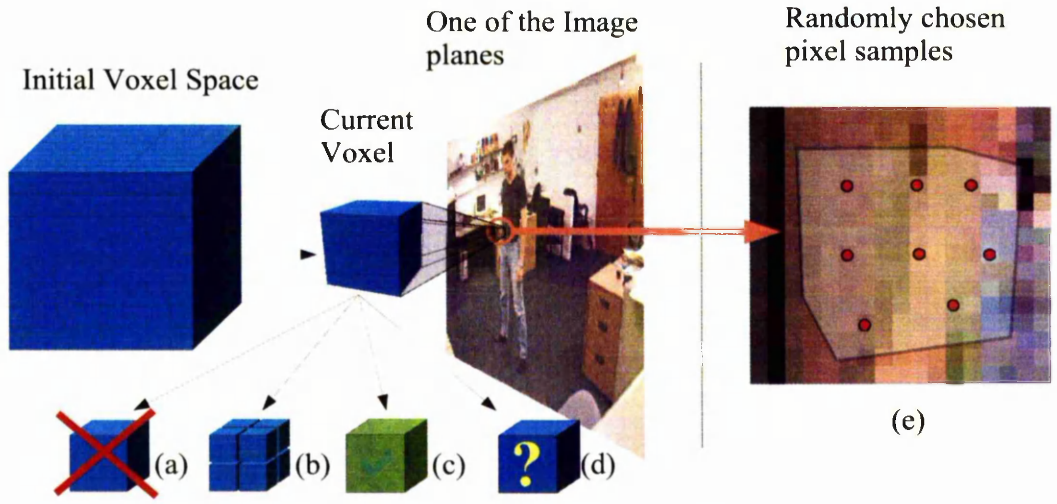


Figure 3.8: *Overview of the classification process. The current voxel is successively projected onto the image planes of the available cameras, and a set of pixel samples is chosen for the corresponding projected area (e). Note that only one camera-view is represented here. The classification then takes place, falling into either* background *(a),* edge *(b),* foreground *(c) or* unknown *(d) categories.*

3.4 Incorporating Colour in the Volumetric Reconstruction

The volumetric reconstruction we described only contains information about the shape of the object of interest. Of course, the colour of individual pixels was implicitly used to build the reconstructed volume, but this information was lost in the classification process. Considering that the tracking process relies solely on the generated voxels, it seems reasonable to recover as much useful information as possible from image data. Colour, texture or edges are possible sources of information that could help locate and disambiguate body parts.

Texture can be a powerful cue for tracking, but it is 2-D information which would be hard to integrate in our voxel-based reconstructed volume. Edges are easier to obtain, but incorporating them from multiple views remains a difficult problem. It is also not clear how beneficial edges could be for tracking, if we keep in mind that external ones are already included in the reconstruction itself.

Colour information has the advantages of being relatively easy to collect and to represent in 3-D. Even if colour is not volumetric information (only the colour of the external surface of an object is visible), it can be "spread" so that internal voxels are also assigned a sensible colour. In the context of human-body tracking, where body parts need to be differentiated, colour is particularly important. For example, using

Algorithm 3.4: Overview of the classification of a voxel

```
Data: Voxel $\mathcal{V}$ (position and size)
Result: Classification of $\mathcal{V}$ as either foreground, background, edge or unknown
TempClass ← unknown ;
NbVisibleViews ← 0;
forall Camera Views, $c_i$ do
    ▷ project $\mathcal{V}$ onto the image plane of camera $c_i$ (cf. Section 3.3.1);
    if $\mathcal{V}$ is visible from the camera $c_i$ then
        NbVisibleViews ← NbVisibleViews +1;
        ▷ sample uniformly the projected area (cf. Section B);
        ▷ classify the set of samples $S^i$ (cf. Section 2.4.3 and Algorithm 3.3);
        if $S^i \in$ background then
            return background;
        else if $S^i \in$ foreground & TempClass $\neq$ edge then
            TempClass ← foreground;
        else if $S^i \in$ edge then
            TempClass ← edge;
        end
    end
end
if NbVisibleViews $> 0$ then
    return TempClass ;
else
    return background;
end
```

colour helps with localising the hands and the face.

Unlike most existing systems which encourage tight and uni-colour clothing, our system highly benefits from real-life clothing. No restriction is therefore imposed on the type or the colour of clothes: the shape and colour of the appearance model are learnt automatically during the first frames of the tracking process (Section 4.3). Of course, the more important the colour differences, the more valuable colour becomes for tracking.

3.4.1 Including All Possible Colours into each Voxel

Extracting the true colour of each voxel would require a visibility test from each camera. Voxels would then be assigned a weighted average of the colours of their projections onto the cameras from which they are visible. Typically, a bigger weight would be given to cameras with a view angle closer to the surface normal at the position of the voxel. This is clearly impractical for our purpose, not only because internal voxels are then omitted or because we do not have surface normals, but mainly for the computational cost associated with visibility tests.

We assume that pieces of clothing are relatively uniform in colour, so that a given point of a body part can always be seen with the same colour from at least one camera. This means that, for example, a voxel on the hand of the subject always projects onto a skin-coloured patch in at least one camera view. This is not an unreasonably restrictive constraint, since most clothes are already uniform in colour, and so is the skin. A voxel placed in the middle of the torso will then be seen with the colour of the shirt from most cameras, and, even when this point is occluded from a few camera views, it will almost certainly remain visible with the same colour from at least one view.

If the "correct" colour of each voxel is visible from at least one camera view, then we simply have to select the camera view giving the best colour (with respect to the model), and give the corresponding colour to the voxel. There is obviously no such thing as a correct colour, but since we will have a model with an expected colour for each body part, the colour closest to the expectation is considered as the correct one. In practice, this means that the colours from each camera view are included into the voxels, and the best colour is selected later on, during the tracking process (Section 4.2).

The colour of a voxel as seen from a camera view is determined by averaging the pixel samples that were used for classification in the previous section. In theory, voxels can overlap areas with different colours, making a simple average of the samples inaccurate. However, in practice, the patches with uniform colour are relatively large, so we assume that voxels project onto zones of uniform colour in at least one camera view. Even when this is not the case, the model is robust enough to incorporate resembling colours. The colour assignment could be made more robust by using the median instead of the mean colour of the samples, but it would be slower as it involves sorting the colours associated with the samples.

A voxel $\mathcal{V}$ is then fully described by its 3-D position $X_{\mathcal{V}}$, its size $s_{\mathcal{V}}$, and its mean colours $\mathcal{C}_{\mathcal{V}}=\{\mathcal{C}_{\mathcal{V}}^1,\dots,\mathcal{C}_{\mathcal{V}}^{N_c}\}$ as seen from the N_c available camera views:

$$\mathcal{V} = \{X_{\mathcal{V}}, s_{\mathcal{V}}, \mathcal{C}_{\mathcal{V}}\} \quad \text{with} \quad \mathcal{C}_{\mathcal{V}}^i = \frac{1}{N_s}\sum_{j=1}^{N_s} s_j^{c_i} \tag{3.2}$$

Storing the colours from all camera views inside each voxel would normally consume too much memory to be practical. However, for tracking purposes, voxels do not

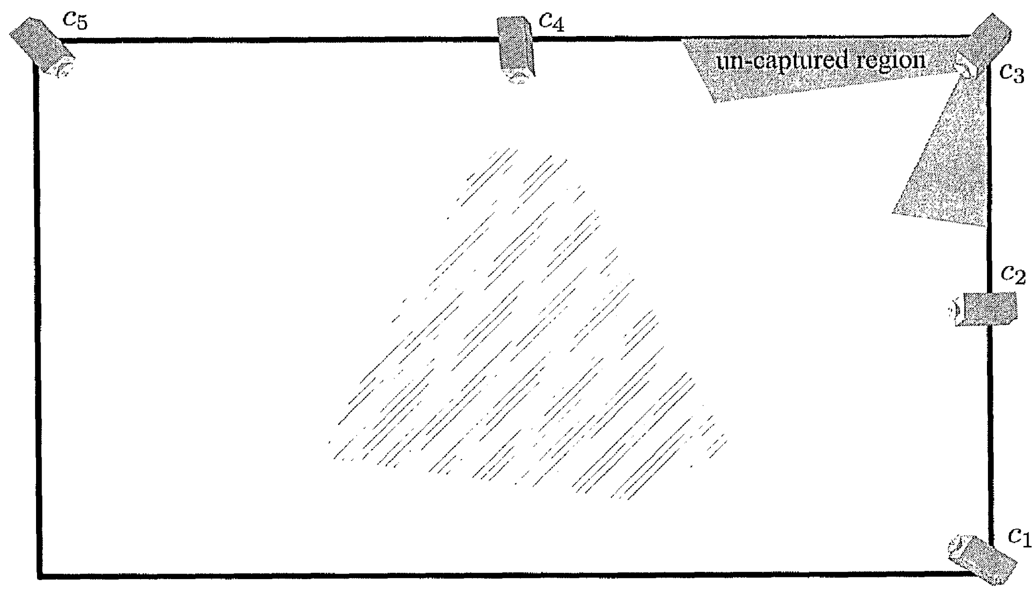


Figure 3.9: *Placement of the cameras. The hatched zone represents the space where the subject is visible from all cameras.*

need to be stored in memory: as we will see in Chapter 4, as soon as a voxel is created and its colours assigned, it is incorporated into the blobs of the model. The only overhead is then the test of each colour against the appearance model to determine the most appropriate colour, but this is minimal compared to the cost of performing visibility tests.

3.5 Results

The placement of the 5 cameras used for our tests is depicted in Figure 3.9. All cameras were placed at the height of the ceiling (approximately 2.5 metres), and orientated towards the same tracking space (hatched on the figure). Note the presence of an un-captured space, where additional people can stand without being visible from any camera.

An example of volumetric reconstruction using a maximal recursion depth of 7 and 4 camera views $\{c_1, c_2, c_3, c_4\}$ is shown in Figure 3.10. The pixels sampled during the reconstruction are superimposed onto the original input images with red spots. It can be noticed that fewer and fewer samples are taken outside the subject in successive camera views, making the algorithm efficient and robust to partial changes in the background. The reconstructed volume is shown from 3 different viewpoints, with voxels displaying

	3 views	4 views	5 views
Max. Depth 6	11.0ms (90.4fps)	11.1ms (90.2fps)	12.8ms (78.2fps)
Max. Depth 7	24.1ms (41.4fps)	22.8ms (43.8fps)	28.1ms (35.6fps)
Max. Depth 8	73.5ms (13.6fps)	66.0ms (15.1fps)	78.6ms (12.7fps)

Table 3.1: *Performance of the volumetric reconstruction.*

the average of the 4 view-colour.

Evaluating the accuracy and the robustness of the reconstruction is tricky because no ground truth measurement is available. In Figure 3.11, we compare the total reconstructed volume across time for different numbers of camera views. The reconstructed volume should ideally remain stable, at a value approximating the true "volume" of the subject. A minimum number of 4 camera views seems required to obtain a relatively stable reconstructed volume. Five or more camera views certainly improve the results, but in lower proportions. Note that the relative placement of the cameras has a strong influence on the result, but an optimal placement was not pursued in this thesis.

The benefits of the hierarchical reconstruction scheme are demonstrated in Figure 3.12, where the contributions of the voxels are aggregated by size. Only half of the total volume is represented by voxels with maximal subdivision (mainly near the edges), while the rest of the reconstructed volume is advantageously represented by higher-level voxels.

Performance measurements of the full reconstruction algorithm for various numbers of camera views and maximal recursion depths are presented in Table 3.1. These measurements were performed on a 2 GHz Pentium computer, and averaged over 1000 frames. Quite remarkably, the number of camera views does not seem to have a strong influence on the computing cost. Background voxels are indeed more likely to be discarded at an early stage as more views are available. The cost of the projections on the extra views seems to be compensated by the benefits of a faster classification. We could not test this hypothesis further because of the limited number of cameras we had access to, but we could expect the computing cost to grow more slowly as more views are added.

The performance results from Table 3.1 can also be compared to the cost of background segmentation on single input images (Table 2.4). Using combined information from all available views and per-sample segmentation, the cost of the full reconstruction is lower than for the individual segmentation of all the input views.

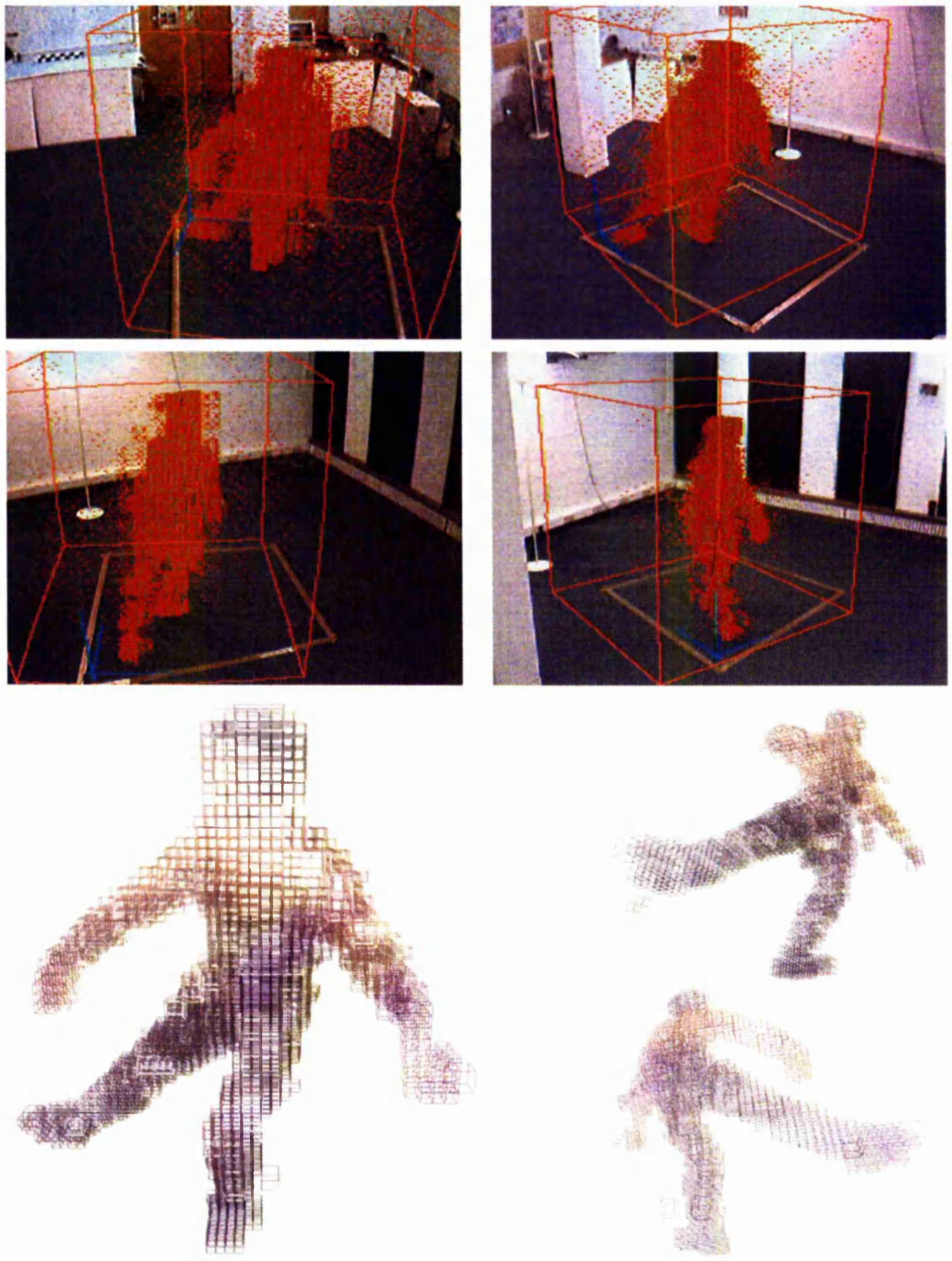


Figure 3.10: *The pixel samples used to reconstruct the visual-hull are represented by red dots in the 4 top input images. In the first camera view, the samples cover all the tracking area, but quickly concentrate on the foreground sections in the following views. The corresponding reconstructed volume is shown at the bottom from 3 different viewpoints.*

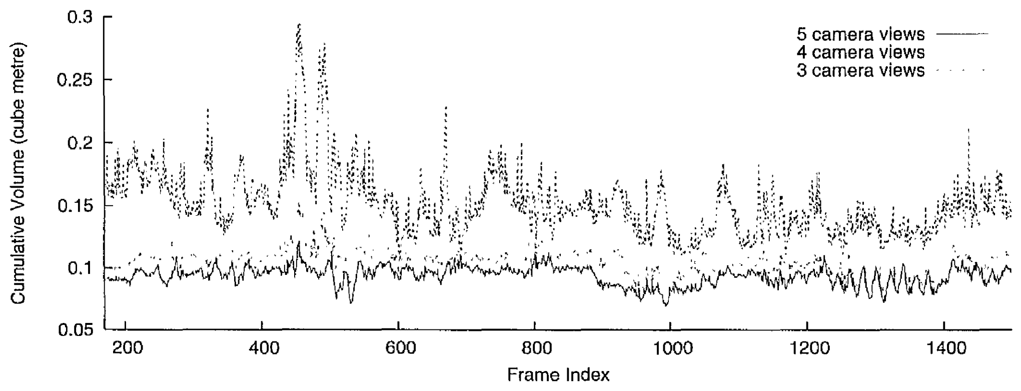


Figure 3.11: *Influence of the number of camera views on the reconstructed volume.*

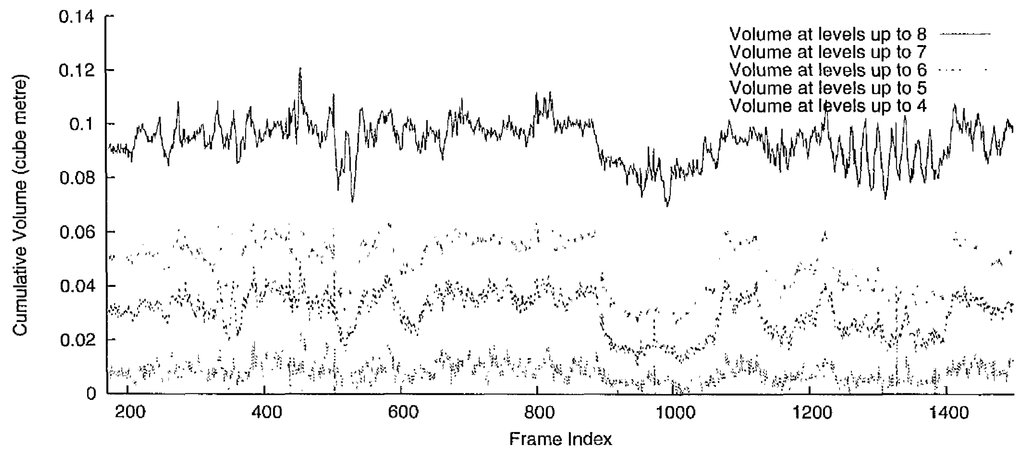


Figure 3.12: *Decomposition of the total reconstructed volume with* 5 *camera views into the contribution of the voxels at each recursive level.*

3.6 Discussion and Conclusion

In this chapter, we presented a novel shape-from-silhouette reconstruction method. Original contributions fall into the main categories:

- *Efficiency*: hierarchical reconstruction framework with minimal memory footprint.
- *Flexibility*: the zone of tracking can be dynamically adjusted, and the initial voxel subdivision is defined by the tracked model.
- *Robustness*: the voxel classification uses the full statistical properties of the sets of pixel samples. The introduction of a *unknown* category allows information from all views to be combined for disambiguation.

- *Colour*: inclusion of colour information in the reconstructed volume, at very low computing cost.

As future work, the statistical framework could be further developed by augmenting each voxel with its probability of being correctly classified. Voxels reaching the maximal subdivision level, which are classified as *foreground* under the current scheme, could then contribute to the reconstructed volume in a more nuanced way.

The inclusion of colour information could be made more compact and efficient by combining similar colours for each voxel. However, in order to compare colours across different views, cameras should be colour-calibrated. We believe that colour calibration is an important issue which could greatly improve the performance of the method. Automatic colour calibration could be achieved, for example, by using the blob-based appearance model to learn the mean colour of each body part and subsequently define a colour-correction function for each camera view.

Chapter 4

Tracking Body Parts with 3-D Blobs

This chapter presents the use of 3-D blobs for tracking individual body parts. The blobs are Gaussian models, giving them a strong statistical predisposition towards robust optimisation algorithms like Expectation-Maximisation. We shall describe the specifics of the Expectation-Maximisation framework when tracking sets of coloured voxels with blobs. We shall finally present a scheme for automatically acquiring and constraining the attributes of the blobs.

4.1 Blobs as Feature Trackers

Human body tracking involves finding the global position of the body as well as the relative position of each body part, for each frame of a video sequence. In order to recognise and follow these body parts from frame to frame, an *appearance model* is necessary. Given that the data available for tracking is the set of coloured voxels reconstructed in Chapter 3, the appearance model of a body part should also be a volumetric description of both its shape and colour.

Acknowledging that the human body is articulated as a whole, individual body parts (thigh, forearm, etc.), on the contrary, remain relatively static in appearance. Of course, the exact shape of each body part depends on the underlying muscles attached to the skeleton, so that their shape changes slightly as the body moves. With clothing, any movement combined with the laws of physics effectively changes the appearance (shape and illumination) of each part of the body. These considerations, despite being valid for high-quality models, are currently inapplicable to human body tracking because of both impractical complexity and the relative coarseness of the 3-D reconstruction. We neglect these small variations in shape and illumination, and assume the existence of a manageable number of elementary body parts, whose appearance is self-coherent (unimodal in space and colour) and remains constant over time.

After a review of some alternative methods, this section introduces a statistical appearance model for tracking individual body parts directly from voxel data. We refer to this as the "blob" model. The properties of this blob model is then examined to pave to way for the tracking process itself, which is presented in Section 4.2.

4.1.1 Appearance Models in the Literature

Various types of model have been described in the literature for identifying body parts. In the vast majority of cases, the models are purely geometric and attached to an underlying articulated skeleton. When the final purpose is tracking, the models are kept simple to achieve an acceptable level of performance. For example, Hogg [Hog83] uses cylinders attached to the bones of a hierarchical kinematic model. Candidate configurations of the model are evaluated in 2-D by comparing the projected cylinders and edges extracted from the image. Mikic *et al.* [MTHC03] use a cylinder for the torso and ellipsoids for the limbs: tracking is then done from 3-D voxels with an error function accounting for the number of voxels left outside these geometric primitives. Mitchelson *et al.* [MH03] use similar primitives, but this time in association with a particle filtering framework and a combination of image filters. Delamarre *et al.* [DF99] have a slightly more complex representation including parallelepipeds, truncated cones and spheres. The model is then projected onto camera images and compared to edge information.

These simple geometric primitives have proved sufficient to model the shape of the limbs with reasonable accuracy. However, in an attempt to improve realism, more complex representations based on polyhedral meshes [YSK$^+$98, BL01b] or superquadrics [GD96] have been proposed. Pushing further towards realism, Plänkers *et al.* [PF03] and Carranza *et al.* [CTMS03] propose a muscular model using "metaballs". These metaballs are in fact Gaussian density distributions, from which the skin is defined as an iso-surface. The idea is related to the blobs we propose in this chapter, but the purpose of [PF03, CTMS03] is to achieve higher realism as opposed to robustness. Their underlying idea is that more realistic models should lead to a better evaluation of the pose. While this is an incontestable statement, one must keep in mind that realism has always a limit, especially with complex clothing. It is then not clear whether a more detailed model handles the variability of the data as flexibly as a simple one.

Focusing more on robustness than on accuracy, Borovikov *et al.* [BD02] devise a density-based model, where body parts are defined as isosurfaces of customised statistical distributions. The optimisation process is based on the distances between reconstructed voxels and these distributions, following gradients of the distributions to converge towards a solution. This scheme, used in conjunction with hierarchical fitting, seems to have the potential to estimate body poses robustly, even when no prior estimation is available. As we will see, the blobs we propose have similar desirable properties, but with the advantage of a simpler statistical description, making them more suitable for real-time tracking.

Body Representation using Blobs

A *blob* is one of the simplest possible descriptions of a unimodal set of data: it encapsulates only the average (or expected) value of the set of data, and the possible variations of the data around this average value. In statistical terms, a blob can be regarded as a Gaussian distribution, so that a mean vector and a covariance matrix fulfil these roles. With respect to spatial information, a blob is often represented by an ellipsoidal shape, which is actually an iso-surface around the mean value. Similarly, the colour information is modelled by a mean colour and a model of variations in colour-space. By contrast with previously-described appearance models based on geometrical primitives, a blob is a statistical entity without hard boundaries. A given voxel has then a probability of belonging to the blob, based on its position and colour.

Blobs are well-adapted to describe the shape and colour information of body parts. Indeed, these elementary body parts (the limbs) can all be approximated by ovoid shapes at a certain level of accuracy: the most complex body parts can always be modelled by more than one blob. Moreover, as seen in Section 3.4 about voxels and colour information, the colour of individual body parts is assumed to be uniform. This means that a single blob should be able to represent both the shape and the colour of an individual body part.

Blobs in the literature

In the Pfinder algorithm [WADP97], Wren *et al.* use 2-D blobs to model the limbs. The blobs are described by mean vectors in position and colour, and block-diagonal covariance matrices. The attributes of the blobs are re-evaluated in each frame from their *support map* of pixels. The support map of a blob is the set of data which is believed to correspond best to the blob model. Since no articulated model is used to constrain the tracking, the blobs are simply propagated between each frame using linear dynamics. Initialisation relies on the extraction of contours, with the identification of specific body parts such as the hands and the head. Pfinder remains a crucial reference in the field of human-body tracking because it was one of the first real-time algorithms that was sufficiently robust to be really usable. However, the fact that it works only in 2-D, and the absence of an underlying articulated model leave some areas of improvement, many of which are addressed in this chapter.

Blobs have also been used more recently to model body parts, but often under different names and formulations. Aguiar *et al.* [dATM$^+$04] use an "ellipsoid" formulation, where each 3-D ellipsoid is described by a translation vector, three rotation coefficients, and three elongation coefficients. This geometrical formulation is roughly equivalent to the statistical approach which we propose, where a blob is assimilated to a Gaussian distribution. The statistical formulation, however, is more compact and lends itself better to formal analysis. Ellipsoid shells have also been used by Cheung *et al.* in [CKBH00, Che03, CBK03a], with once again a geometrical description preferred to a statistical one. The parameters of the blobs are then estimated using the three first-order moments of the underlying data, which is equivalent to estimating the mean and covariance matrix. Bregler *et al.* [BM98, BMP04] define 3-D blobs geometrically in the local coordinate system of each body part. They also describe an Expectation-Maximisation procedure with pixels support maps, used to re-estimate iteratively the parameters of the kinematic model (as opposed to these of the blobs).

To the best of our knowledge, the only example of 3-D blobs used for human-body tracking and described with a Gaussian distribution formulation is due to Jojik *et al.* [JTH99]. In their paper, the upper-body is tracked from dense disparity maps, using 3-D Gaussian models for each body part. The tracking itself is performed using Expectation-Maximisation, and the parameters of an underlying articulated model are computed in an Extended Kalman Filtering (EKF) framework [Kal60, WB01]. A scheme is also introduced to detect self-occlusions. The algorithm is reported to run in real-time. While this work is clearly targeted at narrow-baseline stereo setups, it has a lot of common ground with the method we describe in this chapter. Nevertheless, we shall describe in this chapter various extensions to the formulation of the blobs, such as the inclusion of colour or the automatic acquisition at initialisation. The scope of our work is also different, as we are interested in full-body tracking from a volumetric reconstruction, instead of the upper-body from disparity maps as in [JTH99].

The main advantage of Gaussian blobs over geometrical shape primitives is their natural integration into a statistical framework. Compactness and simplicity are other decisive advantages of this blob description, which has a minimal amount of free parameters. Keeping in mind that all voxels have to be tested against the appearance models of all body parts, the test itself must be simple enough to keep real-time performance. As will be seen in the next sections, computing the probability that a pixel belongs to a blob is much faster than the corresponding test with most other shape primitives, such as cylinders or parallelepipeds.

Finally, because of their relatively small number of parameters, blobs are very flexible and can be dynamically modified. This is especially important for body parts whose shape and colour have to be learnt during tracking. The next section formalises the description of blobs and explains their use for tracking through a standard technique called "Expectation-Maximisation" [DLR77]. Another use of blobs for Bayesian tracking will be presented in Chapter 6.

4.1.2 Theoretical Background and Notation

The data modelled by the blobs are the 3-D locations of the voxels in Euclidean xyz space, augmented by their colours in YUV colour space: each datum is a 6-dimensional vector. A blob is formally defined as a 6-dimensional multivariate Gaussian distribution of mean vector $\boldsymbol{\mu}$ and covariance matrix Σ. The voxels (position and colour) belonging to the blob are assumed to be distributed in a Gaussian way around the mean vector $\boldsymbol{\mu}$, hence the ellipsoidal shape mentioned earlier. The mean vector $\boldsymbol{\mu}$ of a blob is composed of a mean position μ_X and a mean colour μ_C as follows:

$$\boldsymbol{\mu} = (\, \underbrace{\mu_x\ \mu_y\ \mu_z}_{\mu_X}\ \underbrace{\mu_Y\ \mu_U\ \mu_V}_{\mu_C}\,)^T = \begin{pmatrix} \mu_X \\ \mu_C \end{pmatrix} \tag{4.1}$$

The vectors μ_X and μ_C are different in nature, but due to the normalising effect of the covariance matrix, they can be associated coherently. The 6×6 covariance matrix, Σ, is decomposed into semantic blocs in a similar way:

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} & \sigma_{xY} & \sigma_{xU} & \sigma_{xV} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} & \sigma_{yY} & \sigma_{yU} & \sigma_{yV} \\ \sigma_{xz} & \sigma_{yz} & \sigma_z^2 & \sigma_{zY} & \sigma_{zU} & \sigma_{zV} \\ \sigma_{xY} & \sigma_{yY} & \sigma_{zY} & \sigma_Y^2 & \sigma_{YU} & \sigma_{YV} \\ \sigma_{xU} & \sigma_{yU} & \sigma_{zU} & \sigma_{YU} & \sigma_U^2 & \sigma_{UV} \\ \sigma_{xV} & \sigma_{yV} & \sigma_{zV} & \sigma_{YV} & \sigma_{UV} & \sigma_V^2 \end{pmatrix} = \begin{pmatrix} \Sigma_X & \Sigma_{XC} \\ {\Sigma_{XC}}^T & \Sigma_C \end{pmatrix} \tag{4.2}$$

where Σ_X is the spatial covariance matrix, Σ_C is the colour covariance matrix, and Σ_{XC} is a joint covariance, describing the dependencies between position and colour. In the few previous uses of coloured blobs in the literature [WADP97], this joint covariance matrix was ignored and set to zero. This was due to the fact that blobs were only in 2-D, leaving the orientation of blobs uncorrelated with colour. In 3-D, however, position and colour can be correlated as long as the blobs are kept in the local coordinate system of the body part that they are tracking. For example, a blob representing a forearm in global coordinates can be transformed into the local coordinate system of the forearm, with the x axis always pointing from the elbow to the wrist. In this local coordinate system, the first column of the joint covariance matrix Σ_{XC} will always represent the colour variations between the elbow and the wrist, which can be a very valuable information for the repartition of the blobs. Note that an underlying kinematic model (described in Section 5.1) will be necessary to keep track of the hierarchy of coordinate systems, and perform the appropriate transformations.

Nevertheless, all dependencies between position and colour are not pertinent, especially considering the extra cost they generate. Setting Σ_{XC} to zero effectively splits the matrix Σ into 2 sub-matrices, considerably speeding-up matrix inversion. Furthermore, the 3-D reconstruction is often not detailed enough to account for colour variations along short axes: for most blobs, the only observable dependence between colour and position occurs along the main axis only. When such a dependence is observed, we prefer to reorganise the blobs dynamically in order to minimise this dependence. So, without disregarding the correlations between position and colour (Σ_{XC}) completely, we will at this point of the analysis neglect their influence for efficiency reasons. Dynamic blob behaviour and the use of Σ_{XC} for re-organising the blobs is detailed in Section 4.3.

Since blobs are multivariate Gaussians, all the reasoning described in Chapter 2 remains valid. In particular, considering a voxel $\mathcal{V}$ at position $X_{\mathcal{V}}$ and of colours $C_{\mathcal{V}} =$

$\{C^1_\mathcal{V}, \dots, C^m_\mathcal{V}, \dots, C^{N_c}_\mathcal{V}\}$ as seen from the N_c available camera views, the Mahalanobis distance $D_M(\mathcal{V}, B)$ between this voxel and a blob $B(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is:

$$D_M(\mathcal{V}, B) = (\mathcal{V} - \boldsymbol{\mu}) \cdot \boldsymbol{\Sigma}^{-1} \cdot (\mathcal{V} - \boldsymbol{\mu})^T \tag{4.3}$$

we can expand this as:

$$\begin{aligned} D_M(\mathcal{V}, B) &= \begin{pmatrix} X_\mathcal{V} - \mu_X \\ C^m_\mathcal{V} - \mu_C \end{pmatrix} \cdot \begin{pmatrix} \Sigma_X & \Sigma_{XC} \\ {\Sigma_{XC}}^T & \Sigma_C \end{pmatrix} \cdot \begin{pmatrix} X_\mathcal{V} - \mu_X \\ C^m_\mathcal{V} - \mu_C \end{pmatrix}^T \\ &\simeq \begin{pmatrix} X_\mathcal{V} - \mu_X \\ C^m_\mathcal{V} - \mu_C \end{pmatrix} \cdot \begin{pmatrix} \Sigma_X & 0 \\ 0 & \Sigma_C \end{pmatrix} \cdot \begin{pmatrix} X_\mathcal{V} - \mu_X \\ C^m_\mathcal{V} - \mu_C \end{pmatrix}^T \\ &= \underbrace{(X_\mathcal{V} - \mu_X) \cdot {\Sigma_X}^{-1} \cdot (X_\mathcal{V} - \mu_X)^T}_{D_M(X_\mathcal{V}, B)} + \underbrace{(C^m_\mathcal{V} - \mu_C) \cdot {\Sigma_C}^{-1} \cdot (C^m_\mathcal{V} - \mu_C)^T}_{D_M(C_\mathcal{V}, B)} \end{aligned} \tag{4.4}$$

The decomposition of the 6-dimensional Mahalanobis distance into a sum of distances on position and colour is only valid if $\Sigma_{XC} = 0$, but this approximation is important for the real-time feasibility of the computation. The other remarkable part of the equation is the Mahalanobis distance on colours, which uses only the colour vector $C^m_\mathcal{V}$ minimising the distance to the blob model:

$$m = \arg\min_{i=1\dots N_c} (C^i_\mathcal{V} - \mu_C) \cdot {\Sigma_C}^{-1} \cdot (C^i_\mathcal{V} - \mu_C)^T \tag{4.5}$$

Even if this formulation can seem expensive in terms of computational cost, it is actually efficient and accurate even in cases of severe self-occlusions. The Mahalanobis distance in 3 dimensions is indeed fast to evaluate.

The distance formulations (Equations 4.4 and 4.5) presented above are very similar to the ones used for background segmentation. A major difference, however, is that we are not trying here to decide whether a voxel belongs to a single blob (or Gaussian model), but rather to assign a voxel to the most probable blob. In the first case we had only one point of comparison, and had consequently to find a threshold for classification. In the current scheme, there is no need for a threshold since a voxel is assigned to the blob with highest probability $P(\mathcal{V}|B)$.

4.2 Tracking with Expectation-Maximisation

This section explains how Expectation-Maximisation (known as EM) can be used for tracking, first in a general context and then more specifically for the estimation of the parameters of the blobs from the voxel data. A general overview of the algorithm brings us to discuss its pitfalls and how they were addressed in the literature. The main two steps of the algorithm are then detailed.

4.2.1 Overview

Expectation-Maximisation was first introduced by Dempster *et al.* [DLR77] in 1977, as a way to compute iteratively the maximum likelihood estimate of the parameters of a model from incomplete data. This algorithm subsequently became very popular in many areas of Computer Science, mainly because it can be efficiently implemented as a loop over two simple steps. The general principle of the method is summarised in Algorithm 4.1. At each iteration, the parameters of the model are re-evaluated from the data that were the most likely to corroborate the model in the first place. Figure 4.1 presents a visual overview of the EM algorithm for the fitting of blobs.

Algorithm 4.1: General overview of the Expectation-Maximisation algorithm.

```
repeat
    ▷ Expectation: each datum is assigned with the probability that it has been
    generated by the current estimate of the model;
    ▷ Maximisation: the model parameters are re-evaluated using the data
    weighted by the probabilities;
until convergence;
```

The problem of Initialisation

In our case, the models are Gaussians distributions (the blobs) and the data are the reconstructed voxels. A first limitation of the algorithm is that the "structure" of the model has to be given *a priori* because only the free parameters of the model are optimised. Practically, this means that the optimal number of Gaussian blobs is not estimated by the algorithm, and must be chosen beforehand. Providing a good initial number of blobs is not an easy task, but using a skeletal model of the human body, we can have a rough idea of the number of body parts to track. The refinement of this estimation will be the topic of Section 4.3.

