Chapter 11

Colour-based Tracking

This chapter describes the implementation of the Colour-based Tracking method. The first section gives an overview of how colour information can be used to track objects by means of statistical methods. This is followed by a description of how colour is modelled using a Gaussian Mixture Model and how the Expectation-Maximisation (EM) algorithm is used to fit the model to an object. The tracking procedure is described next, together with how the object mixture model is updated to handle variations in the appearance of the object. The chapter finishes off with some results obtained from this method.

11.1 Colour Models

It was shown in the previous chapter (§10.5) that colour information is a very useful feature for representing objects. Colour is robust to changes in the appearance of an object such as deformations, rotation, and scaling. And by using certain types of colour spaces, the colour information about an object can be made independent of any light variations in the scene[1] – such as the HSV colour space, which separates the chromaticity values from the illumination value (others are given in §8.7.2). In addition, since colour is a global property of the object, it is also robust to partial occlusion[2]. Another advantage to using colour is the ease with which it can be extracted from an image (compared to finding other features like corners, edges, etc.)

[1] This property is referred to as *colour constancy* [HORP99].

[2] This may not be strictly true for multi-coloured objects, since during partial occlusion, one or more of the colours may be hidden from view, leaving only some of the colours visible.

Objects in the world are normally multi-coloured, either consisting of parts made up of individual colours or else having a multi-coloured texture. Therefore representing an object by single values such as the average colour is not a useful model. One possibility is to use a histogram *H*, made up of a certain number N of bins, like the model used in §10.3.4 for blob tracking. If the histogram is normalised so that $\sum_{i=1}^{N} H(i) = 1$, then the histogram can be interpreted statistically[3] as a discrete probability density function, where $H(i)$ gives the probability $p(c)$ that a given pixel of the object takes the colour *c*. An example is shown in Figure 11.1.

When using histograms as colour models for an object, there is the question of how many bins to use. Using a large number of bins gives a higher resolution but at the cost of being more prone to error from camera noise – since for error ε, it becomes more probable that colour *c*, affected by noise $c \pm \varepsilon$, falls into one of the neighbouring bins *c-1* or *c+1* instead of in *c*. Using a small number of bins leads to a poor probability resolution (colours are ‘averaged’ out) as well as increasing the impact of quantisation errors.

Another way of statistically modelling the object’s colour information is to explicitly fit a probability distribution to the colour data, based on the assumption that the chosen distribution is a close approximation of the real one that generated the colour data. So in this case, the colour model of the object consists of some set of parameters θ that describe the chosen distribution. This method is referred to as *parametric statistical colour modelling*, while the histogram-based method is an example of a *non-parametric statistical colour modelling* [ELGA02].

For this application, the parametric approach was adopted. An advantage of using the parametric approach is that since the distribution is known, established mathematical techniques from the field of statistics can be applied to it. A disadvantage is the *assumption* that the chosen distribution is the correct one for the underlying data.

[3] That is, the object’s colour data can be seen as arising from a random variable defined in some colour space.

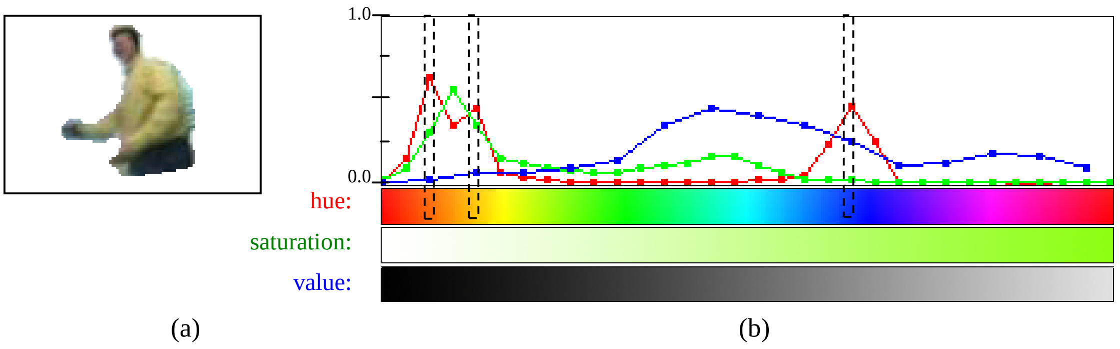


Figure 11.1: Non-parametric statistical colour model based on a HSV-colour histogram. (a) The object shown is from the PETS-ICVS dataset, frame #12775; (b) The object is multi-coloured and therefore the distribution is multimodal (evident from the three main peaks highlighted for the hue histogram – red curve).

11.2 Mixture Models

As mentioned in the previous section, objects are normally multi-coloured and therefore their histogram will be *multimodal*. The common way of handling multimodal data is through the use of *mixture distribution models* [MCLA97 §1.4.3] or *mixture models* for short. A mixture model $q(x)$, for some vector x, is defined as:

$$q(x) = \sum_{i=1}^{M} \pi_i p_i(x) \tag{11.1}$$

which is a weighted sum of M individual probability distributions $p_i(x)$, called *component distributions*, and π_i are called the *mixing weights*, with the condition $\sum_{i=1}^{M} \pi_i = 1$. Therefore, the parameters of the model $q(x)$ consist of the set of weights π_i and the individual parameters θ_i of the component distributions:

$$\theta = (\pi_1, \ldots, \pi_M, \theta_1, \ldots, \theta_M) \tag{11.2}$$

11.2.1 Gaussian Mixture Models

For representing colour data, the individual distributions are often taken to be Gaussian (Normal) distributions [RAJA98; GROV98; OR00]. In this case, the mixture is called a *Gaussian Mixture Model (GMM)* or *Mixture of Gaussians (MoG)*. Theoretically, a GMM with an infinite number of components can model any data distribution. Since colour data is multi-dimensional (usually 2 or 3 dimensions), the Gaussian components N_i are multivariate distributions [BILM98]. If $x = (x_1, x_2, \ldots, x_d)$ is the colour vector with dimensions d, then N_i is defined by:

$$p_i(x) = N_i(\mu_i,\Sigma_i) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma_i|}} e^{-\frac{1}{2}[x-\mu_i]^T \Sigma_i^{-1}[x-\mu_i]} \qquad (11.3)$$

where μ_i is the mean of the distribution, and Σ_i is the covariance matrix (having determinant $|\Sigma_i|$):

$$\Sigma_i = \begin{bmatrix} (\sigma_1)^2 & \sigma_{1,2} & \cdots & \sigma_{1,d} \\ \sigma_{1,2} & (\sigma_2)^2 & \cdots & \sigma_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1,d} & \sigma_{2,d} & \cdots & (\sigma_d)^2 \end{bmatrix} \qquad (11.4)$$

with $(\sigma_j)^2$ being the variance of the j'th colour component and $\sigma_{j,k}$ is the covariance between the j and k'th colour components. From (11.4) it can be seen that Σ_i is a symmetric matrix (that is, $\Sigma^T=\Sigma$), since $\sigma_{j,k} = \sigma_{k,j}$. In addition, the multivariate Gaussian distribution requires the covariance matrix to be *positive definite,* that is:

$$x^T\Sigma_i x > 0 \quad \forall x \in \Re^d, x \neq 0 \qquad (11.5)$$

The parameters θ of the GMM are now given by:

$$\theta = (\pi_1,...,\pi_M,\mu_1,\ldots,\mu_M,\Sigma_1,\ldots,\Sigma_M) \qquad (11.6)$$

In the case of the OmniTracking application, colour data is expressed in the HSV colour space and, as mentioned in §11.1, only the hue and saturation values are used for the object's probability colour model, to achieve a level of colour constancy. Therefore, the mixture model used for the application consists of bivariate Gaussian distributions. If colour values in the HS-space are expressed as $x = (x_h,x_s)$, then for $N_i(\mu,\Sigma)$ we have: $\mu = (\mu_h,\mu_s)$, $\Sigma = \begin{bmatrix} \sigma_h^2 & \sigma_{hs} \\ \sigma_{hs} & \sigma_s^2 \end{bmatrix}$, $\Sigma^{-1} = \frac{1}{|\Sigma|}\begin{bmatrix} \sigma_s^2 & -\sigma_{hs} \\ -\sigma_{hs} & \sigma_h^2 \end{bmatrix}$

and $|\Sigma| = \sigma_h^2\sigma_s^2 - {\sigma_{hs}}^2$. Substituting the values into (11.3):

$$N_i = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2|\Sigma|}\begin{bmatrix} x_h-\mu_h & x_s-\mu_s \end{bmatrix}\begin{bmatrix} \sigma_s^2 & -\sigma_{hs} \\ -\sigma_{hs} & \sigma_h^2 \end{bmatrix}\begin{bmatrix} x_h-\mu_h \\ x_s-\mu_s \end{bmatrix}}$$

and simplifying the quadratic in the exponent, yields:

$$N_i = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2|\Sigma|}\left[(x_h-\mu_h)^2{\sigma_s}^2 - 2(x_h-\mu_h)(x_s-\mu_s)\sigma_{hs} + (x_s-\mu_s)^2{\sigma_h}^2\right]} \qquad (11.7)$$

Figure 11.2 shows how a GMM models an object's colours. Colours are expressed in the HS-space in polar form with hue x_h defining the angle and saturation x_s runs from 0 at the boundary to 1 in the centre, as indicated in Figure 11.2(a). Three component distributions are used, fitted automatically, and as can be seen from Figure 11.2(c), the components correspond approximately to the 3 main peaks of the histogram (at least for the hue) highlighted in Figure 11.1.

