Chapter 10

Blob Tracking

This chapter describes the implementation of the Blob Tracking method used by the OmniTracking application. The first section describes the initial grouping of foreground pixels into blobs using connected components, followed by how these blobs are processed to eliminate noise and handle object fragmentation. Then, the similarity criteria for matching blobs to objects are introduced together with how the matching process and tracking is performed. The chapter finishes off by mentioning some results and the limitations of the blob tracking method.

10.1 Connected Components

Blob tracking is based on the idea that world objects are projected into spatially compact areas in the image plane, appearing as sets of contiguous pixels, called *blobs*, in the image.

The output of the background subtraction algorithm is a classification of pixels into foreground, background and shadow pixels, given by the label map $l(x,y)$ (described in §8.7.4)[1]. This label map defines a segmentation given by the binary image S where:

$$S(x,y) = \begin{cases} 1 & \text{if label } l(x,y) = \text{'foreground'} \\ 0 & \text{otherwise} \end{cases} \qquad (10.1)$$

Grouping foreground pixels into blobs is done by finding *connected components* in $S(x,y)$. A connected component is a set of pixels where each pixel is connected to the

[1] For the purpose of object tracking, the pixels labelled as shadow are considered as background from this point onwards.

others through a path made up of adjacent pixels within the same set. Adjacency between pixels in this case is defined to be 8-neighbours. A component labelling algorithm is used to traverse $S(x,y)$ and assign a unique label to each blob (connected component), giving a list B of blobs as its result [JAIN95 §2.5.2]. At the same time as the labelling is being done, some basic information about each blob is gathered, most of it is accumulated as each pixel is visited by the algorithm:

Blob information	
Bounding box	The rectangular bounds of the blob.
Area	Number of pixels.
Seed point	An arbitrary pixel belonging to the blob. Set to the first pixel encountered by the labelling algorithm while it is scanning the image.

Figure 10.1(b) shows the results of the component labelling algorithm for a sample frame from the PETS-ICVS dataset.

10.1.1 Size Filtering

The hysteresis thresholding algorithm described in §8.7.5 reduces most of the noise-induced pixels, by removing those with little support from their neighbours. But hysteresis thresholding is not able to remove noise caused by camera movement, for example. Some of this remaining noise can be eliminated at this stage by using a *size filter*, since it can be assumed that very small blobs are more likely to arise from noise[2]. A blob with an area smaller than some pre-defined threshold A_{min}, will be deleted from the list of blobs B. This simple filtering is very effective and is not expensive to do, because the area of a blob is already known and the number of blobs in an image will normally be limited. In Figure 10.1(c), 4 blobs were removed using size filtering.

10.1.2 Blob Clustering

One major problem that affects background subtraction is when parts of an object have the same colour as the background (camouflage problem – see §8.6) and hence are undetectable. This causes the object to be fragmented into several blobs, as in the case of two of the persons in Figure 10.1(b). But it can be assumed that the fragments

[2] The remaining noise will be dealt with during the object tracking phase by using temporal constraints.

will be spatially close to each other and therefore a *blob clustering* algorithm can be used.