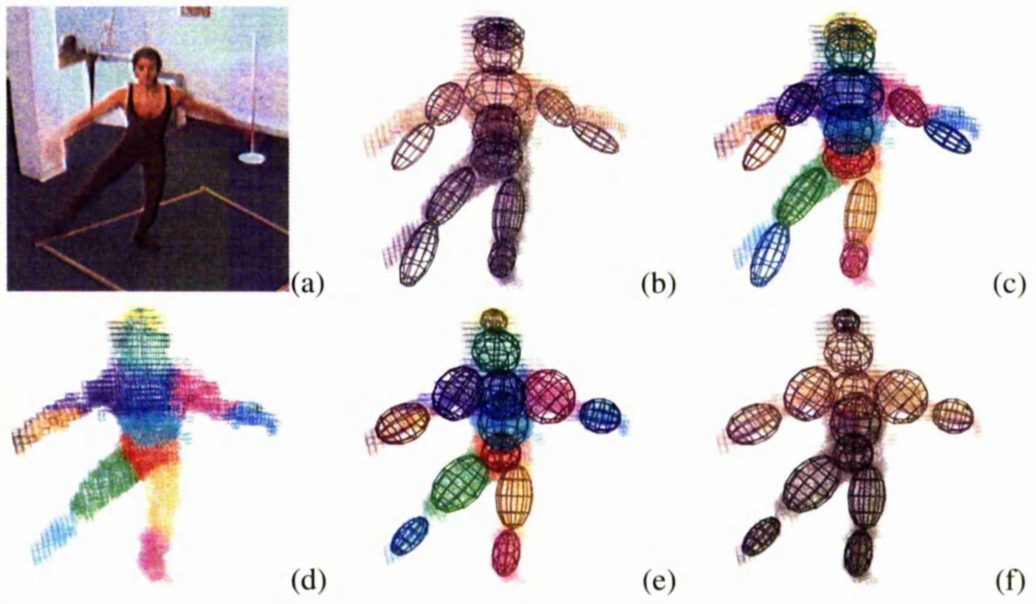


Figure 4.1: *Overview of the blobs fitting process. (a) One of the input views and (b) the corresponding volumetric reconstruction with the predicted blobs. (c) The voxels are assigned to the nearest blob using both position and colour. (d) The sets of voxels assigned to each blob are exhibited with arbitrarily chosen colours. (e) The attributes of the blobs are then re-evaluated from the set of voxels that were assigned to them, (f) giving the new set of blobs.*

Another pitfall of EM comes from its iterative nature, where the data are selected with respect to their agreement with the model. The initial parameters of the model must then explain the data in a satisfactory way, to avoid wrong data selection and sub-optimal convergence. So, blobs must then be sufficiently "close" to the data at each frame. The "closeness" is here understood in the sense of the Mahalanobis distance (Equation 4.4) between blobs and voxels, hence including colour as well as position. Since tracking is a continuous process, the pose of the subject at a given frame is assumed to be sufficiently close to the pose in the previous frame: the initial spatial parameters are then simply taken from the previous frame. The colour of body parts is also assumed to remain relatively constant over time, so that the tracked blobs from the previous frame should provide a sufficient initialisation.

Figueiredo and Jain [FJ02] proposed a few extensions to the standard EM algorithm designed primarily at initialising the algorithm in an unsupervised (automatic) way. The optimal number of Gaussian models is chosen with a Minimum Description Length (MDL) criterion, and the problem of the initial placement of the Gaussian models is solved by first creating a Gaussian model for each data point and then iteratively discarding those with low support from the data. For an analysis of the required level of support, based on the MDL, the reader is referred to [FJ02]. These additions can prove very helpful when no assumption can be made about the data, hence avoiding a totally arbitrary choice of model. Unfortunately, several numbers of components have to be tried before selecting the best one, which makes the algorithm computationally demanding. Furthermore, in our case, the human-body model can provide the expected number of components (blobs), at a significantly lower cost.

Soft Assignment versus Support Maps

The standard EM algorithm uses *soft-assignment* of the data with respect to the model components. Each voxel can then contribute to the re-estimation of the parameters of all the blobs, with a weight proportional to the probability that it had been generated by each particular blob. While this approach is theoretically valid and allows the algorithm to perform well even with a small data set, it can also cause specific issues. The main problem concerns the overlapping of the blobs, which can be desirable in some applications, but makes little sense when blobs are supposed to represent solid body-parts. Soft-assignment tends to favour overlapping in dense regions, making blobs "fuzzier" than they really should. A standard way to solve this particular problem is to use hard-assignment or *support maps*. With hard-assignment, a voxel is assigned only to the blob that is the most likely to have generated it, hence building a binary support map for each blob. The parameters of the blobs are re-estimated only from the voxels belonging to its support map, which prevents overlapping, and improves performance. Note that with support maps, EM is very similar to the K-Means algorithm. The only difference is that in EM each cluster is modelled by a full statistical model, while K-Means uses only the centres (means) of the clusters.

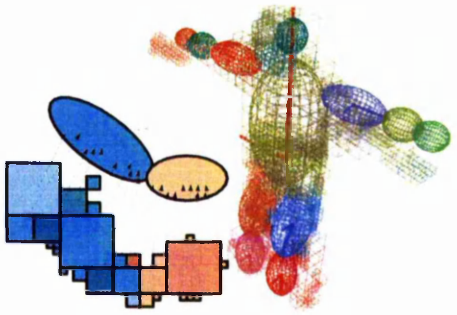
Using support maps also has some drawbacks. Since the assignment of the voxels is binary with respect to each blob, the support maps do not have real Gaussian distributions. Indeed, the "tails" of the Gaussians would normally be modelled by voxels with low probabilities, but instead, those voxels have high chances of being allocated to more likely blobs. This theoretical concern does not seem to be a practical issue as long as a relatively high number of voxels is available in each support map, allowing a sufficiently accurate estimation of the blobs' parameters. Support maps are widely used to constrain EM, and have successfully been used in the context of human-body tracking by Wren *et al.* [WADP97] and Jojic *et al.* [JTH99].

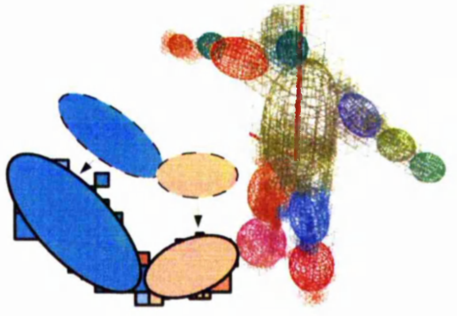
4.2.2 The EM Algorithm for Tracking

The two steps of the EM algorithm are summarised in Algorithm 4.2. During the Expectation step, each voxel $\mathcal{V}_i$ is assigned to the most probable blob B_j. This involves computing for all voxels the probability that they have been generated by each blob $P(\mathcal{V}_i|B_j)$, and finally selecting the blob that gave the highest probability (a slightly more efficient scheme is described in Section 4.2.3). The voxels assigned to a given blob (support map) are then used to re-evaluate its mean and covariance in the Maximisation step. While the Expectation step is relatively straightforward, a correct Maximisation is critical for the convergence of the algorithm. After each iteration of EM, some constraints are applied to the parameters of the blobs (more details in Section 4.3).

Algorithm 4.2: Overview of the use of Expectation-Maximisation for tracking.

```
▷ Initialise with the blobs tracked from the previous frame, and current voxels;
repeat
    ▷ Expectation (Section 4.2.3);
    forall voxels $\mathcal{V}_i$ do
        forall blobs $B_j$ do
            Compute the probability $P(\mathcal{V}_i|B_j)$ that $\mathcal{V}_i$ was generated by $B_j$;
        end
        Support Map: assign $\mathcal{V}_i$ to the blob $B_j$ that maximises $P(\mathcal{V}_i|B_j)$;
    end
    ▷ Maximisation (Section 4.2.4);
    forall blobs $B_j$ do
        Re-evaluate the parameters of $B_j$ from the set of voxels previously assigned with $B_j$;
    end
    Apply constraints on blobs using the kinematic model;
until convergence;
```





Each iteration of the EM algorithm brings the blobs closer to the voxel data. If the blobs are initially sufficiently close to the data, then each voxel should be assigned to the most appropriate blob during the first Expectation step, and a single iteration is therefore sufficient. Multiple iterations are only necessary when some voxels are assigned to wrong blobs. These wrong assignments can be detected when the re-evaluated blob parameters are in strong disagreement with the expected values. For example, if after the Maximisation step, some of the blobs change too radically in size or in colour, subsequent iterations of EM could converge towards a better solution.

4.2.3 Expectation Step

As we saw in Algorithm 4.2, the main difficulty of the Expectation step is the evaluation of the probability for a given voxel $\mathcal{V}_i$ to have been generated by the Gaussian blob B_j. Voxels are assimilated to 6-dimensional Gaussian vectors. Each blob B_j is described by a mean vector $\boldsymbol{\mu}_j$ and a covariance matrix Σ_j. The probability that $\mathcal{V}_i$ was generated by B_j is the probability $P(\mathcal{V}_i|B_j)$ of observing $\mathcal{V}_i$ knowing B_j for a multivariate Gaussian distribution:

$$P(\mathcal{V}_i|B_j) = \frac{1}{(2.\pi)^3.\sqrt{|\Sigma_j|}}.e^{-\frac{1}{2}D_M(\mathcal{V}_i,B_j)} \tag{4.6}$$

where $D_M(\mathcal{V}_i, B_j)$ is the Mahalanobis distance from Equation 4.4 and $|\Sigma_j|$ is the determinant of the covariance matrix of B_j. Considering that the mixed covariance between position and colour is neglected, $|\Sigma_j|$ can be reduced to the product of determinants on position and colour covariance matrices:

$$|\Sigma_j| = \begin{vmatrix} \Sigma_{Xj} & \Sigma_{XCj} \\ \Sigma_{XCj}^T & \Sigma_{Cj} \end{vmatrix} = |\Sigma_{Xj}| \times |\Sigma_{Cj} - \underbrace{\Sigma_{XCj}^T \cdot \Sigma_{Xj}^{-1} \cdot \Sigma_{XCj}}_{\simeq 0}| \simeq |\Sigma_{Xj}| \times |\Sigma_{Cj}| \tag{4.7}$$

A very standard optimisation is to compare the logarithm of the probabilities instead of the probabilities themselves. The logarithm function is indeed monotonic, so that a maximum of probability is also a maximum of log-probability. Taking the logarithm of both sides of Equation 4.6 gives:

$$\log P(\mathcal{V}_i|B_j) = -3.\log 2.\pi - \frac{1}{2}.\log|\Sigma_j| - \frac{1}{2}.D_M(\mathcal{V}_i, B_j) \tag{4.8}$$

which can be further simplified by discarding the constant terms and constant multiplicative factors yielding a function $\phi()$, whose minimisation remains equivalent to the maximisation of the original probability:

$$\phi(\mathcal{V}_i, B_j) = \log|\Sigma_j| + D_M(\mathcal{V}_i, B_j) \tag{4.9}$$

For a given voxel, minimising $\phi()$ with respect to the choice of blob B_j mainly depends on the minimisation of the distance $D_M()$ between the blob and a voxel. The term $\log|\Sigma_j|$ is constant for a given blob, and acts as a normalising factor, raising the chances of "smaller" blobs in the sense of the variances encoded in the covariance matrix. It is particularly useful when two blobs are very similar (in position and colour), because if only $D_M()$ was used, the blob with the largest variance would always have the advantage. This consideration is important for human body tracking, where some blobs are frequently close to each other, while representing different body parts: for example, a blob representing an arm which comes close to the one of the torso.

Keeping in mind that in theory the function $\phi()$ has to be evaluated for every voxel against all blobs (see Algorithm 4.2), any way to reduce the number of tests is highly beneficial for the system performance. The distance function $D_M()$ is composed of two asymmetrical parts in terms of computational cost: the distance on position is indeed much faster to compute than the distance on colour (Equation 4.5). A simple but very efficient optimisation is then to compute first the Mahalanobis distance between the position of a voxel and a blob, and if this distance is obviously too great for the voxel to belong to the blob, then the blob is disqualified and the next one is considered in turn. In order to determine an appropriate threshold, the analysis previously done for background segmentation (Chapter 2) is still valid. The Mahalanobis distance follows a Chi-Square distribution, and a threshold $T_3(\alpha)$ can be defined for the 3 degrees of freedom of the position. A high level of confidence $\alpha \simeq 0.999$ is appropriate to detect virtually all possible candidate voxels for a given blob. This scheme has the additional advantage of discarding outlier voxels that are too far from all blobs.

Likewise, another simple optimisation is to stop testing other blobs when the total Mahalanobis distance including colour for a given blob is small. Because of non-overlapping constraints, it is very unlikely that a voxel could be simultaneously close to the centre of two blobs. A threshold $T_6(\alpha')$ with a relatively tight confidence level is used for the 6 dimensions in colour and position. In practice, taking $\alpha' \simeq 0.8$ means that we retain the 80% of voxels that are closest to the blob among all possible candidates.

The full process of the Expectation step is detailed in Algorithm 4.3. After selecting the most appropriate blob, each voxel is passed to the Maximisation process which is running in parallel. This allows voxels to be "pipelined" without any need to store the whole set of voxels. Because the full set of voxels does not need to be stored, memory usage is minimal. The only drawback of this scheme is that multiple iterations of EM are not possible because they would require subsequent accesses to the voxels. However, as previously stated, a single iteration of EM is sufficient.

Algorithm 4.3: Details of the Expectation Step

```
forall blobs $B_j$ do pre-compute $\log|\Sigma_j|$;
forall voxels $\mathcal{V}_i$ do
    ChosenBlob $\leftarrow \emptyset$;
    CurrentMin $\leftarrow 0$;
    forall blobs $B_j$ do
        if $D_M(X_{\mathcal{V}i}, B_j) < T_3(\alpha \simeq 0.999)$ then
            if $\phi(\mathcal{V}_i, B_j) <$ CurrentMin or ChosenBlob $= \emptyset$ then
                ChosenBlob $\leftarrow B_j$;
                CurrentMin $\leftarrow \phi(\mathcal{V}_i, B_j)$;
                if CurrentMin $< T_6(\alpha' \simeq 0.8)$ then
                    break loop;
                end
            end
        end
    end
    if ChosenBlob $\neq \emptyset$ then
        Add the voxel $\mathcal{V}_i$ to the support map of ChosenBlob ;
    end
end
```

It can be noted that no constraint is placed on the blobs during EM itself. All corrections concerning the expected size of the blobs, their colour, or their movement take place in a later step using a kinematic model (Section 4.3). Placing constraints such as a maximal volume of voxels per blob, would only deteriorate the convergence of EM while being redundant with the later corrections.

4.2.4 Maximisation Step

Following the Expectation step described in Section 4.2.3, the parameters of the blobs need to be re-evaluated from the assigned set of voxels. The difficulty is that voxels have different, non-negligible sizes. The computation of the mean and covariance has to take the shape and volume of voxels into account. Let $\{\mathcal{V}_1 \ldots \mathcal{V}_N\}$ be the set of voxels (support map) from which the new parameters $(\boldsymbol{\mu}, \Sigma)$ of the blob B are to be evaluated. Additionally, let $s_{\mathcal{V}i}$ be the size (side of the cube) of the voxel $\mathcal{V}_i$. The new mean vector $\boldsymbol{\mu}$ of the blob B is the weighted average of the voxel vectors:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_X \\ \mu_C \end{pmatrix} = E[\{\mathcal{V}_1 \ldots \mathcal{V}_N\}] = \frac{1}{\sum_{i=1}^{N} s_{\mathcal{V}i}{}^3} . \sum_{i=1}^{N} s_{\mathcal{V}i}{}^3 . \begin{pmatrix} X_{\mathcal{V}i} \\ \mathcal{C}_{\mathcal{V}i}^m \end{pmatrix} \tag{4.10}$$

In this formulation, colour and position are handled simultaneously. Note that at this point, voxels contain only a single colour $\mathcal{C}_{\mathcal{V}}^m$, which was selected during the Expectation step (Equation 4.5). The computation of the mean is also appropriate for iterative evaluation where the weighted sum is updated as new pixels are included, and finally normalised by the total volume when all voxels have been processed.

In contrast, the computation of the covariance matrix is more delicate. For clarity, let us decompose the problem into the computation of the sub-matrices Σ_X, Σ_C and Σ_{XC}, standing respectively for position, colour and mixed position-colour covariance matrices. Starting with the computation of Σ_C, an important assumption is that the colour inside each voxel is uniform. This is obviously a simplification of the reality since the internal colour variance of each voxel exists and could have been computed during the 3-D reconstruction from the image samples. Nevertheless, a uniform colour vector $\mathcal{C}_{\mathcal{V}i}$ for each voxel $\mathcal{V}_i$ is sufficient for our purpose because colour tends to be uniform on large space regions. The standard formula of the variance can then be applied, weighting the contribution of each voxel with its volume, as follows:

$$\begin{aligned} \Sigma_C &= \frac{1}{\sum_{i=1}^{N} s_{\mathcal{V}i}{}^3} . \sum_{i=1}^{N} s_{\mathcal{V}i}{}^3 . (\mathcal{C}_{\mathcal{V}i} - \mu_C) \cdot (\mathcal{C}_{\mathcal{V}i} - \mu_C)^T \\ &= \frac{1}{\sum_{i=1}^{N} s_{\mathcal{V}i}{}^3} . (\sum_{i=1}^{N} s_{\mathcal{V}i}{}^3 . \mathcal{C}_{\mathcal{V}i} \cdot \mathcal{C}_{\mathcal{V}i}^T) - \mu_C \cdot \mu_C{}^T \end{aligned} \tag{4.11}$$

The second formulation is adapted to iterative computation because the mean vector μ_C can be used at the end of the integration of all voxels, when it is finally known.

The covariance matrix on positions, Σ_X, is slightly more delicate to evaluate. Indeed, a given voxel $\mathcal{V}_i$ cannot be reduced to the single position of its centre $X_{\mathcal{V}i}$. Since voxels have non-negligible volumes, the sizes of the voxels themselves should contribute to the total variance of the blob. The step-by-step computation of the covariance matrix Σ_X is detailed in Appendix C, reaching the following formula:

$$\Sigma_X = \frac{1}{\sum_{i=1}^{N} s_{\mathcal{V}i}{}^3} . \left(\sum_{i=1}^{N} \underbrace{\tfrac{1}{12} . s_{\mathcal{V}i}{}^5 \cdot I_3}_{Internal\ cov.} + \underbrace{s_{\mathcal{V}i}{}^3 . X_{\mathcal{V}i} \cdot X_{\mathcal{V}i}{}^T}_{External\ cov.} \right) - \mu_X \cdot \mu_X{}^T \tag{4.12}$$

where I_3 is the identity matrix in 3 dimensions. This formulation is essentially the same as for the colour covariance of Equation 4.11, except for the addition of an internal covariance term reflecting the fact that a voxel cannot simply be assimilated to a weighted point in space.

The last covariance matrix to compute, Σ_{XC}, is the mixed covariance between position and colour. It reflects the correlations between the variations of colour and the axis of the blob. These correlations (or dependencies) are normally small because the blobs are designed to track parts of the body with relatively uniform colour. Actually, the minimisation of the correlations between colour and position is the main criterion of efficiency for a blob tracker. The computation of Σ_{XC} has the purpose of evaluating the strength of these correlations, and consequently adapt the distribution of the blobs in order to minimise them (this is why Σ_{XC} was neglected during the Expectation step). The actual computation of Σ_{XC} is very similar to the one of Σ_C (Equation 4.11). Since a uniform colour is assumed for the voxels, the internal variance of voxels plays no role in this equation.

$$\Sigma_{XC} = \frac{1}{\sum_{i=1}^{N} s_{\mathcal{V}_i}{}^3} . (\sum_{i=1}^{N} s_{\mathcal{V}_i}{}^3 . X_{\mathcal{V}_i} \cdot C_{\mathcal{V}_i}{}^T) - \mu_X \cdot \mu_C{}^T \tag{4.13}$$

The covariance matrices Σ_X and Σ_{XC} are computed in the global coordinate system of the voxels, and not in the local coordinate system of each blob. Efficiency is the main justification of this approach: it is indeed much faster to transform the few blobs composing the appearance model after the completion of the maximisation than to transform all of the voxels beforehand. Interpretation of the parameters of the blobs, however, necessitates this transformation into the local coordinate systems. In the local coordinate system of each body part, the blobs have in most cases constant shape and colour properties, whereas in the global coordinate system, these attributes depend on the positions and orientations of the blobs. The global to local transformation necessitates the position and orientation of the body parts, which will be made available by an underlying kinematic model (Section 5.1). The actual transformation of the blobs' parameters and the use of these parameters to maintain a coherent appearance model over time are detailed in Section 4.3.

4.3 Constraining EM with Learnt Models of Blobs

Expectation-Maximisation is a powerful optimisation method which finds the maximum likelihood estimate of the blobs model given the data. However, EM is not sufficiently constrained to avoid the gradual degeneration of the blobs when presented to imperfect data. When left unconstrained, blobs tend to drift from the features they are supposed to track, and end up loosing track completely. This problem is particularly visible when using the reconstructed voxels as a basis for tracking, because body parts are prone to change shape depending on the viewpoints of the cameras.

The obvious solution is to constrain some parameters of the blobs between each iteration of EM, thus avoiding their degeneration. The constraints are learnt directly from the data during an "acquisition" stage. Likewise, the initial repartition of the blobs can prove inefficient or simply sub-optimal: a scheme is presented to dynamically reorganise the blobs during the acquisition stage. The appearance model is then fully acquired from the data, and subsequently used to constrain EM.

4.3.1 Run-Time Correction of Blobs Parameters

All the parameters of a blob B are encoded into its mean $\boldsymbol{\mu}$ and covariance matrix Σ. While the mean vector $\boldsymbol{\mu}$ represents the position and colour of the blob, the covariance matrix Σ encodes both its rotation and its variations in shape and colour. All these parameters need to be constrained in different ways because their physical nature is different. For example, the shape of a blob is expected to remain constant over time, while its orientation cannot be constrained a priori. In order to formulate the constraints, we need at this point to assume that an underlying skeletal model is available, driven by the blobs which are "attached" onto its bones (Section 5.2). The corrected positions and orientations of the blobs, satisfying the kinematic constraints, can then be obtained at each frame from this skeletal model.

Assuming that the blob B is attached at an offset $\widehat{\alpha}$ along a bone of the skeletal model, let us denote as P the global position of this bone obtained after application of the kinematic constraints, and R the associated rotation matrix (see Figure 4.2). The position of the point of attachment of the blob is described in the local coordinate system of the bone, allowing the offset $\widehat{\alpha}$ to remain constant across time. The corrected mean position vector μ'_X is then computed from the model as a simple conversion from

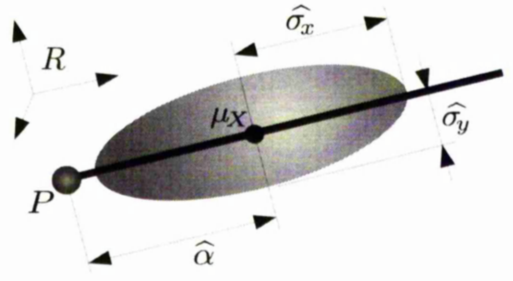


Figure 4.2: *Notations for the attributes of a blob B, attached onto a bone of the skeletal model.*

local to global coordinates:

$$\mu'_X = P + R \cdot \begin{pmatrix} \widehat{\alpha} \\ 0 \\ 0 \end{pmatrix} \tag{4.14}$$

The shape of the blob, encoded in its covariance matrix Σ_X, is summarised by its expected standard deviations $\{\widehat{\sigma_x}, \widehat{\sigma_y}, \widehat{\sigma_z}\}$ along the 3 axes of the local coordinate system of the bone segment (Figure 4.2). If we assume that the axes of the bone segment are approximately aligned with the main directions of the blob, the conversion from global to local coordinates is equivalent to an eigenvalue decomposition. The corrected covariance matrix Σ'_X of the blob B is again re-generated from the model:

$$\Sigma'_X = R \cdot \begin{pmatrix} \widehat{\sigma_x}^2 & 0 & 0 \\ 0 & \widehat{\sigma_y}^2 & 0 \\ 0 & 0 & \widehat{\sigma_z}^2 \end{pmatrix} \cdot R^T \tag{4.15}$$

The colour usually remains constant during the tracking period. Therefore, as soon as a model is acquired for the mean colour vector μ_C and the colour covariance matrix Σ_C, these can simply be re-generated at each frame from the model. If we denote as $\widehat{\mu_C}$ the acquired model for the mean colour vector, and $\widehat{\Sigma_C}$ the corresponding model for the colour covariance matrix, the correction at each frame is:

$$\mu'_C = \widehat{\mu_C} \qquad \Sigma'_C = \widehat{\Sigma_C} \tag{4.16}$$

Finally, the mixed position-colour covariance matrix Σ_{XC} is re-generated as equal to zero because the blobs are re-distributed (Section 4.3.3) during the acquisition stage so as to minimise dependencies between position and colour, and thus keep Σ_{XC} as

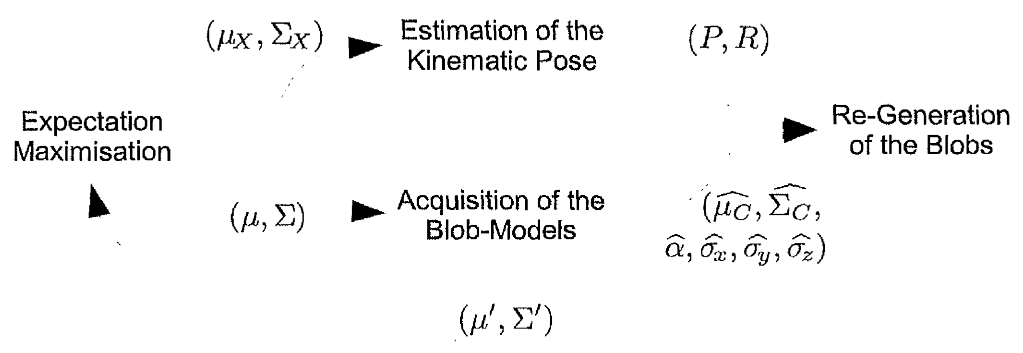

(μ_X, Σ_X)
Estimation of the Kinematic Pose
(P, R)
Expectation Maximisation
Re-Generation of the Blobs
(μ, Σ)
Acquisition of the Blob-Models
$(\widehat{\mu_C}, \widehat{\Sigma_C}, \widehat{\alpha}, \widehat{\sigma_x}, \widehat{\sigma_y}, \widehat{\sigma_z})$
(μ', Σ')

Figure 4.3: *Data-Flow diagram of the parameters of a blob between each frame. The dashed arrow denotes the fact that the acquisition process is only active during a preliminary stage of tracking.*

close to zero as possible:

$$\Sigma'_{XC} = 0 \tag{4.17}$$

Once the parameters of the model have been learnt, the run-time correction of the blobs is integrated into the tracking cycle according to Figure 4.3. The main steps of this data-flow are:

1. *Expectation-Maximisation:* The re-generated blobs are used as an initial estimation for EM, where they are adjusted to fit the voxel data.
2. *Kinematic Pose Estimation:* The fitted blobs are used to drive a skeletal model towards a kinematically correct tracked position (see Chapter 5).
3. *Blobs Re-Generation:* The blobs are re-generated onto the bones of the skeletal model and converted into global coordinates. The re-generated blobs are the new initial estimations for EM in the next frame.
4. *Acquisition:* During an initial acquisition stage, the models for the various attributes of the blobs are learnt automatically. This acquisition stage is the topic of Section 4.3.2.

4.3.2 Automatic Acquisition of Blobs Models

In the previous section, the models for the attributes of the blobs were simply assumed to be readily available. While using manual settings is a possibility for a few parameters like the expected shape standard deviations $\{\widehat{\sigma_x}, \widehat{\sigma_y}, \widehat{\sigma_z}\}$, it becomes impossible for others such as the expected colour covariance matrix $\widehat{\Sigma_C}$. An automatic scheme for learning all the models of parameters during an acquisition stage is presented in this section.

Initial Positioning and Colour Estimation

During the acquisition stage, the subject is asked to adopt a pose generating as few ambiguities and occlusions as possible, and to remain in this pose, relatively immobile, during a few seconds. In practice, 2 to 3 seconds representing 50 to 100 frames are sufficient. The first step of the acquisition consists in positioning the blobs onto the parts of the body, and learning a first approximation of their colour. This process is done by disabling the use of colour during Expectation (a prior colour model is rarely available), and using initial default values for the shape of the blobs.

When no initial model of colour is available for a blob B, the acquisition of the colour model is based on all visible colours of the voxels belonging to the support map of B. A simple average of the colours seen from the N_c available cameras gives a sufficient approximation of the real colour of the body part, as long as the pose adopted by the subject during initialisation does not generate too many self-occlusions. The original formulae of the Maximisation step (Equations 4.10 and 4.11) are therefore modified to include all available colours:

$$\begin{aligned} \mu_C &= \frac{1}{N_c . \sum_{i=1}^{N} s_{\mathcal{V}i}{}^3} \sum_{i=1}^{N} s_{\mathcal{V}i}{}^3 . \sum_{j=1}^{N_c} C_{\mathcal{V}i}^j \\ \Sigma_C &= \frac{1}{N_c . \sum_{i=1}^{N} s_{\mathcal{V}i}{}^3} \sum_{i=1}^{N} s_{\mathcal{V}i}{}^3 . \sum_{j=1}^{N_c} C_{\mathcal{V}i}^j \cdot {C_{\mathcal{V}i}^j}^T - \mu_C \cdot {\mu_C}^T \end{aligned} \tag{4.18}$$

The colour model is build incrementally as a running average of the values returned by this Maximisation step. For a given timestep $t > 0$, the colour models are acquired as follows:

$$\begin{aligned} \widehat{\mu_C}_t &= \frac{1}{t} . \mu_{Ct} + (1 - \frac{1}{t}) . \widehat{\mu_C}_{t-1} \\ \widehat{\Sigma_C}_t &= \frac{1}{t} . \Sigma_{Ct} + (1 - \frac{1}{t}) . \widehat{\Sigma_C}_{t-1} \end{aligned} \tag{4.19}$$

During the preliminary phase of the acquisition process, only the colour parameters for which no prior value is available are learnt with this special treatment. The other models for the blobs attributes are learnt incrementally using at each frame the outputs of the standard Maximisation formulae.

Occasionally, some blobs can be initialised with a prior knowledge about their colour. For example, if we create some blobs supposed to track the hands, giving them an initial model of skin colour helps them focusing on the hands, instead of trying to learn the colour of another body part. When such a prior model is present, it is included in the acquired models from Equation 4.19 by starting the running average at a timestep $t' > 1$ depending on the strength of the prior.

Acquisition of the Full Models with EM

After only a few frames, when the blobs are positioned onto the body parts and an initial colour model has been acquired, the complete acquisition process can start, based on the actual EM algorithm from Section 4.2. At each frame t, a new mean μ_t and covariance matrix Σ_t of a blob B are obtained from the voxels through the standard Maximisation formulae (Section 4.2.4). These new values are then incrementally integrated into the blob-models with the running average formulation from Equation 4.19.

The shape models $\{\widehat{\sigma_x}, \widehat{\sigma_y}, \widehat{\sigma_z}\}$ are updated by converting the spatial covariance matrix $\Sigma_{X\,t}$ into the local coordinate system of the body part: $R^T \cdot \Sigma_{X\,t} \cdot R$. The diagonal elements of the transformed covariance matrix are the variances (squared standard deviations) along the local axes of the bone:

$$\begin{pmatrix} \widehat{\sigma_x} \\ \widehat{\sigma_y} \\ \widehat{\sigma_z} \end{pmatrix}_t = \frac{1}{t} \cdot \sqrt{diag(R_t^T \cdot \Sigma_{X\,t} \cdot R_t)} + (1 - \frac{1}{t}) . \begin{pmatrix} \widehat{\sigma_x} \\ \widehat{\sigma_y} \\ \widehat{\sigma_z} \end{pmatrix}_{t-1} \tag{4.20}$$

The last remaining model parameter to acquire is the offset of the point where the blob is attached onto the bone, $\widehat{\alpha}$. The current offset is computed by projecting the mean of the blob μ_X onto the first axis R_1 of the rotation matrix of the bone, as illustrated in Figure 4.4. Like other parameters, $\widehat{\alpha}$ is then incrementally refined with a running average:

$$\widehat{\alpha}_t = \frac{1}{t} . (\mu_{X\,t} - P_t) \cdot R_{1\,t} + (1 - \frac{1}{t}) . \widehat{\alpha}_{t-1} \tag{4.21}$$

Even if the acquisition process is fully automatic, a series of constraints can be imposed on the blobs attributes. In order to avoid obvious errors, the shape models $\{\widehat{\sigma_x}, \widehat{\sigma_y}, \widehat{\sigma_z}\}$ are bounded by minimal and maximal values. For example, a common maximal value for $\widehat{\sigma_x}$ is half the length of the bone onto which the blob is attached. Likewise, the offset of the point where the blob is attached, $\widehat{\alpha}$, is clamped between zero and the length of the bone segment. More advanced constraints are used when

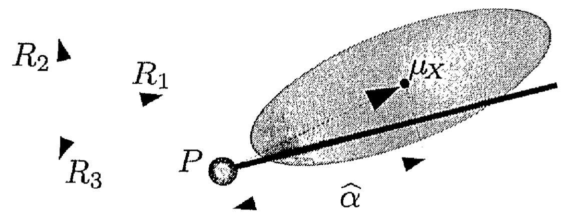


Figure 4.4: *The offset $\widehat{\alpha}$ of the attach point of a blob along a bone is acquired by scalar projection of the mean position μ_X of the blob onto the bone. The direction of the bone is the first column of the rotation matrix R.*

more than one blob is attached on the same bone segment. For example, we limit overlapping by keeping the distance between the points of attachment greater that the sum of their standard deviations.

4.3.3 Dynamic Splits

The acquisition process described so far is able to optimise all the attributes of the blobs, making them fit the body part they are supposed to track. However, the choice of the initial number of blobs is still an important open question. A blob is a single Gaussian distribution, and thus a strongly uni-modal tracker, unadapted to the tracking of multi-modal data. When confronted to the problem of tracking the human body, exhibiting non-homogeneous colour and spatial data, the obvious solution is to use as many blobs as there are *self-coherent* body parts to track. This is actually the generic problem of the choice of the number of Gaussian components for EM, mentioned earlier in Section 4.2.

A constraint to take into consideration, when choosing the number of blobs, is that each of them has to be attached along a bone of the underlying skeletal model. This means that the blobs cannot be placed freely in space, but are rather stacked one after the other along the bones of the kinematic model. This constraint is a consequence of both the axial symmetry of most limbs, and the fact that colour variations on small scales are inconclusive. The main consequence of this constraint is that the meaningful modes (statistical maxima) of the data should be separable along a bone segment to be modelled by distinct blobs.

The strategy we adopt is to start with the minimal number of blobs, and to iteratively split those which are trying to acquire multi-modal data. We are mainly interested in the modes of the colour information. Indeed, the spatial data for a single body part is normally relatively compact and self-coherent, regardless of the noise and reconstruction errors. So, if we were not using colour information, the body part around each bone could be represented by a single Gaussian blob. With colour, however, we have to take into account the radical changes that can happen along the same bone. For example, the forearms often exhibit an important change in colour because of the sleeves.

The main issue is therefore to detect a significant change of colour along the main axis of a blob. The mixed covariance matrix between position and colour Σ_{XC} was precisely built during Maximisation (Section 4.2.4, Equation 4.13) for this purpose. Although not detailed in the previous section, a running average of the matrices Σ_{XC} is maintained during the acquisition stage. This matrix, denoted as $\widehat{\Sigma_{XC}}$, encodes the variations of colour with respect to the spatial variations along the axes of the global coordinate system. The matrix Λ, which encodes the colour variations along the axes of the local coordinate system of the bone, is obtained by a change of coordinate system, followed by a normalisation of each axis by its standard deviation:

$$\Lambda = \widehat{\Sigma_{XC}} \cdot R \cdot \begin{pmatrix} \frac{1}{\widehat{\sigma_x}} & 0 & 0 \\ 0 & \frac{1}{\widehat{\sigma_y}} & 0 \\ 0 & 0 & \frac{1}{\widehat{\sigma_z}} \end{pmatrix} \tag{4.22}$$

The columns $\{\Lambda_1, \Lambda_2, \Lambda_3\}$ of the matrix Λ are the colour standard deviations along each axis of the local coordinate system of the blob. For example, Λ_1 represents the variation of colour which is correlated with a shift of one standard deviation along the axis R_1. In order to measure the relative strength of the colour variations along the first axis, we define λ as the ratio between the norm of the first column and the norm of the second greatest one:

$$\lambda = \frac{|\Lambda_1|}{\max(|\Lambda_2|, |\Lambda_3|)} \tag{4.23}$$

For example, $\lambda = 2$ would mean that the colour variations along the bone segment are twice as strong as in any other direction. Unfortunately, this does not constitute a real proof that the colour distribution along the axis of the bone is multi-modal. Nevertheless, even if distinct modes are not assured, we know that the mean colour significantly differs at the two poles of the blob. This alone justifies the use of two blobs instead of one, in the hope of making each tracker more specialised and efficient.