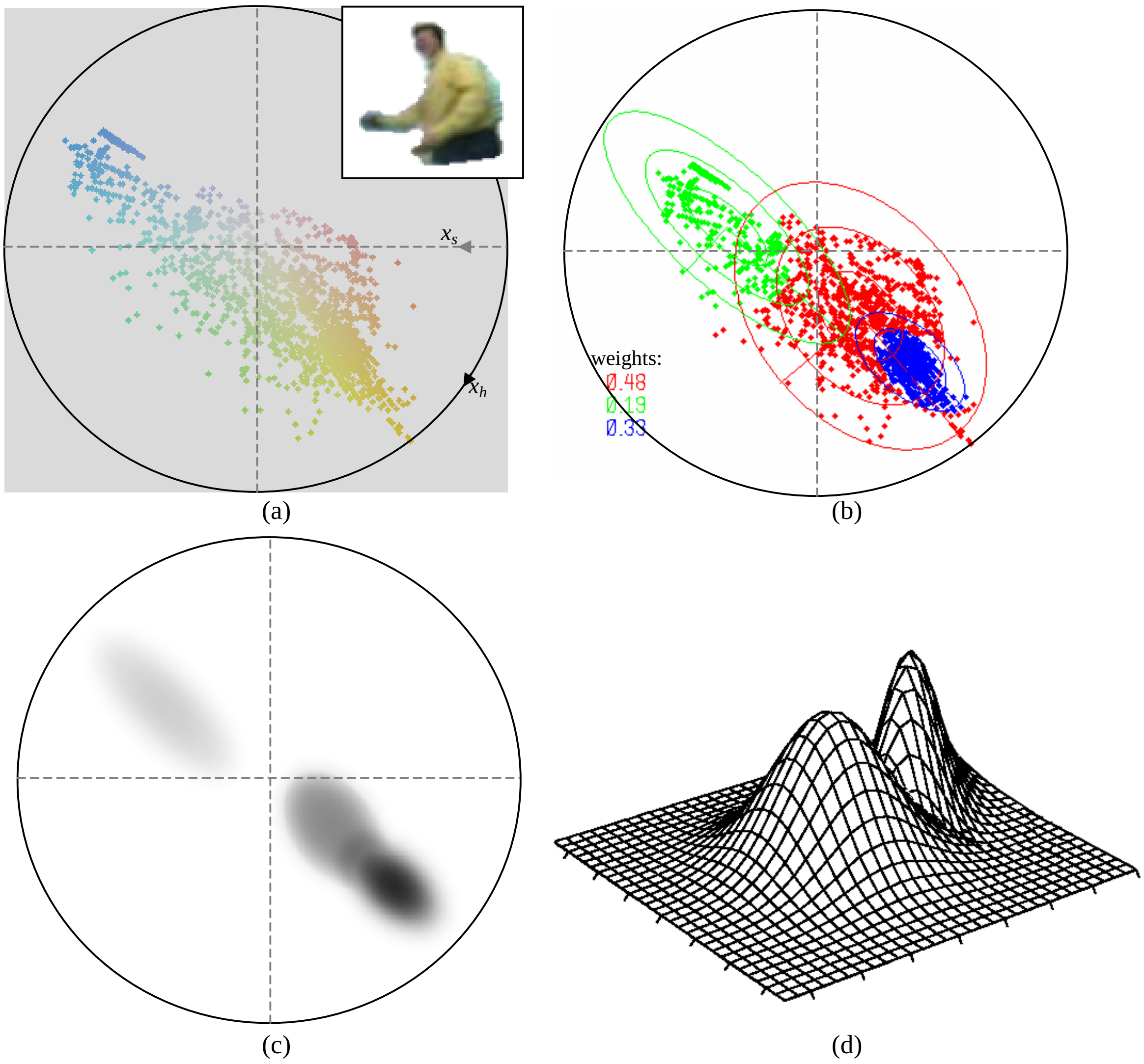


Figure 11.2: Parametric statistical colour model based on a Gaussian Mixture Model. (a) Colours of the object (PETS-ICVS dataset, frame #12775) plotted in the HS-space; (b) A 3-component Gaussian mixture fitted to the colour data; (c) The probability map of the mixture $q(x)$; (d) A 3D version of the two lower components of (c) as seen from near the origin.

The components in Figure 11.2(b) are shown as 3 concentric ellipses, with each ellipse representing standard deviations of σ, 2σ, and 3σ respectively from the mean. These ellipses are called *contours of constant probability density* and can be used to define a threshold for each component, beyond which the probability is set to 0 (to avoid working with very small numbers)[4]. The axes of the *kσ*-ellipse are given by:

$$\mu \pm k\sqrt{\lambda_1}e_1 \quad \text{and} \quad \mu \pm k\sqrt{\lambda_2}e_2 \tag{11.8}$$

where λ_1,λ_2 are the eigenvalues and e_1,e_2 are the eigenvectors of Σ:

$$\lambda_i = \left(\sigma_1^2 - \sigma_2^2\right)^2 \pm 4\sigma_{12}^2 \qquad e_i = \frac{1}{\sqrt{\left(\lambda_i - \sigma_2^2\right)^2 + \sigma_{12}^2}}\left(\lambda_i - \sigma_2^2, \sigma_{12}\right) \tag{11.9}$$

The symmetric, positive definite property of Σ ensures that $\lambda > 0$. Figure 11.2(c) shows the probability of the full mixture *q*, so the probability that an object's pixel has colour *x* is given by $q(x_h,x_s)$, calculated using (11.7) and (11.1).

11.3 Tracking with Colour Models

Given a Gaussian mixture model *q*(*x*) describing the colour information of an object, the location of the object at time *t* can be found by simply computing the probability for each pixel of the image (that is, computing the probability that the colour of the pixel was generated by the GMM) and finding the region in the image with maximum probability. Figure 11.3 shows how this is applied to an image using the colour mixture model of Figure 11.2, where the position of the object can be easily identified. Some areas of the image have a constant value – this is because of the application of a 3*σ* threshold for each of the distribution components.

In this example, the object's main colour (yellow) is similar to that of parts of the background (yellowish-white colour of the walls and roof). This results in non-zero probability values being assigned to most of the image. Although these are low probability values, they can make the location of the maximum-probability region harder to find. One way of eliminating this is to use the result from the motion detection phase and compute the probability only for foreground pixels [PÉRE02]. The result is shown in Figure 11.4(a). An advantage of this method is that the tracker does not need to calculate the probability for each image pixel, but only for those labelled as foreground.

[4] Note that the exponent term of the multivariate Gaussian distribution has the general form $x^T(\Sigma^{-1})x$ (which becomes a quadratic equation in *x* for the bivariate case). This expression is the *Mahalanobis distance* [MCLA97 §2.6] with the covariance matrix serving to scale each dimension of *x* and rotate the axes to better fit the colour data, which is normally skewed and has different variability in each dimension – hence the ellipsoids $x^T(\Sigma^{-1})x=c^2$.