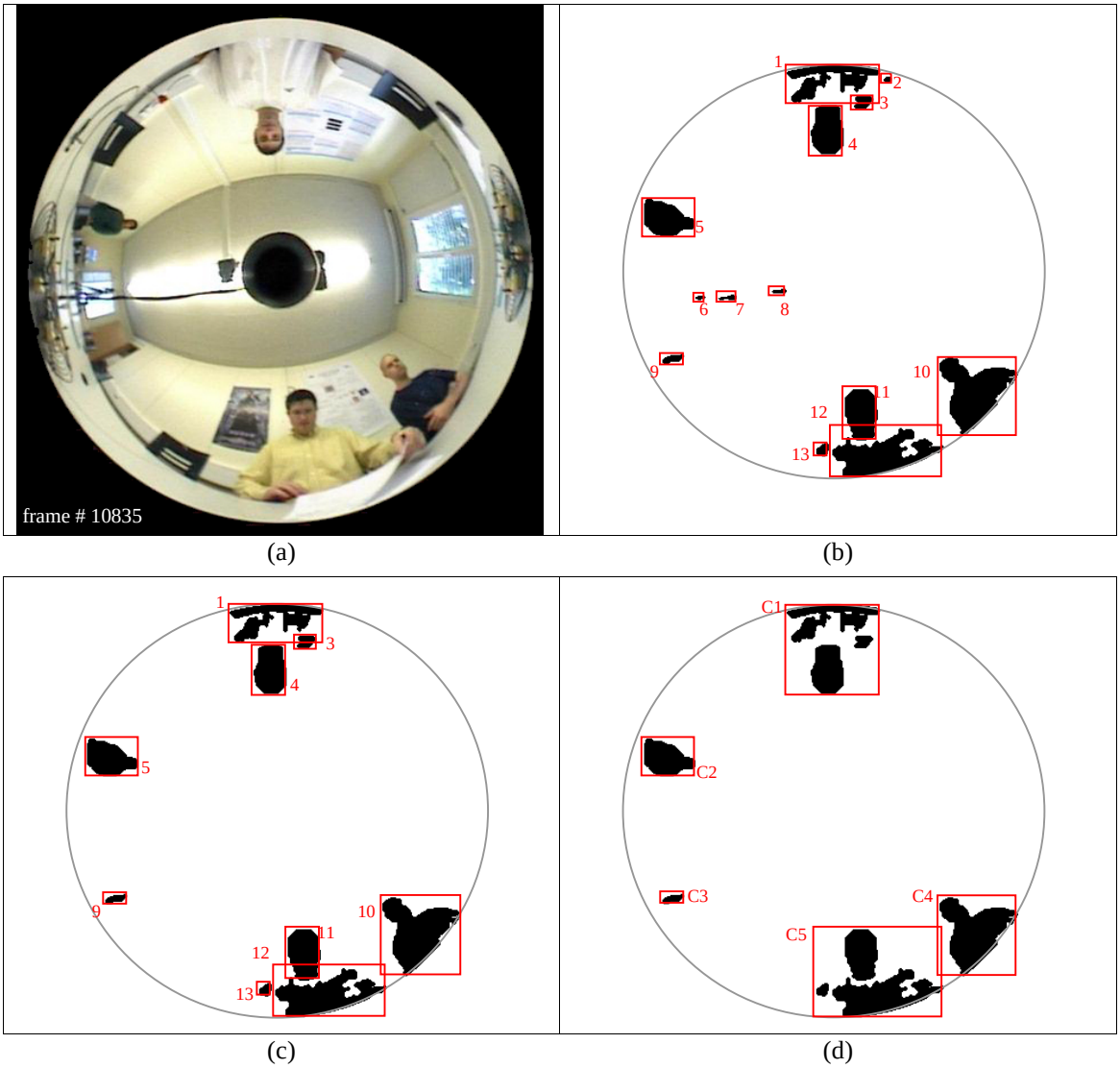

Figure 10.1: Connected Components and Blob Clustering. (a) original image from the PETS-ICVS dataset; (b) blobs found by the component labelling algorithm; (c) result of running size filtering to eliminate small blobs; (d) blob clustering to correct for object fragmentation.

Many existing blob trackers use a bounding-box approach to blob clustering: if the bounding boxes of two blobs overlap (or are sufficiently close to each other), then the blobs are merged into one [COLL00]. This is an efficient method, but assumes that the bounding box is a close representation of the blob's boundary – which is not the case for elongated, diagonally-oriented objects. This is not much of a drawback for rectilinear images, where objects normally appear in an "upright" position and diagonal objects are not a common occurrence. But in the case of omnidirectional images, there is no preferred orientation. For example, in Figure 10.1(c), blob #10 has

a diagonal orientation. If using bounding boxes, blob #10 appears 'closer' to blob #12 than #13 is (in fact they overlap!), when in fact the opposite is true. Therefore it was decided to use a combination of the faster bounding-box approach together with a slower distance calculation method. The algorithm is described below:

Given two blobs B_1 and B_2, their bounding boxes can be used to check if the blobs are too far from each other or not:

d_x = distance between bounding boxes of B_1 and B_2 in *x*-direction, *(10.2)*
d_y = distance between bounding boxes of B_1 and B_2 in *y*-direction[3].

if $d_x \le d_{max\_clustering}$ *and* $d_y \le d_{max\_clustering}$ *then* :
    use distance calculation method on B_1, B_2.
else :
    skip these blobs.