The notations for the splitting procedure of a blob B with a large ratio ($\lambda \geq 2$) are illustrated in Figure 4.5. When splitting the blob, the only important requirement is to

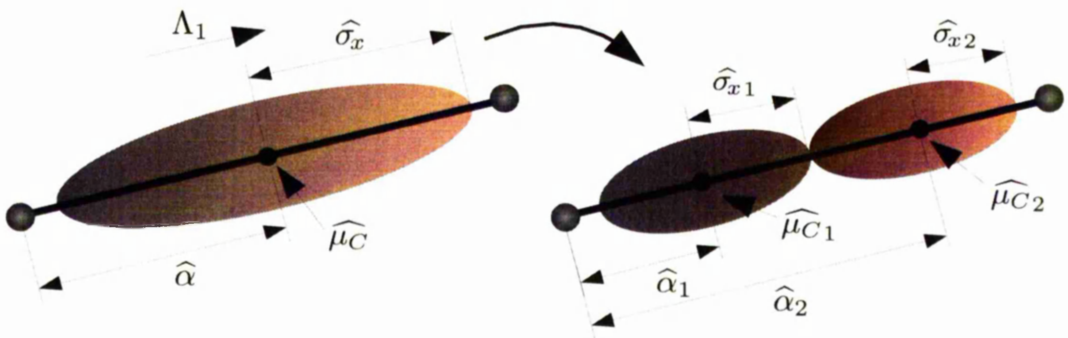


Figure 4.5: *Splitting of a blob having a high colour variance along the bone axis.*

make the new blobs as distinct from each other as possible. All the attributes of the new blobs are subsequently re-optimised during the rest of the acquisition stage. The new colour means, $\widehat{\mu_C}_1$ and $\widehat{\mu_C}_2$ are first defined by shifting the original mean colour by one standard deviation along both directions of the bone axis:

$$\widehat{\mu_C}_1 = \widehat{\mu_C} - \Lambda_1 \qquad \widehat{\mu_C}_2 = \widehat{\mu_C} + \Lambda_1 \tag{4.24}$$

Similarly, the new points of attach $\widehat{\alpha}_1$ and $\widehat{\alpha}_2$ are shifted apart by a standard deviation $\widehat{\sigma_x}$ along the axis of the bone, while the new standard deviations $\widehat{\sigma_x}_1$ and $\widehat{\sigma_x}_2$ are simply half the original one to reflect the split:

$$\widehat{\alpha}_1 = \widehat{\alpha} - \frac{\widehat{\sigma_x}}{2} \qquad \widehat{\alpha}_1 = \widehat{\alpha} + \frac{\widehat{\sigma_x}}{2} \tag{4.25}$$

$$\widehat{\sigma_x}_1 = \frac{\widehat{\sigma_x}}{2} \qquad \widehat{\sigma_x}_2 = \frac{\widehat{\sigma_x}}{2} \tag{4.26}$$

The other parameters are left untouched from the original blob. An illustration of the acquisition and splitting process is presented in Figure 4.6, over 20 frames. As described in this section, the blobs acquire their colour model in a first step, and the blobs exhibiting a large colour variance along their main axis are split in a second step. The appearance model obtained at the end of these two steps is much more accurate that any prior generic model.

4.4 Discussion and Conclusion

This chapter introduced a framework for using blobs as feature trackers. The blob-models were first formally described, and a fast fitting procedure based on EM with

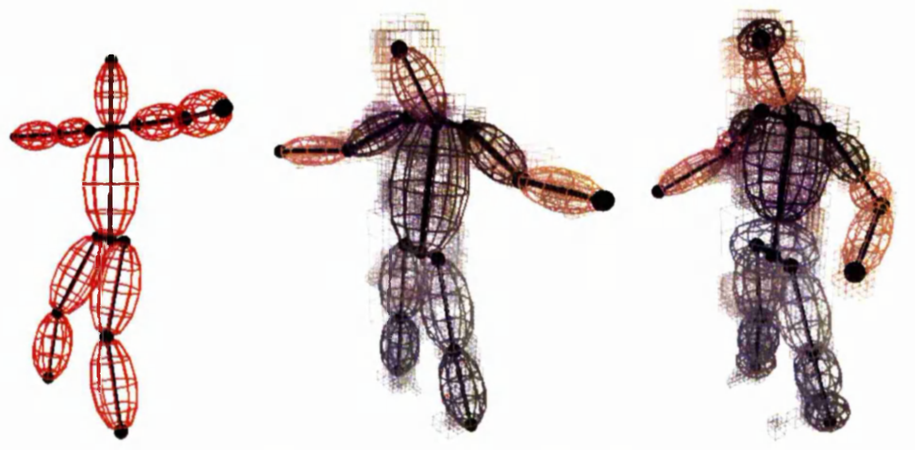


Figure 4.6: *Automatic reconfiguration of the blobs during* 20 *frames of the acquisition stage. (Left) the initial repartition of blobs before the start of the acquisition. (Middle) The blobs have acquired their shape and colour from the voxels, but are none has yet split. (Right) The blobs with the greatest colour variances have automatically split to reflect the morphology and clothing of the subject.*

binary support maps was presented. Automatic acquisition and dynamic optimisation of the blobs-based appearance model was the subject of the last section. Novel contributions reside in the formulation of EM, based on the hierarchical coloured voxel reconstruction, and in the automatic reconfiguration of the blob-models using a colour consistency criterion.

A possible extension of the current framework could be to allow more general distributions than simple Gaussians. While the ellipsoidal shape of the Gaussian blobs is well adapted to most body parts, and more complex features can always be modelled by multiple blobs, other "customised" distributions would be interesting to evaluate. A multimodal colour model, for example, could allow a single blob to represent a wider range of body parts.

Chapter 5

Hierarchical Tracking with Inverse Kinematics

This chapter presents the direct recovery of the full kinematic pose in a bottom-up way: the blobs from Chapter 4 are used to guide a kinematic model towards the tracked pose in a two-step Inverse-Kinematics process. In the first step, the positions for the joints of the kinematic model are computed from the blobs. The second step is then a standard case of inverse kinematics. The kinematic model itself constrains the possible movements, and allows the interpretation of the tracking data.

5.1 Kinematic Model

The blobs introduced in Section 4.1 are *models* describing the properties (shape and colour) of the body parts. This type of model is commonly called an *appearance model*, and is only a partial description of the properties of an object. The appearance of the object is by no means fully and accurately described just by the blobs, but since this model is sufficient for our particular application, we use the term *appearance model*.

This section focuses on another type of model describing the mechanical relationships between the moving parts of the object of interest. The moving parts of the human body are, of course, the limbs and bones which are articulated by joints. This type of model is called a *kinematic model*. Kinematics is defined as "the branch of mechanics concerned with motion without reference to force or mass", which simply means that a kinematic model describes the spatial relationships between body parts without any actual physical model of motion. A model describing the physical properties of each part is often called a *dynamic model*. While a kinematic model can only describe the relative position of the body parts, a dynamic model can correct and predict body poses through physical simulation.

After identifying the requirements of a kinematic model in the context of real-time human body tracking, the next section presents a range of kinematic models which have been used in the literature. A description of the chosen model parametrisation follows. The last two sections focus on the links between the appearance model (the blobs) and the kinematic model, and describe an efficient way to estimate the model parameters through inverse kinematics.

5.1.1 Requirements of a Kinematic Model

A kinematic model should obviously describe the relationship between each related moving body part. Less obvious is the level of detail that is most appropriate for real-time tracking. Regarding the issue of accuracy, the first parameter to consider is the number of body parts to represent in the kinematic "skeleton". While the major limbs (legs, arms) are necessary, other articulated parts like the fingers are not necessarily compatible with the achievable level of detail. The voxel-based 3-D reconstruction has indeed a strong limitation in accuracy (typically of the order of one centimetre), making the tracking of small body parts impossible. Another argument against the inclusion of hands and fingers is the inherent complexity of hand tracking and modelling, which in itself, is the subject of much research [SGH05].

A second issue concerns the modelling of the joints themselves: while all relationships between adjacent body parts are rotations, the constraints associated with these rotations can be relatively complex. For example, the physiologically possible rotation of the elbow depends on the rotation of the shoulder, which itself depends on the global pose of the body. In [HUF04] and [HUF05], Herda *et al.* describe a method for capturing and using implicit joint constraints for tracking. Valid joints configurations are first captured using a hardware tracker. A hierarchy is then established between the joints, allowing the valid sub-space of a joint to be recovered from its parents. The constraints themselves are modelled by implicit surfaces, delimiting the cloud of captured valid positions. [HUF04] and [HUF05] also show how to interpolate between key positions to compensate for the missing data, and incorporate the joint constraints in a standard least squares error minimisation framework for full human-body tracking. Another way to model constraints, based on Support Vector Machines (SVM) classifiers, is proposed by Demirdjian *et al.* in [DKD03]. An SVM classifier is trained to differentiate valid and invalid poses, and subsequently to constrain the optimisation process. Both these types of constraints model appear promising and can make the tracking process more robust. In this section, we shall devise a simple scheme handling basic constraints. A more advanced scheme, capable of constraining the full body pose to learnt configurations, will be described in Chapter 6.

The number of parameters is a determinant criterion in the choice of a good model. The parameters of the kinematic model are divided into two main groups of variables. The morphological parameters such as the sizes of the various body parts belong to the first group that are evaluated prior to the tracking or during an initialisation step, but which remain constant during the tracking process. By contrast, the rotation angles of the joints and the global position of the skeleton belong to the second category of parameters which are evaluated "online" during the tracking process, and for which no prior value is available. Minimising the number of parameters in both groups is important to keep the model compact, efficient and robust. The compactness and performance follow naturally from a reduced number of parameters. The increased robustness comes from the fact that, as the number of parameters is reduced, the space spanned by those parameters is smaller, therefore imposing more constraints on possible movements. Naturally, the space of online parameters has to remain big enough to include all possible body poses.

The remaining details to consider when choosing a model are the ease and cost of updates. The main operations expected from a kinematic model are *forward kinematics* and *inverse kinematics*. Forward kinematics is the standard way of updating the model from the online parameters (recovering the global positions of the body parts from the joint angles). Inverse kinematics works the other way around by recovering the online parameters of the model from the position (or desired position) of the body parts. While forward kinematics is usually considered an easy problem, and is handled efficiently by all types of kinematic models, inverse kinematics is non-trivial and is often formulated as an optimisation problem. The following section highlights the advantages and issues of some kinematic models described in the literature.

5.1.2 Kinematic Human Body Models in the Literature

The most common type of kinematic model is a hierarchy of *bones* (analogy with the term "skeleton") and *joints*. The kinematic model is then a tree with a root usually placed at the pelvis, as illustrated in Figure 5.1. The global positioning of the model is defined by 6 parameters (3 for rotation and 3 for translation) which refer to the placement of the root of the kinematic tree. The position of each node of the tree is then defined relatively to its parent, so that computing the global position and orientation of a leaf of the tree (one hand, for example) requires applying recursively all transformations along the kinematic chain linking the root to this leaf.

In the literature, complexity varies between 20 and 32 degrees of freedom for the full body. The most standard type of model includes only the main limbs (torso, legs, arms and head), already amounting to between 16 and 20 dimensions. When adding the 6 dimensions of the root of the tree, it is easy to realise that even these basic models represent a challenge for tracking. Kinematic trees including the main limbs are widely used in the human body tracking literature [CBK03a, MH03, GD96, BD02, YSK$^+$98, CTMS03]. The coarseness of the models is often imposed by the quality of input images: wrists and ankles are indeed impossible to discern from many videos. Some authors, however, have tried more detailed models including ankle rotations [BL01b] or wrists [DDR01], but no evaluation is provided regarding their tracking accuracy.

Parametrisation of the Joints

Regarding the formulation of the rotations, Euler angles are still widely used despite a number of pitfalls. The main problem with Euler angles is that they generate singularities for specific rotation values: individual rotations around the basis axes are applied consecutively, so that a rotation in one axis could override a rotation in another, effectively loosing a degree of freedom. This phenomenon is called “Gimbal Lock”. It can be avoided by imposing constraints on the joint angles, hence bounding the rotation angles to a “safe” zone, but the parametrisation is then tedious. For further details on Euler angles, and the way to alleviate their weaknesses for forward and inverse kinematics, the reader is referred to [Wel93].

Unit quaternions [Hor87] are an elegant way to tackle the Gimbal Lock problem by encoding any arbitrary 3-D rotation as a hyper-sphere in 4-D space. The 3-D rotation encoded by a quaternion is equivalent to a single rotation around an axis which changes with the quaternion. The absence of fixed rotation axes poses the problem of data interpretation and constraints enforcement. Herda *et al.* [HUF04, HUF05] use quaternions to represent rotations with 3 degrees of freedom, and learn an implicit valid sub-space from motion capture data. The quaternions are then constrained into this subspace during the optimisation process.

Twists and products of exponential Maps [MSZ94] are another alternative to Euler angles. Like quaternions, they can encode 3-D rotations without singularities. While their formulations are slightly more complex than the one of the quaternions, their derivatives are easier to obtain. This is particularly useful for problems like inverse kinematics, which rely on the Jacobian matrix for gradient-based optimisation. Twists and exponential maps were first used in the context of human-body tracking by Bregler *et al.* [BM98, BMP04], where they permit pose estimation as a linear optimisation problem. Mikic *et al.* [MTHC03] also encode the rotations of their kinematic model with twists and exponential maps, and perform inverse kinematics with the help of a Kalman Filter. Finally, Demirdjian *et al.* [DKD03] learn morphological constraints with Support Vector Machines, and incorporate those in an optimisation framework.

Other Types of Kinematic Models

In an attempt to simplify the inverse kinematics problem, Theobalt *et al.* [TMSS02] propose a 2-layer kinematic model. A very coarse and unconstrained layer is first fitted onto the tracked body parts. The second layer, containing the correct kinematic constraints, is then adjusted onto the data under constraints from the first layer. More specifically, in the first layer, an arm is only represented by the vector linking the shoulder to the hand. The possible positions for the elbow are therefore constrained to a circle in the second layer, and the best solution is found iteratively. The main problem with this approach is that it assumes that specific body parts (hands and feet) can be tracked reliably, which is rarely the case.

5.1.3 Model Description and Parametrisation

We used a classic kinematic tree rooted at the pelvis. The global positioning of the model is described by three parameters representing the position of the pelvis and three rotation parameters describing its global orientation. Euler angles were chosen mainly because of their simplicity, and because other formulations like quaternions or exponential maps would have required a large amount of motion capture data to learn useful constraints. A more advanced kinematic model is the topic of future work, but we shall show that even these simple kinematics are sufficient to demonstrate the efficiency of our tracking framework.

In order to avoid singularities, joints with 3 degrees of freedom had to be avoided.

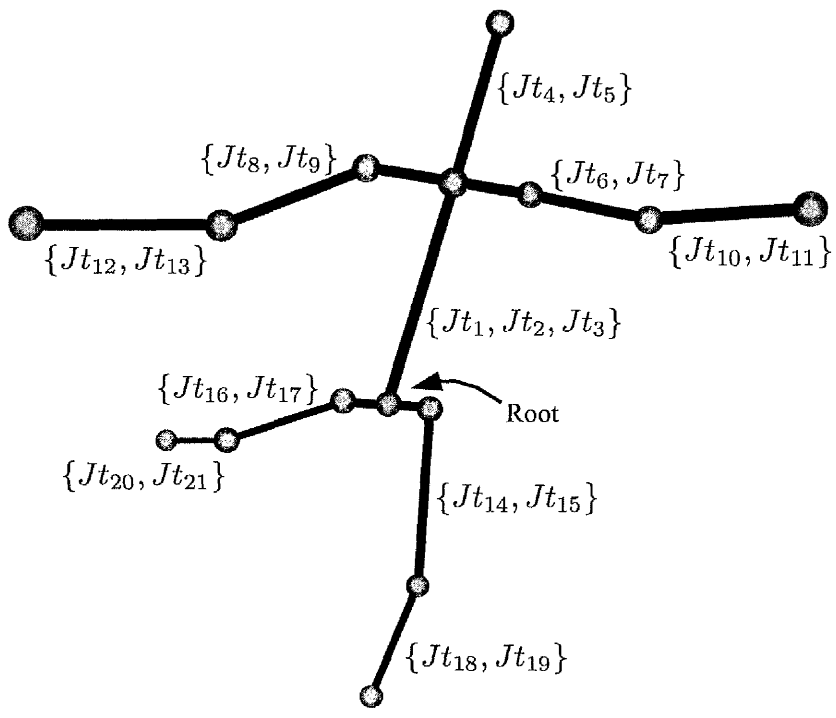


Figure 5.1: *Repartition of the joints in the kinematic tree. To avoid singularities, most of the body parts have only two degrees of freedom. The complete parametrisation of each joint is given in Table 5.1.*

The parametrisation of the arms and legs therefore redistributes one degree of freedom from the most complex joints (shoulders and hips) to the simpler ones (elbows and knees). The only drawback of this redistribution is that it modifies the semantics of the model, which no longer matches the human morphology. This is actually a minor concern because data classification and interpretation do not rely on a specific parametrisation. Even in cases where a specific mapping is needed, like for rendering, a conversion can recover the morphological parametrisation. The repartition of joints, $\{Jt_1 \ldots Jt_{21}\}$, is illustrated in Figure 5.1.

For the sake of both performance and simplicity, all the joints are implemented with only one degree of freedom (one rotation around a fixed axis). More complex joints are then built by putting successively two or three of these 1-dof joints in a kinematic chain: this is the way Euler angles work. The length of the bone of the first joints would then be null, giving the illusion of a single joint with more degrees of freedom. Such a choice simplifies the implementation of update algorithms.

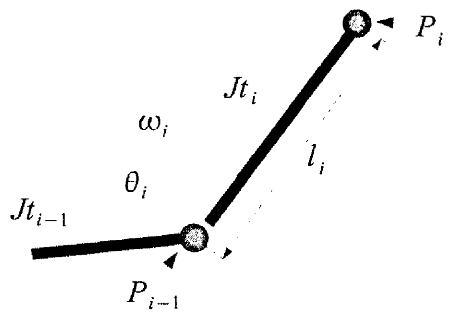


Figure 5.2: *Parametrisation of a joint Jt_i with relation to its parent Jt_{i-1}. The transformation is composed of a rotation of angle θ_i around the axis ω_i, and is followed by a translation of length l_i.*

Notations

In the following, the term "joint" will refer to both the joint itself and the bone attached to it. Let us now consider a kinematic chain (for example, the chain linking the pelvis and the right hand) instead of the whole kinematic tree. In such a chain of N_J joints $\{Jt_{\pi(1)} \dots Jt_{\pi(N_J)}\}$, the function $\pi(.)$ is a partial mapping from $\{1 \dots N_J\}$ to $\{1 \dots 21\}$ and the parent of a given joint $Jt_{\pi(j)}, j \in \{1 \dots N_J\}$ is $Jt_{\pi(j-1)}$. In order to keep the notation simple and readable, we shall denote in the rest of this thesis the current joint in the kinematic chain as Jt_i and its parent as Jt_{i-1}.

Each joint is defined in the coordinate system of its parent, which means that a joint contains only the transformation needed to compute its position with respect to its parent's. Since all joints have only one degree of freedom (one rotation), the transformation between Jt_{i-1} and Jt_i consists of a rotation of angle θ_i around the axis ω_i. Note that the rotation axis ω_i is defined in the coordinate system of Jt_{i-1}. The rotation is followed by a translation of the length of the bone l_i, which is performed in the local coordinate system of Jt_i, so that we can arbitrarily decide that it always happens along the first axis of the local coordinate system. We finally define the global 3-D position of the joint Jt_i as P_i, obtained after rotation and translation. These notations are summarised in Figure 5.2.

Using the joints as defined in Figure 5.1, and the notations from Figure 5.2, we now define the actual parametrisation of each individual joint in Table 5.1. The constraints are enforced under the form of bounding values on the joint angles, $\theta_i \in [\theta_i^-, \theta_i^+]$. These constraints are very simplistic, missing all dependencies between joints, and are therefore clearly insufficient to limit the space of possible model configurations to only the valid ones. Figure 5.3 illustrates the rotation axes and constraints. As stated

Joint Id	Description	ω	θ^-	θ^+	l/scale	Parent
Jt_1	Torso Left/Right	Z	$-\pi/8$	$\pi/8$	0	Root
Jt_2	Torso Front/Back	Y	$-\pi/4$	$\pi/2$	0	Jt_1
Jt_3	Torso Twist	X	$-\pi/4$	$\pi/4$	0.281	Jt_2
Jt_4	Head Left/Right	Z	$-\pi/8$	$\pi/8$	0	Jt_3
Jt_5	Head Front/Back	Y	$-\pi/4$	$\pi/4$	0.179	Jt_4
Jt_6	Shoulder L. Up/Down	$X+Z$	$-2\pi/5$	$\pi/2$	0	Jt_3
Jt_7	Shoulder L. Front/Back	Y	$-\pi/5$	$7\pi/10$	0.153	Jt_6
Jt_8	Shoulder R. Up/Down	$-X-Z$	$-2\pi/5$	$\pi/2$	0	Jt_3
Jt_9	Shoulder R. Front/Back	Y	$-\pi/5$	$7\pi/10$	0.153	Jt_8
Jt_{10}	Elbow L. 1	$2X+Z$	$-3\pi/4$	$3\pi/4$	0	Jt_7
Jt_{11}	Elbow L. 2	Y	0	$4\pi/5$	0.164	Jt_{10}
Jt_{12}	Elbow R. 1	$-2X-Z$	$-3\pi/4$	$3\pi/4$	0	Jt_9
Jt_{13}	Elbow R. 2	Y	0	$4\pi/5$	0.164	Jt_{12}
Jt_{14}	Hip L. Left/Right	Z	$\pi/10$	$3\pi/4$	0	Root
Jt_{15}	Hip L. Front/Back	Y	$-\pi/3$	$2\pi/5$	0.25	Jt_{14}
Jt_{16}	Hip R. Left/Right	$-Z$	$\pi/10$	$3\pi/4$	0	Root
Jt_{17}	Hip R. Front/Back	Y	$-\pi/3$	$2\pi/5$	0.25	Jt_{16}
Jt_{18}	Knee L. 1	$-X+Z$	0	$\pi/2$	0	Jt_{15}
Jt_{19}	Knee L. 2	Y	$-\pi/2$	0	0.235	Jt_{18}
Jt_{20}	Knee R. 1	$X-Z$	0	$\pi/2$	0	Jt_{17}
Jt_{21}	Knee R. 2	Y	$-\pi/2$	0	0.235	Jt_{20}

Table 5.1: *Parametrisation of each joint of the kinematic model, including its rotation axis ω, its bounding angle values $[\theta^-, \theta^+]$ and its relative length.*

previously, more accurate constraints would require motion capture data, and are the subject of future work. The lengths of the bones are reported as ratios of the total size of the subject, so that different body sizes are produced with a simple scaling factor. It can also be noticed that some axes of rotation for the right-hand limbs are flipped compared to the corresponding ones in the left-hand limbs: this allows a symmetry of the joint angles across each side of the body, which in turn facilitates data interpretation.

Forward Kinematics

Let us denote as (P_0, R_0) the global position and orientation of the root of the kinematic tree, r_i the local rotation of angle θ_i around the axis ω_i, and t_i the translation of length l_i along the first axis or the local coordinate system. The global position P_i of a joint Jt_i in a kinematic chain $\{Jt_1 \dots Jt_{N_J}\}$ is computed recursively:

$$P_i = P_{i-1} + R_i \cdot t_i \qquad \text{where} \qquad R_i = R_{i-1} \cdot r_i \tag{5.1}$$

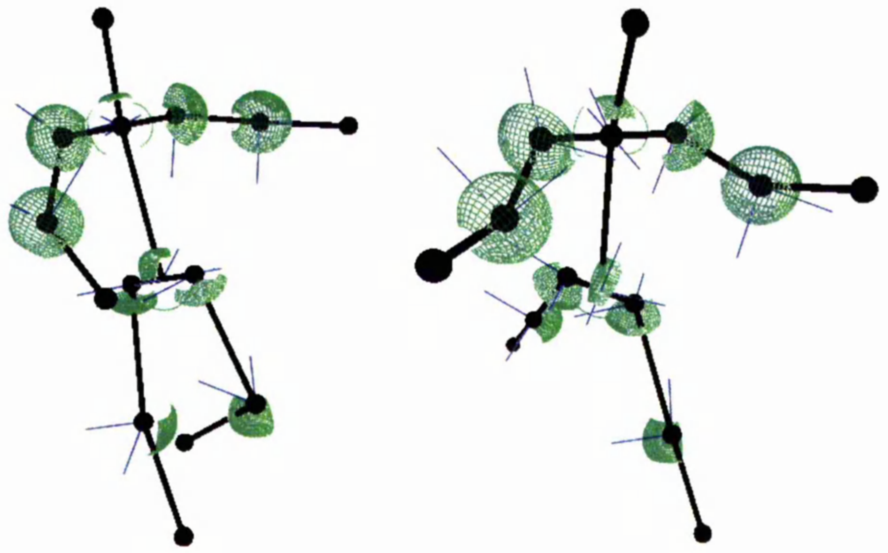


Figure 5.3: *Screenshots of the model in kinematically valid poses. The blue lines are the rotation axes of the joints. Notice the inversions between the two sides of the body, to keep a coherent parametrisation. The green surface patches represent the allowed movement of a bone with respect to its parent in the kinematic tree.*

leading to the standard formulation of forward kinematics:

$$P_i = P_0 + R_0 \cdot r_1 \left(t_1 + r_2 \left(t_2 + \left(\cdots + r_{i-1} \left(t_{i-1} + r_i t_i\right)\right)\right)\right) \tag{5.2}$$

The formulation of forward kinematics in the full kinematic tree is exactly the same, although the intermediate positions are cached to avoid unnecessary re-computations. Another optimisation, when realising that many of the t_i are null, is to implement the local rotations r_i with quaternions which make the composition of rotations faster.

5.2 Linking the Blobs to the Model

The kinematic model described in the previous section is used in conjunction with the blobs. As we saw in Chapter 4, the actual feature tracking is performed solely by the blobs during Expectation-Maximisation. The role of the kinematic model with respect to the blobs is twofold. First, it is used to constrain and correct the movements of the blobs (as seen in Section 4.3), and second, the kinematic model is necessary to interpret the data collected from tracking: it gives a semantic meaning to the blobs.

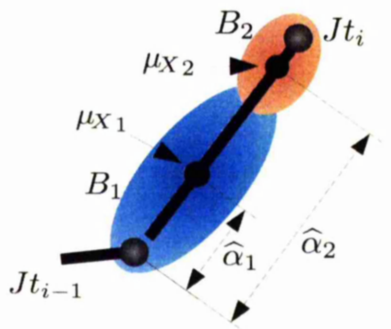


Figure 5.4: *Two blobs, B_1 and B_2, attached to a joint Jt_i. The blobs are attached by their centres $\mu_{X\,1}$ and $\mu_{X\,2}$ along the main axis of the joint. the actual offset of the point of attachment is noted by a coefficient ($\widehat{\alpha}_1$ and $\widehat{\alpha}_2$).*

Furthermore, the parameters of the kinematic model are a much more compact representation of the tracked subject than the blobs.

Each blob is attached to a bone (a joint with a non-zero length) of the kinematic model. The "attachment" between a blob and a bone can be pictured as a virtual spring that would drive the corresponding joint towards the mean position of the blob. When the blobs have a clear orientation (typically, the greatest eigenvalue being at least twice the next one), then the virtual spring also drives the bone in alignment with the main direction of the blob.

The point of attachment of a blob onto the kinematic model is relative to the local coordinate system of the joint to which it is attached. Considering the relative symmetry of the body parts around the bones of the model, we only allow the points of attachment to lie along the bone itself. For example, a blob representing the hand will be attached near the extremity of the bone of the forearm. This assumption simplifies the estimation of the kinematic pose at a reasonably small cost in accuracy. As illustrated by Figure 5.4, each blob B_k is attached at its centre $\mu_{X\,k}$ to an offset $\widehat{\alpha}_k$ along the bone of a joint Jt_i.

The process of driving the kinematic model to match the position of the blobs is decomposed into two steps. Firstly, some "goal positions" are computed for the joints, taking into account the position and orientation of the blobs. Note that these goals are not necessarily all reachable by the kinematic model because no kinematic constraints were enforced during the fitting of the blobs. An iterative algorithm for computing the goal positions is described in the rest of this section. The second step then consists in finding the kinematically valid configuration of the model that satisfies best the goal positions. This last problem is a typical case of inverse kinematics, and is dealt with in

Section 5.3.

5.2.1 Evaluation of Goal Positions

We are now concerned with the evaluation of goal positions for the joints, using the blobs obtained after Expectation-Maximisation. Even if these goal positions do not need to be correct in the sense of kinematic constraints, they should nonetheless be computed using as much prior knowledge as possible because the subsequent evaluation of the model configuration relies solely on them. In addition to the positions and directions of the blobs attached to the current joint, the computation of a goal position should also be influenced by the blobs attached to neighbouring joints in the kinematic tree. For instance, the position of the knee should be conditioned by both the blobs on the upper leg and the blobs on the lower leg. Finally, the current position of each joint is an important hint for the computation of the goals, especially when the orientation of the blobs is not decisive.

All joints do not need to have a goal position defined because the inverse kinematics scheme used to recover the full model configuration is able to interpolate the position of joints for which no goal is available. Nevertheless, a guess of the goal position of a joint is preferable to no information at all. Possible errors should indeed be robustly tackled by the kinematic constraints, but a total lack of information leads to a wild guess of the position of the joint, which is hardly desirable in the absence of a dynamic model. With a more elaborate model of dynamics or behaviour, guessing the position of joints with no information would still be possible. For example, Grochow *et al.* [GMHP04] describe a style-based inverse kinematics method, where the optimal body pose is found under learnt constraints of movement-specific styles. The full pose of the body is then recovered from only a few goal positions, such as the recovery of the full gait cycle from the positions of the feet only. Such a method would be highly beneficial to our tracking algorithm, but in order to remain general enough, a large amount of behaviours would have to be captured and learnt. It is then not clear whether a method like [GMHP04] can learn all these diverse behaviours without loosing in efficiency.

For most interactive applications, it is especially important to compute as accurately as possible the goal positions of the end effectors (hands, feet...). The algorithm used to compute the goal positions of the joints computes first the goal positions at the leaves of the kinematic tree, and propagates the computation up to the root. Since several concurrent constraints need to be satisfied for each joint, a general optimisation

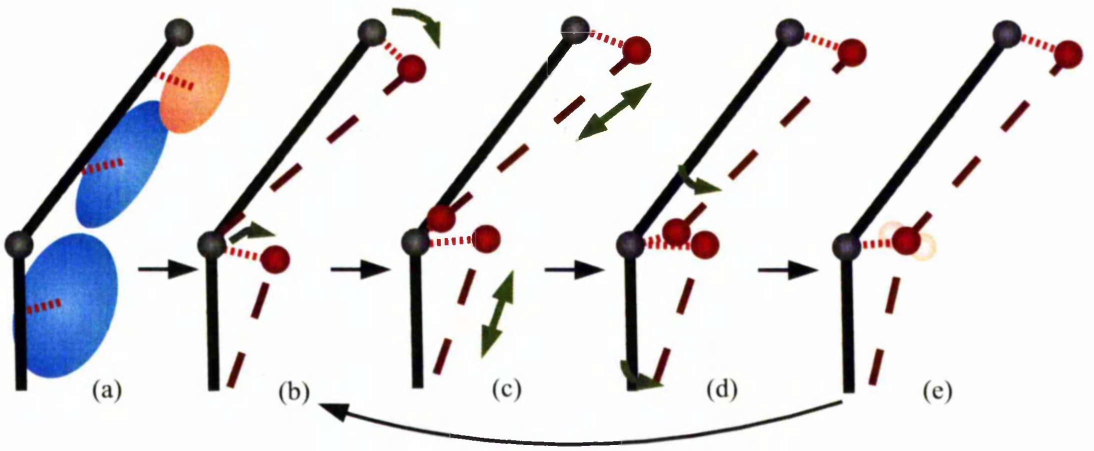


Figure 5.5: *Computation of goal positions from the blobs. From left to right: (a) After Expectation-Maximisation, the blobs attached to the bones of the kinematic model have new parameters. (b) The goal position of the tip of each joint is first computed, followed by the computation of the goal position of the base by firstly (c) translating to minimise re-projection error, and secondly (d) re-aligning with the blobs. The estimate from each joint is then (e) merged with the one from its parent before iterating.*

algorithm would be very complex and hence computationally too expensive. We use instead a simple iterative scheme, which optimises in turn the position of the tip and the base of each bone, eventually converging to a satisfying solution. This iterative process is illustrated in Figure 5.5, and includes the following steps:

1. We compute the goal position for the tip of the current joint using the base of the joint as a fixed rotation point (Figure 5.5b).
2. We translate the goal positions of the base and tip of each joint so as to minimise the projection error of the mean of the blobs onto the bone (Figure 5.5c).
3. The goal position of the base of the joint is optimised using this time the goal position of the tip of the joint as a fixed rotation point (Figure 5.5d).
4. The goal position coming from both the current joint and its parent are merged into a single goal position (Figure 5.5e). The algorithm is then iterated from step 1 using the newly computed goal positions as basis.

The algorithm typically needs only a few iterations to generate a satisfying solution. The stopping condition is then simply that the algorithm has converged, meaning that the sum of the squared distances between the goals from the last iteration and the

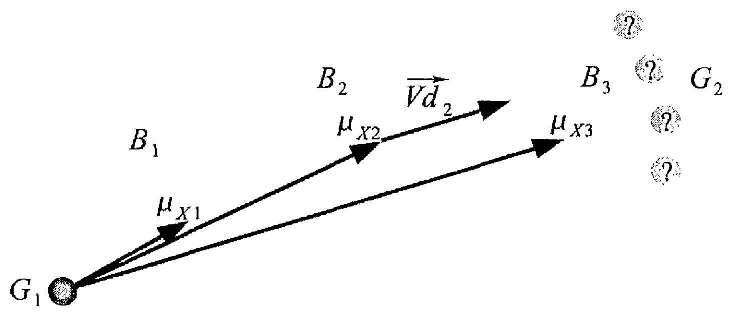


Figure 5.6: *Computation of the direction of a goal position from a fixed point* G_1 *and some blobs* $\{B_1, B_2, B_3\}$.

current ones is below a pre-defined threshold (manually-set, in our implementation). The algorithm is kept simple and efficient by optimising only one goal at a time. We will now look more closely into the steps (b)–(e) of Figure 5.5 before exploiting these goal positions to recover a correct kinematic pose in Section 5.3.

Aligning a Bone Segment with the Associated Blobs

A non-trivial part of the algorithm from Figure 5.5 occurs twice, at steps (b) and (d) and involves aligning a moving goal with a fixed rotation point (a previously determined goal) and a set of blobs. As an illustration of the problem, let us consider Figure 5.6, where we want to determine the direction of the vector $\overrightarrow{G_1G_2}$ pointing to the moving goal position G_2 from the fixed point G_1. The blobs $\{B_1, B_2, B_3\}$ of means $\{\mu_{X1}, \mu_{X2}, \mu_{X3}\}$ are attached to the same bone segment at the offsets $\{\widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha}_3\}$. Additionally, the blob B_2 has an elongated shape so that a direction vector $\overrightarrow{Vd_2}$ could be computed.

Let us first deal with the position of the blobs alone and incorporate the possible intrinsic directions in a second step. A simple sum of vectors $\sum_i \overrightarrow{G_1\mu_{Xi}}$ would have the desirable property that the blobs furthest to G_1 have a greater influence. Unfortunately, it would not be robust to misplaced blobs: the influence of a blob should indeed be proportional to its real position along the bone segment, instead of its distance to the goal position, which is not reliable.