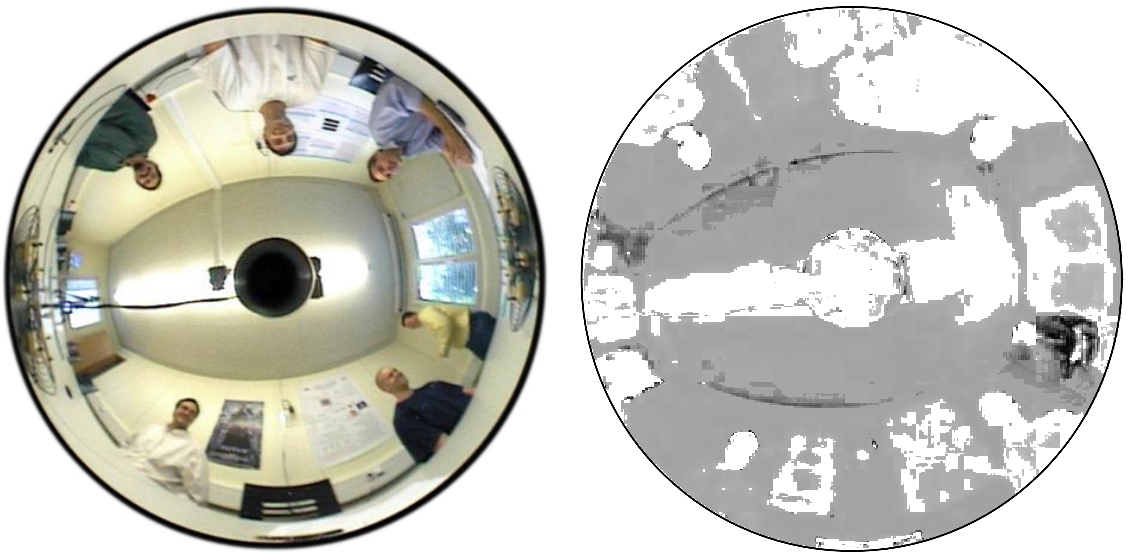

(a) (b)

Figure 11.3: Locating an object by finding the region with highest colour probability. (a) The source image from the PETS-ICVS dataset, frame #12790; (b) Probability values computed for each pixel of the image (black = highest probability). The colour GMM used is the one shown in Figure 11.2 and fitted to the object 15 frames earlier.

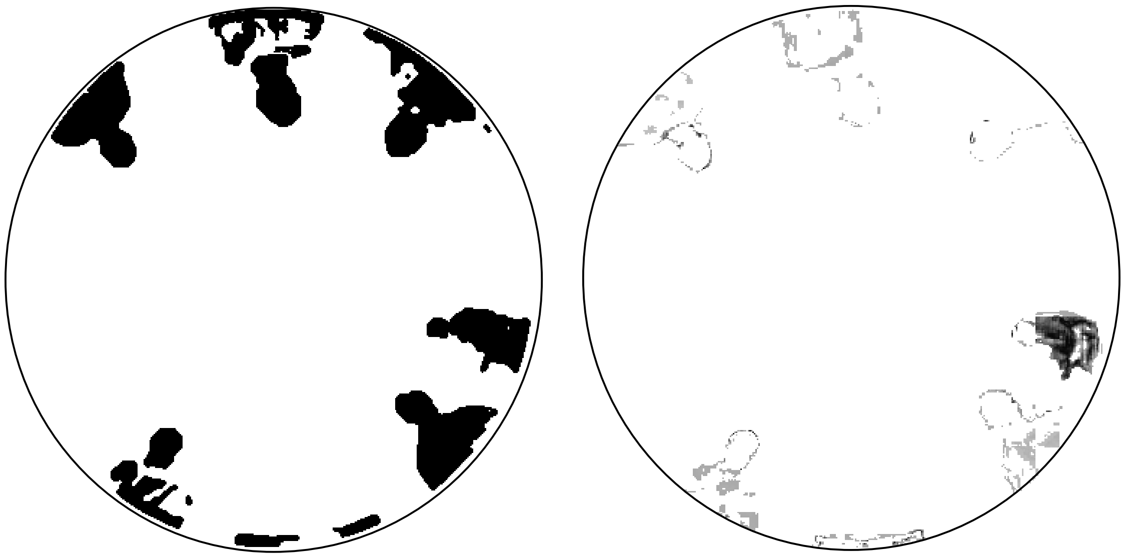

(a) (b)

Figure 11.4: Combining colour tracking with motion detection. (a) The result from the background subtraction algorithm (foreground pixels shown in black); (b) Combined result.

In addition, temporal constraints could be used to further restrict the area of the image to be searched, as the inter-frame movement exhibited by an object should be limited. And the object's size should also change slowly. These constraints also help to eliminate the possibility of the tracker 'jumping' from one object to another one, which happens to have the same colour distribution [RAJA98].

Given that the object's position at time *t-1* was at image position μ_{t-1}, then a search window is defined on the image, centred on this position. From within this search window, only those pixels that belong to the foreground are used (as determined by the motion detection algorithm). For this set of foreground pixels *R*, having position *l*, the probability of their colour c_l is calculated using the mixture model *q*. Then the new position for the object is estimated to be:

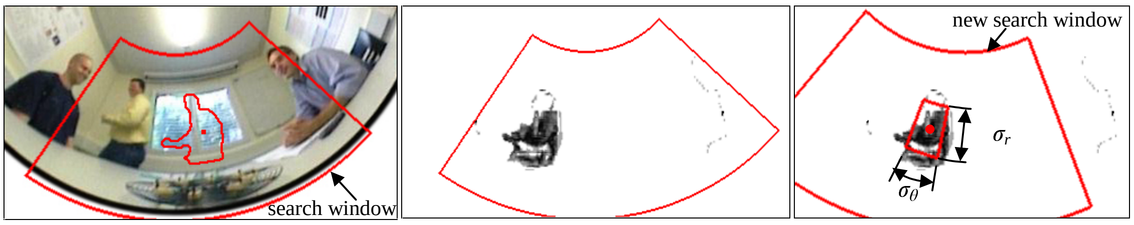
$$\mu_t = \mu_{t-1} + \frac{1}{q(R)}\left(\sum_{l \in R} q(c_l)(l - \mu_{t-1})\right) \tag{11.10}$$

where $q(R) = \sum_{l \in R} q(c_l)$ is the total probability of the colour model within *R*. The extent of the object is expressed as:

$$\sigma_t^{\,2} = \frac{1}{q(R)}\left(\sum_{l \in R} q(c_l)\left[(l - \mu_{t-1}) - \mu_t\right]^2\right) \tag{11.11}$$

The new position μ_t and the object extent σ_t can then be used to define the next search window at time *t+1*, as shown in Figure 11.5 below.

As in the case of the bounding box used for blob tracking in §10.2, the object extent is specified in polar coordinates (θ, r), the range in azimuth θ and the range in the radial distance from the image centre *r*. Therefore, pixel positions *l* in (11.10) and (11.11) are given as (θ, r) and σ is expressed as σ_θ and σ_r.



(a) (b) (c)

Figure 11.5: Using temporal constraints for the search window. (a) Search window derived from object's position at time *t-1*; (b) Probability for foreground pixels within the search window; (c) The new object's position, extent and the search window to be used at time *t+1*.

11.4 Colour Model Learning

One of the issues with mixture models (and parametric models in general) is determining explicitly what model to use for (best explains) a particular data set. In the case of the Gaussian mixture, the model is described by the parameter set θ, where

$$\theta = (\pi_1, ..., \pi_M, \mu_1, \ldots, \mu_M, \Sigma_1, \ldots, \Sigma_M) \qquad (11.12)$$

In the majority of cases, the parameter set θ is not known beforehand, but has to be determined automatically from the data itself. This process is known as "learning" the model.