The direction of the goal position G_2 can then be computed by giving to each blob $B_i, i \in [1, \ldots, N_b]$ a contribution proportional to its offset $\widehat{\alpha}_i$ along the bone segment:

$$\overrightarrow{G_1G_2} = \frac{l}{\|\overrightarrow{G_1G_2}\|} \sum_{i=1}^{N_b} \widehat{\alpha}_i \frac{\overrightarrow{G_1\mu_{Xi}}}{\|\overrightarrow{G_1\mu_{Xi}}\|} \tag{5.3}$$

where l is the length of the current bone segment. The remaining task is to include the contribution of possible intrinsic direction, such as the vector $\overrightarrow{Vd_2}$ in Figure 5.6. A fast algorithm to compute the main direction of a Gaussian blob is described in Appendix D. The contribution of this intrinsic direction does not depend on the offset of the blob along the bone segment, but only on the strength of the direction vector. A strongly elongated blob should contribute with greater strength than a smaller and rounder one. The ratio between the first and the second eigenvalue of the covariance matrix Σ_X of a blob reflects strength of the elongation. The full computation of the goal position G_2 is performed according to the formula:

$$\overrightarrow{G_1G_2} = \frac{l}{\|\overrightarrow{G_1G_2}\|} \cdot \left(\sum_{i=1}^{N_b} \widehat{\alpha}_i \frac{\overrightarrow{G_1\mu_{X_i}}}{\|\overrightarrow{G_1\mu_{X_i}}\|} + \sum_{i, \frac{\lambda_{i,1}}{\lambda_{i,2}} > 2} \frac{\lambda_{i,1}\overrightarrow{Vd_i}}{\lambda_{i,2}} \right) \tag{5.4}$$

Translating the Goal Position

At step (c) of the algorithm illustrated in Figure 5.5, the goal positions G_1 and G_2 are translated along the current direction of the vector $\overrightarrow{G_1G_2}$. With this translation of offset T, we wish to minimise the square distances between the projected centres of the blobs and their actual points of attachment (see Figure 5.7). This is a standard least square optimisation problem:

$$T = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(\overrightarrow{G_1\mu_{X_i}} \cdot \frac{\overrightarrow{G_1G_2}}{\|\overrightarrow{G_1G_2}\|} - \widehat{\alpha}_i \right) \tag{5.5}$$

5.2.2 Complete Algorithm

The recursive computation of all the goal positions in the kinematic tree is presented in Algorithm 5.1. This algorithm is applied to the root of the kinematic tree, and iterated until convergence. The goal position for a given joint is either enabled if sufficient information can be gathered from the blobs or the children joints, or disabled otherwise. The joints with zero-length are treated as a special case which transmit the goal positions but disable it for themselves. The algorithm also takes into account joints with no blobs attached, for which a goal position is inferred from the children, when possible.

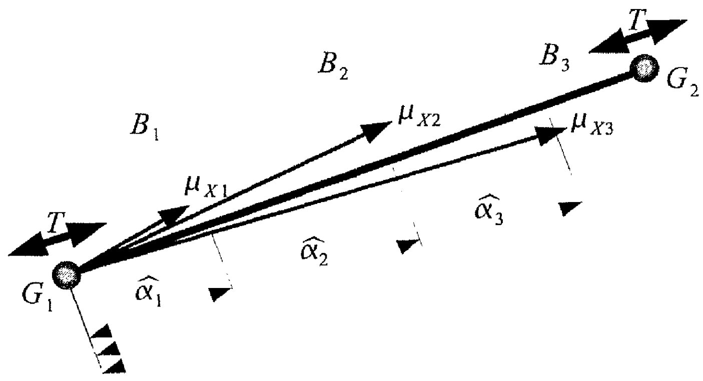


Figure 5.7: *Translation of the goal positions G_1 and G_2 by an offset T along the direction of $\overrightarrow{G_1G_2}$, to minimise the error between the projected centres of the blobs and their actual point of attachment.*

5.3 Inverse Kinematics

Inverse kinematics is concerned with recovering the parameters of the model from the desired global positions of the joints. In our case, the parameters of the model are the joint angles $\Theta = \{\theta_1, \ldots, \theta_{N_J}\}$ and the global position and orientation (P_0, R_0) of the root of the kinematic tree. The desired global positions of the joints are the goal positions $\mathcal{G} = \{G_1, \ldots, G_{N_G}\}$ computed in the previous section. Note that the number of computed goal positions N_G is usually smaller than the number of joints N_J: this makes inverse kinematics an under-constrained problem. The relation between the parameters of the model and the subset of the joint positions $\mathcal{P} = \{P_1, \ldots, P_{N_G}\}$ is given by a set of forwards kinematics functions $F = \{f_1, \ldots f_{N_G}\}$ similar to that defined in Equation 5.1:

$$\mathcal{P} = F(P_0, R_0, \Theta) \Leftrightarrow \begin{cases} P_1 = f_1(P_0, R_0, \theta_1, \ldots, \theta_{N_J}) \\ P_2 = f_2(P_0, R_0, \theta_1, \ldots, \theta_{N_J}) \\ \quad\vdots \\ P_{N_G} = f_{N_G}(P_0, R_0, \theta_1, \ldots, \theta_{N_J}) \end{cases} \tag{5.6}$$

If F was invertible, one could compute the desired model parameters from the goal positions as $F^{-1}(\mathcal{G})$. Unfortunately, because of singularities, redundant configurations (Figure 5.8) and kinematic constraints, the function F is non-invertible. Moreover, the goal positions $\mathcal{G}$ are unlikely to be kinematically valid, and a solution can therefore not be reached.

Algorithm 5.1: Recursive computation of the goal positions

```
▷ Get recursively the goal positions from the children;
if at least one blob attached to the current joint then
    ▷ Compute the goal positions for the current joint as in Figure 5.5;
    ▷ Merge the computed goal position of the tip of the joint with the ones
    returned by the children;
    ▷ Enable the goal position for the current joint;
    return the goal position of the base;
else
    if at lest one children returned a goal position then
        if the length of the joint is null then
            ▷ Disable the goal position for the current joint;
            return the average of the goal positions returned by the children;
        else
            ▷ Enable the current goal as the average of the goal positions
            returned by the children;
            return nothing;
        end
    else
        ▷ Disable the goal position for the current joint;
        return nothing;
    end
end
```

A standard idea is to linearise F about the current configuration of the model. It should then be possible to invert this linear local approximation of F to get iteratively closer to the solution. The local linearisation of F is called the Jacobian matrix:

$$J = \frac{dF(P_0, R_0, \Theta)}{d(P_0, R_0, \Theta)} = \begin{pmatrix} \frac{\partial f_1}{\partial P_0} & \frac{\partial f_1}{\partial R_0} & \frac{\partial f_1}{\partial \theta_1} & \cdots & \frac{\partial f_1}{\partial \theta_{N_J}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{N_G}}{\partial P_0} & \frac{\partial f_{N_G}}{\partial R_0} & \frac{\partial f_{N_G}}{\partial \theta_1} & \cdots & \frac{\partial f_{N_G}}{\partial \theta_{N_J}} \end{pmatrix} \tag{5.7}$$

A small variation of the global positions of the joints $\Delta\mathcal{P}$ is then propagated to the model parameters using the inverse of the Jacobian matrix:

$$\Delta(P_0, R_0, \Theta) = J^{-1} \cdot \Delta(P_1, \ldots, P_{N_G}) \tag{5.8}$$

Unfortunately, the Jacobian matrix is generally not any more invertible than the original function F. The Moore-Penrose pseudo-inverse ($(J^T J)^{-1} J^T$) can be used instead,

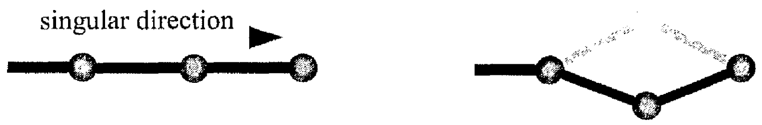


Figure 5.8: *(left) Singular configuration: no rotation of the joints can move the effector along the singular direction. The corresponding row of the Jacobian matrix contains only zeros, and is therefore non-invertible. (right) Redundant configuration: the system is under-constrained. The Jacobian matrix is not square and again non-invertible.*

but singularities and the problem of the unreliable goal positions remain. The computational cost is also high since the Jacobian matrix has to be re-computed at each iteration, until convergence.

A common variation around the Jacobian method is to integrate the estimation of parameters in an Extended Kalman Filtering (EKF) framework. This has been done for human-body tracking [MTHC03, JTH99] with a state transition matrix equal to the identity matrix, so that the main benefit of the Kalman filter is to smooth the measurement noise. Unfortunately, this measurement noise is rarely evaluated, and the assumption that it should be zero-mean, white and Gaussian is never tested. Actually, the goal positions of the body parts are subject to many biases making these assumptions unlikely.

Alternatively, inverse kinematics can be formulated as an optimisation problem. The aim is then to minimise the error between the current positions $\mathcal{P}$ of the joints and their goal positions $\mathcal{G}$. The squared Euclidean distance is often used to define the error function:

$$|\mathcal{G} - \mathcal{P}| = |\mathcal{G} - F(P_0, R_0, \Theta)| = \sum_{i=1}^{N_G} (G_i - f_i(P_0, R_0, \Theta))^2 \tag{5.9}$$

Standard optimisation methods like Gauss-Newton of Levenberg-Marquardt can be applied to compute the set of joint angles minimising the error function. A local derivative of the error function is often used to get the direction of the gradient, leading to a convergence similar to that of the Jacobian matrix described earlier. Some constraints can also be incorporated into the system, leaving to the chosen optimisation algorithm the care to avoid local minima. Concerns about computational efficiency deter us from using these general optimisation methods, and a simpler iterative scheme is preferred.

Motivated by the difficulty of obtaining of the gradient of the error and the need to avoid local minima, Monte-Carlo methods are an interesting way to estimate a set of model parameters minimising the error function. They work by sampling (more or less randomly) a large number of model configurations in the parameter space, evaluating each of them, and finally, estimating the mode(s) of the weighted discrete distribution. We shall present in Chapter 6 an efficient pose estimation method based on discrete Monte-Carlo sampling.

In the rest of this chapter, we describe a simple direct evaluation scheme, aimed primarily at computational performance. The position and orientation of the root of the kinematic tree are first evaluated from the available goal positions of the torso and the hips. Each of the limbs is subsequently processed with an iterative local optimisation method called *Cyclic-Coordinate Descent* (CCD) [Wel93].

5.3.1 Estimation of the Root Position and Orientation

During normal tracking circumstances, the goal positions for the joints of the hips ($G_{hip\,1}$ and $G_{hip\,2}$) and the pelvis (G_{pelvis}) are available because some blobs are always attached to the torso and the legs. The global position of the root of the kinematic tree (the pelvis) can be evaluated as a simple average of the goal positions of the hips and the pelvis:

$$P_0 = \frac{G_{hip\,1} + G_{hip\,2} + G_{pelvis}}{3}$$

The positions of the hips are mainly taken into account for extra robustness. The global orientation is then computed using the goal position of the neck (G_{neck}), which can be computed as the average of the goal positions of the shoulders when not available:

$$\begin{aligned}
R_{0x} &= \frac{G_{hip\,2} - G_{hip\,1}}{\|G_{hip\,2} - G_{hip\,1}\|} \\
R_{0y} &= \frac{(G_{neck} - G_{pelvis}) - ((G_{neck} - G_{pelvis}) \cdot R_{0x}).R_{0x}}{\|(G_{neck} - G_{pelvis}) - ((G_{neck} - G_{pelvis}) \cdot R_{0x}).R_{0x}\|} \\
R_{0z} &= R_{0x} \wedge R_{0y}
\end{aligned}$$

5.3.2 Initialisation of the Root Position and Orientation

At initialisation, the root parameters are estimated using the main axes of the volumetric reconstruction. A root blob is computed from all the voxels, using the Maximisation procedure described in Section 4.2.4. The eigenvectors, or principal axes, of this root blobs are obtained using the iterative algorithm described in Appendix D. The principal axis of the blob defines the vertical orientation of the model, always pointing

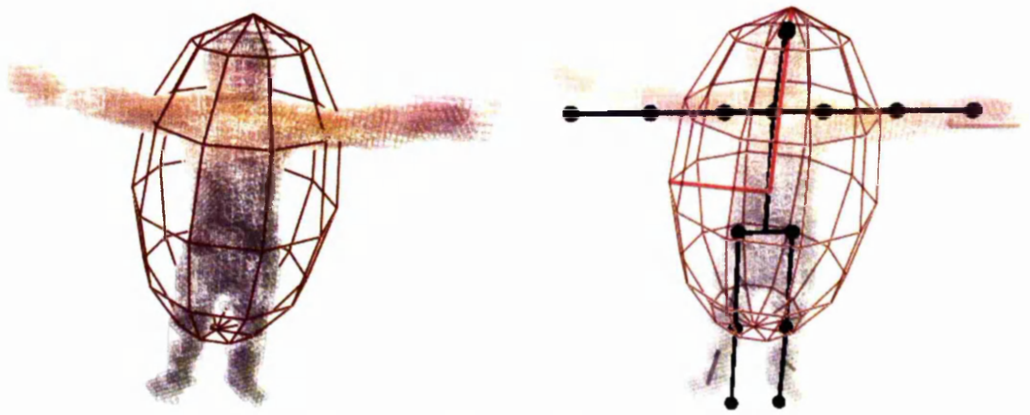


Figure 5.9: *Initialisation of the root position and orientation of the model. The mean and covariance of a root blob are evaluated from all the voxels. The two greatest eigenvectors define the orientation of the model, while the root position is found along the greatest eigenvector.*

towards increasing y values. The horizontal orientation is then defined by the second greatest eigenvector, up to "back-front flipping" ambiguity. We currently assume that the subject is always facing the same direction at initialisation, but a simple way to recover the direction faced by the subject would be to keep both hypotheses until a kinematic constraint is violated when tracking with the incorrect one. The position of the root of the kinematic tree is found along the main axis, by imposing the constraint that at least one of the feet should be on the ground. Figure 5.9 illustrates this initialisation process.

5.3.3 Cyclic-Coordinate Descent

The Cyclic-Coordinate Descent (CCD) is an iterative local optimisation method. Each joint of the kinematic tree is optimised independently, starting with the leaves (or end-effectors) and progressing towards the root of the tree. The optimisation at a given joint consists in minimising the error between both itself and its children in the kinematic tree, and the corresponding goal positions. Let us denote the current joint as Jt_i and its children as $\{Jt_{i,1}, \ldots, Jt_{i,n}\}$ with global position vectors $\{P_i, P_{i,1}, \ldots, P_{i,n}\}$ and associated goal positions $\{G_i, G_{i,1}, \ldots, G_{i,n}\}$.

Let us first consider the simpler problem of optimising the joint Jt_i when either only itself or only one of its children has a goal position to satisfy. The only degree of freedom of the current joint Jt_i is a rotation of angle θ_i about the axis ω_i. The joint

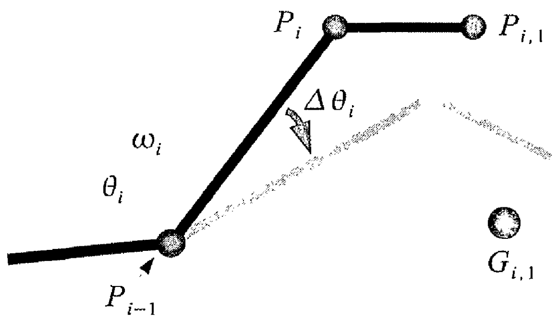


Figure 5.10: *CCD with a single goal position, minimising the error between the joint of current position $P_{i,1}$ and associated goal position $G_{i,1}$.*

with a goal position to satisfy is arbitrarily denoted as $P_{i,1}$ with associated goal position $G_{i,1}$. Our aim is to find the variation of angle $\Delta\theta_i$ which minimises the error between $P_{i,1}$ and $G_{i,1}$, as illustrated in Figure 5.10.

The position of the base of the joint Jt_i is P_{i-1}: let us then denote $\overrightarrow{P} = \overrightarrow{P_{i-1}P_{i,1}}$ and $\overrightarrow{G} = \overrightarrow{P_{i-1}G_{i,1}}$. It can be shown [Wel93] that the variation of angle $\Delta\theta_i$ minimising the distance between $P_{i,1}$ and $G_{i,1}$ also maximises the scalar product between $\overrightarrow{P}$ and $\overrightarrow{G}$, and has the following closed form expression:

$$\Delta\theta_i = \arctan \frac{\omega_i \cdot (\overrightarrow{P} \wedge \overrightarrow{G})}{\overrightarrow{G} \cdot \overrightarrow{P} - (\overrightarrow{G} \cdot \omega_i).(\overrightarrow{P} \cdot \omega_i)} \tag{5.10}$$

This formulation only finds a solution in the range $\Delta\theta_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. While other solutions can theoretically be found using the periodicity of tan, it is very unlikely in practice that a variation of more than $\frac{\pi}{2}$ radians could occur between two consecutive frames. The range of the solution is therefore sufficiently wide for our purpose.

The angle constraints defined in Table 5.1 are enforced by clamping the rotation angle θ_i to the range $[\theta_i^-, \theta_i^+]$ after each local optimisation. When constraints associated with Euler angles are too restrictive, the allowed sub-space of a given joint can be non-convex. In such a case, independent optimisation of joints is insufficient to bypass the lock enforced by the constraints. A practical solution is to disable the constraints for the first few iterations of the CCD, allowing angles to get past possible mutual locks, and then re-enforce the constraints in a second pass.

To allow a smooth repartition of movement between all the joints in the kinematic tree, an attenuation coefficient η is introduced. This attenuation (or stiffness) coefficient also limits undesired "oscillations" which tend to happen when optimising simultaneously multiple joints with contradictory sub-goals. At each iteration, the rotation angle θ_i is updated in the following way:

$$\theta_i = \theta_i + \eta.\Delta\theta_i \tag{5.11}$$

where the coefficient η controls directly the rate of convergence. We found $\eta = 0.3$ to be a good compromise between smoothness and efficiency.

Let us now deal with the full inverse kinematics problem, when more than one child of the current joint Jt_i has a goal position to satisfy. Finding analytically the variation of angle $\Delta\theta_i$ which minimises the total error between all the children and their goal positions would be too complex. Instead, we use an approximate heuristic which combines the individual optimisations of all the sub-goals. Let us denote as $\{\Delta\theta_{i,1}, \ldots, \Delta\theta_{i,k}\}$ the angle variations computed with Equation 5.10, that optimise the sub-goals $\{G_{i,1}, \ldots, G_{i,k}\}$. The combined angle variation for the joint Jt_i is a weighted sum of the individual ones:

$$\Delta\theta_i = \frac{1}{\sum_{j=1}^{k} \lambda_{i,j}} . \sum_{j=1}^{k} \lambda_{i,j} \Delta\theta_{i,j} \tag{5.12}$$

where $\{\lambda_{i,1}, \ldots, \lambda_{i,k}\}$ are weighting coefficients designed to give more importance to goals nearer to the root of the kinematic tree. Intuitively speaking, if the weights were uniform, the goals nearer to the root of the kinematic tree would only be optimised partially, whereas the goals near the leaves would be advantaged by their more frequent updates. In practice, we take $\lambda_{i,j}$ equal to the inverse of the number of joints separating Jt_i to the joint associated with the goal $G_{i,j}$. An illustration of the iterative optimisation process with two goals is presented in Figure 5.11.

The overall convergence rate of the CCD is difficult to analyse, because changes at each joint are incrementally taken into account when optimising the next ones. Welman [Wel93] reported a faster convergence rate with CCD compared to a standard Jacobian optimisation method. The CCD method has the additional advantages of behaving well around singular configurations, and of incorporating constraints in straightforward manner.

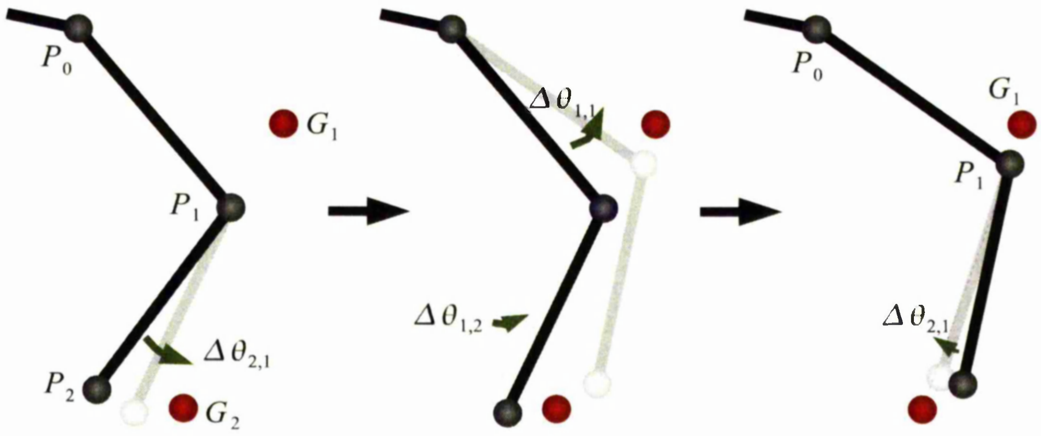


Figure 5.11: *CCD with two goal positions G_1 and G_2 to satisfy. (left) The algorithm starts by optimising the joint nearest to the end of the kinematic chain. (middle) The parent joint is then optimised, integrating the optimisations of both sub-goals. (right) Successive iterations allow the convergence of the method.*

5.4 Results

The computation of the goal positions and the fitting of the kinematic model are demonstrated in Figure 5.12. The kinematic constraints are also shown to illustrate the fact that some goal positions cannot be fully satisfied, for example at the right foot. The processing cost for these two steps is in average 0.55ms per frame ($\simeq 1800$fps) with 10 iterations of IK, which is clearly within our objectives.

Figure 5.13 presents some tracking results, sampled from sequences of slow movements, captured by 4 cameras at 15 Hertz. These results show the potential of the approach, faced with noisy images, a cluttered environment and a poor inter-cameras synchronisation. Overall, we found that the simple hierarchical method presented in this chapter worked very well, as long as the root of the kinematic tree can be located. The inclusion of colour allows to keep track of limbs which would be lost using reconstructed volume alone, as illustrated by Figure 5.14.

Unfortunately, while locating the torso works well with relatively static sequences involving limited self-occlusions, more challenging types of motions remain problematic. For example, the hierarchical tracker could not cope with the ballet dancing sequences presented in Chapter 7 because of the very fast movements and rotations of the torso. Additional constraints and a prior knowledge of dynamics would be required to make the tracking more robust to missing or ambiguous evidences: this idea will be further developed in Chapter 6.

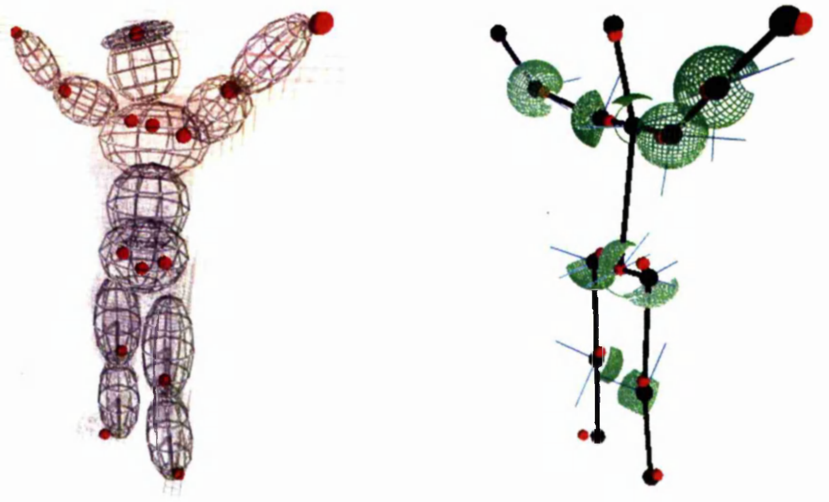


Figure 5.12: *(left) The goal positions, illustrated with red dots, are computed using the previous tracked position and the blobs from the current frame. (right) The full kinematic pose is then recovered by positioning the root of the kinematic tree, and optimising the remaining goals with inverse kinematics.*

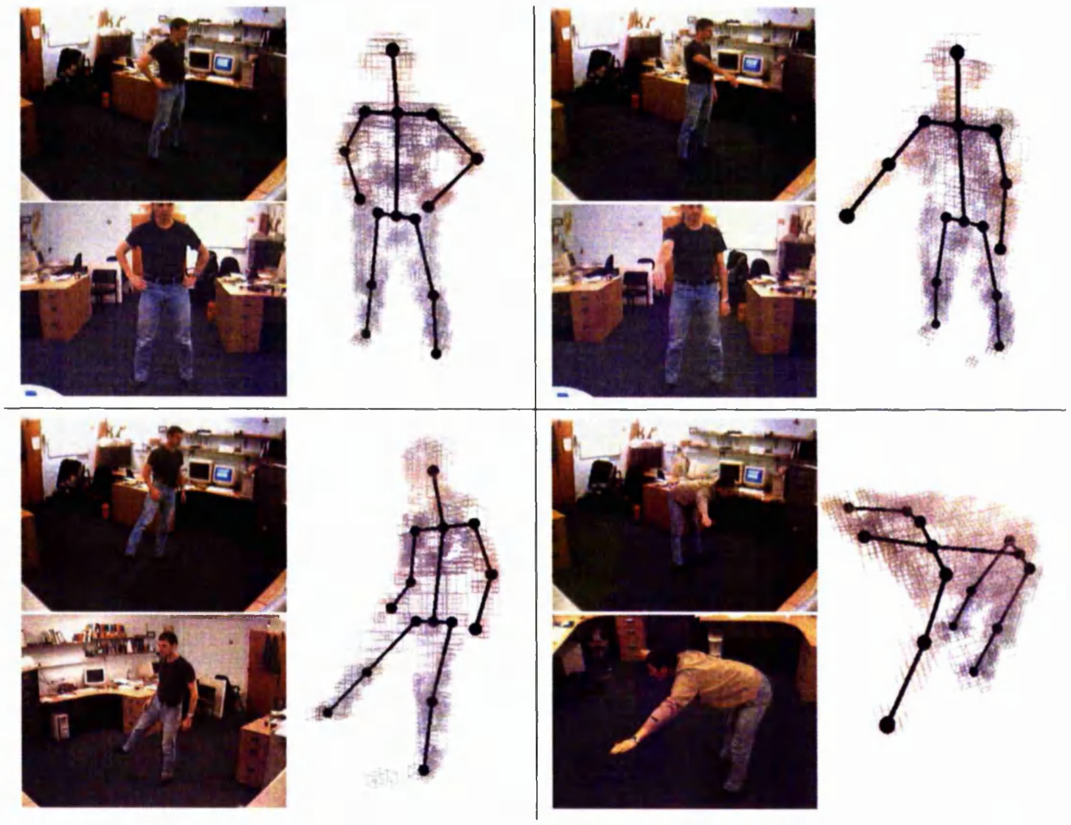


Figure 5.13: *Results of the hierarchical tracking method using* 4 *camera views.*

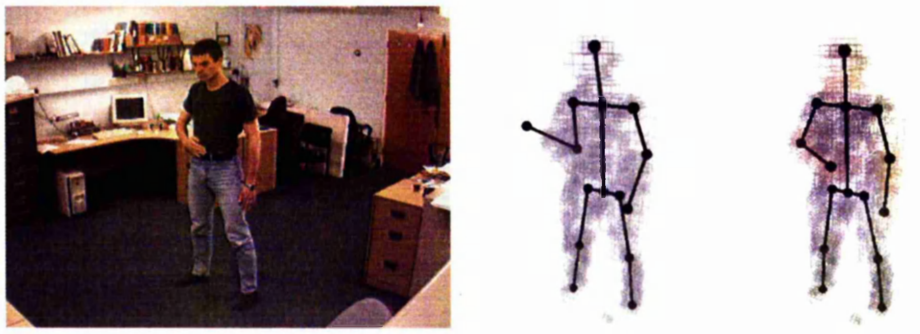


Figure 5.14: *Colour is important to disambiguate poses with self-occlusions.*

5.5 Discussion and Conclusion

We started this chapter with a description of the kinematic model used for tracking, and a choice of parametrisation. We then introduced an iterative algorithm to compute the goal positions of the joints of the kinematic model, based on the 3-D blobs. These goal positions were finally used to optimise the pose of the kinematic model using inverse kinematics.

Due to its very low computational cost, the hierarchical tracking method presented in this chapter is a practical solution for many human-computer interaction setups. It can be used, for example, in conjunction with a hand-tracker to recover gestures and control virtual interfaces. Hierarchical tracking methods are particularly adapted to upper-body tracking because the location of the root of the kinematic tree is fixed and known.

A number of improvements could be brought to the kinematic model and the hierarchical tracking framework. The kinematic model would benefit from a better joint parametrisation, like quaternions or exponential maps. Euler angles were chosen for simplicity, but singularities prevent the definition of joints with 3 degrees of freedom.

The model would also greatly benefit from better kinematic constraints: preventing inter-penetration between body parts, for example, would make the whole tracking process more robust. Soft constraints, either based on physical simulation or learnt from training data could also guide inverse kinematics, even with low evidences from the blobs. Motion prediction would make the blob-fitting procedure more robust, and allow the tracking of fast motions. These last issues will be addressed in Chapter 6.

Chapter 6

Bayesian Tracking with Monte-Carlo Sampling

This chapter builds on the volumetric reconstruction and blob-fitting frameworks, treating tracking as a global optimisation problem. The parameter space, augmented with first order derivatives, is automatically partitioned into Gaussian clusters each representing an elementary motion: hypothesis propagation inside each cluster is therefore accurate and efficient. The transitions between clusters use the predictions of a Variable Length Markov Model which can explain high-level behaviours over a long history. Using Monte-Carlo methods, the evaluation of large numbers of model candidates is critical for both speed and robustness. We present a new evaluation scheme based on volumetric reconstruction and blobs-fitting, where appearance models and image evidences are represented by Gaussian mixtures.

6.1 Introduction

Tracking is a global optimisation process: because of kinematic constraints, body parts are all linked to each other, and cannot be optimised individually without affecting the whole body configuration. So, when a set of detected image features is available (such as the blobs discussed in Chapter 4) limbs must "compete" with each other to fit onto their own detected features. The global solution is likely to be a compromise between the optimal placement of each body part and the enforcement of kinematic constraints.

6.1.1 Global Optimisation Techniques

One approach to tracking as a global optimisation problem is to start from image data, trying to detect features independently in each frame, and to search for a kinematically valid pose optimally satisfying the detected features. The configuration of the model is then recovered from the "bottom-up" [RF03], that is, from detected body part locations (features) to model parameters (joint angles). The feature detection stage is purely image-based, as no kinematic constraints or prior knowledge about the relative arrangement of the body parts are used. This has the important advantages of solving the hard problems of initialisation and recovery after failure, but the detectors are also weaker and the detected features are considered highly unreliable, typically producing many false-positives. The most likely model configuration must therefore be found robustly, and belief propagation techniques [SISB04, SBR$^+$04] are a way to address this problem. A global solution is iteratively reached by message-passing between the nodes of an interconnected graph. The graph itself encodes the relationships between body parts in a probabilistic manner, with strong priors about human kinematics acquired from training examples. Sigal *et al.* [SBR$^+$04] extend this graph to include relationships between body configurations at the previous, current and next time-steps, resulting in a potentially robust tracker. Ramanan *et al.* [RFZ05] include a collection of characteristic poses into the model, which can then automatically start tracking people striking these poses.

While bottom-up belief propagation techniques are theoretically appealing, they rely on the detection of specific features. This can become an issue for the exact same reasons that played against hierarchical methods: detecting individual body parts is not always possible because of occlusions or loose clothing. In the context of 3-D tracking, the problem is worsened by the difficulty of locating and matching 3-D features. Another important issue with the method is its computational complexity, which is currently too high for real-time applications.

Alternatively, one can use the body configuration in the current frame and a dynamic model to predict the next configuration candidates. These candidates are then tested against image data to find the most likely configuration. As opposed to the previous approach, we now start with estimations of the model parameters and test those hypotheses against image evidence: this is a top-down approach. In 1983, Hogg [Hog83] proposed a framework to evaluate model configurations of a walking person with extracted image features (edges). Tracking with particle filters works along those lines, approximating the *posterior* distribution by a set of representative elements, and updating these particles with the Monte Carlo importance sampling rule [GSS94]. A more in-depth description of particle filters is presented in Section 6.2.

Particle filtering has a number of advantages over hierarchical and bottom-up approaches. Firstly, the evaluation of particles does not pre-suppose any particular feature detection: this is important because it potentially makes the optimisation process very general. Any criterion or combination of criteria [MH03] can be used as objective function. Secondly, particle filters are simpler, and more flexible than most optimisation techniques. The number of particles and the complexity of their model of propagation can be adjusted for the application's needs. It can also be argued that particle filters are more robust than most standard optimisation methods, in the sense that they do not get trapped in local minima, and can track multiple hypothesis simultaneously.

Of course, particle filters also suffer from a number of drawbacks, the most obvious being the high number of particles required. In full body tracking problems, the dimensionality of the parameter space is far too high to accurately represent the probability distribution of the parameters given the image evidences (*posterior*) across the whole parameter space. In practice, a maximum of a few thousand particles can be managed in real-time, which is definitely too little to populate a parameter space with $d = 27$ dimensions. Particles then tend to concentrate in only a few of the most significant modes, leading to possible failures when too few particles are propagated to represent a new peak in the *posterior*. Since the particles cannot cover the whole space, they must be guided toward the portions of the parameter space that are the most likely to contain the solutions. Using the previous tracked configuration as a predictive basis is a strong advantage, but more advanced schemes have to be used when dealing with fast movements. In Section 6.3 of this chapter, we shall describe a predictive method capable of capturing the high-level behaviour of the subject.

The second drawback of particle filters comes as a direct consequence of the potentially large number of required particles, the main performance bottleneck being the evaluation of the *likelihood* function. Evaluating the *likelihood* of each particle usually involves generating a 3-D appearance model from the particle state, projecting this appearance model onto the available image planes, and finally comparing it with some extracted image features such as silhouettes or edges. Various simplifications or optimisations [CTMS03] have been attempted, but none of them were able to make full use of image information in real-time. Building upon the volumetric reconstruction technique from Chapter 3 and the blobs framework from Chapter 4, we shall propose in Section 6.4 a fast evaluation method enabling real-time use of our particle filter.

6.2 Bayesian Framework and Particle Filter

Image evidence is noisy and unreliable. Belief about the current configuration of the model should therefore be built incrementally from all available observations, up to the current one. At each time step, the probability of the model given all observations follows a distribution called the *posterior*. The estimation of the distribution of the *posterior* is the purpose of Bayesian tracking. The tracked position is the global maximum of the distribution of the *posterior*.

6.2.1 Bayesian Framework

We start by defining the evolution of the model as the discrete sequence of configuration vectors $\{\mathbf{C}_t, t \in \mathbb{N}\}$ which consist of the joint angles, and the global positions and orientations of the root of the kinematic tree. In the case of human-body tracking, each state $\mathbf{C}_t$ is then a $d = 27$ dimensional vector (although we shall see in Section 6.3 that the configuration vector can be augmented with the first derivatives of the joint angles, reaching a total of 44 dimensions). The evolution of the state vector is modelled as a discrete process:

$$\mathbf{C}_t = f_t(\mathbf{C}_{t-1}, v_{t-1}) \tag{6.1}$$

where $f_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a non-linear function describing the evolution of the model at time t, and $\{v_t, t \in \mathbb{N}\}$ is the process noise. The model process function f_t is unknown, and impossible to observe directly. Being able to evaluate f_t would enable us to estimate the model state at each frame, consequently solving the tracking problem. The objective of tracking is to recursively estimate the state vector $\mathbf{C}_t$ from a series of measurements:

$$\mathbf{z}_t = h_t(\mathbf{C}_t, w_t) \tag{6.2}$$

where $h_t : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^{d_z}$ is the non-linear observation function, and $\{w_t, t \in \mathbb{N}\}$ is the measurement noise. We seek filtered estimates of $\mathbf{C}_t$ based on the set of all available measurements $\mathbf{Z}_t = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t\}$ up to time t. The sum of these estimates form the distribution of the *posterior*. To estimate the *posterior*, we use the "Bayes' rule". Considering two hypothesis A and B, the Bayes' rule relates their conditional probabilities as follows:

$$\left.\begin{array}{l} P(A|B).P(B) = P(A,B) \\ P(B|A).P(A) = P(A,B) \end{array}\right\} \Rightarrow P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{6.3}$$

Despite its simplicity, the Bayes' rule is a powerful tool which allows the estimation of a range of functions which are not directly evaluable. It is frequently the case that a conditional probability is easier to evaluate by transposing the problem. The probability $P(\mathbf{C}_t|\mathbf{Z}_t)$ of a model configuration $\mathbf{C}_t$ given all previous measurements $\mathbf{Z}_t$ is a typical example of function which is hard to evaluate directly. However, we can evaluate the probability of the current observation $\mathbf{z}_t$ given a model configuration: $P(\mathbf{z}_t|\mathbf{C}_t)$. This is called the *likelihood*. We therefore apply the Bayes' rule to transform the expression of the *posterior* into an evaluable formulation depending on the *likelihood*:

$$\begin{aligned} P(\mathbf{C}_t|\mathbf{Z}_t) &= \frac{P(\mathbf{Z}_t|\mathbf{C}_t).P(\mathbf{C}_t)}{P(\mathbf{Z}_t)} \\ &= \frac{P(\mathbf{z}_t, \mathbf{Z}_{t-1}|\mathbf{C}_t).P(\mathbf{C}_t)}{P(\mathbf{z}_t, \mathbf{Z}_{t-1})} \end{aligned} \tag{6.4}$$

The current observation $\mathbf{z}_t$ is not independent of the previous observations $\mathbf{Z}_{t-1}$, but using Equations 6.2 and 6.1, we can show that given a model configuration $\mathbf{C}_t$, all observations become functions of the Gaussian noise. We can therefore de-correlate the current observation from the previous ones:

$$P(\mathbf{C}_t|\mathbf{Z}_t) = \frac{P(\mathbf{z}_t|\mathbf{C}_t).P(\mathbf{Z}_{t-1}|\mathbf{C}_t).P(\mathbf{C}_t)}{P(\mathbf{z}_t|\mathbf{Z}_{t-1}).P(\mathbf{Z}_{t-1})} \tag{6.5}$$

We can again apply the Bayes' rule on the expression $P(\mathbf{Z}_{t-1}|\mathbf{C}_t)$:

$$\begin{aligned} P(\mathbf{C}_t|\mathbf{Z}_t) &= \frac{P(\mathbf{z}_t|\mathbf{C}_t).P(\mathbf{C}_t|\mathbf{Z}_{t-1}).P(\mathbf{Z}_{t-1}).P(\mathbf{C}_t)}{P(\mathbf{z}_t|\mathbf{Z}_{t-1}).P(\mathbf{Z}_{t-1}).P(\mathbf{C}_t)} \\ &= \frac{1}{P(\mathbf{z}_t|\mathbf{Z}_{t-1})}.P(\mathbf{z}_t|\mathbf{C}_t).P(\mathbf{C}_t|\mathbf{Z}_{t-1}) \end{aligned} \tag{6.6}$$

In order to exploit the incremental nature of the tracking process, we marginalise $P(\mathbf{C}_t|\mathbf{Z}_{t-1})$ over the previous tracked configurations of the model $\mathbf{C}_{t-1}$, leading to

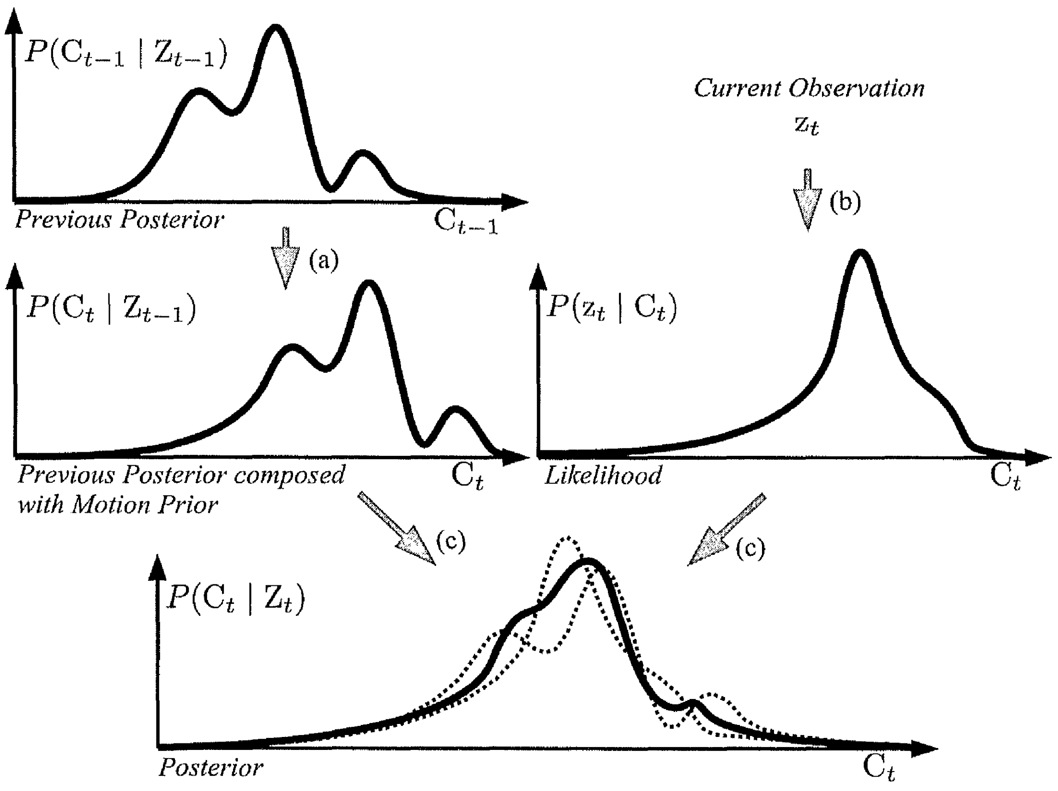


Figure 6.1: *Tracking in the Bayesian framework. The previous posterior is used as a basis to estimate the distribution of the posterior for the current frame. It is first composed with the motion prior (a), giving the expected distribution of the posterior. This expected distribution is finally evaluated against the likelihood from the observations (b) to produce the final estimation of the posterior (c).*

the formulation of the *posterior*:

$$\underbrace{P(\mathrm{C}_t|\mathrm{Z}_t)}_{Posterior} = \kappa.\underbrace{P(\mathrm{z}_t|\mathrm{C}_t)}_{Likelihood}. \int \underbrace{P(\mathrm{C}_t|\mathrm{C}_{t-1})}_{Motion\ Prior}.\underbrace{P(\mathrm{C}_{t-1}|\mathrm{Z}_{t-1})}_{Previous\ posterior} d\mathrm{C}_{t-1} \tag{6.7}$$

where $\kappa = \frac{1}{P(\mathrm{z}_t|\mathrm{Z}_{t-1})}$ is a normalising constant. At initialisation ($t = 0$) no observation is available, so that the initial distribution of the posterior is equal to the initial distribution of the state vector: $P(\mathrm{C}_0|\mathrm{Z}_0) \equiv P(\mathrm{C}_0)$, also known as the *prior*. The Bayesian tracking framework of Equation 6.7 is illustrated in Figure 6.1.

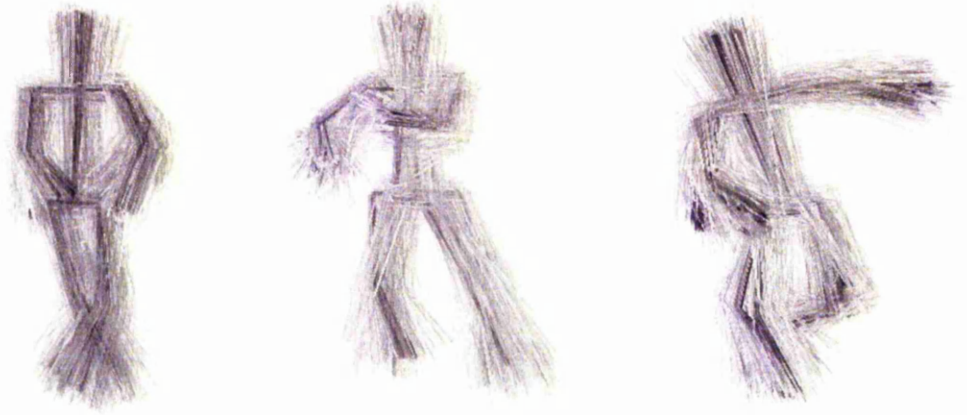


Figure 6.2: *Sets of* 1000 *particles locally populating the parameter space of the model configurations, and weighted so as to estimate the distribution of the posterior.*

6.2.2 Sequential Monte-Carlo Approach

Particle filters estimate the probability density function (pdf) of the *posterior* with discrete samples (*particles*). Each particle is a possible configuration $\mathbf{C}_t$ of the model in the parameter space. The general idea of the particle filter is then to "throw" an important number of these particles to populate the parameter space, and to estimate the *posterior* for each particle using the Bayesian framework described in the previous section. Figure 6.2 illustrates the particles (model configurations) populating the parameter space around the expected peaks of the *posterior*.

Particle filters also belong to the more general class of Monte Carlo methods, which all share a "random" component. Various names are used in the literature to denote particle filtering: it is also known as *Sequential Monte Carlo, bootstrap filtering, CONDENSATION* algorithm [IB98a, IB98b] and *survival of the fittest.*

The *posterior* distribution is approximated by a set of discrete weighted particles, each representing a body configuration in the parameter space. Let us denote as $\{\{\mathbf{C}_t^1, \mathbf{w}_t^1\}, \ldots, \{\mathbf{C}_t^{N_p}, \mathbf{w}_t^{N_p}\}\}$ the set of N_p weighted particles representing the probability density function of $P(\mathbf{C}_t|\mathbf{Z}_t)$. The weight are normalised such that $\sum_{i=1}^{N_p} \mathbf{w}_t^i = 1$. A set of particles is a *random measure* of the *posterior*, which can then be approximated as:

$$P(\mathbf{C}_t|\mathbf{Z}_t) \approx \sum_{i=1}^{N_p} \mathbf{w}_t^i.\delta(\mathbf{C}_t - \mathbf{C}_t^i) \qquad (6.8)$$

where $\delta()$ is the Dirac delta function. The approximation from Equation 6.8 approaches the true *posterior* as $N_p \to \infty$.

The Sampling Importance Resampling (SIR) particle filter follows the iterative formulation of the *posterior* from Equation 6.7. At the current timestep t, the particles are assumed to approximate the *previous posterior*. We propagate each particle using a dynamic model (*motion prior*) approximating the process function $f_t(\mathrm{C}_{t-1}, v_t)$ defined in Equation 6.1. In Section 6.3, we shall introduce a novel scheme to learn high level behaviours and approximate the true dynamic model. After propagation, the updated set of particles is a discrete approximation of the *previous posterior* combined with the *motion prior* (the right hand side of Equation 6.7).

In the next step, called "evaluation", the weights of the particles are re-assigned according to the *likelihood* of the observation $\mathbf{z}_t$ given the current particle. The evaluation of the *likelihood* function is critical for both robustness and performance: we shall present in Section 6.4 a novel evaluation method based on the 3-D blobs. After evaluation, the set of particles approximates the distribution of the *posterior*. The particles with highest weights can then be used to find the most likely configuration of the model (mode of the distribution).

The remaining step of the algorithm is called "resampling". It optimises the distribution of the particles, and is discussed in Section 6.2.3. These three steps, common to most particle filters, are illustrated in Algorithm 6.1.

6.2.3 Resampling

Resampling is designed to match the density of the distribution of the particles with the probability density function of the *posterior*. Practically, this is equivalent to eliminating the particles with small weights in order to concentrate on the particles with significant weights. The rationale behind resampling is that, since the total number of particles is limited, accuracy is mostly needed in regions where the expectation for true model configuration is high.

It has been shown [AMGC02] that without resampling, the variance of the weights can only increase over time. This *degeneracy problem* implies that only a few particles gain a high weight, while the vast majority contribute very little to the approximation of the *posterior*. This situation is obviously undesirable because computational resources are then wasted updating unimportant particles, while the peaks of the posterior are not accurately represented.

An estimate of degeneracy for a set of weighted particles is the effective sample

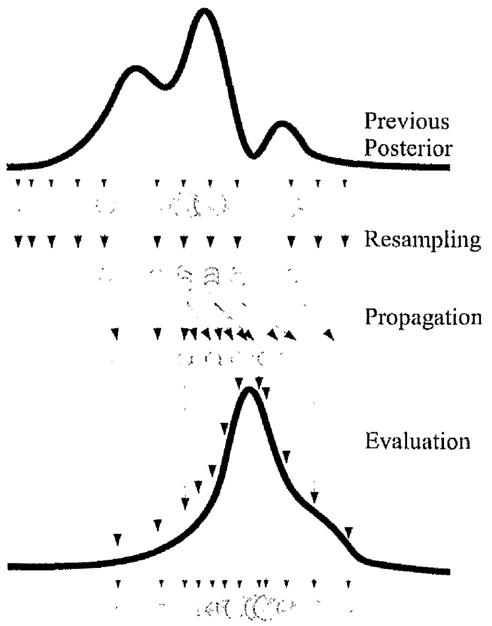
Algorithm 6.1: Generic SIR Particle Filter.

```
repeat
    ▷ Resampling (Section 6.2.3);
    for $i = 1$ to $N_p$ do
        Draw $C_t^i \sim P(C_{t-1}|Z_{t-1})$;
        Reset weights as $w_t^i \leftarrow \frac{1}{N_p}$;
    end

    ▷ Propagation using the motion prior (Section 6.3);

    ▷ Evaluation (Section 6.4);
    for $i = 1$ to $N_p$ do
        $w_t^i \leftarrow P(z_t|C_t^i)$;
    end
    for $i = 1$ to $N_p$ do
        $w_t^i \leftarrow \frac{w_t^i}{\sum_{k=1}^{N_p} w_t^k}$;
    end

    $t \leftarrow t + 1$;
until end tracking;
```



size $\widehat{N_{eff}}$, introduced by Liu *et al.* [LC98], and defined as:

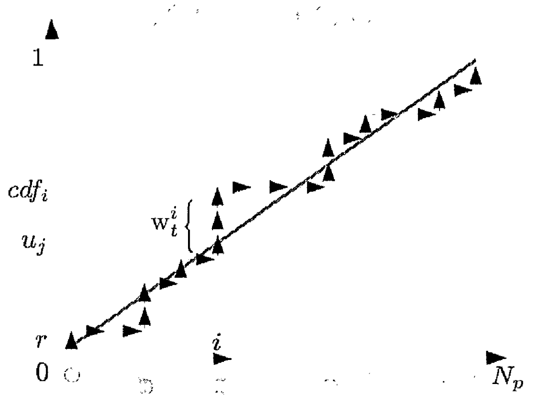
$$\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_p} (w_t^i)^2} \tag{6.9}$$

Small values of the effective sample size $\widehat{N_{eff}}$ indicate severe degeneracy, and in many implementations [AMGC02], trigger the resampling of the set of particles. However, the SIR implementation (Algorithm 6.1) resamples systematically the set of particles at each frame: the effective sample size is then unnecessary but can still be used as a performance indicator of the filter.

The resampling step involves generating a new set of particles, sampled from the approximate discrete approximation of the *posterior* (Equation 6.8). The *Systematic Resampling* [Kit96] algorithm was chosen because of its linear complexity in the number of particles: $O(N_p)$. The principle of Systematic Resampling is described and illustrated in Algorithm 6.2. The resulting set of particles is a random sample from the discrete approximation of the *posterior*, which explains that all weights are reset to $\frac{1}{N_p}$.

Algorithm 6.2: Systematic Resampling Algorithm.

```
$cdf_1 \leftarrow \mathrm{w}_t^1$;
for $i = 2$ to $N_p$ do
    $cdf_i \leftarrow cdf_{i-1} + \mathrm{w}_t^i$;
end
Draw $r \sim \mathrm{U}[0, N_p^{-1}]$;
$i \leftarrow 1$;
for $j = 1$ to $N_p$ do
    $u_j = r + \frac{j-1}{N_p}$;
    while $cdf_i < u_j$ do
        $i \leftarrow i + 1$;
    end
    $\{\mathrm{C}_t^j, \mathrm{w}_t^j\} \leftarrow \{\mathrm{C}_t^i, N_p^{-1}\}$;
end
```



A particularity of Systematic Resampling is that existing particles are used as supports to initialise the new ones: the particles that have high weights are therefore statistically selected and replicated many times in the new set. This phenomenon, called *sample impoverishment* characterises itself by a loss of diversity among the particles as the resultant sample will contain many repeated points. This problem is usually harmless when the process noise is strong enough, allowing particles to differentiate from each other during the propagation step. In presence of smaller noise however, alternative techniques such as *regularisation* [AMGC02] have to be used, with an impact on performance.

6.3 Propagation of the Particles

In this section, we introduce a novel method for propagating particles in an efficient way. The dynamics of the subject are learnt at two levels of granularity: local dynamics are encoded with second-order Gaussian models, while higher level behaviours are represented by a Variable Length Markov Model (VLMM).

6.3.1 Theory and Related Work

In the context of particle filtering, the role of the *motion prior* is to propagate the particles using a dynamic model, so that the distribution of the particles after the propagation step approximates the expected distribution of the *likelihood*. Ideally, the evaluation of the *likelihood* has then the sole role of correcting wrong predictions, which are bound to be frequent because of the necessary simplifications of the motion model. General-purpose prediction of human movements is very complex because of the high dimensionality of the parameter space combined with the highly non-linear nature of most motions. Unfortunately, the literature on motion prediction is too vast to be fully reviewed in this thesis. We shall therefore focus on some of the most pertinent contributions. The reader is invited to refer to some relatively recent surveys [Gav99, MG01] for further details.

In the CONDENSATION algorithm [IB98a], Isard *et al.* model the dynamics of the system with a single second order Auto-Regressive Process (ARP) in the full parameter space. When the number of particles is sufficient with respect to the size of the parameter space, enough particles are assumed to be propagated in the direction of the true motion. In practice, since the number of manageable particles is limited, problems start occurring when too few particles are propagated to represent peaks in the distribution of the *likelihood*. In the ICONDENSATION framework [IB98b], Isard *et al.* propagate the particles either using the same scheme as in the standard CONDENSATION algorithm, or guide them towards the most relevant regions of the parameter space using an auxiliary measurement obtained directly from the image data. More specifically, in [IB98b], Isard *et al.* use some colour blobs to track the hands and propagate a predefined proportion of the particles towards the corresponding image regions. This scheme compensates for the prediction errors of the dynamic model, but necessitates a reliable auxiliary measurement, which can become difficult to obtain in high dimensionality problems.

Annealing [DBR00] was introduced by Deutscher *et al.* as a coarse to fine approach that can help focus the particles on the global maxima of the *likelihood*, at the price of multiple iterations per frame. A smoothing function is used to "broaden" the peaks of the *likelihood*, hence increasing their chances of being initially represented by a sufficient number of particles. In subsequent iterations, the smoothing decreases simultaneously with the propagation noise, so that the particles are resampled nearer and nearer to the maxima of the *likelihood*. Due to the complexity of human dynamics, no dynamic model is learnt and the particles are propagated using solely the Gaussian process noise. The tracking process therefore relies mostly on the evaluation of the *likelihood.* Of course, as the dimensionality of the parameter space increases, more and more particles are needed to explore all possible directions, which can be very wasteful in computational resources. Compared with CONDENSATION, the extra cost generated by the multiple iterations per frame is compensated by the lower number of required particles. However, problems still occur when either the number of particles is too small, or when the movements are too fast, placing the new peaks of the *likelihood* out of reach by the random propagation.

In [DDR01], Deutscher *et al.* propose two new methods to help focus particles on the relevant parts of the parameter space. Hierarchical partitioning (also known as "scaled sampling") adjusts the propagation noise to the variance of the set of particles after resampling. The dimensions of the parameter space with the greatest influence on the evaluation of the *likelihood* have a lower variance than the less influential dimensions. For example, the global position of the kinematic model is more influential than the joint angle of a hand, so that its variance after the first evaluation and resampling should be lower. By iteratively adjusting the propagation noise to the variance of each dimension, the most influential dimensions are fitted first, making the algorithm behave like an automatic hierarchical method. Another extension is the crossover operator, which exchanges subset of parameters between high-weighted particles, borrowing the idea from genetic algorithms. The convergence rate using these techniques is reported in [DDR01] to be four times better than with standard annealing, but remains too high for real-time processing.

While the previously-mentioned schemes are general enough to track all types of motion, their robustness fully depends on the estimation of the *likelihood.* When observations become too noisy or ambiguous, these methods unavoidably fail, and have no means of recovery. Human-body tracking is a typical example of application where the observations are unreliable (self-occlusions, camera noise, motion blur, and so on), and more robust schemes must therefore be employed.

Prediction

Using a good predictive model, inconclusive observations can be recovered from. Particles are indeed propagated only in plausible directions, which means that even in the absence of strong evidence, the tracking can hold for a while by relying mostly on the *motion prior.* Outliers are also automatically discarded since no particles are propagated towards them. Another advantage of a good prediction scheme is that the limited number of particles is used much more efficiently than with random propagation: full body tracking becomes practical in real-time. Of course, since motion prediction is itself very complex and sometimes unreliable, wrong predictions are bound to happen, possibly resulting in a loss of tracking. When possible, recovery schemes should then be implemented.

Extending incrementally the CONDENSATION framework, Isard and Blake propose in [IB99] a mixed state dynamic model for tracking. A set of second order ARPs is learnt from manually segmented data, and the transitions between these dynamic models is performed at runtime according to a manually-set transition matrix (first order Markov model). Tests validate the benefits of the approach on simple dynamics composed of only 2 or 3 classes. In [BNI99], the same authors devise a scheme for learning automatically the transition matrix from the data. The number of dynamic models is still manually set and the models themselves are very constrained (only two free parameters in their example), but this method represents a first step towards the automatic learning of complex motions.

Using a Bayesian tracking framework, Sidenbladh *et al.* [SBF00] propose a variational model for simple activities such as walking and running. The parameters (joint angles and global position) are first projected into a lower dimensionality space using a standard Principal Component Analysis (PCA), and their distributions are modelled by a single Gaussian. The *motion prior* is therefore evaluated using this Gaussian model of variations between the previous frame and the current one. Despite the relative simplicity of the dynamic model, results suggest that the *motion prior* improves greatly the accuracy and robustness of the tracking. However, the number of required particles is still very high (10,000), and the dynamic model remains limited to very simple types of motion, such as the walking cycle.

Agarwal *et al.* [AT04] extend the work of Sidenbladh *et al.* [SBF00] by first clustering the parameter space using K-Means. Each cluster then represents a simpler activity which is easier to learn than trying to learn dynamics over the whole parameter space. This clustering allows the learning of complex activities, provided that elementary sub-activities can be identified. Inside each cluster, the dimensionality of the parameter space is reduced with a PCA, and dynamics are learnt using a second order Auto-Regressive Process. Transitions between clusters are based on the current configurations of the particle, the conditional probability of each cluster being modelled by a Gaussian distribution. In the absence of a higher level model of behaviour to guide the transitions between clusters, it is not clear how Agarwal's system copes with tracking failures. A limited evaluation is also suggesting the success of the method on short activities, but more challenging and diverse sequences remain untested.

Urtasun *et al.* [UF04] learn separately the dynamics of various walking and running rhythms under PCA projection. The variations (derivatives) of the joint angles are stored in a database, and retrieved during tracking. The switching between the dynamic models (for example, walking to running) seems to be implemented by testing a posteriori the likelihood of each model. The method also necessitates manual segmentation of various activities, which is impractical for complex sequences. In [UFF05], the same method is applied to the tracking of the golf swing, with more details on the least square optimisation framework that optimises the selection of the dynamic model along with the other model parameters. Unfortunately, the optimisation necessitates the computation of the gradients from the objective function, which is both costly and not always possible. Finally, in [UFHF05], the PCA is replaced by Scaled Gaussian Latent Variable Models (SGPLVMs) [Law04] which perform better for non-linear motions, and for small training sets.

In Style Machines [BH00], Brand and Hertzmann learn the general structure of the training data at the same time as the stylistic variations exhibited by individual subjects. To achieve this, specific models learning the training data of individual subject interact with a general model supposed to learn the common structure of all the sequences. Both specific and general models are Hidden Markov Models with Gaussian emission probabilities. During learning, the interactions between the specific models and the general one are formulated as a sum of entropies, including a cross-entropy term. The full objective function is designed to maximise overlapping between the specific models and the general one, while keeping the specific models specialised and the general model simple. A modified Expectation-Maximisation loop is used to fit the Gaussian models on the joint angles data, previously summarised by a PCA. Once both the general and the specific HMMs are learnt, the stylistic variations are recovered as the parameters allowing the transition between specific Gaussian emission models inside the general one. In practice, under PCA, only a few parameters are sufficient to explain the variability between specific models. While they were mainly designed for retargeting applications, stylistic models can be an interesting way to learn accurate *motion priors* once the stylistic parameters for the tracked subject have been discovered. For prediction, it is possible to follow a smooth path between Gaussian distributions, as detailed in [Bra99].

In [SBS02b], Sidenbladh *et al.* take a different approach to particles propagation:

instead of trying to learn a predictive model, they build a database of model configurations from a large amount of training data. An efficient search algorithm, based on a PCA and a binary tree, allows the lookup of a particular pose. More interestingly, a "heat" coefficient controls a probabilistic perturbation of the search algorithm, so that similar body configurations can be sampled from. Using a standard particle filtering framework, the propagation of the particles is therefore strongly constrained by the poses included in the database. When the target motions are very similar to the training examples included in the database, this scheme has the potential to be very robust and efficient. However, this data-based approach does not readily generalise to new motions, and as the size of the training data grows, storage can become an issue.

Other proposed methods include the unscented particle filter [vdMDdFW00] which utilises the predictions of an Extended Kalman filter. The dynamic model is then learnt online, with no need for training data. The accuracy of this method is however questionable, especially since the Kalman cannot capture the non-linear variations of the parameters. Another propagation scheme proposed by Choo *et al.* [CF01] utilises the gradient of the *likelihood* to guide the particles. The method assumes that the gradient can be computed, and also takes the risk of attracting particles towards local maxima.

Many other predictive methods are borrowed from the field of Artificial Intelligence, with a strong emphasis on movement classification and recognition. For example, Hong *et al.* [HTH00] use finite state machines to classify gestures. Brand *et al.* [BOP97] introduce coupled hidden Markov models to model interactions between limbs.

6.3.2 Learning Dynamics

We now describe the learning of our dynamic model. The parameter space is clustered into Gaussian states, over which high-level behaviour is modelled by a variable length Markov model.

Description and Pre-Processing of the Training Data

The training data consists of the joint angles $\Theta = \{\theta_1, \ldots, \theta_{19}\}$ of the kinematic model augmented by their first derivatives $\dot{\Theta} = \{\dot{\theta}_1, \ldots, \dot{\theta}_{19}\}$, amounting to 38 dimensions. Note that the degrees of freedom of the head were not included in the model because the corresponding training data was unavailable. The global position P_0 and orientation R_0 of the kinematic model are also not included in the training data to keep the dynamics invariant to the placement of the subject. For convenience, let us denote as $\Theta = \{\Theta, \dot{\Theta}\}$ the feature vector composed of the joint angles and their derivatives. The full configuration of a kinematic model at time t is therefore described by the configuration vector $\mathbf{C}_t = \{P_{0t}, R_{0t}, \Theta_t\}$ with a total of 44 dimensions.

A first part of our training data is acquired from a Vicon tracking system, which returns accurate and smooth estimates of the 3-D positions of the body parts. The rest of the training data is obtained from manually annotated video sequences, producing sparse and noisy 3-D positions for the body parts. In both cases, the values of the joint angles are recovered using inverse kinematics (Section 5.3). The joint angles corresponding to manually annotated sequences are then smoothed using a Gaussian kernel $\mathcal{N}(0,1)$, and the data is completed by interpolating between annotated positions with Cubic Splines.

The first derivatives (velocity) of the joint angles are computed in a discrete manner as $\dot{\Theta}_t = \Theta_t - \Theta_{t-r}$, where r is the number of interpolated configurations between two keyframes taken at the original framerate. A Gaussian noise (with a variance of typically $1/100$ of the total variance of each dimension) is finally added to all the parameters to ensure a good generality of the learnt model and avoid overfitting.

Clustering the Parameter Space

Due to the complexity of human dynamics, we break down complex behaviours into elementary movements for which local dynamic models are easier to infer. The problem is then to automatically find, isolate and model these elementary movements from the training data. We achieve this by clustering the feature space into Gaussian clusters using the EM algorithm proposed by Figueiredo and Jain [FJ02]. Their proposed method automatically addresses the main pitfalls of traditional EM, that is, the delicate initialisation, the arbitrary choice of the number of components, and the possibility of singularities. Body configurations sampled from a few clusters on ballet-dancing data are shown in Figure 6.3.

Learning Transitions between Clusters with a VLMM

Complex human activities such as dancing (or even simpler ones such as walking), can be viewed as a sequence of primitive movements with a high level structure controlling the temporal ordering. A suitable way to obtain probabilistic knowledge of the underlying behavioural structure is variable length Markov models (VLMMs) [RST96].

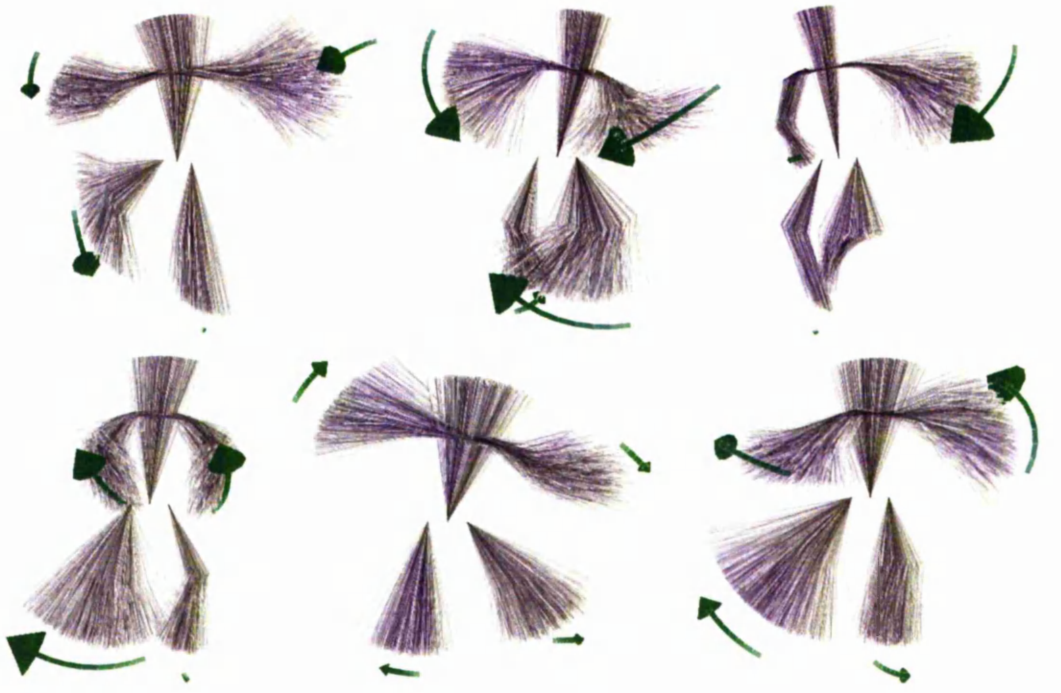


Figure 6.3: *Model configurations sampled from various Gaussian clusters. The mean values of the derivatives are represented at each end effectors by a green arrow with a size proportional to the absolute value of the derivatives.*

Variable length Markov models deal with a class of random processes in which the memory length varies, in contrast to n-th order Markov models. They have been previously used in the data compression [CH87] and language modelling domains [RST96, GP95]. More recently, they have been successfully introduced in the computer vision domain for learning stochastic models of human activities with applications to behaviour recognition and behaviour synthesis [GJH99a, GJH99b, GJH01, GCMH02]. Their advantage over a fixed memory Markov model is their ability to locally optimise the length of memory required for prediction. This results in a more flexible and efficient representation which is particularly attractive in cases where we need to capture higher-order temporal dependencies in some parts of the behaviour and lower-order dependencies elsewhere. A detailed description on building and training variable-length Markov models is given by Ron *et al.* [RST96].

A VLMM can be thought of as a probabilistic finite state automaton (PFSA) $\mathcal{M} = (Q, \mathcal{K}, \tau, \gamma, s)$, where $\mathcal{K}$ is a set of tokens that represent the finite alphabet of the VLMM, and Q is a finite set of model states. Each state corresponds to a string in

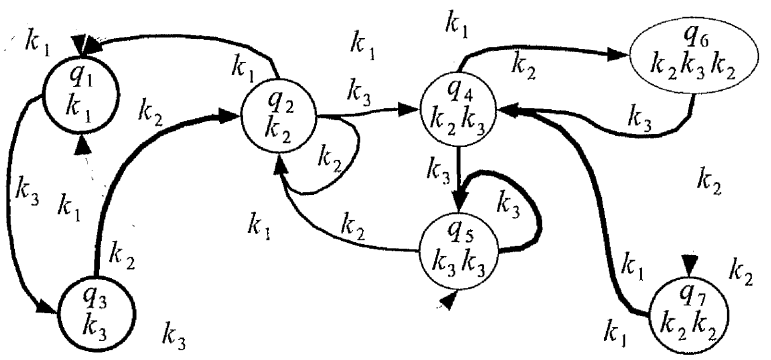


Figure 6.4: *Synthetic VLMM with alphabet $\mathcal{K} = \{k_1, k_2, k_3\}$, and 7 states. The thickness of the arcs represent the output probabilities γ, and the thickness of the state ellipses stands for the probability distribution s over the start state.*

$\mathcal{K}$ of length at most $N_{\mathcal{M}}$ ($N_{\mathcal{M}} \geq 0$), representing the memory for a conditional transition of the VLMM. The transition function τ, the output probability function γ for a particular state, and the probability distribution s over the start states are defined as:

$$\begin{aligned} \tau &: Q \times \mathcal{K} \to Q \\ \gamma &: Q \times \mathcal{K} \to [0,1], \qquad \forall q \in Q, \sum_{k \in \Sigma} \gamma(q,k) = 1 \\ s &: Q \to [0,1], \qquad \sum_{q \in Q} s(q) = 1 \end{aligned} \tag{6.10}$$

The VLMM is a generative probabilistic model: by traversing the model's automaton $\mathcal{M}$ we can generate sequences of the tokens in $\mathcal{K}$. By using the set of Gaussian clusters as the alphabet, we can capture the temporal ordering and space constraints associated with the primitive movements. Consequently, traversing $\mathcal{M}$ will generate statistically plausible examples of the behaviour.

6.3.3 Predicting Using the VLMM

In this section, we describe the way particles are propagated (*motion prior*). The VLMM guides the transitions between clusters of elementary movements. For each particle, the joint angles are propagated with local dynamics encoded by the current Gaussian cluster, while the global parameters are propagated stochastically.

Transitions Between Clusters with the VLMM

The particles are augmented with their current VLMM state q_t, from which the cluster k_t to which they belong may be easily deduced. Transitions (or jumps) between clusters are conditional on the particle's feature vector Θ_t as well as the transition probabilities $\gamma(\cdot,\cdot)$ in the VLMM. The probability of transition towards a new Gaussian cluster k_{t+1} of mean $\mu_{k_{t+1}}$ and covariance $\Sigma_{k_{t+1}}$ is derived using the Bayes' rule:

$$\begin{aligned}
P(k_{t+1} \mid \Theta_t, q_t) &= \frac{P(\Theta_t|k_{t+1}).P(q_t|k_{t+1}).P(k_{t+1})}{P(\Theta_t).P(q_t)} \\
&= \frac{P(\Theta_t|k_{t+1}).P(k_{t+1}|q_t).P(q_t).P(k_{t+1})}{P(\Theta_t).P(q_t).P(k_{t+1})} \\
&\propto P(\Theta_t \mid k_{t+1}).P(k_{t+1} \mid q_t) \\
&= \frac{1}{\sqrt{(2\pi)^d \left|\Sigma_{k_{t+1}}\right|}}.e^{-\frac{1}{2}.(\Theta_t-\mu_{k_{t+1}})^T.\Sigma_{k_{t+1}}^{-1}.(\Theta_t-\mu_{k_{t+1}})}.\gamma(q_t, k_{t+1})
\end{aligned} \tag{6.11}$$

At each frame, the state transition is chosen according to the above probabilities for each neighbouring cluster. In practice, only a few transitions are encoded in the VLMM, making the evaluation efficient. If the same cluster is chosen ($k_{t+1} = k_t$), the particle is propagated using local dynamics, as formulated in the next section. If a new cluster is selected, the particle's parameters are re-sampled from the new Gaussian cluster.

Local Dynamics Inside Each Cluster

Inside each Gaussian cluster, a new model configuration can be stochastically predicted from the previous feature vector Θ_{t-1}. Since the Gaussian clusters include derivatives, the prediction effectively behaves like a second-order model. Let us consider a Gaussian cluster of mean $\mu_\Theta = \left(\begin{smallmatrix}\mu_\Theta \\ \mu_{\dot\Theta}\end{smallmatrix}\right)$ and covariance matrix $\Sigma_\Theta = \begin{pmatrix}\Sigma_{\Theta\Theta} & \Sigma_{\Theta\dot\Theta} \\ \Sigma_{\Theta\dot\Theta}^T & \Sigma_{\dot\Theta\dot\Theta}\end{pmatrix}$. The noise vector is directly sampled from the cluster's covariance matrix with an attenuation coefficient λ, leading to the formulation:

$$\begin{aligned}
\dot\Theta_t &= \dot\Theta_{t-1} + \lambda.d\dot\Theta_t \\
\Theta_t &= \Theta_{t-1} + \Theta_t + \lambda.d\Theta_t
\end{aligned} \qquad \text{with} \qquad \begin{pmatrix} d\Theta_t \\ d\dot\Theta_t \end{pmatrix} \sim \mathcal{N}(0, \Sigma_\Theta) \tag{6.12}$$

The random noise vector is drawn as $\left(\,d\Theta_t\ d\dot\Theta_t\,\right)^T = \sqrt{\Sigma_\Theta} \cdot X$ with $X \sim \mathcal{N}(0, I)$. The square-root of the covariance matrix is computed by performing the eigenvalue

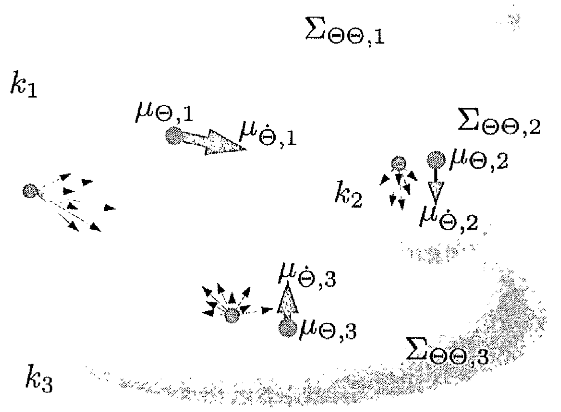


Figure 6.5: *Probabilistic propagation of the particles using local dynamics from the Gaussian clusters $\{k_1, k_2, k_3\}$. The ellipsoidal shape of the clusters represents the covariances $\Sigma_{\Theta\Theta}$ of the joint angles. The mean values of the derivatives of the joint angles $\mu_{\dot{\Theta}}$ are represented by green arrows.*

decomposition, $\Sigma_\Theta = V \cdot D \cdot V^T$, and taking the square root of the eigenvalues on the diagonal of D, so that $\sqrt{\Sigma_\Theta} = V \cdot \sqrt{D} \cdot V^T$.