An example is learning the parameters of the 3-component mixture model used for the object shown in Figure 11.2. The mixture model $q(x)$ defines the probability that a particular colour value x is generated by the object's model. But the mixture model depends on the choice of θ, so it can be written as $q(x|\theta)$, that is, the probability of generating colour x given the parameter set θ. If X is the dataset of colour values of the object in question, where $X = \{x_1, x_2, \ldots, x_N\}$, with N being the number of pixels of the object, then $q(X|\theta) = q(x_1 x_2 ... x_N|\theta)$. And since the colour values are assumed to be independent (and identically distributed), this gives:

$$q(X|\theta) = q(x_1|\theta)\, q(x_2|\theta) \ldots q(x_N|\theta) = \prod_{i=1}^{N} q(x_i|\theta) \qquad (11.13)$$

Learning the parameters θ from a dataset X, can be viewed as keeping X fixed in (11.13), and letting θ vary over the space Ω of all possible parameter values. Finding θ that yields the highest probability in (11.13) is equivalent to finding θ that best describes the given dataset X. This is called the process of finding the *maximum likelihood estimate*, and (11.13) can be written in terms of the *likelihood function L* (the likelihood of parameter set θ given the data) [BILM98]:

$$L(\theta|X) = q(X|\theta) = \prod_{i=1}^{N} q(x_i|\theta) \qquad (11.14)$$

and the maximum likelihood process is then described by:

$$\theta_{\max} = \arg\max_{\theta} L(\theta|X) \qquad (11.15)$$

which means, finding $\theta = \theta_{\max} \in \Omega$ that maximises the likelihood function L.

A common technique in statistics is to maximise the logarithm of $L(\theta|X)$ instead of working directly with $L(\theta|X)$ as this simplifies things (eliminates the log-of-product expression) and it is easy to see that a value of θ that maximises L will also maximise $\log L$:

$$\log L(\theta|X) = \log\left(\prod_{i=1}^{N} q(x_i|\theta)\right) = \sum_{i=1}^{N}\left[\log q(x_i|\theta)\right] \qquad (11.16)$$

Substituting the mixture model defined by (11.1) into (11.16) gives:

$$\log L(\theta|X) = \sum_{i=1}^{N}\left[\log q(x_i|\theta)\right] = \sum_{i=1}^{N}\left[\log\left(\sum_{m=1}^{M}\pi_m p_m(x_i|\theta_m)\right)\right] \qquad (11.17)$$

which is not possible to solve analytically because of the log-of-sum expression [FORS02 §18.1.1]. Therefore for colour mixture models, a different approach is required for maximising the log-likelihood equation, and hence learning the model.

11.4.1 The Expectation-Maximisation (EM) Algorithm

A popular technique for finding the maximum likelihood estimate is the *Expectation-Maximisation (EM)* algorithm [MCLA97]. EM is a general-purpose and powerful algorithm used for maximising likelihood functions, especially in the case of *incomplete data problems*. The algorithm was first formally presented in a paper by A. Dempster, N. Laird, and D. Rubin in 1976, and has been widely used in diverse fields such as astronomy, synthetic aperture radar, and medical imaging. It has also been the subject of much research and [MCLA97] quotes a bibliographic list about the EM algorithm with more than a thousand papers referenced.

When used for incomplete data problems, the EM algorithm simplifies the likelihood function by taking into account the *missing data*. Normally, incomplete data means that elements of some of the data vectors of dataset X cannot be observed. But the term 'missing data' can also be used to represent model parameters whose value is unknown or by artificially adding some variables into the model which are assumed to be hidden [SCHA97 §3.2].

The latter definition to missing data (that is, hidden variable) is used for colour mixture models. Given a mixture model $q(x)$, made up of M component distributions, and dataset $X = \{x_1, x_2, \ldots, x_n\}$ being the object's colours, it is assumed that each vector x_i is generated by one of the component distributions of the mixture model. An artificial variable y_i is introduced, which specifies which of the M distributions generates colour value x_i [BILM98]. That is, for x_i:

$$q(x_i) = \sum_{m=1}^{M} \pi_m p_m(x_i) \quad \text{simplifies to} \quad q(x_i) = \pi_{y_i} p_{y_i}(x_i) \qquad (11.18)$$

where y_i takes a value between [1..M]. The set of these artificial variables is denoted by $Y = \{y_1, y_2, \ldots, y_N\}$, their values are assumed to be hidden and so form the 'missing data'. And the dataset X (the set of observable values) is considered to be 'incomplete' (because the y_i variables are not part of the observable values). Combining the two into one set gives the 'complete' data set Z=(X,Y) and the log-likelihood function is now:

$$\log L(\theta \mid Z) = \log L(\theta \mid X, Y) = \log\big(q(X, Y \mid \theta)\big) = \sum_{i=1}^{N} \left[\log\big(q(x_i \mid y_i) q(y_i)\big)\right] \qquad (11.19)$$

and substituting (11.18) into (11.19) gives:

$$\log L(\theta \mid Z) = \sum_{i=1}^{N} \left[\log\big(\pi_{y_i} p_{y_i}(x_i \mid \theta)\big)\right] \qquad (11.20)$$

which means that working with the complete-data log-likelihood log(L(θ|Z)) is much simpler than working with the incomplete-data log-likelihood log(L(θ|X)), because the log-of-sum expression has disappeared by virtue of (11.18) and the introduction of the y_i variable (the missing data). The EM algorithm, described below, uses the simplified complete-data log-likelihood for learning the mixture parameters.

The EM algorithm is an iterative algorithm that basically performs the following two steps:

1. "Guesses" the missing data Y by performing an averaging (calculates the expectation). This is called the *expectation step,* **E-step** for short.
2. Estimates (maximises) the parameters θ by using the guessed values. This is called the *maximisation step*, **M-step** for short.

The steps are repeated to improve the estimated value of the parameters θ.

The algorithm starts with some initial values for the parameters, denoted by θ_0. During the E-step, the EM algorithm finds the mathematical expectation value Q of the complete-data log-likelihood function log(L(θ|X,Y)), using parameters θ_{t-1} obtained from the previous iteration of the algorithm. For the purpose of expectation calculation, function L can be viewed as being constant in X and in θ_{t-1}, but variable in Y – hence the idea of finding the missing data Y.

$$Q(\theta, \theta_{t-1}) = E\left[\log\big(q(X, Y \mid \theta)\big) \mid X,\ \theta_{t-1}\right] \qquad (11.21)$$

The expectation value $Q(\theta,\theta_{t-1})$ is an expression in θ that is evaluated during the E-step in terms of θ_{t-1}. The M-step then consists of maximising the expectation Q:

$$\theta_t = \arg\max_{\theta} Q(\theta,\theta_{t-1}) \qquad (11.22)$$

The actual expressions used for the expectation calculation and for argument maximisation in (11.21) and (11.22) depend on the probability distribution model being used. For this reason, although it is called the EM 'algorithm', it can be seen as being more of a general-purpose process rather than an algorithm in the strictest sense of the word [MCLA97].

For the particular case of the bivariate Gaussian mixture model used for this application, the equations of the EM algorithm are given below[5].

E-Step: *(11.23)*

$$p(y_i \mid x_i,\theta^{t-1}) = \frac{\pi_{y_i}^{t-1} p_{y_i}\left(x_i \mid \theta_{y_i}^{t-1}\right)}{p(x_i \mid \theta^{t-1})} = \frac{\pi_{y_i}^{t-1} p_{y_i}\left(x_i \mid \theta_{y_i}^{t-1}\right)}{\sum_{m=1}^{M} \pi_m^{t-1} p_m\left(x_i \mid \theta_m^{t-1}\right)}$$

where $p_m(x_i \mid \theta_m)$ is calculated using (11.7)

M-Step: *(11.24)*

$$\pi_m^t = \frac{1}{N}\sum_{i=1}^{N} p(m \mid x_i,\theta^{t-1})$$

where $p(m \mid x_i,\theta)$ is calculated using the E-step result with $y_i=m$

$$\mu_m^t = \frac{\sum_{i=1}^{N} x_i \; p(m \mid x_i,\theta^{t-1})}{\sum_{i=1}^{N} p(m \mid x_i,\theta^{t-1})}$$

$$\Sigma_m^t = \frac{\sum_{i=1}^{N} p(m \mid x_i,\theta^{t-1})\left(x_i - \mu_m^t\right)\left(x_i - \mu_m^t\right)^T}{\sum_{i=1}^{N} p(m \mid x_i,\theta^{t-1})}$$

The EM algorithm has several important mathematical properties, such as monotonicity, where the likelihood function satisfies the condition $L(\theta^t) \geq L(\theta^{t-1})$, and the guarantee of convergence to at least a local maximum (and possibly a global

[5] The derivation of these equations is not given here. Details can be found in [BILM98].

maximum) in the parameter space [MCLA97 §§3.2, 3.4]. These two properties can be used to test for the termination condition of the EM algorithm by checking the complete-data log-likelihood, calculated using (11.20). When the algorithm converges, the complete-data log-likelihood does not change any longer. Convergence is normally quite rapid and is achieved with a few number of iterations.

The EM algorithm offers other advantages such as:

- its strong statistical basis and theoretical guarantees on optimality,
- robustness to noise and highly-skewed data, and
- its simplicity of implementation.

Among its disadvantages, one can find that:

- the algorithm may converge to a poor local maximum,
- although the EM algorithm converges rapidly, the actual number of iterations needed for a particular dataset is unknown,
- computations are unstable when the variances of the probability mixture model are close to 0, and
- the algorithm is hard to initialise, that is, to estimate the initial values of the parameters θ_0.

Figure 11.6 shows the partial results of the EM algorithm while being used to learn the colour model of the object shown in Figure 11.2. The model is a 3-component Gaussian mixture, and a total of 17 iterations were required for this particular case, with the change in the log-likelihood value (shown in Figure 11.7) tested for convergence with a precision of 3 decimal places.

11.4.1.1 Selecting the Initial Parameters

As mentioned further above, one of the problems of the EM algorithm is in deciding what initial parameter values θ_0 to use. Normally, the parameters are assigned random values. But this suffers from the problem that if the initial choice of values happens to be an incorrect one, then the EM algorithm converges to a poor local maximum. This is especially important for mixture models because as the number of components increases, so does the number of maxima in the log-likelihood function space[6]. Some applications use the K-means clustering method to get a good initial estimate for the parameters θ_0, but this can be expensive for real-time applications.

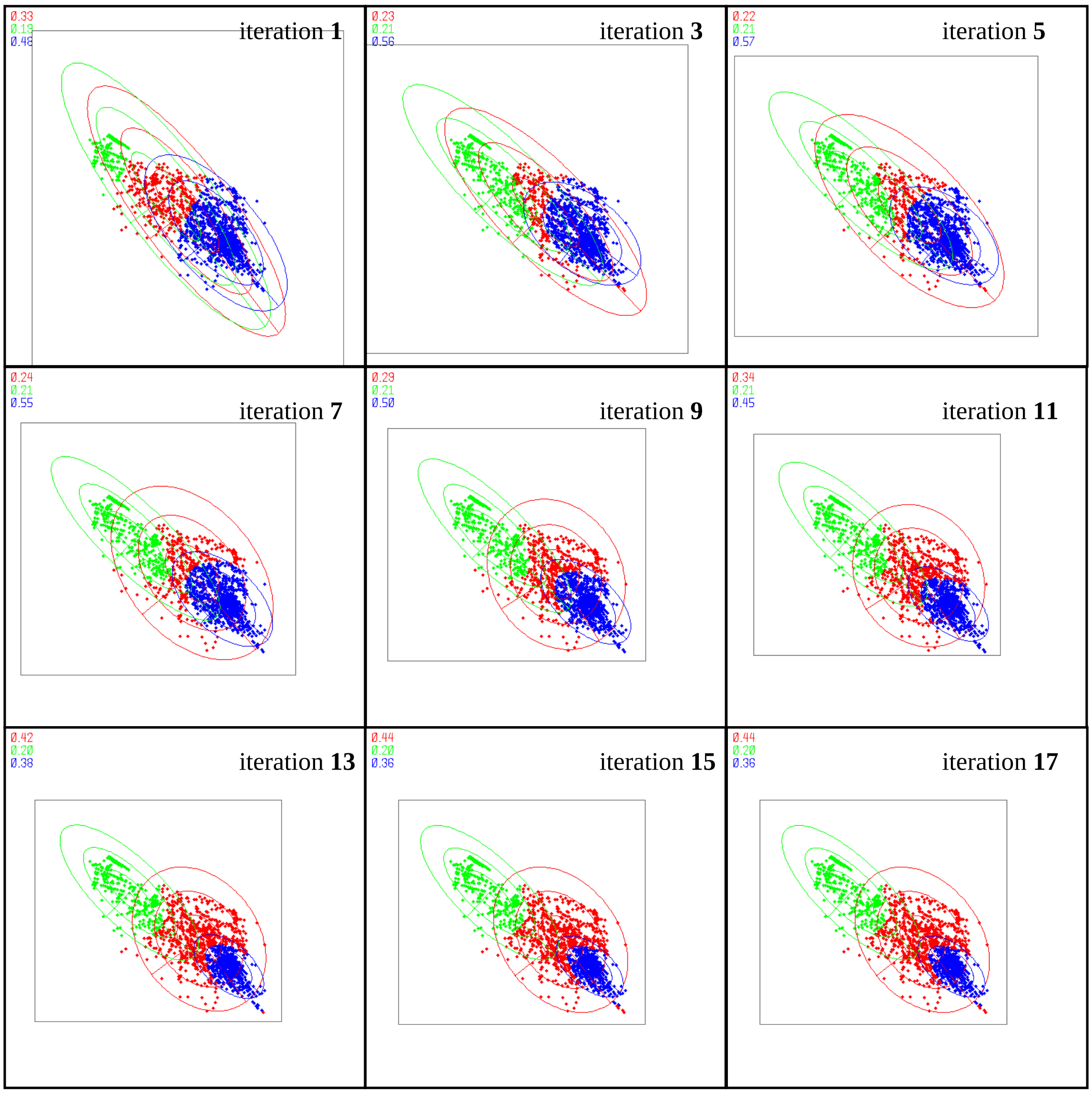


Figure 11.6: Learning a colour model using the EM algorithm. Partial results from some of the iterations are shown. A total of 17 iterations were required for this model. The colour data is for the object of Figure 11.2.

[6] It is easy to see that there are several trivial maxima in the log-likelihood space of mixture models. For example, the solutions for the first 2 components (θ_1,θ_2) and (θ_2,θ_1) occur at separate maxima in the likelihood space, but as far as the mixture model is concerned the two solutions are equivalent because the order of the components within the mixture is not important. Apart from these trivial maxima, there are also other maxima, the number of which increases as more components are added to the mixture model.

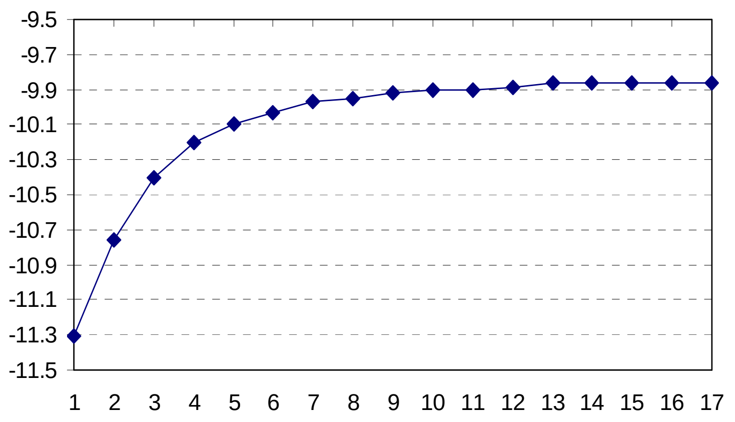


Figure 11.7: Convergence of the log-likelihood value of the colour model of Figure 11.6.

In the case of the OmniTracking application, the mixture parameters are initialised with colour values taken from the dataset, that is, the object's pixels. A pixel is selected from the object one for each component, and the component's mean colour values are set to the pixel's colour values. In addition, the variance of each component is set to a large value, while the covariance is initially set to 0. Since neighbouring pixels tend to have similar colours, the pixels selected must be a certain spatial distance from each other. When tested on the PETS datasets, this initialisation method performed satisfactorily, in the sense that the initialisation values (colours) for the 3 components were distinct from each other.

11.5 Colour Model Update

The appearance of an object changes over time due to several factors such as object motion, rotation, light variations, etc. (see §9.2 for more details). As a result, any model describing the object starts getting out-of-date unless it is updated on a regular basis. This also applies to colour models (on a long-term basis), even though colour is usually more robust to many of these changes (§11.1).

In §11.4.1 it was shown that the EM algorithm is a very powerful method for learning the initial colour models of objects. This may lead one to think of 'updating' an object's colour model by re-using the EM algorithm of §11.4.1 to fit a new model over the old one – in other words, by 're-learning' the model. But using the EM algorithm for model update can prove to be a computationally intensive process and not suitable for real-time applications. Under most circumstances, changes in the colour model of an object are small, which means that only slight adjustments are needed; using the EM algorithm means duplicating most of the work done to learn the previous model.