This predictive model has to be understood in the context of Monte-Carlo sampling, where noise is introduced to model uncertainty in the prediction: the properties of the noise vector are therefore almost as important as the dynamics themselves. The covariance matrix of the current cluster provides a good approximation of this uncertainty, and sampling the noise vector from the cluster itself makes propagation of uncertainty much closer to the training data than uniform Gaussian noise.

Random Propagation of the Global Parameters

The six parameters describing the global position P_0 and orientation R_0 of the model are not included in the dynamic model. They are therefore propagated with Gaussian noise, in a similar way to the CONDENSATION algorithm [IB98a]. The amplitude of the Gaussian noise has to be sufficient to follow fast motions. We therefore define some scaling coefficients ρ_g and λ_g, respectively for the global position and the global orientation of the kinematic tree. These coefficients depend on the type of motion and the framerate of the cameras capture. In our experiments, we set $\rho_g = 60$ millimetres and $\lambda_g = 0.1$ radians. The global parameters are stochastically propagated according to:

$$\begin{aligned} P_{0t} &= P_{0t-1} + \rho_g.X \\ R_{0t} &= R_{0t-1} + \lambda_g.X \end{aligned} \quad \text{with} \quad X \sim \mathcal{N}(0, I_3) \tag{6.13}$$

6.4 Fast Evaluation of the Likelihood

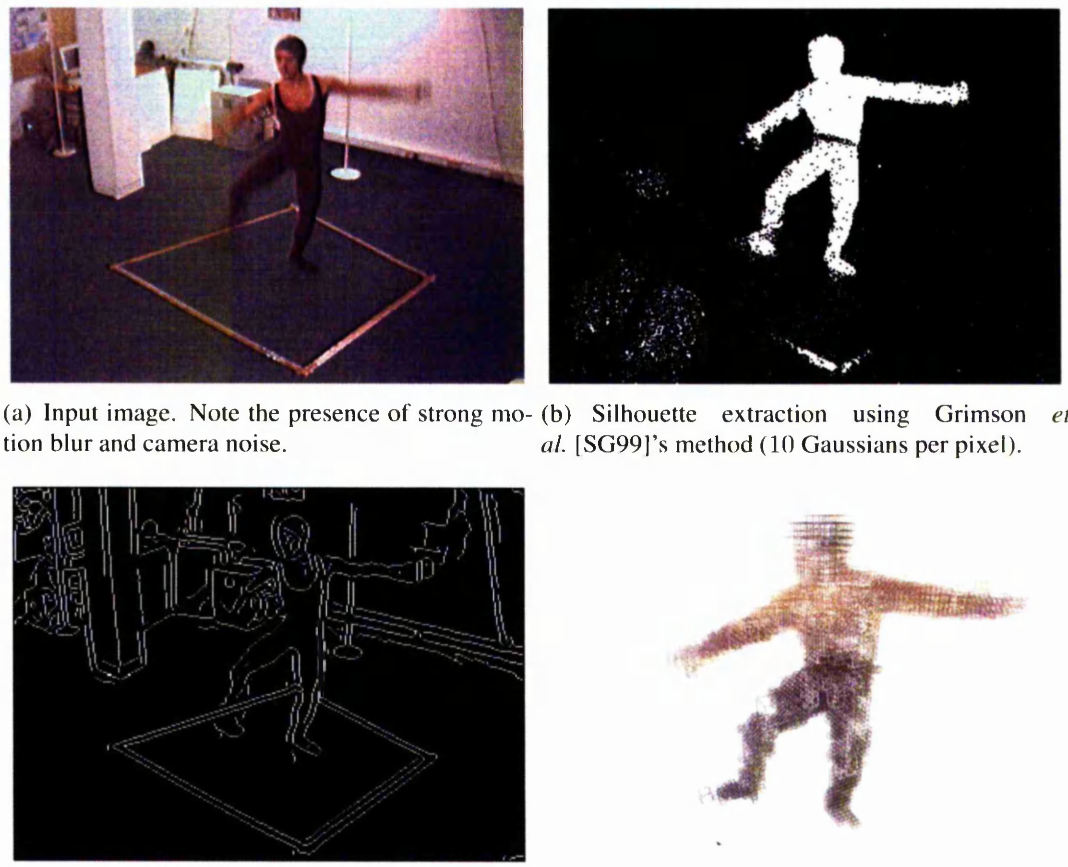
In this section we present a fast evaluation technique for the particles. Because of its good compactness, the 3-D blobs representation introduced in Chapter 4 provides an ideal basis for the evaluation of the likelihood function. We shall see how the blob description can be integrated into the particle filtering framework, alleviating most of the pitfalls of the hierarchical tracking.

6.4.1 Introduction

The distribution of the *likelihood* is approximated at timestep t by a set of N_p weighted particles $\{\{\mathbf{C}_t^1, \mathbf{w}_t^1\}, \ldots, \{\mathbf{C}_t^{N_p}, \mathbf{w}_t^{N_p}\}\}$. The evaluation stage consists in weighting each particle proportionally to the probability of the current measurement $\mathbf{z}_t$ given the model configuration encoded by the particle: $\forall i \in [1 \ldots N_p], \mathbf{w}_t^i \overset{\propto}{\leftarrow} P(\mathbf{z}_t \mid \mathbf{C}_t^i)$.

To estimate the probability of the observations given a model configuration, an appearance model is first generated from the model parameters, and is subsequently evaluated against the observations. The goal of appearance models is to encode in a compact way all the pertinent information about the subject, that can be extracted from the current observation. The evaluation of the appearance model therefore occurs at a "middle ground" between raw observations (image inputs) and the parameter space of the model. The choice of this "middle ground", or *feature space*, is critical for the overall performance of the system because an appearance model must be generated and evaluated for each particle at each frame.

A variety of features are used in the literature to characterise the configuration of the subject. Silhouettes, extracted using background segmentation techniques (see Chapter 2), are very popular [MH03, CTMS03, DBR00, DDR01] because of their simplicity and good overall robustness. The appearance model is often projected onto all available image planes, and the number of matching pixels defines the objective function. More or less complex appearance models, ranging from a set of cylinders to deformable mesh models, have been used for evaluation: the reader is invited to refer to Section 4.1.1 for a short review of appearance models. The intrinsic limitations of silhouettes include the loss of internal features, the absence of colour and the need for a static background. Moreover, in presence of strong camera noise, silhouette extraction can generate outliers, thus requiring costly post-processing. Figure 6.6(b) demonstrates the overall robustness of silhouette extraction, even in extreme conditions, but also exhibits the numerous outliers which are a consequence of processing



(a) Input image. Note the presence of strong motion blur and camera noise.

(b) Silhouette extraction using Grimson *et al.* [SG99]'s method (10 Gaussians per pixel).

(c) Canny [Can86] edge detection.

(d) Volumetric reconstruction using all 5 available camera views.

Figure 6.6: *Comparison of image-based features for particles evaluation.*

each pixel independently.

Edges can also be exploited, with, for example, the distance to the closest edge pixel as objective function. The result of applying the Canny [Can86] edge detection algorithm is shown in Figure 6.6(c). Even if, once again, the results are rather good considering the challenging input image, some important features such as the right hand are missing because of the motion blur. The cluttered background is also problematic as it can distract the objective function from the subject, and lead to a loss of tracking.

When multiple camera views are available, evaluating model configurations based on a volumetric reconstruction becomes an interesting option. In recent years, the visual hull has been exploited [TMSS02, CBK03a, CBHK04, MTHC03, KBG05] to define various objective functions for human-body tracking. However, to our knowledge, volumetric reconstructions have not yet been used in the context of discrete Bayesian tracking. The advantages of the 3-D reconstruction over other image cues have already been advocated throughout this thesis: let us just insist on the increased robustness, the good compactness of the data, and the inclusion of colour. The result of our 3-D reconstruction method (Chapter 3) using all 5 available camera views is shown in Figure 6.6(d).

Various other image cues have been used in the literature. For example, Stefanov *et al.* [SGH05] perform a Hough transform on the silhouettes of the hands to detect the round shape of the fingertips. The particles are then evaluated in the Hough space instead of the image space. Let us finally mention the widely used *boosting* [MR03] framework, which combines the outputs of a collection of specialised detectors to producing a more robust classifier.

The full literature on feature detection extends far beyond the scope of this thesis. However, the area of human-body tracking is relatively conservative regarding image features. Silhouettes, edges, and occasionally disparity maps are indeed the basis of the vast majority of trackers. This short overview, although not exhaustive, is therefore representative of the common current evaluation schemes used for human-body tracking.

The next sections describe how the volumetric reconstruction from Chapter 3 can be used as a basis for the efficient evaluation of the *likelihood*. We start in Section 6.4.2 by introducing a direct voxel-based evaluation scheme, and improve its efficiency in Sections 6.4.3 and 6.4.4 by exploiting the blob-fitting procedure from Chapter 4 into a fast blob-to-blob evaluation scheme.

6.4.2 Direct Voxel-Based Particle Evaluation

The appearance model associated with a given configuration $\mathbf{C}_t$ of the model is a set of blobs $\mathcal{B} = \{B_1, \dots, B_{N_b}\}$. As we saw in Section 4.3, the attributes of the blobs are automatically acquired from the data, and new sets of blobs $\mathcal{B}$ can be readily generated for any model configuration $\mathbf{C}_t$. In other words, the configuration vector $\mathbf{C}_t$ conditions fully the set of blobs $\mathcal{B}$. The evaluation of the *likelihood* distribution at a given configuration vector $\mathbf{C}_t$ is as follows:

$$\begin{aligned} P(\mathbf{z}_t \mid \mathbf{C}_t) &\propto P(\text{Voxels} \mid \mathcal{B}) \\ &= \prod_{\mathcal{V}_i \in \text{Voxels}} s_{\mathcal{V}_i}^3 . P(\mathcal{V}_i \mid \mathcal{B}) \\ &= \prod_{\mathcal{V}_i \in \text{Voxels}} s_{\mathcal{V}_i}^3 . \sum_{j=1}^{N_b} P(\mathcal{V}_i \mid B_j) \end{aligned} \tag{6.14}$$

Using the formulation of $P(\mathcal{V}_i \mid B_j)$ from Equation 4.6 (Section 4.2.3), and taking the logarithm of the expression gives:

$$\log P(\mathbf{z}_t \mid \mathbf{C}_t) \propto \sum_{\mathcal{V}_i \in \text{Voxels}} s_{\mathcal{V}_i}^3 . \log \left(\sum_{j=1}^{N_b} \frac{1}{(2.\pi)^3 . \sqrt{|\Sigma_j|}} . e^{-\frac{1}{2} D_M(\mathcal{V}_i, B_j)} \right) \tag{6.15}$$

This expression can be further simplified by exploiting the fact that blobs are not overlapping. The likelihood of each voxel is then computed exclusively with respect to the most probable Gaussian blob:

$$\log P(\mathbf{z}_t \mid \mathbf{C}_t) \overset{\propto}{\simeq} \sum_{\mathcal{V}_i \in \text{Voxels}} s_{\mathcal{V}_i}^3 . \max_{j=1..N_b} \left(-\log |\Sigma_j| - D_M(\mathcal{V}_i, B_j) \right) \tag{6.16}$$

Even with a low number of blobs, and exploiting the hierarchical nature of the voxel space, it is easy to see how this formulation remains far too computationally expensive for the online evaluation of a full set of particles. One must keep in mind that this scheme is nonetheless much more efficient than most image-based evaluation methods, that would require testing all the pixels for all the camera views. In the next sections, we propose a far more efficient evaluation scheme exploiting the blob fitting procedure from Chapter 4.

6.4.3 Data Density as a Mixture of Gaussians

In Section 4.2, we showed how to fit a mixture of Gaussian blobs onto the 3-D voxels in real-time using an EM-like procedure. Provided that this blob-fitting procedure is reliable enough, the resulting set of blobs constitutes an ideal basis for efficient evaluation of particles. Each set of blobs can be assimilated to a mixture of Gaussians, for which we can derive an efficient measure of similitude based on the cross-entropy.

As with every EM-based algorithm, the reliability of blob-fitting strongly depends

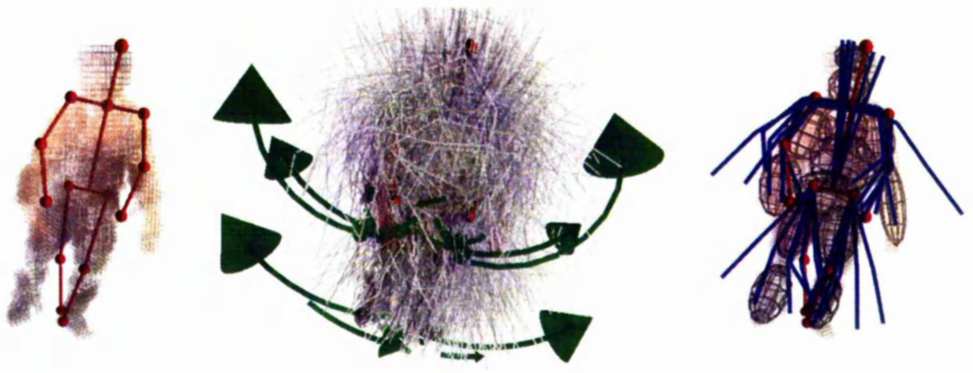


Figure 6.7: *Blobs fitting process. (left) The reconstructed volume for the current frame, and the mode of the posterior corresponding to the last tracked position. (middle) The particles approximating the posterior are propagated using local dynamics and the VLMM. (right) The blob-fitting is performed from the centres of the clusters with high support (displayed in blue), and the maximum-likelihood set of blobs is retained.*

on initialisation. The number of blobs and their attributes are known from the appearance model, but their actual positions depend on the pose of the underlying kinematic model. Initialising EM from the tracked position in the last frame can prove insufficient for fast movements. Fortunately, the VLMM can predict the next possible clusters by traversing the automaton from the last tracked positions. EM is then performed from the means of these clusters, and the maximum-likelihood result is retained.

A strength of the particle filter is the ability to track multiple hypotheses at once. Different states in the VLMM can therefore be represented by meaningful numbers of particles. In order to avoid biasing the evaluation process, EM should be performed from every predicted maximum of the *posterior*. To achieve this, we count the number of particles corresponding to each cluster after resampling and prediction. If this number is greater that a given ratio of the total number of particles, then the corresponding cluster maps to significant values of the predicted *posterior*, and EM must be performed from the centre of the cluster (illustration in Figure 6.7). In practice, we take the threshold ratio to be $1/10$ of the total number of particles, which allows simultaneous tracking of multiple hypothesis and prevents unnecessary computations from clusters with low support.

The likelihood of each set of blobs obtained after performing EM from the clusters with high support is evaluated using Equation 6.16. Even though this formulation is too expensive for real-time evaluation of all particles, its performance is sufficient to evaluate the small number of candidates resulting from EM. Let us denote as $\widehat{\mathcal{B}} =$

$\{\widehat{B_1} \ldots, \widehat{B_{N_b}}\}$ the set of blobs with the greatest likelihood.

This blobs-fitting procedure has the important advantage of detecting tracking failures: if the best mixture $\widehat{\mathcal{B}}$ has a low likelihood, the tracker is lost and needs re-initialisation. Unlike most other trackers, automatic recovery from failures is then possible because the parameter space is clustered in motion prototypes. Performing EM from all clusters might provoke a noticeable lag, depending on the total number of prototypes, but is bound to return a good result. The VLMM state of all particles is then reset, which has the effect of spreading them across the clusters. To ensure a quick recovery, a bias towards the clusters that returned the best mixtures is introduced for the first state transition.

6.4.4 Fast Particle Evaluation as Cross-Entropy Measure

A model configuration (particle) is evaluated by first generating an appearance model from the particle state, and then comparing the produced set of blobs with the maximum likelihood set of blobs $\widehat{\mathcal{B}}$ obtained after EM. Figure 6.8 illustrates the evaluation framework. Let us denote as $\mathcal{B} = \{B_1, \ldots, B_{N_b}\}$ the set of blobs generated from a given particle of configuration $\mathbf{C}_t$ (the generation of the appearance model is performed according to Section 4.3.1). In the following, we shall assimilate the sets of blobs $\widehat{\mathcal{B}}$ and $\mathcal{B}$ to mixtures of Gaussians, which assumes equal priors for each Gaussian blob:

$$\forall x \in \mathbb{R}^6 \qquad P(x|\widehat{\mathcal{B}}) = \sum_{i=1}^{N_b} \frac{1}{N_b} P(x|\widehat{B_i}) \qquad P(x|\mathcal{B}) = \sum_{i=1}^{N_b} \frac{1}{N_b} P(x|B_i) \tag{6.17}$$

The cross-entropy between two statistical distributions reflects the "energy" that is required to transform one distribution into the other. By contrast to some simpler metrics (comparing only the means and the variances between two distributions), the cross-entropy compares all the moments of the statistical distributions simultaneously. The distance corresponding to the measure of cross-entropy is often referred to as the Kullback-Leibler (KL) divergence, and defined as:

$$D_{KL}(\mathcal{B}\|\widehat{\mathcal{B}}) = \int_{\mathbb{R}^6} P(x|\mathcal{B}) \ln \frac{P(x|\mathcal{B})}{P(x|\widehat{\mathcal{B}})} dx \tag{6.18}$$

where $d = 6$ is the dimensionality of the Gaussian blobs. For a mixture of Gaussians,

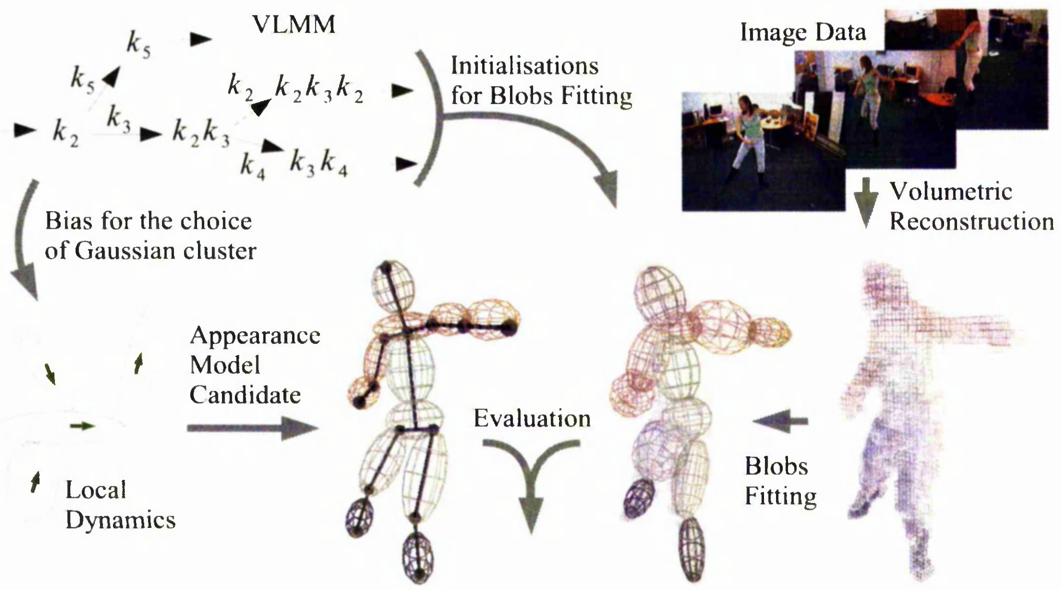

VLMM
Initialisations for Blobs Fitting
Image Data
Volumetric Reconstruction
Bias for the choice of Gaussian cluster
Appearance Model Candidate
Evaluation
Blobs Fitting
Local Dynamics

Figure 6.8: *Overview of the particle evaluation framework. The visual hull is first reconstructed (a) from the input images. The maximum-likelihood set of blobs $\widehat{\mathcal{B}}$ is then found (b) by performing EM from the centres of the clusters with high support (c). The particles are evaluated by generating a blob-model (d) from their configuration vector, and measuring the cross-entropy (e) between the two sets of blobs.*

this formulation can be expanded as:

$$D_{KL}(\mathcal{B}\|\widehat{\mathcal{B}}) = \sum_{i=1}^{N_b} \frac{1}{N_b} \int_{\mathbb{R}^6} P(x|B_i) \ln P(x|\mathcal{B}) dx - \sum_{i=1}^{N_b} \frac{1}{N_b} \int_{\mathbb{R}^6} P(x|B_i) \ln P(x|\widehat{\mathcal{B}}) dx$$

This formulation is simplified by exploiting the fact that the blobs are well separated (minimal overlapping). The mixture $\mathcal{B}$ was generated including non-overlapping constraints, and $\widehat{\mathcal{B}}$ was computed with binary support maps, limiting overlapping during EM. Using the approximation proposed by [GGG03], we approximate the sum over the mixture $\widehat{\mathcal{B}}$ by the contribution of the closest blob only:

$$\begin{aligned} D_{KL}(\mathcal{B}\|\widehat{\mathcal{B}}) &\simeq \frac{1}{N_b}\left(\sum_{i=1}^{N_b}\int_{\mathbb{R}^6} P(x|B_i)\ln\frac{1}{N_b}P(x|B_i)dx \right. \\ &\left. \qquad - \sum_{i=1}^{N_b}\max_{j\in[1..N_b]}\int_{\mathbb{R}^6} P(x|B_i)\ln\frac{1}{N_b}P(x|\widehat{B}_j)dx\right) \\ &= \frac{1}{N_b}\sum_{i=1}^{N_b}\min_{j\in[1..N_b]} D_{KL}(B_i\|\widehat{B}_j) \end{aligned}$$

Moreover, correspondence between blobs is maintained under the form $B_i \leftrightarrow \widehat{B}_{\pi(i)}$, so that the complexity of the run-time evaluation function becomes linear with respect to the number of blobs:

$$D_{KL}(\mathcal{B}\|\widehat{\mathcal{B}}) \simeq \frac{1}{N_b}\sum_{i=1}^{N_b} D_{KL}(B_i\|\widehat{B}_{\pi(i)}) \tag{6.19}$$

This last formulation can be efficiently computed using the closed form solution of the KL divergence between two Gaussian blobs $B \sim \mathcal{N}(\boldsymbol{\mu}_B, \Sigma_B)$ and $\widehat{B} \sim \mathcal{N}(\boldsymbol{\mu}_{\widehat{B}}, \Sigma_{\widehat{B}})$:

$$\begin{aligned} D_{KL}(B\|\widehat{B}) &= \int_{\mathbb{R}^6} P(x \mid B)\ln\frac{P(x \mid B)}{P(x \mid \widehat{B})}dx \\ &= \frac{1}{2}\left(\ln\frac{|\Sigma_B|}{|\Sigma_{\widehat{B}}|} - 6 + tr(\Sigma_B^{-1}\Sigma_{\widehat{B}}) + (\boldsymbol{\mu}_{\widehat{B}} - \boldsymbol{\mu}_B)^T\Sigma_B^{-1}(\boldsymbol{\mu}_{\widehat{B}} - \boldsymbol{\mu}_B)\right) \end{aligned} \tag{6.20}$$

The weighting of a particle is proportional to the inverse of the relative entropy $D_{KL}(\mathcal{B}\|\widehat{\mathcal{B}})$ between the particle and the set of blobs corresponding to image evidence. The proportionality factor is unimportant since the weights are re-normalised before resampling.

6.5 Discussion and Conclusion

This chapter started with a general introduction to Bayesian tracking and Monte-Carlo approaches, where the need of a good prediction scheme for both performance and robustness was highlighted. We then introduced a novel prediction scheme, based on the automatic decomposition of the parameter space into clusters of elementary movements, and the learning of the transitions between these clusters with a VLMM. The second part of the chapter focused on the evaluation of the likelihood. We presented a fast evaluation technique based on the relative entropy between sets of blobs. An extensive evaluation of this tracking framework will be the topic of Chapter 7.

The extension of the dynamic model to include the first derivatives of the global parameters is the subject of future work. We feel that this relatively straightforward extension could improve greatly the accuracy of the propagation scheme. Another potentially important extension could be the learning of stylistic variations between subjects, as proposed by Brand [BH00]. Predictions could then become more accurate for a specific subject, while extrapolation to unseen styles would be possible.

Another worthwhile research topic would be the projection of the parameter space onto a lower dimensionality manifold, where dynamics could be easier to learn and correlations between the parameters implicitly encoded. Recent developments in this area, based on Scaled Gaussian Process Latent Variable Models [Law04, GMHP04, UFHF05], suggest the clear benefits of the method, particularly when a limited amount of training data is available. To date, SGPLVMs have still not been used in the context of Monte-Carlo Bayesian tracking.

Chapter 7

Overall Evaluation

This chapter presents the evaluation of our Bayesian tracking algorithm on challenging sequences of ballet dancing. We start by describing the hardware setup used to capture the video streams. We then detail the content of the video sequences and the acquisition of the ground-truth data. The evaluation consists of a visual inspection, followed by quantitative error measurements. We also compare our algorithm to other standard methods. After some performance results, a discussion about the scalability of the system concludes the chapter.

7.1 Hardware Setup and Test Sequences

This section describes the acquisition of the test sequences, and the training of the dynamic model from Section 6.3 on the training data.

7.1.1 Hardware Setup

All video sequences presented in this thesis were captured using commodity hardware. We used 5 webcams capturing video sequences at 30 frames per second, in 320×240 resolution. The images were acquired in YUV:422 format, which means that the chrominance components were sub-sampled, reducing the quality of the image. The videos were acquired by a single computer to which all cameras were linked through an IEEE1394 (also known as "firewire") bus.

No external triggering being available, the synchronisation between multiple views is approximate. At full framerate (30fps), the maximal delay between frames is quite small, so that the level of synchronisation is sufficient for most applications. The reconstruction of fast movements (more than 10cm per frame), however, suffers from this lack of accurate synchronisation.

The cameras also exhibit a high level of noise (analysed in Chapter 2). The speed of the shutter was kept at the minimum to limit motion blur, but the problem with short exposure times is that a strong lighting is required. Since we could not reach a sufficient level of lighting, we had to increase the gamma correction, which has the undesirable effect of accentuating the noise.

The calibration of both intrinsic and extrinsic camera parameters was performed using Bouguet's Matlab Toolbox [Bou]. This toolkit implements the calibration procedure described by Zhang [Zha00]. A chessboard pattern was used to acquire coplanar points, and perform the full calibration. The placement of the cameras was limited to arrangements where all views had a full coverage of the chessboard pattern.

Finally, even though the tracking space was emptied to allow the dancers to perform safely, no particular effort was made to facilitate the background segmentation. It can be noticed on the video sequences that the background is still cluttered, and that the clothing of the dancers is very similar in colour to some elements in the background.

7.1.2 The Ballet Dancing Sequences

Ballet dancing is an interesting application for the evaluation of human-body tracking algorithms because of its diverse body postures, and its fast and challenging movements. The speed of execution is a key challenge and represents an important test for human-body tracking algorithms. Finally, ballet dancing is a structured activity, allowing some predictability in the succession of the movements, therefore making the learning of behaviour patterns possible.

We evaluated our tracking algorithms on ballet dancing sequences, captured with students of Kate Simmons Dance Ltd. Our dancers were not professional ballet dancers and as a consequence, the choreographies were only reproduced approximately. We insist on this point because it makes the learning of dynamics and the generalisation from examples particularly challenging.

The dancers performed a complex sequence composed of 2 exercises, amounting to approximately 1500 frames (750 frames for each exercise). The choreography of each exercise is given in standard ballet notations in Table 7.1. The test sequences feature

Exercise 1	Exercise 2
1. Glissade derrière	
2. Jeté derrière	1. Deux Balancés
3. Coupé	2. Pas de côté droit soutenu
4. Assemblé dessous	3. Dégagé de côté droit
5. Sissonne fermée devant	4. Préparation pour la Pirouette
6. Sissonne fermée derrière	5. Double Pirouette fermée derrière
7. Sissonne fermée de côté	

Table 7.1: *Description of the dance exercises in standard ballet notation.*

the full choreography (both dance exercises), while the training sequences consist of 3 repetitions of each dance exercise.

7.1.3 Ground-Truth and Training Data

The data used respectively for training and quantitative evaluation were obtained by manual annotation of the video sequences. The 2-D locations of 12 body parts were first annotated for each frame of the sequence, and for all camera views. The 3-D locations of the body parts were then computed as a linear optimisation problem, minimising the reprojection error. The trajectories of the body parts were then smoothed and interpolated from, as described in Section 6.3.2. The joint angles were finally recovered using inverse kinematics. We obtained a total of 13000 frames for training by oversampling and varying the amount of smoothing for each training sequence.

For each dance exercise, we automatically clustered the training data using our own implementation of the EM algorithm proposed by Figueiredo *et al.* [FJ02], as described in Section 6.3.2. A total of 122 clusters were found for the full choreography (both dance exercises), which can seem high but actually reflects the underlying complexity of the motions. As a comparison, the same clustering on a simpler "arms pointing" sequence returned only 5 clusters.

The data-path of the parameters across the Gaussian clusters was used to train a variable length Markov model with various maximal history lengths. Using a maximal memory length of 10, the VLMM learnt 948 distinct states. This number of states rose to 2262 with a maximal length of 20 and 2890 with a maximal memory length of 30. All VLMMs were trained with a threshold on the Kullback-Leibler divergence of

$\varepsilon = 0.0001$ (see [RST96] for details).

7.2 Tracking Results

In this section, we present the results of the tracking of the ballet-dancing sequences using our algorithm. We then compare accuracy and robustness against other standard techniques.

7.2.1 Visual Analysis

Figures 7.1 and 7.2 show the tracked model pose, superimposed on one of the 5 input views used to capture the first dancing exercise. We used for this sequence a VLMM with maximal history of 20 frames and 1000 particles. The tracking is successful over the whole sequence, but it can be noticed that on a few frames, the tracked position does not match exactly the subject. Poor image evidences are mainly to blame, especially in presence of fast movements generating motion blur. A second explanation is the relatively small amount of training data (3 repetitions of the exercise), making the generalisation of the motions difficult. Finally, as for all Monte-Carlo techniques, the true mode of the *posterior* is difficult to reach, so that the displayed pose does not necessarily reflect the underlying distribution of the particles.

Despite these shortcomings, and even if the tracked position is sometimes approximate, the overall pose of the body remains coherent with both the training sequences and image evidence. The dynamic model propagates particles only in plausible directions, so that the model never falls into obviously impossible poses. The consistency of the pose is enforced, even when image evidence is very poor. These results appear very promising for the future, as we could expect tracking accuracy to improve substantially using cameras with better resolution and synchronisation.

A subject performing the second dance exercise is tracked in Figures 7.3 and 7.4. More particles (2000) had to be used for this sequence because of the fast rotation of the root of the kinematic tree, for which no dynamic model is currently learnt. We expect the number of required particles to be significantly reduced with the inclusion of the derivatives of the global parameters into the dynamic model.

Once again, despite occasional inexact positionings, tracking was successful over the whole sequence. This second dance exercise is particularly challenging because the limbs tend to stay close to the body during fast rotations (pirouette). In these cases of

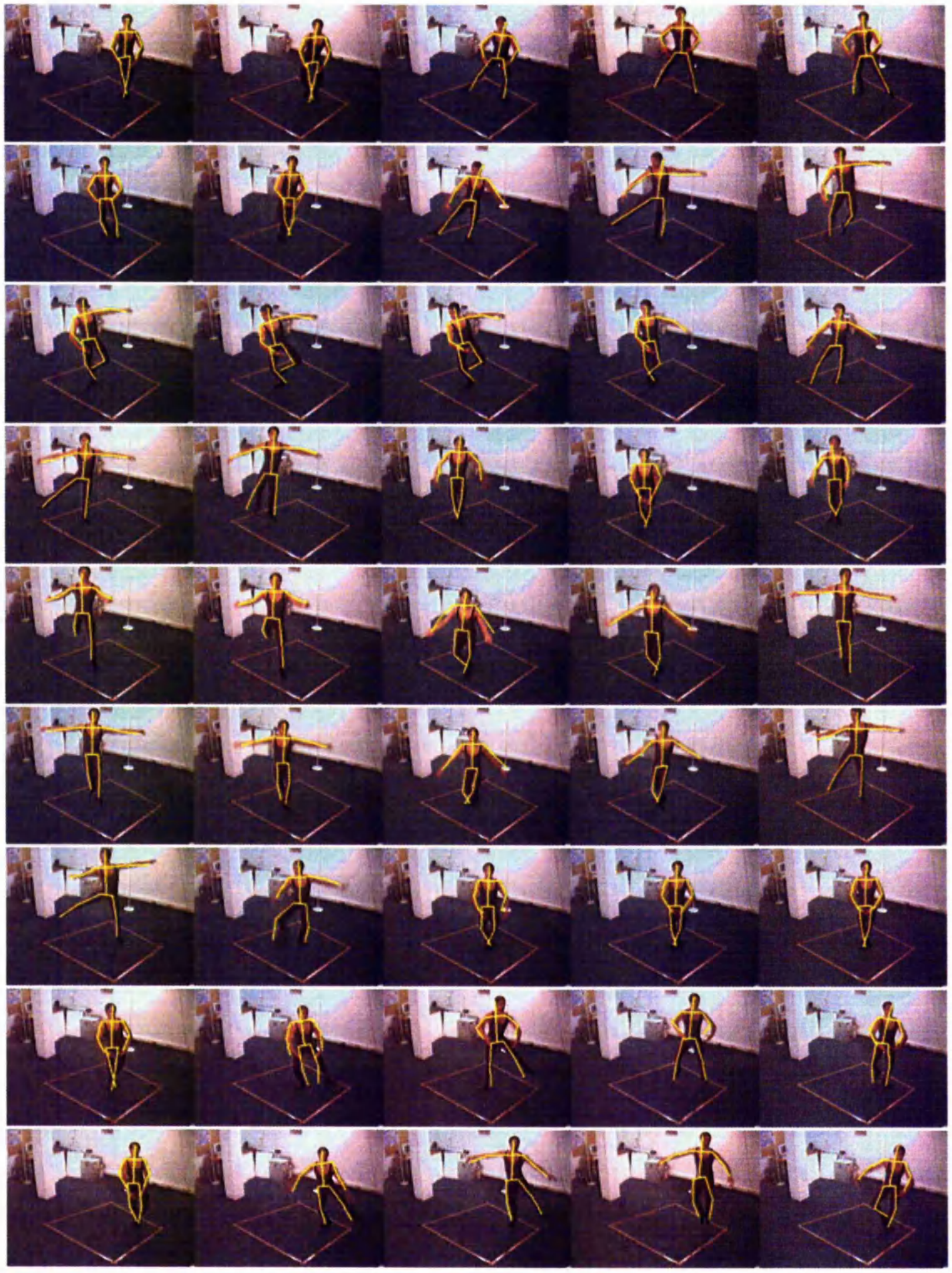


Figure 7.1: *Tracking the first dancing exercise, with a VLMM history of* 20 *frames, and* 1000 *particles. The video sequence is sampled every* 5 *frames, corresponding to* 167*ms (continued in Figure 7.2).*

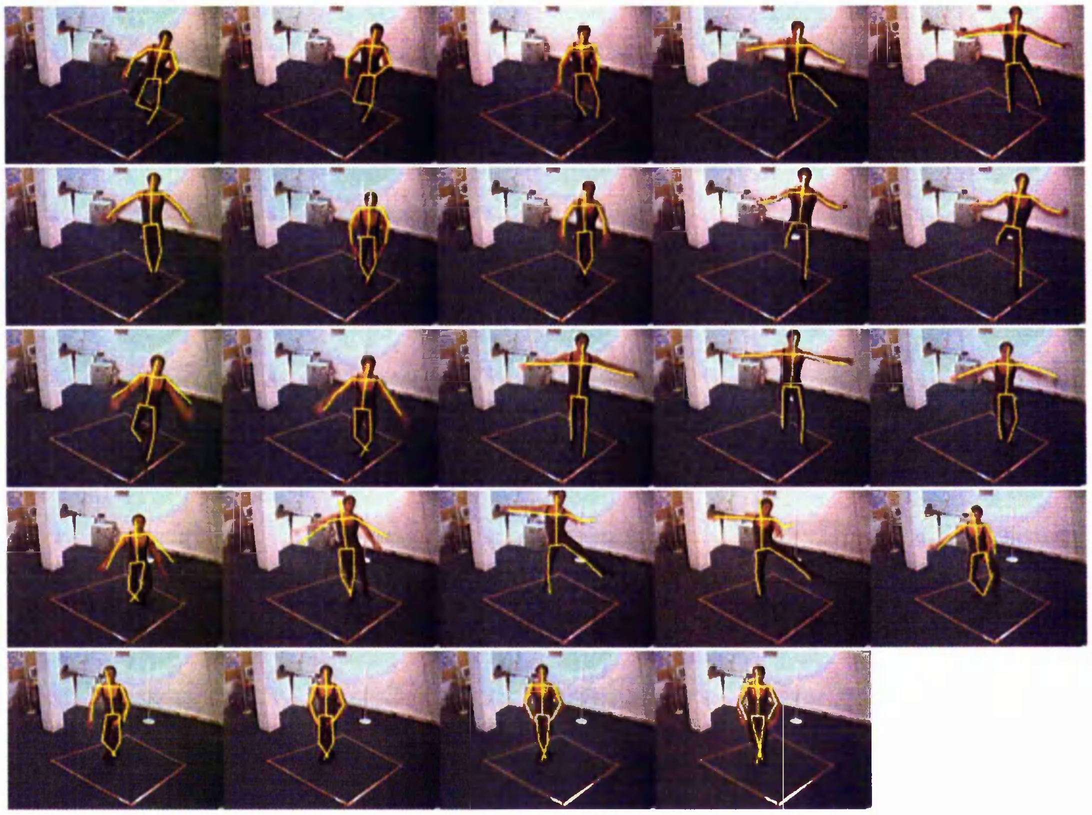


Figure 7.2: *End of the tracking of dance exercise* 1 *(continued from Figure 7.1).*

important self-occlusions combined with motion blur, the reconstructed volume provides very poor image evidence. The *motion prior* is then fully exploited, constraining the poses of the model to the learnt configurations. The ability of the particle filter to keep track of multiple hypothesis is also important for automatic recovery after periods of ambiguous *likelihood* function.