From its first introduction in 1976, the EM algorithm has been the subject of numerous research and many different variants of the EM algorithm[7] have been proposed [NEAL98; NG03; THIE01; ORDO02]. Some of these variants were designed with speed in mind, others with the aim of allowing incremental learning of the model, etc. Most of these variants can be classified into roughly two categories, those methods that concentrate on improving (modifying) the E-step, and those that improve (modify) the M-step. Generally, variants that deal with the E-step tend to offer the greatest advantages as regards to computational speed, faster convergence rates and incremental behaviour, mainly because of the two steps, it is the E-step that is affected most by the number of data elements present [THIE01].

One such example is the *Incremental EM* (*IEM*) algorithm [NEAL98], which partitions the dataset X into blocks. For each data vector x in a block, the IEM algorithm performs a partial evaluation of the E-step, and at the end of each block a single M-step is calculated. The partial evaluation of the E-step is implemented using *expected sufficient statistics*, provided that the probability model being learned is a subfamily of the *exponential family.*

A set of sufficient statistics S is one where all the information that can be gathered from the dataset X about the parameters θ, can also be obtained from the set of statistics S alone [SCHA97 §3.2]. And a distribution $p(x|\theta)$, with parameters $\theta=(\theta_1,\theta_2,\dots,\theta_k)$, belongs to the exponential family if it can be written as:

$$p(x\,|\,\theta) = f_a(x)\,f_b(\theta)\,e^{\left(\sum_{i=1}^{k} f_c(\theta)t(x)\right)} \qquad (11.25)$$

where $f_a()$, $f_b()$, $f_c()$ and $t()$ are real-valued functions.
and the set $T = \left(t_1(x), t_2(x), \dots, t_k(x)\right)$ is the set of sufficient statistics.

In the case of the bivariate Gaussian mixture model used for this application, the individual components are given by (11.7), where it is quite easy to see that this is an exponential function, with the exponent term consisting of a quadratic equation in

[7] Sometimes, the original EM algorithm is referred to as the *generic EM algorithm* or the *standard EM algorithm*, when compared to the other variants.

vector *x*, and the sufficient statistics for this case are [NG03]:

$$T_1 = \sum_{i=1}^{N} p(x_i \mid \theta), \quad T_2 = \sum_{i=1}^{N} p(x_i \mid \theta) x_i, \quad T_3 = \sum_{i=1}^{N} p(x_i \mid \theta) x_i (x_i)^T \qquad (11.26)$$

Using the above sufficient statistics, the Incremental EM algorithm is given below:

Partial E-Step: *(11.27)*

calculate $p(x_i \mid \theta^{t-1}) = \sum_{m=1}^{M} \pi_m^{t-1} p_m\left(x_i \mid \theta_m^{t-1}\right)$

where $p_m(x_i \mid \theta_m)$ is calculated using (11.7).

accumulate T_1, T_2 and T_3 using (11.26)

M-Step: *(11.28)*

$$\pi_m^t = \frac{T_1}{N}$$

$$\mu_m^t = \frac{T_2}{T_1}$$

$$\Sigma_m^t = \frac{1}{T_1}\left[T_3 - \left(\frac{T_2 (T_2)^T}{T_1}\right)\right]$$

As mentioned before, the partial E-steps are performed on a block of data, with a single M-step evaluated at the end of each block. Then the iterations of the IEM algorithm proceed as follows:

```
For each data block $B$:                                  (11.29)
    For all $x_i \in B$:
        perform partial E-step on $x_i$
    perform M-step at end of block $B$
```

If the M-step were to be evaluated after every partial E-step (that is, size of block *B* is set to 1), then the IEM algorithm is known as the *Online EM* algorithm and this is the fastest of all the EM algorithms. On the other hand, if the M-step is performed after all the data vectors have been read (that is, only one block is used with size *N* equal to that of the dataset), then the IEM algorithm is reduced to the standard EM algorithm (though using sufficient statistics instead of the equations in §11.4.1), and no speed improvements are obtained. But as the size of block *B* gets smaller, the IEM algorithm becomes more susceptible to the order of the data vectors *x* and the algorithm may no longer guarantee convergence to a local maximum. Selecting the size of data block *B* is a compromise between speed and guaranteed convergence. In the case of the OmniTracking application, the pixels of an object in an image frame are considered to constitute a new data block *B*.

11.6 Implementation Issues

The complete high-level algorithm used by the OmniTracking application is given in Figure 11.8 below.

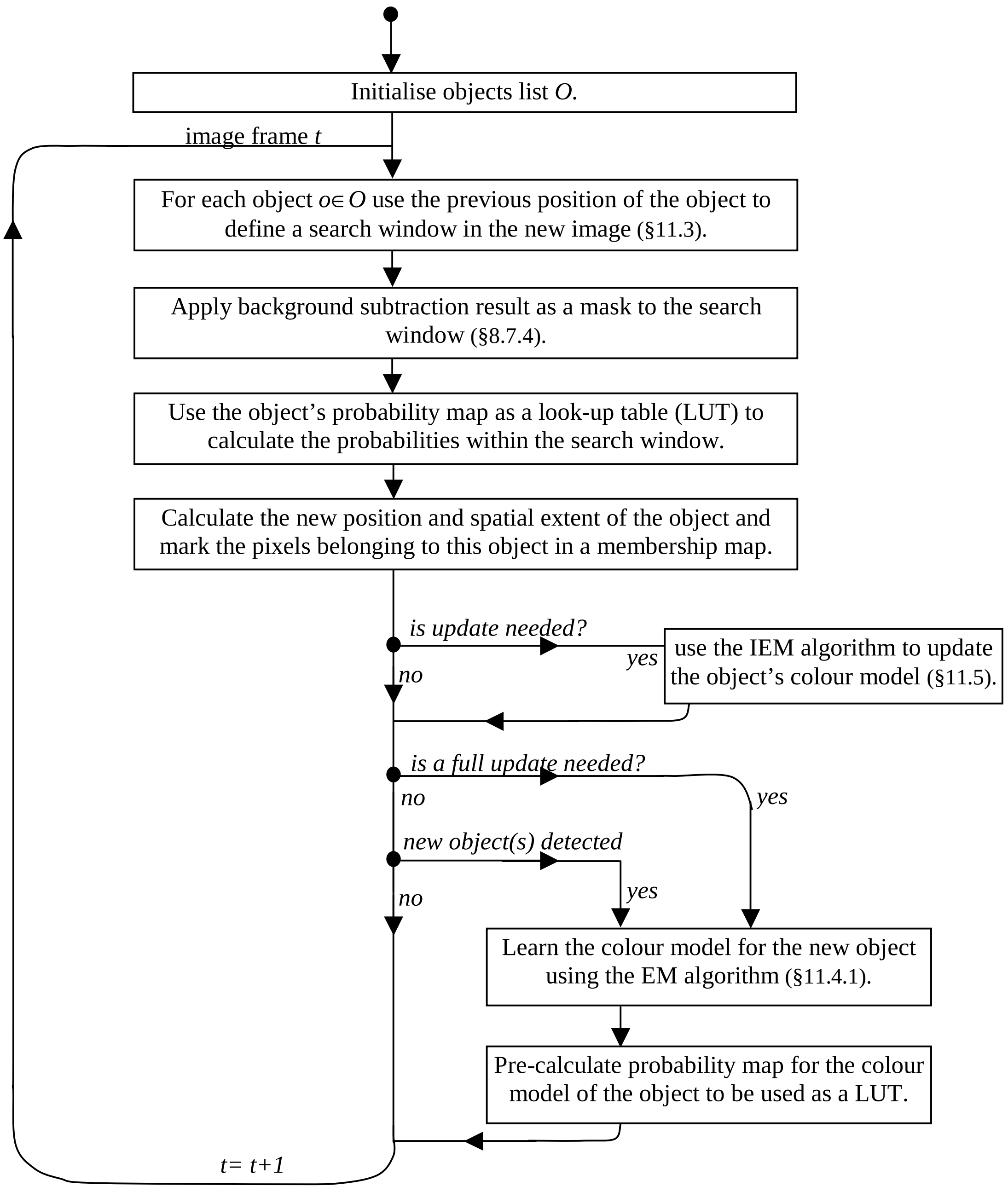

Initialise objects list *O*.
image frame *t*
For each object $o \in O$ use the previous position of the object to define a search window in the new image (§11.3).
Apply background subtraction result as a mask to the search window (§8.7.4).
Use the object's probability map as a look-up table (LUT) to calculate the probabilities within the search window.
Calculate the new position and spatial extent of the object and mark the pixels belonging to this object in a membership map.
is update needed?
yes
use the IEM algorithm to update the object's colour model (§11.5).
no
is a full update needed?
yes
no
new object(s) detected
yes
no
Learn the colour model for the new object using the EM algorithm (§11.4.1).
Pre-calculate probability map for the colour model of the object to be used as a LUT.
t= t+1

Figure 11.8: Colour Tracking Algorithm

As mentioned in §11.2.1, the colour model chosen for the OmniTracking application uses a bivariate Gaussian mixture model with the colour values expressed as vectors of the form (hue,saturation), or *HS* for short, and full covariance matrices are used for the component distributions to allow for correlation between hue and saturation. It was decided to use a mixture of 3 components since this was found to provide a good representation for the PETS datasets. The improvement to the model from adding more components was small compared to the added computational costs. The initial parameter values for the component distributions are chosen from the object's pixels directly as explained in §11.4.1.1.