7.2.2 Quantitative Error Measurements

We now present comparative error measurements between our algorithm, and other standard approaches based on particle filters. The CONDENSATION [IB98a] algorithm propagates particles using the predictions of an auto-regressive process (ARP). Annealing [DBR00] iterates a propagation-evaluation loop over multiple layers, in a "coarse to fine" manner. Having a simple predictive model, these two methods are unable to provide a good initialisation for the blob-fitting procedure, and quickly fail in normal tracking conditions. Even with 5000 particles evaluations, they both quickly

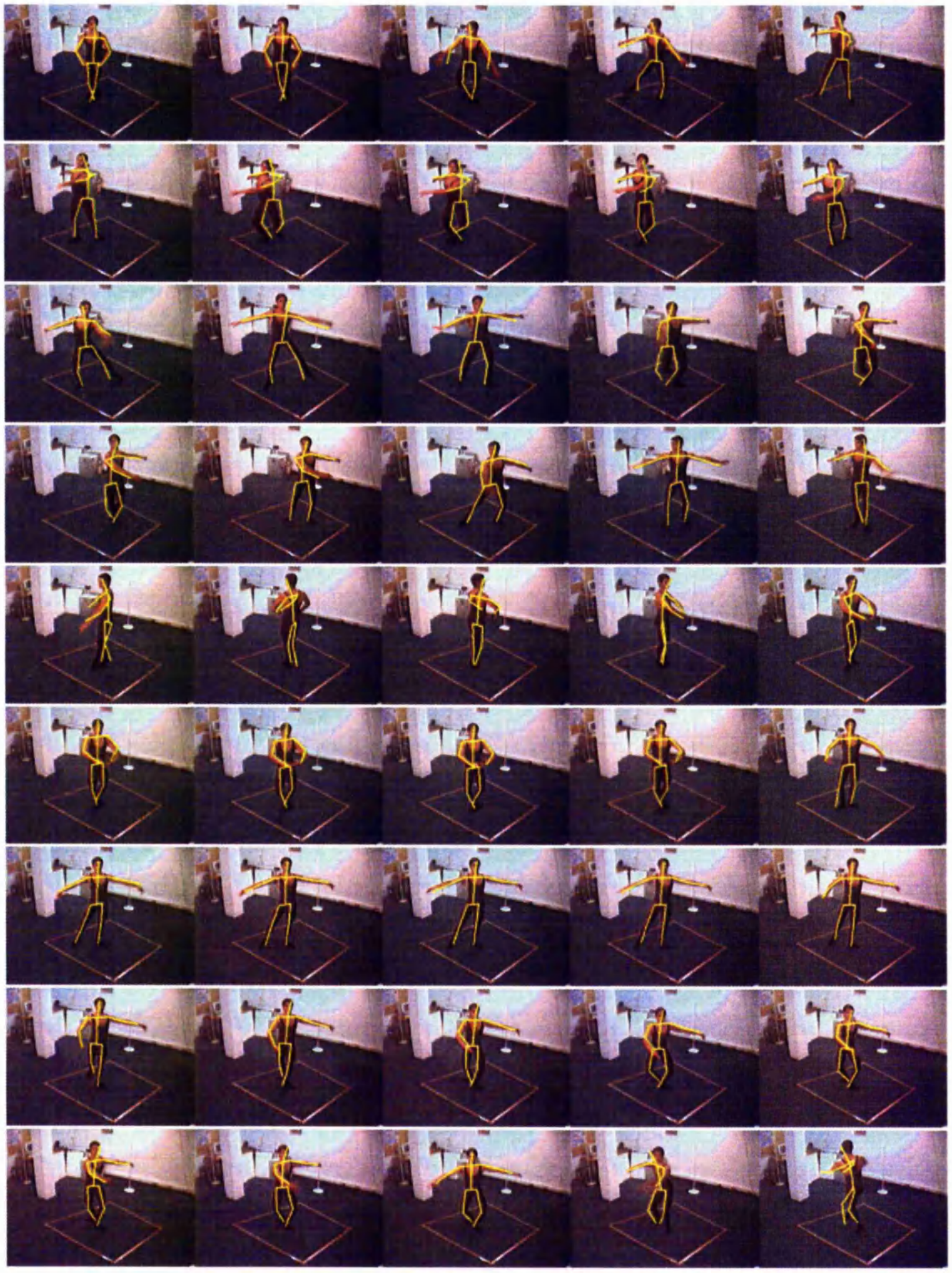


Figure 7.3: *Tracking the first dancing exercise, with a VLMM history of* 20 *frames, and* 2000 *particles. The video sequence is sampled every* 5 *frames, corresponding to* 167*ms (continued in Figure 7.4).*

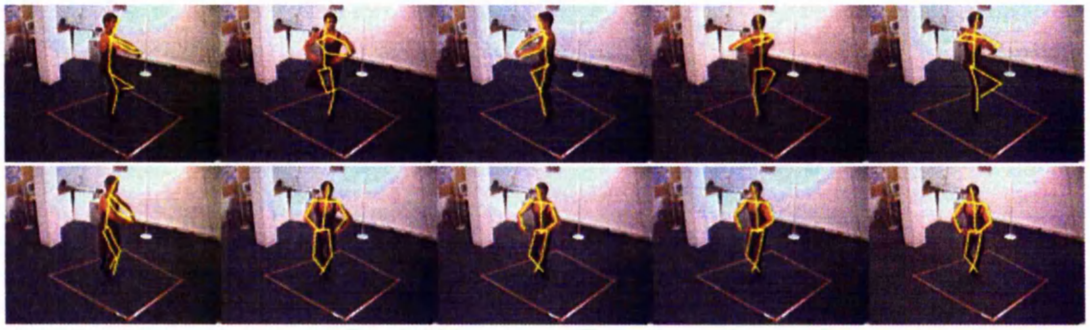


Figure 7.4: *End of the tracking of dance exercise 2 (continued from Figure 7.3).*

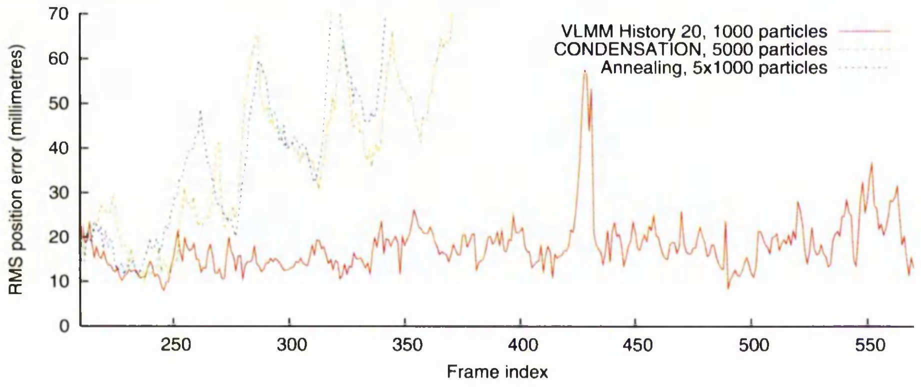


Figure 7.5: *Accuracy comparison between the particles propagation schemes of CONDENSATION, annealing, and our method. The RMS joint position error with the manually annotated ground truth is shown for the first dance exercise.*

lose track when used in our "blobs evaluation" framework, as illustrated in Figure 7.5. Our algorithm, however, maintains a good overall accuracy with only 1000 particles. A momentary tracking failure around frame 420 is automatically detected and recovered from by reinitialising the VLMM.

To keep the comparison focused on the dynamic models, we use the same likelihood distribution for all three algorithms (CONDENSATION, annealing and our method). At each frame, the blob-fitting procedure is initialised from the annotated ground-truth pose of the model. This provides a good, but also realistically noisy, likelihood function for all three algorithms. Results are reported in Figure 7.6. Even using 5000 particles, CONDENSATION is unable to explore the parameter-space in all appropriate directions, resulting in a poor overall accuracy. The Annealed particle filter uses only 1000 particles, but because of the 5 layers of annealing, the computational cost remains equivalent to CONDENSATION. Annealing produces relatively

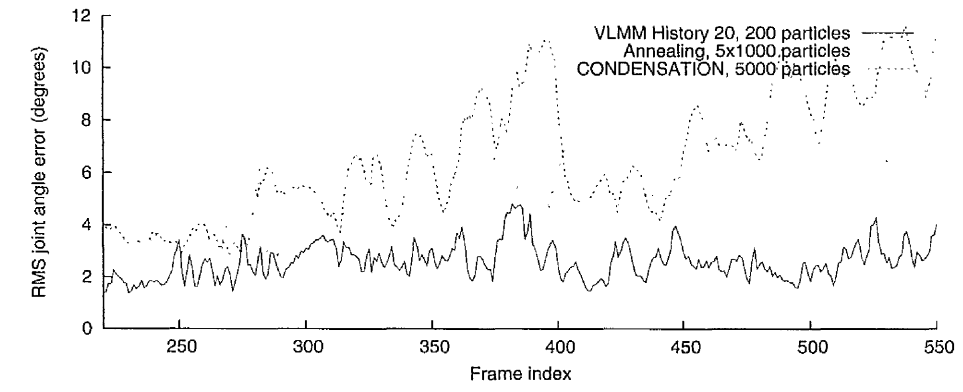


Figure 7.6: *Accuracy comparison between the particles propagation schemes of CONDENSATION, annealing, and our method. All three algorithms are evaluated against a common likelihood function, in which the blob-fitting procedure is initialised from the annotated ground-truth.*

accurate results in most of the test sequence, although it is still distracted by the noisy likelihood function. Annealing also tends to focus particles on a single mode of the posterior, limiting the ability of the tracker to recover from ambiguous situations. We tested our propagation method with only 200 particles. Despite having 25 times less particle-evaluations than the two other methods, accuracy and robustness were maintained throughout the sequence.

Figure 7.7 compares the accuracy of predictions using various memory lengths for the VLMM and only 200 particles. A memory of 1 frame (first order Markov model) is insufficient to capture the complexity of the succession of movements, and wastes particles by propagating them to the wrong clusters. Accuracy is therefore penalised by the smaller number of particles tracking the right pose. With a longer memory, the propagation of the particles is more focused, and the overall accuracy is improved. The accuracy of the prediction is slightly improved by increasing the maximal memory length of the VLMM from 10 frames to 20 frames.

The optimal memory length of the VLMM can be experimentally determined by measuring the relative entropy between the predictions of the trained VLMM and test sequences. Unfortunately, we could not carry out this these types of experimentations, mainly because of the limited amount of training data. A maximal memory length of 20 currently seems to give the best results, but a more systematic testing is needed. For more details about optimising the parameters of variable length Markov Models, the reader is referred to [Gal01].

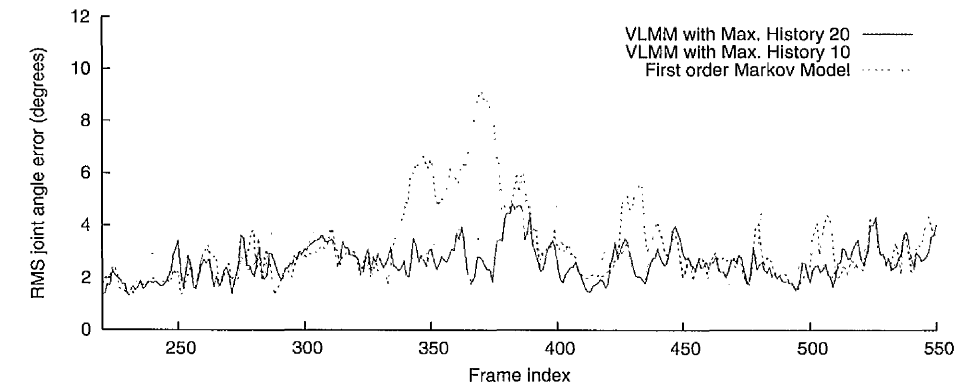


Figure 7.7: *Prediction accuracy for various history lengths of the Markov model, using* 200 *particles.*

7.3 Performance Considerations

Table 7.2 reports performance measurements on a 2 GHz Pentium 4. The timings of the volumetric reconstruction are similar to these presented in Section 3.5. As mentioned earlier, the maximal recursive depth of the 3-D reconstruction is the main parameter which conditions both performance and accuracy. A maximal depth of 6 ($64 \times 64 \times 64$ voxels) produces a coarse reconstruction, but is nevertheless sufficient for some tracking requirements. All the results presented in this section were obtained with a maximal recursive depth of 7 ($128{\times}128{\times}128$ voxels), which gives a good compromise between efficiency and accuracy. Also, the low resolutions of our cameras (320×240) made finer reconstructed volumes meaningless. However, when using higher resolution cameras, a maximal recursive depth of 8 ($256 \times 256 \times 256$ voxels) can be envisaged, still running with a competitive level of performance.

Because of the linear complexity of EM with respect to the data, the cost of the blob-fitting procedure is strongly influenced by the number of reconstructed voxels. The timings are reported in Table 7.2 for a single blob-fitting procedure, and should therefore to be multiplied by the number of actual blob-fittings performed at each frame. The number of candidate clusters for the blobs-fitting highly depends on the predictions of the VLMM. As the predictions are more accurate, less particles are propagated to the wrong clusters, and the effective number of blob-fittings is decreased. When using a VLMM with a maximal history of 20 frames, the number of candidate clusters was below 5 in most of the dance sequence, corresponding to a total time below 74ms per frame. When a re-initialisation of the tracker is needed, however, it implies fitting the blobs from all the clusters. With 122 clusters, each re-initialisation

	3-D Reconstruction			Blobs	Particle Filter	
Max. Depth	3 views	4 views	5 views	Fitting	500	1000
$D^+=6$	11.0ms	11.1ms	12.8ms	3.1ms		
$D^+=7$	24.1ms	22.8ms	28.1ms	14.8ms	20.7ms	45.3ms
$D^+=8$	73.5ms	66.0ms	78.6ms	74.1ms		

Table 7.2: *Overall performance measurements.*

provokes a lag or nearly 2 seconds.

The performance of the particle filter including resampling, propagation and evaluation is also reported for 500 and 1000 particles in Table 7.2. These timings are independent of the maximal recursive depth of the 3-D reconstruction because the evaluation of the particles is only based on the fitted blobs. All algorithms composing the particle filter have a linear complexity with respect to the number of particles. The framework can therefore scale to large numbers of particles, which can be managed in real-time, thanks to the fast evaluation procedure.

The full tracker with a reconstruction depth of 7 and 1000 particles ran at an average of 8.5fps over the dance sequence. While special care has been given to the algorithms, we believe that the efficiency of the implementation could be improved by a significant margin with low-level optimisations. Considering the extra overheads, such as disk accesses or displays, this final performance result satisfies our original objective of running in real-time on a single computer. Let us finally mention the great potential for parallelisation of the whole system. Not only all modules can run concurrently on a multi-processor system, but also each individual module can be parallelised in a very straightforward manner.

7.4 Discussion on the Scaling Issue

Even if our test sequences exhibited diverse and challenging movements, more tests would be needed to confirm the applicability of the method to other types of behaviour. Because of their simple dynamics, atomic and cyclic activities, such as walking, should be straightforward to learn. The main challenge, however, would be to try learning large amounts of diverse activities altogether. Unfortunately, this type of large-scale evaluation could not be performed in the scope of the work described in this thesis because it would have required a readily available motion capture setup. We can nonetheless extrapolate on the potential issues and benefits of our predictive framework, when confronted with larger sets of motions.

Clustering the parameter space into atomic behaviours should not suffer from a larger database of movements. No dimensionality reduction method was employed, so that clustering is performed in the full parameter space in which different motions are naturally separated out. Of course, we expect some atomic movements to be recurrent in diverse types of behaviour, but this is actually a benefit as the number of clusters would not grow linearly with the size of the dataset.

The process of learning transitions between clusters should also scale naturally to a larger dataset. The variable length Markov models are designed to optimise locally their memory length, so that the size of the VLMM should remain manageable. The computational cost of the predictions would remain unaffected by a more complex VLMM because of the finite state automaton structure.

With the increased number of clusters, the advantage of VLMMs over lower order models should however become more prominent A long history should allow behaviours to be differentiated, and predictions to be performed accordingly. For example, even if the walking cycle shares some clusters of atomic movements with other types of behaviours, the VLMM would associate these common clusters with the current behaviour based on contextual history. When a subject is walking, the predictions of the VLMM would then be unaffected by other types of behaviours sharing common atomic motions, but different from their context.

Of course, problems will still appear for movements previously unseen in the training data. In the current implementation of the system, nothing is done to handle these cases, and the tracker has to be reinitialised. As long as the total number of clusters remains significantly lower than the number of particles, this simple reinitialisation of the VLMM works well. However, for larger pools of diverse movements, a simple reinitialisation can prove both computationally expensive and inefficient. Unfortunately, switching back to a stochastic propagation method is not a viable option, as we demonstrated in this chapter that such methods are incapable of exploring the whole parameter space. This difficult problem is left open for future research.

7.5 Conclusion

In this chapter, we evaluated our Bayesian tracking algorithm on challenging sequences of ballet dancing. The visual results exhibited the complexity of the sequences, and the poor image evidence that the tracker had to handle. A series of quantitative error measurements allowed us to evaluate our method against other standard algorithms. We demonstrated the benefits of our predictive scheme for both increased robustness and accuracy. Finally, we evaluated the performance of the system and its suitability for real-time applications with some benchmarks.

Chapter 8

Conclusion and Future Work

We now review the key points of the work presented in this thesis, followed by a summary of achievements, and suggestions of possible extensions and future work.

8.1 Summary of the Thesis

In this thesis, we have introduced two algorithms for real-time full human body tracking. Both techniques are based on the prior volumetric reconstruction of the subject of interest, and the fitting of 3-D Gaussian blobs. An important design constraint is to keep all algorithms efficient enough to reach real-time performance. This is achieved in various ways without compromising on robustness.

Our hierarchical volumetric reconstruction algorithm is based on the more general shape-from-silhouette paradigm, using background segmentation on each input view. Unlike other approaches, the extraction of the silhouettes is not artificially separated from the reconstruction process, which allows the classification of voxels using robust discriminative statistics. Performance is improved by segmenting only the required pixel samples, and robustness benefits from the combination of multiple views for ambiguous cases. We also include colour information into the reconstructed volume, in a fast and straightforward manner.

The blob-fitting procedure is designed to exploit the hierarchical structure of the reconstructed volume. We have detailed the two steps of an EM-like procedure which fits blobs onto reconstructed body parts using both position and colour. We have also introduced an automatic procedure to acquire the number of blobs and their disposition onto the kinematic model.

Our first tracking technique uses direct inference to recover the pose of the kinematic model from the set of fitted blobs. A set of goal positions is computed from the blobs, and the pose of the model is recovered as an inverse kinematics problem. This bottom-up approach has the clear advantage of simplicity and performance, and is adapted to many practical scenarios like human-computer interactions or upper-body tracking.

Our second approach to tracking estimates the distribution of the posterior with a set of discrete samples. The propagation of the particles is based on a relatively complex prediction scheme, decomposing the parameter space into clusters of elementary movements, and predicting the transitions between these clusters with a VLMM. Real-time performance is achieved with an evaluation scheme based on the relative entropy between two sets of blobs. Tests show the good robustness of the method, even confronted with challenging movements such as those found in ballet dancing.

8.2 Summary of Achievements

The final, and most important achievement of the work described in this thesis is the successful tracking of the ballet-dancing sequences presented in Chapter 7. To our knowledge, tracking such fast and complex motions in real-time is presently unique. The novel prediction and evaluation schemes that we developed for this purpose enable robust tracking at a greatly reduced cost when compared to the standard Bayesian Monte-Carlo framework.

With the hierarchical tracking technique, we demonstrate a low cost algorithm capable of handling non-critical motions. While the method is arguably less robust than the Bayesian one, it is also more general in the sense that no motion model has to be learnt.

Although they are not the direct aim of this thesis, the volumetric reconstruction and the blob framework, on which both tracking methods are built, constitute worthy contributions by themselves. The full volumetric reconstruction runs in real-time on a single computer while being robust enough to cope with noisy input images and cluttered backgrounds. Colour is incorporated at a very low extra-cost, making the full reconstruction algorithm very competitive, both in terms of features and performance. The appearance model based on 3-D blobs is acquired automatically and is efficiently fitted to the voxels.

8.3 Future Work and Possible Extensions

While usable and potentially useful for real applications, the work presented in this thesis is not a general solution to the human-body tracking problem. A number of simplifications and shortcuts had to be employed to obtain a running system, but we could expect most of these to disappear in the future. Our background segmentation scheme, for example, currently necessitates an empty scene at start-up and cannot cope with changing environments. An automatic acquisition and update scheme would be very useful, and would make the system more practical. A possible idea for such a scheme would be to update incrementally the models of the pixels which we know (from the tracked model) belong to the background. Other constraints that could be dealt with in a foreseeable future include automatic camera calibration and synchronisation.

Another type of improvement concerns the modelling of the subject, which is quite crude in our current system. The appearance model, composed of blobs, represents efficiently the body parts but is incapable of modelling accurately finer features such as the face or the hands. Experimenting with more complex statistical distributions could lead to more accurate results, but would also involve more processing. Nevertheless, this step might become unavoidable in the ambition of recovering the pose of more body parts, such as the feet or the hands. For these comparatively small parts of the body, learnt motion models are likely to play a major role when image evidence is insufficient.

As mentioned in Chapter 5, the kinematic model would benefit from a parametrisation without singularities. Replacing Euler angles with quaternions should be relatively straightforward and integrate well into the Bayesian framework. Another practical extension would be the automatic acquisition of the kinematic structure. Although we believe that some kind of prior knowledge about the kinematics of the target is necessary, learning automatically the relative sizes of the limbs could help adapt to different morphologies. This process could be coupled with the dynamic learning of the appearance model.

In the current scheme, the blobs-fitting procedure requires a good initialisation to be able to "snap" blobs to the correct voxels. This bias from the predictive model towards the image evidence is an important limitation of our approach. When a correct initialisation cannot be provided, the blobs used as image evidence can become a wrong interpretation of the image data, and generate tracking failures. Breaking this link from tracking (even with a good prediction scheme) to image evidence is a future research goal. Some EM algorithms which alleviate the need for a good initialisation have been proposed [FJ02], but additional experiments are needed to evaluate the computational efficiency of these methods, and how they would maintain consistency when presented with noisy and incomplete voxel data.

Simultaneous tracking of multiple subjects would open a new range of applications. It is still not clear how robustly our system would perform when presented with multiple subjects. The shape-from-silhouette algorithm used for reconstruction is the main potential problem, as multiple subjects would generate an important number of occlusions. The number of cameras and their dispositions would then play a crucial role, which remains to evaluate. The rest of the tracking process should generalise painlessly.

A final research area is the learning of the dynamic and behaviour models. A number of significant improvements could be made in this area, with the common objective of modelling more accurately a wider range of activities. Dimensionality reduction methods, such as SGPLVMs, are a promising direction worth investigating. An important part of human dynamics could be encoded on a lower dimensionality manifold, hence increasing the power of expression of the Gaussian clusters. Modelling stylistic variations would also make it easier to generalise from a simpler model, and provide more accurate predictions. Online learning, where unseen sequences are incrementally integrated into the behaviour model, would also represent a worthy contribution.

Learnt predictions could finally be assisted with physical priors. As a general rule, incorporating more prior knowledge into the model can only help in tracking (as long as this knowledge proves to be valid) and in cutting down the size of the required training dataset. Kinematic constraints can obviously be incorporated into the model to avoid impossible poses. Other priors based on the laws of physics, such as the overall balance of the subject, could also make predictions more accurate, even confronted to movements unseen in the training data.

Bibliography

[AMGC02] Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, February 2002. (Cited on pages 138, 139, and 140).

[AT04] Ankur Agarwal and Bill Triggs. Tracking articulated motion using a mixture of autoregressive models. In *8th European Conference on Computer Vision (ECCV 2004), Prague, Czech Republic, May 11-14, 2004. Proceedings, Part III*, volume 3023 of *Lecture Notes in Computer Science*, pages 54–65. Springer, 2004. (Cited on page 143).

[AV89] Narendra Ahuja and Jack E. Veenstra. Generating octrees from object silhouettes in orthographic views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):137–149, 1989. (Cited on page 57).

[Bau74] Bruce Guenther Baumgart. *Geometric modeling for computer vision.* PhD thesis, Stanford University, 1974. (Cited on page 57).

[BD02] Eugene Borovikov and Larry Davis. 3D shape estimation on density driven model fitting. In *Proceedings of the 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT'02)*, pages 116–126, 2002. (Cited on pages 59, 61, 84, and 110).

[BDC01] Adrian Broadhurst, Tom W. Drummond, and Roberto Cipolla. A probabilistic framework for space carving. In *Proceedings of the*

8th International Conference in Computer Vision (ICCV), pages 388–393, Vancouver, Canada, July 2001. IEEE Computer Society Press. (Cited on page 60).

[BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996. (Cited on page 70).

[BH00] Matthew Brand and Aaron Hertzmann. Style machines. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 183–192. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. (Cited on pages 18, 144, and 159).

[BKC04] Matthiew Brand, Kongbin Kang, and David B. Cooper. Algebraic solution for the visual hull. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'04)*, volume 1, pages 30–35, June 2004. (Cited on page 58).

[BL01a] Andrea Bottino and Aldo Laurentini. Experimenting with motion capture in a virtual environment. *The Visual Computer Journal*, 17(1):14–29, 2001. (Cited on page 59).

[BL01b] Andrea Bottino and Aldo Laurentini. A silhouette based technique for the reconstruction of human movement. *Computer Vision and Image Understanding*, 83(1):79–95, July 2001. (Cited on pages 26, 59, 83, 110, and 195).

[BL03] Andrea Bottino and Aldo Laurentini. Introducing a new problem: Shape-from-silhouette when the relative positions of the viewpoints is unknown. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1484–1493, November 2003. (Cited on page 60).

[BL04] Andrea Bottino and Aldo Laurentini. The visual hull of smooth curved objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(12):1622–1632, December 2004. (Cited on page 58).

[BM98] Christoph Bregler and Jitendra Malik. Tracking people with twists and exponential maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 8–15. IEEE Computer Society, 1998. (Cited on pages 85 and 111).

[BMK+00] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steve Shafer. Easyliving: Technologies for intelligent environments. In *Second International Symposium on Handheld and Ubiquitous Computing (HUC 2000)*, pages 12–27, September 2000. (Cited on page 19).

[BMP04] Christoph Bregler, Jitendra Malik, and Katherine Pullen. Twist based acquisition and tracking of animal and human kinematics. *International Journal of Computer Vision (IJCV)*, 56(3):179–194, March 2004. (Cited on pages 85 and 111).

[BNI99] Andrew Blake, Ben North, and Michael Isard. Learning multi-class dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, pages 389–395, Cambridge, MA, USA, 1999. MIT Press. (Cited on page 143).

[BOP97] Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden markov models for complex action recognition. In *In Proceedings of Computer Vision and Pattern Recognition (CVPR)*, page 994, Washington, DC, USA, 1997. IEEE Computer Society. (Cited on page 145).

[Bou] Matlab Camera Calibration Toolbox (Y. Bouguet) `http://www.vision.caltech.edu/bouguetj`. (Cited on page 161).

[Bra99] Matthew Brand. Shadow puppetry. In *Proceedings of the International Conference on Computer Vision (ICCV '99)*, volume 2, page 1237, Washington, DC, USA, 1999. IEEE Computer Society. (Cited on page 144).

[BSD03] Eugene Borovikov, Alan Sussman, and Larry Davis. A high performance multi-perspective vision studio. In *Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS'03)*, June 2003. (Cited on page 59).

[BV99] Jeremy S. De Bonet and Paul Viola. Roxels: Responsibility weighted 3D volume reconstruction. In *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages

418–425. IEEE Computer Society Press, September 1999. (Cited on page 60).

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. (Cited on page 61).

[CA86] C. H. Chien and J. K. Aggarwal. Volume/surface octrees for the representation of 3D objects. *Computer Vision, Graphics, and Image Processing*, 36(1):100–113, 1986. (Cited on page 57).

[CA89] C. H. Chien and J. K. Aggarwal. Model construction and shape recognition from occluding contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):372–389, 1989. (Cited on page 57).

[Can86] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 679–698, 1986. (Cited on page 152).

[CBHK04] Kong Man Cheung, Simon Baker, Jessica K Hodgins, and Takeo Kanade. Markerless human motion transfer. In *Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission*, September 2004. (Cited on page 152).

[CBK03a] Kong Man Cheung, Simon Baker, and Takeo Kanade. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, volume 1, pages 77–84, June 2003. (Cited on pages 85, 110, and 152).

[CBK03b] Kong Man Cheung, Simon Baker, and Takeo Kanade. Visual hull alignment and refinement across time: A 3D reconstruction algorithm combining shape-from-silhouette with stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, volume 2, pages 375–382, June 2003. (Cited on page 60).

[CBK05] Kong Man Cheung, Simon Baker, and Takeo Kanade. Shape-from-silhouette across time part i: Theory and algorithms. *International Journal of Computer Vision (IJCV)*, 2005. (Cited on page 60).

[CF01] Kiam Choo and David J. Fleet. People tracking with hybrid monte carlo. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 321–328, 2001. (Cited on page 145).

[CH87] G. Cormack and R. Horspool. Data Compression using Dynamic Markov Modelling. *Computer Journal*, 30(6):541–550, 1987. (Cited on page 147).

[Che03] Kong Man Cheung. *Visual Hull Construction, Alignment and Refinment for Human Kinematic Modeling, Motion Tracking and Rendering*. PhD thesis, Carnegie Mellon University, June 2003. (Cited on pages 60, 85, and 196).

[CKBH00] Kong Man Cheung, Takeo Kanade, Jean-Yves Bouguet, and Mark Holler. A real time system for robust 3D voxel reconstruction of human motions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '00)*, volume 2, pages 714–720, June 2000. (Cited on pages 24, 26, 46, 59, 61, 85, and 196).

[Coh98] Michael Cohen. Design principles for intelligent environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, Madison, WI, 1998. (Cited on page 19).

[Coh99] Michael Cohen. The future of human-computer interaction or how i learned to stop worrying and love my intelligent room. *IEEE Intelligent Systems*, March 1999. (Cited on page 19).

[CTMS03] Joel Carranza, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Free-viewpoint video of human actors. In *Proceedings of ACM SIGGRAPH, San Diego*, pages 569–577, 2003. (Cited on pages 24, 83, 110, 133, and 151).

[dATM$^+$04] Edilson de Aguiar, Christian Theobalt, Marcus Magnor, Holger Theisel, and Hans-Peter Seidel. M^3: Marker-free model reconstruction and motion tracking from 3D voxel data. In *Proceedings of Pacific Graphics*, 2004. (Cited on page 85).

[DBC$^+$99] Larry Davis, Eugene Borovikov, Ross Cutler, David Harwood, and Thanarat Horprasert. Multi-perspective analysis of human action. In *Proceedings of Third International Workshop on Cooperative Distributed Vision*, November 1999. (Cited on page 59).

[DBR00] Jonathan Deutscher, Andrew Blake, and Ian D. Reid. Articulated body motion capture by annealed particle filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 126–133. IEEE Computer Society Press, June 2000. (Cited on pages 141, 151, and 165).

[DDR01] Jonathan Deutscher, Andrew J. Davison, and Ian D. Reid. Automatic partitioning of high dimensional search spaces associated with articulated body motion capture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 669–676. IEEE Computer Society Press, December 2001. (Cited on pages 110, 142, and 151).

[DF99] Quentin Delamarre and Olivier Faugeras. 3D articulated models and multi-view tracking with silhouettes. In *Proceedings of International Conference in Computer Vision (ICCV'99)*, volume 2, pages 716–721, 1999. (Cited on pages 24 and 83).

[DH04] Hannah Dee and David Hogg. Detecting inexplicable behaviour. In *Proceedings of the British Machine Vision Conference (BMVC)*, Kingston, UK, Steptember 2004. (Cited on page 19).

[DKD03] David Demirdjian, T. Ko, and Trevor Darrell. Constraining human body tracking. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV'03)*, page 1071. IEEE Computer Society, 2003. (Cited on pages 109 and 111).

[DLR77] 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 Series B*, 39(1):1–38, November 1977. (Cited on pages 86 and 89).

[Dye01] Charles R. Dyer. Volumetric scene reconstruction from multiple views. In L. S. Davis, editor, *Foundations of Image Understanding*, pages 469–489. Kluwer, 2001. (Cited on page 59).

[FB03] Jean-Sébastien Franco and Edmond Boyer. Exact polyhedral visual hulls. In *Fourteenth British Machine Vision Conference (BMVC'03)*, pages 329–338, September 2003. Norwich, UK. (Cited on page 58).

[Fel45] W. Feller. The fundamental limit theorems in probability. *Bull. Amer. Math. Soc.*, 51:800–832, 1945. (Cited on page 28).

[FJ02] Mario A. T. Figueiredo and Anil K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396, 2002. (Cited on pages 90, 91, 146, 162, and 176).

[FR97] Nir Friedman and Stuart Russel. Image segmentation in video sequences: A probabilistic approach. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI 97)*, 1997. (Cited on pages 26, 46, and 194).

[Gal01] Aphrodite Galata. *Learning Variable Length Markov Models of Behaviour*. PhD thesis, School of Computing, University of Leeds, 2001. (Cited on page 168).

[Gav99] Dariu M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding: CVIU*, 73(1):82–98, 1999. (Cited on page 141).

[GCMH02] Aphrodite Galata, Anthony G. Cohn, Derek Magee, and David Hogg. Modeling interaction using learnt qualitative spatio-temporal relations and variable length markov models. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'02)*, pages 741–745, 2002. (Cited on page 147).

[GD96] Dariu Gavrila and Larry Davis. 3D model-based tracking of humans in action: a multi-view approach. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 73–80, 1996. (Cited on pages 83 and 110).

[GGG03] Jacob Goldberger, Shiri Gordon, and Hayit Greenspan. An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 487–493, 2003. (Cited on page 157).

[GJH99a] Aphrodite Galata, Neil Johnson, and David Hogg. Learning Behaviour Models of Human Activities. In *Procedings of the British Machine Vision Conference (BMVC)*, pages 12–22, 1999. (Cited on page 147).

[GJH99b] Aphrodite Galata, Neil Johnson, and David Hogg. Learning structured behaviour models using variable length markov models. In *In IEEE Workshop on Modelling People, Corfu, Greece*, September 1999. (Cited on page 147).

[GJH01] Aphrodite Galata, Neil Johnson, and David Hogg. Learning Variable Length Markov Models of Behaviour. *Computer Vision and Image Understanding*, 81(3):398–413, 2001. (Cited on page 147).

[GMHP04] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popovic. Style-based inverse kinematics. *ACM Trans. Graph. (Proceedings of the 2004 SIGGRAPH Conference)*, 23(3):522–531, 2004. (Cited on pages 117 and 159).

[GP95] I. Guyon and F. Pereira. Design of a Linguistic Postprocessor using Variable Memory Length Markov Models. In *ICDAR*, pages 454–457, 1995. (Cited on page 147).

[GSD03a] Kristen Grauman, Gregory Shakhnarovich, and Trevor Darrell. A Bayesian approach to image-based visual hull reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, volume 1, pages 187–194, June 2003. (Cited on page 60).

[GSD03b] Kristen Grauman, Gregory Shakhnarovich, and Trevor Darrell. Inferring 3D structure with a statistical image-based shape model. In

Proceedings of the IEEE International Conference on Computer Vision (ICCV'03), volume 1, pages 641–649, October 2003. (Cited on pages 24 and 60).