Calculating the probability value for a pixel is an expensive operation, involving computing the inverse of the covariance matrix, its determinant, and the exponential term, and this has to be done for all of the 3 components of the mixture. Therefore, once the mixture model for an object is known, a lookup-table (LUT) is created and the probability values are computed over the range of the HS-space. An example of such a lookup-table, called a probability map, is displayed in Figure 11.9(a), and as indicated in 11.9(b), a 3σ threshold is used for each component. This helps to improve speed, since pixels outside 3σ-ellipses will have zero probability and will be skipped by later processing steps, but more importantly it eliminates small probability values which can be numerically unstable. A bounding box (shown in grey) is also defined for each colour model – *HS* values that fall outside this bounding box will not be looked-up in the probability map.

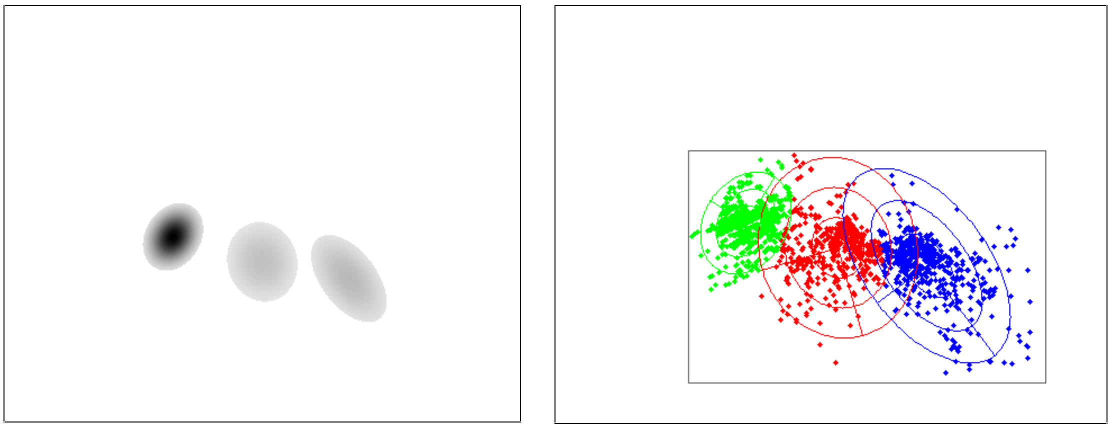

(a) (b)

Figure 11.9: Probability Map used as a look-up table. (a) The probability map; (b) the data points and colour model used for the maps.

As described in §11.3, when searching for an object in frame t, a search window is used which is derived from the object's position in frame *t-1*. This window defines a range in azimuth and a radial range within the omnidirectional image, and its size is determined from the object's spatial extent at *t-1*, expressed in terms of σ_θ and σ_r. This window is defined to have a size of $3\sigma_\theta$ by $3\sigma_r$ and a certain minimum size (defined by the maximum motion expected within a frame) is used to ensure a meaningful size.

While tracking objects in image frame t, the OmniTracking application maintains a global map, called a membership map. Pixels that belong to object O_1 (that is, those pixels that have a non-zero probability within the search window of O_1), are labelled as being members of O_1 in the membership map. This is done for all the objects that are currently being tracked. Any remaining regions of foreground pixels that belong to none of the current objects, are then considered to be new objects, their colour model is created, and they are added to the object list O.

The standard EM algorithm is used for learning the colour model of a new object, but as mentioned in §11.5, the model update is performed using the Incremental EM (IEM) algorithm. Each object has an internal counter that indicates how long it has been (in number of frames) since the last update of the colour model. When this counter reaches a threshold, the colour model is updated using the IEM algorithm and the counter is reset. Currently the update is defined to occur every 15 frames.

The IEM algorithm, in contrast to the standard EM algorithm, is not guaranteed to converge to a local maximum, and it is also susceptible to the order in which colour values are processed. Because of this, errors in the model can start to accumulate over time until the model no longer provides a correct representation of the object's colour information. The OmniTracking program solves this problem by occasionally running the EM algorithm to re-learn the model. By default, this happens every 1000 frames and is user-configurable.

Finally, when objects are no longer visible in the omnidirectional image, the objects are deleted from the list O, using the same deletion rules as the ones described in §10.4.3.

11.7 Results

The colour-based tracking method as implemented for the OmniTracking application produces good results and is especially robust to partial occlusion. In both of the PETS datasets, this method is able to track most of the objects successfully from the time they are first detected in the omnidirectional image until the moment they are no longer in view.

An example of tracking through partial occlusion from the PETS-ICVS dataset is shown in Figure 11.10 below. It can be seen from the probability map (full screen version shown here, that is, without limiting it to the search window and to foreground pixels only) that the colour probability is able to discriminate between the two objects. Unlike in the case of the blob tracking method (§10), the two objects do not affect each other (unless they have a common colour).

Figure 11.11 shows an example from the PETS-ICVS dataset where two objects share a common colour and one is fully occluded by the other, and is recovered successfully after the occlusion events, even though it has quite low chromaticity values (the object is basically white). In this case, the common colour is skin colour, but the mixture component that roughly represents skin colour has very low weight (skin area is small) and on its own, the probability is below the mixture threshold, which explains why the object is lost in images 5 and 10. In image 7, one can also just discern the search window being used to track the object. Although the tracker incorrectly matches the common skin-areas of the other objects with this one, the position of the object (indicated by a red dot in the figure) is still quite accurate.

Finally, in the PETS-2001 dataset, some objects are lost just after they are first detected. After some investigation, this was found to be caused by the small size of the objects in question (mostly <100 pixels), resulting in a poor colour model being generated by the EM algorithm. Figure 11.12 displays one such case. After the object is lost, then it is reacquired as a separate object and is successfully tracked for the rest of the sequence.