[GSS94] N. Gordon, J. Salmond, and A. Smith. Novel approach to non-linear/non-gaussian bayesian state estimation. In *IEE Proceedings of Radar and Signal Processing*, volume 140, pages 107–113, April 1994. (Cited on page 133).

[HGW01a] Michael Harville, Gaile Gordon, and John Woodfill. Adaptive video background modeling using color and depth. In *Proceedings of the IEEE International Conference on Image Processing (Thessoloniki, Greece)*, October 2001. (Cited on pages 26 and 195).

[HGW01b] Michael Harville, Gaile Gordon, and John Woodfill. Foreground segmentation using adaptive mixture models in color and depth. In *Proceedings of the IEEE Workshop on Detection and Recognition of Events in Video (Vancouver, Canada)*, July 2001. (Cited on pages 26 and 195).

[HLGB03] Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, and Edmond Boyer. Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics*, pages 49–56. Eurographics, Elsevier, 2003. (Cited on page 59).

[HLS04] Jean-Marc Hasenfratz, Marc Lapierre, and François Sillion. A real-time system for full body interaction. *Virtual Environments*, pages 147–156, 2004. (Cited on page 59).

[Hog83] David Hogg. Model-based vision: a program to see a walking person. *Image and Vision Computing*, 1(1):5–20, February 1983. (Cited on pages 18, 83, and 132).

[Hor87] Berthold Klaus Paul Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4:629, April 1987. (Cited on page 110).

[HTH00] Pengyu Hong, Matthew Turk, and Thomas S. Huang. Gesture modeling and recognition using finite state machines. In *Proceedings of* *the Fourth International Conference on Automatic Face and Gesture Recognition*, pages 410–415, March 2000. (Cited on page 145).

[HUF04] Lorna Herda, Raquel Urtasun, and Pascal Fua. Hierarchical implicit surface joint limits to constrain video-based motion capture. In Tomás Pajdla and Jiri Matas, editors, *Proceedings of the European Conference on Computer Vision (ECCV'04)*, volume 2, pages 405–418. Springer, May 2004. (Cited on pages 108 and 110).

[HUF05] Lorna Herda, Raquel Urtasun, and Pascal Fua. Hierarchical implicit surface joint limits for human body tracking. *Computer Vision and Image Understanding*, 99(2):189–209, August 2005. (Cited on pages 108 and 110).

[IB98a] Michael Isard and Andrew Blake. CONDENSATION – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998. (Cited on pages 137, 141, 150, and 165).

[IB98b] Michael Isard and Andrew Blake. ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework. In *Proceedings of the 5th European Conference in Computer Vision (ECCV)*, volume 1, pages 893–908, 1998. (Cited on pages 137 and 141).

[IB99] Michael Isard and Andrew Blake. A mixed-state condensation tracker with automatic model-switching. In *Proceedings of the 6th International Conference in Computer Vision (ICCV)*, pages 107–112, 1999. (Cited on page 143).

[JSS02] Omar Javed, Khurram Shafique, and Mubarak Shah. A hierarchical approach to robust background subtraction using color and gradient information. In *Proceedings of Workshop on Motion and Video Computing*, pages 22–27, December 2002. (Cited on pages 26 and 194).

[JTH99] Nebojsa Jojic, Matthew Turk, and Thomas S. Huang. Tracking self-occluding articulated objects in dense disparity maps. In *International Conference on Computer Vision (ICCV)*, pages 123–130, September 1999. (Cited on pages 85, 91, and 123).

[Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. In *Transaction of the ASME - Journal of Basic Engineering*, volume 82(Series D), pages 35–45, 1960. (Cited on page 85).

[KBG05] Roland Kehl, Matthew Bray, and Luc Van Gool. Full body tracking from multiple views using stochastic sampling. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 129–136, June 2005. (Cited on pages 56 and 152).

[Kit96] K. Kitagawa. Monte carlo filter and smoother for non-gaussian non-linear state space models. *Journal of Computational and Graphical Statistics*, 5:1–25, 1996. (Cited on page 139).

[Lau94] Aldo Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, February 1994. (Cited on page 58).

[Lau95] Aldo Laurentini. How far 3D shapes can be understood from 2d silhouettes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):188–195, February 1995. (Cited on page 58).

[Law04] Neil Lawrence. Gaussian process models for visualisation of high dimensional data. In L. Saul S. Thrun and B. Schlkopf, editors, *Advances in Neural Information Processing Systems (NIPS)*, 2004. (Cited on pages 144 and 159).

[LBP01] Svetlana Lazebnik, Edmond Boyer, and Jean Ponce. On computing exact visual hulls of solids bounded by smooth surfaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01)*, volume 1, pages 156–161, December 2001. (Cited on page 58).

[LC98] Jun S. Liu and Rong Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998. (Cited on page 139).

[LSL01] Jason P. Luck, Daniel E. Small, and Charles Q. Little. Real-time tracking of articulated human models using a 3D shape-from-silhouette method. In *Proceedings of the International Workshop* *on Robot Vision*, pages 19–26. Springer-Verlag, 2001. (Cited on pages 24, 56, and 58).

[LY02] Benny P. L. Lo and Guang-Zhong Yang. Neuro-fuzzy shadow filter. In *Proceedings of the 7th European Conference on Computer Vision*, pages 381–392. Springer-Verlag, May 2002. (Cited on page 47).

[MA83] W. N. Martin and J. K. Aggarwal. Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):150–158, March 1983. (Cited on page 57).

[MBM01] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering*, pages 115–125, June 2001. (Cited on page 58).

[MBR$^+$00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of ACM SIGGRAPH*, pages 369–374. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. (Cited on page 55).

[Met] MetaMotion$^{\text{TM}}$ `http://www.metamotion.com`. (Cited on page 19).

[MG01] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding (CVIU)*, 81(3):231–268, 2001. (Cited on pages 23 and 141).

[MH03] Joel Mitchelson and Adian Hilton. Simultaneous pose estimation of multiple people using multiple-view cues with hierarchical sampling. In *Proceedings of the British Machine Vision Conference (BMVC)*, September 2003. (Cited on pages 83, 110, 133, and 151).

[Mit97] Tom M. Mitchell. *Machine Learning*. Computer Science. McGraw-Hill, 1997. (Cited on page 34).

[MPC$^+$05] Craig D. Murray, Steve Pettifer, Fabrice Caillette, Emma Patchick, and Toby Howard. Immersive virtual reality as a rehabilitative technology for phantom limb experience. In *Proceedings of the 4th International Workshop on Virtual Rehabilitation*, September 2005. (Cited on page 19).

[MR03] R. Meir and G. Rtsch. An introduction to boosting and leveraging. In S. Mendelson and A. Smola, editors, *Advanced Lectures on Machine Learning*, pages 119–184. Springer, 2003. (Cited on page 153).

[MSZ94] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*, chapter 2, pages 22–34. CRC Press, Inc., Boca Raton, FL, USA, 1994. (Cited on page 111).

[MTHC03] Ivana Mikic, Mohan Trivedi, Edward Hunter, and Pamela Cosman. Human body model acquisition and tracking using voxel data. *International Journal of Computer Vision (IJCV)*, 53(3):199–223, July/August 2003. (Cited on pages 24, 56, 58, 61, 83, 111, 123, and 152).

[NBH05] Matti Niskanen, Edmond Boyer, and Radu Horaud. Articulated motion capture from 3-D points and normals. In A. W. Fitzgibbon W. F. Clocksin and P. H. S. Torr, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, volume 1, pages 439–448, September 2005. (Cited on page 61).

[NFA88] Hiroshi Noborio, Shozo Fukuda, and Suguru Arimoto. Construction of the octree approximating a three-dimensional object by using multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):769–782, 1988. (Cited on page 58).

[OB80] J. O'Rourke and N. I. Badler. Model-based image analysis of human motion using constraint propagation. In *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, volume 2(6), pages 522–536, November 1980. (Cited on page 18).

[PF03] Ralf Plänkers and Pascal Fua. Articulated soft objects for multiview shape and motion capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1182–1187, September 2003. (Cited on page 83).

[Pot87] Michael Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, 1987. (Cited on page 58).

[RF03] Deva Ramanan and D. A. Forsyth. Finding and tracking people from the bottom up. In *Computer Vision and Pattern Recognition (CVPR'03)*, volume 2, pages 467–475, June 2003. (Cited on page 132).

[RFZ05] Deva Ramanan, D. A. Forsyth, and Andrew Zisserman. Strike a pose: Tracking people by finding stylized poses. In *Computer Vision and Pattern Recognition (CVPR'05)*, 2005. (Cited on page 132).

[RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996. (Cited on pages 146, 147, and 163).

[SBF00] Hedvig Sidenbladh, Michael J. Black, and David J. Fleet. Stochastic tracking of 3D human figures using 2d image motion. In *Proceedings of the 6th European Conference on Computer Vision*, volume 2, pages 702–718, London, UK, 2000. Springer-Verlag. (Cited on page 143).

[SBR$^+$04] Leonid Sigal, Sidharth Bhatia, Stefan Roth, Michael J. Black, and Michal Isard. Tracking loose-limbed people. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 421–428. IEEE Computer Society Press, July 2004. (Cited on page 132).

[SBS02a] Miguel Sainz, Nader Bagherzadeh, and Antonio Susin. Hardware accelerated voxel carving. In X. Pueyo M.P. dos Santos, L. Velho, editor, *Proceedings of the 1st Iberoamerican Symposium in Computer Graphics (SIACG'2002)*, pages 289–297, Guimarães, Portugal, 2002. (Cited on page 60).

[SBS02b] Hedvig Sidenbladh, Michael J. Black, and Leonid Sigal. Implicit probabilistic models of human motion for synthesis and tracking. In *Proceedings of the 7th European Conference on Computer Vision*, volume 1, pages 784–800, 2002. (Cited on page 144).

[SG99] Chris Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, volume 2, pages 252–259, June 1999. (Cited on pages 26, 46, 49, 50, 52, 152, and 194).

[SGH05] Nikolay Stefanov, Aphrodite Galata, and Roger Hubbold. Real-time hand tracking with variable-length markov models of behaviour. In *Workshop on Vision for Human-Computer Interaction (V4HCI)*, June 2005. (Cited on pages 108 and 153).

[SISB04] Leonid Sigal, Michael Isard, Benjamin H. Sigelman, and Michael J. Black. Attractive people: Assembling loose-limbed models using non-parametric belief propagation. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004. (Cited on page 132).

[SVZ00] Dan Snow, Paul Viola, and Ramin Zabih. Exact voxel occupancy with graph cuts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'00)*, volume 1, pages 345–353, June 2000. (Cited on page 60).

[Sze90] Richard Szeliski. Real-time octree generation from rotating objects. Technical Report 90/12, Digital Equipment Corporation, Cambridge Research Lab, December 1990. (Cited on pages 26, 58, and 59).

[TD02] Leonid Taycher and Trevor Darrell. Range segmentation using visibility constraints. *International Journal of Computer Vision (IJCV)*, 2002. (Cited on page 27).

[TMSS02] Christian Theobalt, Marcus Magnor, Pascal Schüler, and Hans-Peter Seidel. Combining 2D feature tracking and volume reconstruction for online video-based human motion capture. In *Proceedings of Pacific Graphics, Beijing, China*, pages 96–103, 2002. (Cited on pages 26, 56, 58, 61, 111, and 152).

[UF04] Raquel Urtasun and Pascal Fua. 3D human body tracking using deterministic motion models. In *European Conference on Computer Vision*, May 2004. (Cited on page 144).

[UFF05] Raquel Urtasun, David J. Fleet, and Pascal Fua. Monocular 3D tracking of the golf swing. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 932–939, June 2005. (Cited on page 144).

[UFHF05] Raquel Urtasun, David J. Fleet, A. Hertzmann, and Pascal Fua. Priors for people tracking from small training sets. In *International Conference in Computer Vision (ICCV)*, October 2005. (Cited on pages 144 and 159).

[vdMDdFW00] R van der Merwe, A Doucet, Nando de Freitas, and E Wan. The unscented particle filter. In T.G. Dietterich T.K. Leen and V. Tresp, editors, *Advances in Neural Information Processing Systems (NIPS13)*, 2000. (Cited on page 145).

[Vic] Vicon Peak^{TM} `http://www.vicon.com`. (Cited on page 19).

[WADP97] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997. (Cited on pages 26, 46, 84, 87, 91, and 195).

[WB01] Greg Welch and Gary Bishop. An introduction to the kalman filter. In *Course 8 of SIGGRAPH 2001*, 2001. (Cited on page 85).

[Wel93] C. Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993. (Cited on pages 110, 124, 126, and 127).

[YSK^{+}98] Masanobu Yamamoto, Akitsugu Sato, Satoshi Kawada, Takuya Kondo, and Yoshihiko Osaki. Incremental tracking of human actions from multiple views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 2–7. IEEE Computer Society, 1998. (Cited on pages 83 and 110).

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. (Cited on page 161).

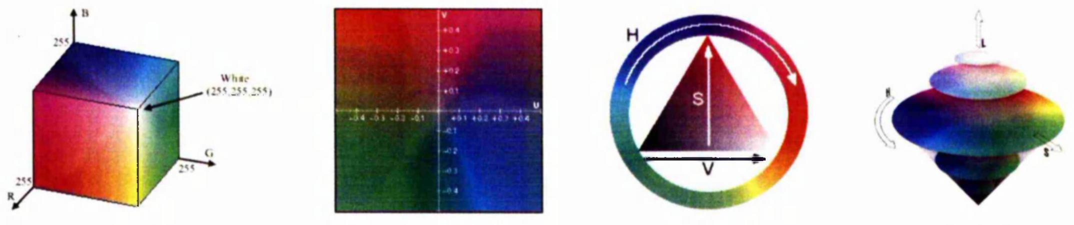
[ZN02] Tao Zhao and Ram Nevatia. Stochastic human segmentation from a static camera. In *Proceedings of Workshop on Motion and Video Computing*, pages 9–14, December 2002. (Cited on page 27).

Appendix A

Colour Representation

Most background segmentation techniques rely solely on the colour of the pixels for classification. The position of the pixels in the image is usually not considered because that would require a prohibitively complex model of the object of interest. The colour of a pixel is usually encoded with discrete values in a given a number of channels. Monochrome pixels are represented by a single intensity value whereas colour pixels have 3 to 4 components. While the coding of monochrome pixels is straightforward, there are various ways to code the colour of a pixel on 3 or 4 channels.

RGB (Red-Green-Blue) encoding is the most common way of defining a pixel colour. Each of the 3 channels has a value (usually an integer) representing the intensity of the corresponding base colour. This representation has the advantages of simplicity as well as a direct mapping with hardware pixel format. RGB colour-space has been used in various image segmentation techniques (for example [JSS02, SG99, FR97]). However the human perception of colours is very different from the one of the machine (the human eye is more than 5 times more sensitive to green than blue), and RGB encoding is not necessarily the most appropriate format for colour comparison.



(a) RGB (b) UV plane, Y=0.5 (c) HSV "wheel" (d) HLS

Figure A.1: *Graphical representation of RGB, YUV, HSV and HLS colour-spaces.*

YUV (Luminance-Chrominance) colour space is used mainly for video broadcasting: it is part of the PAL television standard. The luminance (Y component) represents the intensity or brightness of the colour. The other two channels (U and V) define the colour itself for a given brightness, as illustrated by Figure A.1(b). The YUV colour space is an affine transformation of RGB which has the advantage of being closer to human perception:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.332 & 0.5 \\ 0.5 & -0.419 & -0.0813 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128.0 \\ 128.0 \end{pmatrix} \tag{A.1}$$

The human eye is indeed far more sensitive to brightness changes than to colour changes. This leads to interesting applications: as a compression method (JPEG compression), the chrominance channels are often sub-sampled while keeping an accurate luminance. In computer vision and background segmentation, it is interesting to separate luminance from chrominance information because illumination artifacts (shadows) are then easier to discard. Just like RGB, YUV colour-space has been widely used for segmentation and tracking [HGW01b, HGW01a, WADP97, BL01b].

HSV (Hue-Saturation-Value) is another common colour space. The Hue is an angular value on a virtual colour wheel representing the chrominance of the colour, the Saturation can be seen as the "purity" of the colour and finally the Value is its brightness (Figure A.1(c)). The HSV colour-space is mainly used in computer graphics because of its convenience, but the transformation from RGB and YUV is non-linear, leading to singularities. This is particularly problematic in presence of random noise, where small perturbations in RGB or YUV colour-spaces can lead to big jumps in HSV space. Moreover, the conversion is inconsistent for some colours – like greys – where the Hue or the Saturation are undefined. There has been some rare examples of use of HSV in computer vision, but it is mostly inappropriate for our purpose, especially in the context of a coherent statistical framework.

Appendix B

Uniform Pixel Sampling

All pixels lying inside the projected area of a voxel should theoretically be inspected to decide on the classification of this voxel. However, an approximation has to be used to maintain the performance of the system. Following a method similar to Cheung *et al.* [CKBH00, Che03], the projected area of a voxel is sampled uniformly, and the classification is based on these samples alone. This approach makes sense in our case because the distances to the background model are then computed per-sample, and not for the whole image as when background segmentation is performed as a pre-processing step of the reconstruction.

The optimal number of samples is a critical parameter for robust classification. In the so-called SPOT (Sparse Pixel Occupancy Test) algorithm, Cheung *et al.* [CKBH00, Che03] use only 2 samples out of an average of 10 pixels in the projected area of each voxel. This number is derived from measured error rates in silhouette extraction. The total misclassification probability per voxel is then reported to be below 1%. Unfortunately, the same kind of reasoning cannot be applied to our case because we do not perform any binary segmentation of the silhouette pixels, and consequently misclassifications cannot be measured on a per-pixel basis.

Instead of sampling from the irregular shape of the projected area, one could sample uniformly inside voxels in 3-D space, and then use the projection of these 3-D samples. However, depending on the angle of view, these 3-D samples often project onto the same pixels or fail to uniformly cover the area. Alternatively, picking some pixels randomly from a 2-D area is straightforward, but for small numbers the resulting distribution can be uneven. A bad distribution of the samples can be damaging to the 3-D reconstruction considering that voxels are discarded from a single view. A small image feature or an edge can easily be missed if samples leave some part of the projected area uncovered. Samples should then be chosen so as to cover a maximum of space inside the projected area of the voxel. Several standard techniques exist to achieve this goal:

- *Systematic sampling:* There is no random element in this method. Pixels are linearly sampled at a regular interval, determined by dividing the total number of pixels in the area by the number of wanted samples. This method has the advantage of simplicity but can result in an uneven 2-D distribution, especially if the shape of the area is irregular. Moreover, the random aspect is important if we want all pixels to be sampled with equal probability.

- *Random sampling with heuristic:* The desired number of pixels is sampled randomly from the area. A heuristic measuring the goodness of the repartition is then evaluated and the samples are re-chosen iteratively until a criterion is satisfied. Depending on the heuristic used, the final repartition can be relatively uniform. Unfortunately, there is no guarantee of convergence, especially as the number of samples increase.

- *Clustered random sampling:* The area is first divided into as many clusters as desired samples. This clustering can be done in a number of ways, but using a regular grid is one of the easiest. In a second step, a single pixel is chosen randomly from each cluster. This technique is relatively simple and samples cover most of the area.

The "random sampling with heuristic" approach is very appealing from a statistical point of view, but its convergence problems make it hardly usable in practice. We choose to use a hybrid approach where, initially, the projected area is coarsely divided in equal clusters. The number of clusters is proportional to the number of desired samples, a practical ratio of 3 to 4 samples per cluster being sensible. The desired number of samples is then randomly picked from each of the clusters, insuring both maximal covering of space and randomness.

To avoid cases where the overall distribution of samples is still poor, a global heuristic is used to choose the most uniform distributions among a number of tries. The heuristic is chosen as an energy function $\mathcal{H}$ maximising the spacing (Euclidean distance) between all samples $\{s_1 \ldots s_{N_s}\}$:

$$\mathcal{H}(s_1 \ldots s_{N_s}) = \sum_{i=1}^{N_s} \sum_{j=1}^{N_s} distance(s_i, s_j)^2 \qquad \text{(B.1)}$$

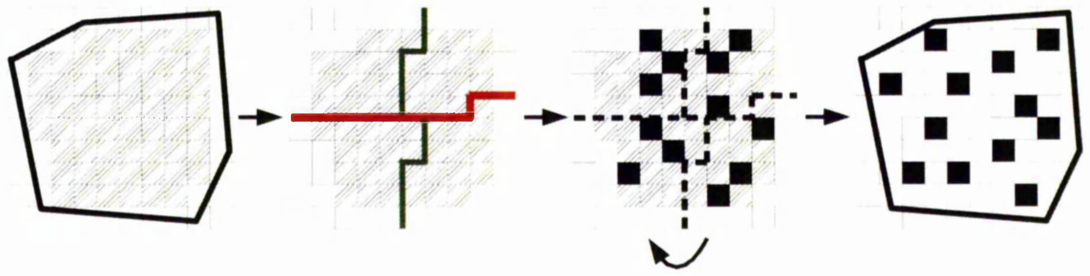


Figure B.1: *From left to right: The pixels lying totally inside the projected voxel area are selected, and depending on their number and the desired number of samples, they are divided into clusters of equal size. Pixels are then randomly sampled inside each cluster, the goodness of the distribution being evaluated for the whole area using a heuristic function. Finally, distributions satisfying the heuristic are retained.*

Figure B.1 shows a step-by-step illustration of the sampling process for a fairly large voxel projected area. For smaller projected areas, the clustering step can be unnecessary and pixels are then sampled randomly from the whole area. The heuristic function is still used to ensure a good repartition of samples.

Unfortunately, this type of sampling is still far too complex for real-time use. Actually, even the simplest sampling schemes would be too demanding for real-time implementation because they all require convex-hull computation and numerous tests. We have to pre-compute patterns of samples instead. The underlying assumption is that for a given camera, all voxels projecting on a relatively small part of the image plane are seen from a similar view angle. If we divide the image plane Π into such regions $\{\Pi_1 \ldots \Pi_{N_r}\}$, the projections of all voxels falling into the same region Π_i will have a similar shape. We can then approximate these projections by the same pre-computed area S_{Π_i} (see Figure B.2 for an illustration). A single projection area needs to be computed for all voxels projecting onto the same image region: different voxel sizes or distances to the camera are just matters of real-time scaling. Using the sampling method described earlier, patterns are pre-computed for different numbers of samples. The two steps below detail the offline computation of patterns of samples:

1. *Pre-compute the projected areas:* For each camera-view, divide the image plane Π into N_r regions $\{\Pi_1 \ldots \Pi_{N_r}\}$ (in our implementation, $N_r = 9$). For each region Π_i, pre-compute the approximated shape of the projected area of voxels falling into these regions. This is done by projecting (without applying lens distortions) all eight vertices of a voxel of unitary size, situated at a unitary distance from the camera,

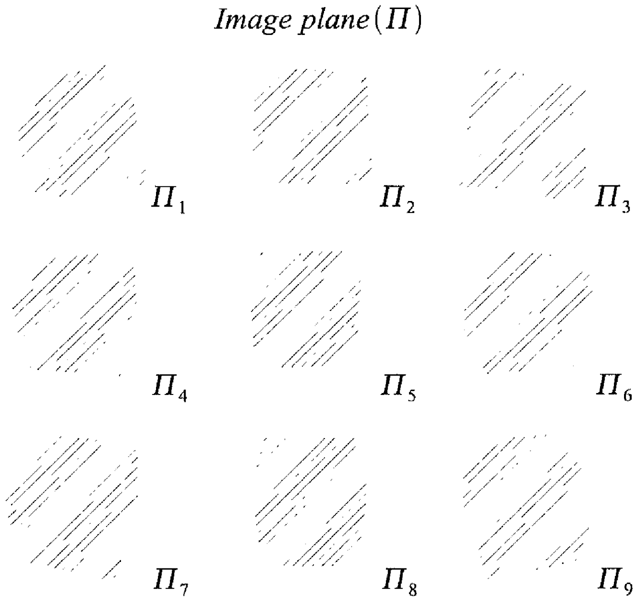


Figure B.2: *Voxels projecting onto a same image region (Π_i) have a similar projected area. These projected areas can therefore be approximated by pre-computed mean shapes S_{Π_i}.*

and which projects at the centre of region Π_i. The actual size of the projected area depends on the focal length of the camera, but is always big enough so that pixel discretisation is not a problem. The convex-hull of the eight projected vertices is then computed and the corresponding area is denoted as S_{Π_i}. The centre of S_{Π_i} is the projection of the centre of the voxel.

2. *Pre-compute patterns of samples:* For each projected area S_{Π_i} and desired number of sample N_s, pre-compute patterns of samples using the clustered random sampling with heuristic method (Figure B.1). For each projected area S_{Π_i} and desired number of samples, a large number of patterns are first generated without using the heuristic function $\mathcal{H}$. Only a fraction of these patterns which maximise $\mathcal{H}$ is retained (in practice, we retained the best 10 out of 100 patterns). The position of each sample inside the pattern is stored relatively to centre of S_{Π_i}. Patterns of samples can subsequently be retrieved depending on image-plane region Π_i, number of samples N_s and index of random pattern j.

Using these pre-computed patterns in real-time is now relatively straightforward, but in order to understand the different phases of voxel projection, let us first decompose the camera projection matrix. For a given calibrated camera $c_i, i \in [1..N_c]$ of focal length fl_i, principal point pp_i, orientation matrix R_i and position vector T_i, we define:

$$K_i = \begin{pmatrix} fl_{i,x} & 0 & pp_{i,x} \\ 0 & fl_{i,y} & pp_{i,y} \\ 0 & 0 & 1 \end{pmatrix} \qquad K_{Ri} = K_i \cdot R_i \qquad K_{Ti} = K_i \cdot T_i \tag{B.2}$$

The matrices K_{Ri} and K_{Ti} are pre-computed for each camera. Also using the non-linear function $kk_i() : \mathbb{R}^2 \to \mathbb{R}^2$ which applies camera distortions to a 2-D point, the complete projection of a 3-D vertex X onto the corresponding pixel $\left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right)$ can be written as:

$$\begin{pmatrix} x \\ y \end{pmatrix} = kk_i \begin{pmatrix} \frac{u}{w} \\ \frac{v}{w} \end{pmatrix} \quad \text{where} \quad \begin{pmatrix} u \\ v \\ w \end{pmatrix} = K_{Ri} \cdot X + K_{Ti} \tag{B.3}$$

This is a standard technique which would not be worth mentioning if we did not need to use intermediate results to place the pattern of samples on the image (for example, the distance of X to the camera is w). The few steps below detail the online process for a voxel $\mathcal{V}$ of size $s_{\mathcal{V}}$ and 3-D position $X_{\mathcal{V}}$:

1. *Voxel projection:* Using Equation B.3 on $X_{\mathcal{V}}$ gives the distance from the voxel to the camera (w) and the image coordinates of the projection of $X_{\mathcal{V}}$ before applying distortions ($x = u/w$ and $y = v/w$). Note that only the centre of the voxel is projected, which represents a significant speedup as compared to the projection of the eight vertices of the same voxel.

2. *Pattern selection:* The 2-D coordinates (x, y) of the projected centre designate the region Π_i, while the scaling factor of the pattern is the size of the voxel divided by its distance to the principal point ($s_{\mathcal{V}}/w$). The number of samples is then computed as a function of the scale factor (typically, N_s = constant $\times$ $(s_{\mathcal{V}}/w)$). Finally, a random number j is drawn to select the pattern that will be applied.

3. *Pattern positioning and scaling:* Since the coordinates of the samples in the patterns are normalised and relative to the projection of the centre, they need to be scaled by $s_{\mathcal{V}}/w$ and translated by (x, y).

4. *Apply distortions:* The function $kk_i()$ applies camera distortions to the 2-D coordinates of the samples. The distortions have to be applied at the end of the process because they depend on the positions of samples on the image, which are only known when placing the pattern. To speedup the evaluation of $kk_i()$, we build a lookup table for the whole image in a pre-processing step.

Appendix C

Spatial Covariance of a Voxel

Let us consider a voxel $\mathcal{V}$ of centre $X_{\mathcal{V}}$ and size $s_{\mathcal{V}}$. We aim at computing the spatial contribution of this voxel to the covariance matrix of a Gaussian distribution centred on μ_X, as illustrated in Figure C.1. The partial covariance matrix $d\Sigma_X$ contributed by the voxel $\mathcal{V}$ is defined as:

$$d\Sigma_X = \iiint_{\mathcal{V}} (X - \mu_X) \cdot (X - \mu_X)^T dX \tag{C.1}$$

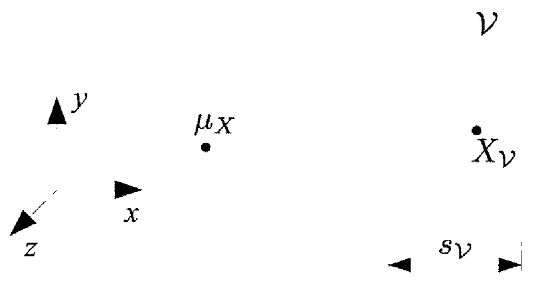


Figure C.1: *We wish to compute the contribution of the voxel $\mathcal{V}$ to the covariance matrix of a Gaussian distribution of mean μ_X. The main axes of the voxels are aligned with the axes of the coordinate system.*

A closed form solution can be obtained by introducing the centre of the voxel $X_\mathcal{V}$, and integrating over the volume:

$$\begin{aligned}
d\Sigma_X &= \iiint_\mathcal{V} (X-\mu_X)\cdot(X-\mu_X)^T dX \\
&= \iiint_\mathcal{V} ((X-X_\mathcal{V})+(X_\mathcal{V}-\mu_X))\cdot((X-X_\mathcal{V})+(X_\mathcal{V}-\mu_X))^T\, dX \\
&= \iiint_\mathcal{V} (X-X_\mathcal{V})\cdot(X-X_\mathcal{V})^T + 2.(X-X_\mathcal{V})\cdot(X_\mathcal{V}-\mu_X)^T \\
&\qquad + (X_\mathcal{V}-\mu_X)\cdot(X_\mathcal{V}-\mu_X)^T dX \\
&= \iiint_\mathcal{V} (X-X_\mathcal{V})\cdot(X-X_\mathcal{V})^T dX + 2.\int_\mathcal{V}(X-X_\mathcal{V})\cdot(X_\mathcal{V}-\mu_X)^T dX \\
&\qquad + s_\mathcal{V}{}^3.(X_\mathcal{V}-\mu_X)\cdot(X_\mathcal{V}-\mu_X)^T \\
&= \int_{-\frac{s_\mathcal{V}}{2}}^{\frac{s_\mathcal{V}}{2}}\int_{-\frac{s_\mathcal{V}}{2}}^{\frac{s_\mathcal{V}}{2}}\int_{-\frac{s_\mathcal{V}}{2}}^{\frac{s_\mathcal{V}}{2}} \begin{pmatrix} x^2 & x.y & x.z \\ x.y & y^2 & y.z \\ x.z & y.z & z^2 \end{pmatrix} dx\, dy\, dz + s_\mathcal{V}{}^3.(X_\mathcal{V}-\mu_X)\cdot(X_\mathcal{V}-\mu_X)^T \\
&= s_\mathcal{V}{}^2.\begin{pmatrix} \int_{-\frac{s_\mathcal{V}}{2}}^{\frac{s_\mathcal{V}}{2}} x^2 dx & 0 & 0 \\ 0 & \int_{-\frac{s_\mathcal{V}}{2}}^{\frac{s_\mathcal{V}}{2}} y^2 dy & 0 \\ 0 & 0 & \int_{-\frac{s_\mathcal{V}}{2}}^{\frac{s_\mathcal{V}}{2}} z^2 dz \end{pmatrix} + s_\mathcal{V}{}^3.(X_\mathcal{V}-\mu_X)\cdot(X_\mathcal{V}-\mu_X)^T \\
&= \frac{1}{12}.s_\mathcal{V}{}^5.I_3 \quad + \quad s_\mathcal{V}{}^3.(X_\mathcal{V}-\mu_X)\cdot(X_\mathcal{V}-\mu_X)^T
\end{aligned}$$

It can be noticed that this internal covariance appears only on the diagonal of the covariance matrix: it is intuitively explained by the fact that a voxel is a cube which sides are aligned with the axis of the coordinate system. It follows that the internal distribution is the same on each axis, with no dependencies on other axes.

Appendix D

Fast Computation of the Direction of Blob

When tracking the human body, one must notice that most of the trackable features have a clearly defined direction. The torso, the legs and the arms all have an elongated shape that necessarily reflects on the blobs, giving them a clear main direction. Extracting the main direction from blobs proves useful to drive the underlying kinematic model to an accurate position estimate. Of course, this scheme cannot apply to some features (head, hands, etc.) because of their rounder shape, and the system should be able to handle these special cases.

The main axis of a Gaussian blob is the eigenvector of the covariance matrix Σ_X associated with the greatest eigenvalue. The eigenvalues are first found by solving the characteristic polynomial $P_C(\lambda) = det(\Sigma_X - \lambda.I_3)$. The eigenvectors can then be computed as the vectors X satisfying $\Sigma_X \cdot X = \lambda.X$. The characteristic polynomial is a cubic which admits general analytical solutions, but at a high computational cost. Also, even with the eigenvalues computed, finding the eigenvectors is still a relatively expensive optimisation problem. A faster scheme is needed, at the possible price of a sacrifice in accuracy. Our accuracy requirements are relatively low since the directions of the blob are rather unstable with respect to noisy data.

The main axis of each blob from the last frame is also available, constituting a good first order approximation of the new main axis. Convergence is ensured by the fact that the main eigenvector is the only local maximum close to the approximate previous direction. Actually, the multiplication of any vector by the covariance matrix converges towards the main eigenvector, so that multiplying a sufficient number of times the initial estimate by Σ_X should converge towards the solution. Unfortunately, the convergence of this method is very slow, typically needing hundreds of iterations

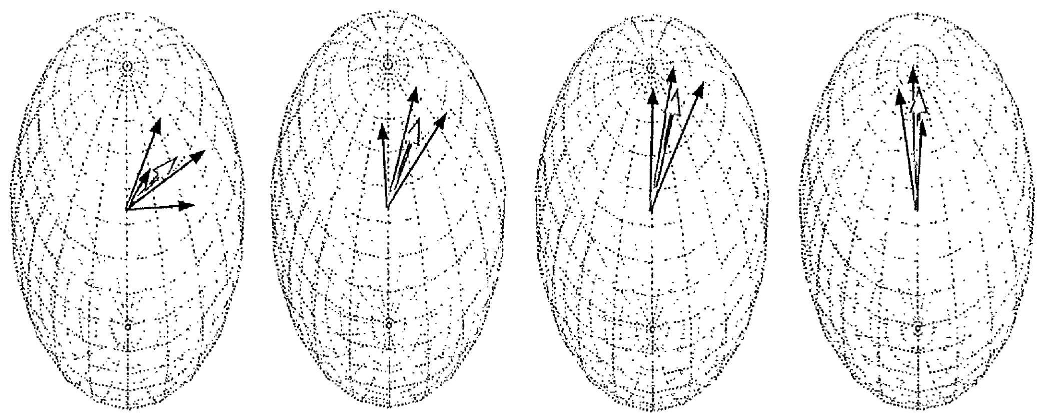


Figure D.1: *Iterative gradient descent converging towards the main eigenvector of the blob (from left to right). At each step, the current estimate (red arrow) is rotated around the axis y and z, and the sample with the minimal distance (blue arrow) is retained for the next step. It can be noticed that from the second iteration, only 3 samples are drawn since there is no need to re-evaluate the previous guess.*

to produce a good result. Another method for simple iterative gradient descent is proposed to achieve faster convergence.

For convenience, the above problem of maximisation of the eigenvalue is transposed into the equivalent problem of minimisation of the Mahalanobis distance between the unitary direction vector X and the blob of spatial covariance matrix Σ_X. The solution of the problem is then the vector X minimising the distance $D_M(X, B) = X \cdot {\Sigma_X}^{-1} \cdot X^T$ under the constraint $\|X\| = 1$. The inversion of Σ_X is not a penalty since it is also used during the Expectation step of the blob fitting process.

Starting with the direction estimate from the last frame, the distance $D_M()$ is evaluated on 4 direction vectors, sampled by rotating the initial estimate around the two main rotation axis. If we represent the direction vectors with quaternions, then the 4 samples mentioned above are easily obtained by multiplying the current estimate with pre-computed quaternions representing rotations around the axis y and z. For each of the 4 samples, the distance $D_M()$ is evaluated and the algorithm is iterated using the best sample as new initial guess. A schematic illustration of this process is presented in Figure D.1.

The chosen angle of rotation has a direct influence on the convergence rate. In practice, an angle equal to $\frac{\pi}{36}$ gives a good constant convergence rate. However, to improve even more on accuracy and performance, the angle of rotation is chosen hierarchically in a coarse to fine scheme. In practice, 2 levels of accuracy with rotation angles taken successively at $\frac{\pi}{12}$ and $\frac{\pi}{50}$ give a very good final result in only a few iterations.

Once the main direction is computed, it can be useful to find the remaining axes of the blob. The second axis is computed in a similar way than the main one. The problem is however more constrained since the second axis is perpendicular to the main direction: convergence is even faster. The third axis is immediate to compute as the cross product of the first two.