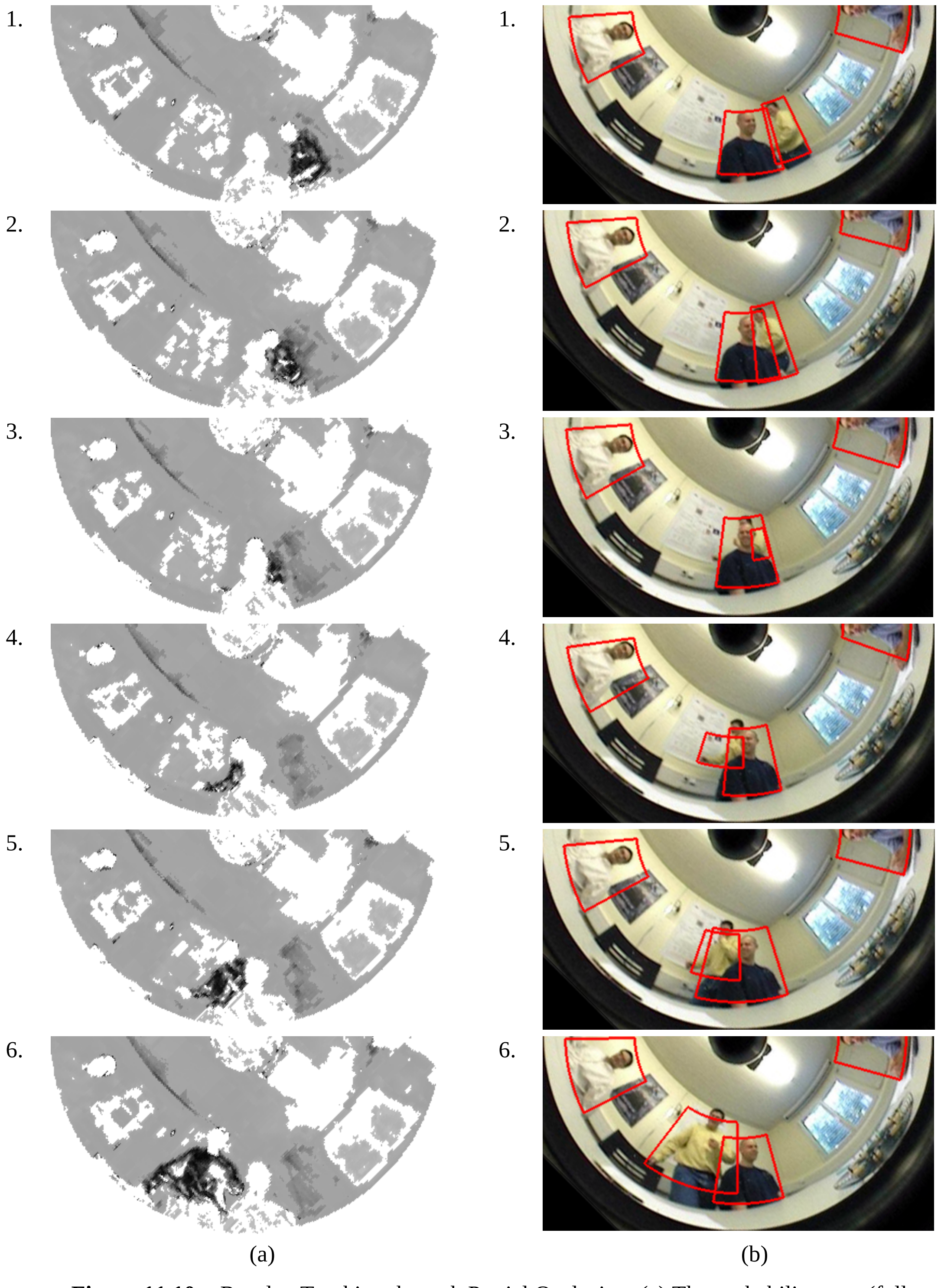


Figure 11.10: Results: Tracking through Partial Occlusion. (a) The probability map (full image version) for the object being tracked; (b) The bounds of the tracked objects.

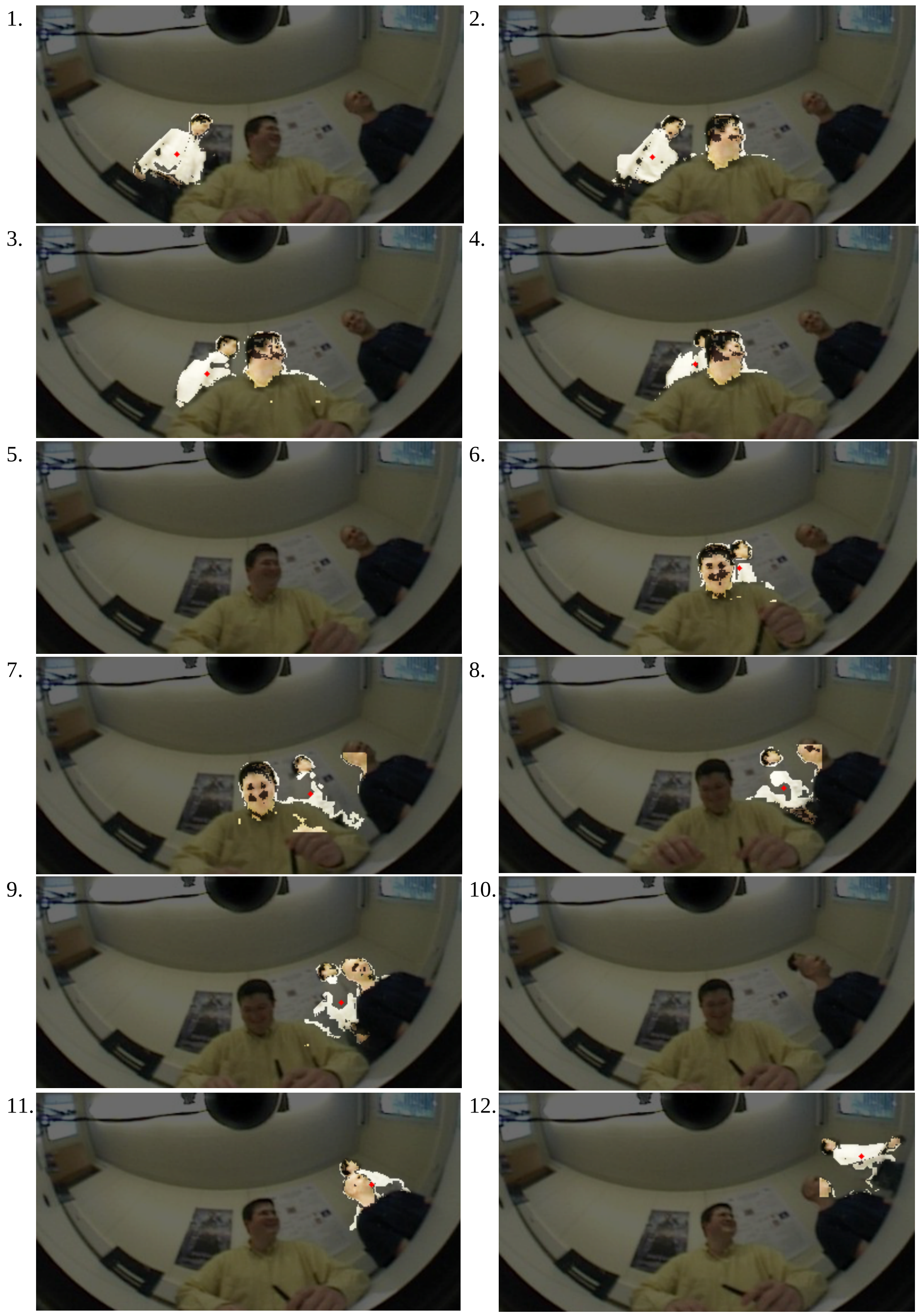
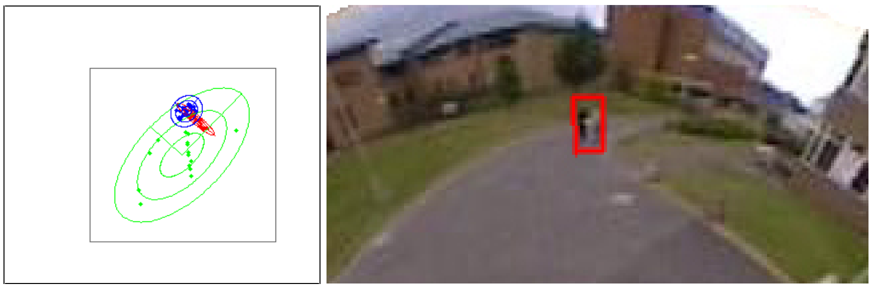


Figure 11.11: Results: Tracking through Occlusion. The pixels belonging to the object are brighter. These images were generated from the membership map (§11.6) that the program uses during tracking.



(a) (b)

Figure 11.12: Results: Poor Colour Model. (a) The colour model built from a size just above the detection threshold of 50 pixels; (b) The object being tracked (PETS-2001 dataset)

11.8 Conclusion

This chapter described how the colour-based tracking method was implemented for the OmniTracking program. The colour information of objects (hue and saturation) is modelled statistically using a mixture model consisting of 3 Gaussian distributions. The Expectation-Maximisation (EM) algorithm is used to automatically learn the colour model from the object's pixel values, and the model is periodically kept up-to-date using a faster version of the EM algorithm called the Incremental EM (IEM) algorithm. The chapter also described how the tracking is performed through a probabilistic search using the colour model and how temporal constraints are used to limit the search area and provide tracking consistency. This method achieves very good tracking results, especially when objects are partially occluded. Compared with the blob tracking method of §10, this is the method of choice.