The value $d_{max\_clustering}$, is a global constant that is user-configurable and represents the maximum distance that two blobs can be apart and still be clustered together.

The distance calculation method uses the *distance transform* to get an estimate of the smallest distance between two blobs. The distance transform is a procedure that generates an 'image' containing the approximate distance of every pixel belonging to some set *S*, from its background $\overline{S}$ [JAIN95 §2.5.9]. This image is calculated using a pre-defined distance mask and a sequence of additions. For the OmniTracking program, the distance transform function available in the OpenCV library was used, since this is optimised for MMX processors. A 3x3 mask is used that gives an approximation for the Euclidean distance.

For blobs B_1 and B_2, the smallest distance between them is given by:

roi = minimum bounding box containing both B_1 and B_2 *(10.3)*
DT = distance transform of $\{p = (x, y) : p \in \overline{B}_1, p \in roi\}$
$d_{min} = \min[\{d : d = DT(x, y), (x, y) \in B_2\}]$
if $d_{min} \le d_{max\_clustering}$ *then* :
    combine B_1 and B_2 into one blob cluster.

[3] If either of the distance components along the *x*- and *y*-axis is larger then the maximum distance threshold, then the actual 2D distance between the two bounding boxes will also be larger.

At first, using the distance transform may seem expensive, but this is only used for a limited number of blobs and within the image area covered by their minimum bounding box. In addition, through the combined use of hysteresis thresholding, size filtering and blob clustering (all three working locally), the OmniTracking application does not require any morphological operations to be applied (globally on the image).

Figure 10.2 shows how the blob clustering algorithm is applied to the blobs numbered #10 and #12 in Figure 10.1(c). While Figure 10.1(d) shows the result of the algorithm and how the objects are correctly labelled.

Figure 10.2: Blob clustering algorithm. (a) The common bounding box *roi* is calculated; (b) complement of B_1; (c) doing distance transform on $\overline{B}_1$ (black represents smallest distance); (d) using B_2 as a mask on the DT image and finding the minimum distance.

Because of the non-uniform geometrical nature of omnidirectional images and to add more flexibility to the blob clustering algorithm, distances can also be expressed in terms of their components in polar coordinate form (r,θ) with the origin being the image centre. That is, the threshold $d_{max\_clustering}$ can be specified in terms of the distance threshold along θ and along r. The former one is an angular distance and is specified in degrees. By default, these are set to –1, meaning that the component-wise thresholds are not used. For example, in the case of the PETS-ICVS dataset, a maximum azimuth distance threshold (along θ) was used in conjunction with $d_{max\_clustering}$, to place a further constraint on blob clustering and based on the observation that the objects being tracked in this dataset consists of people which are normally elongated in a 'vertical' direction. The conversion of a point or distance from image coordinates (x,y) to polar coordinates (r,θ) is done using a fast look-up table, created by the calibration module (§7.4.4).

Given a list of blobs $B = \{B_1, B_2, B_3, \ldots, B_n\}$ as input, the blob clustering algorithm generates an output list of blob clusters C:

$$C = \{\ C_1 = \{B_i, B_j : \mathrm{d}(B_i,B_j) < d_{max\_clustering}\}, C_2, \ldots\ \}. \qquad (10.4)$$

For example, the cluster list for Figure 10.1(d) is:
{ C_1={B_1,B_2,B_4}, C_2={B_5}, C_3={B_9}, C_4={B_{10}}, C_5={B_{11},B_{12},B_{13}} }.

10.2 Object Features

An important aspect of object tracking is selecting features that can act as a representation for ('describe') detected objects. For blob tracking, these features are extracted directly from the blob clusters found in the image. The set of features (or feature vector) is then used, in conjunction with temporal constraints, to match blobs from one frame to the next and hence track objects over time.

The two main constraints applied in this case are the temporal constraint, where an object is expected to show only a small movement in its position from one frame to the next, and the similarity constraint, where the object's appearance is also expected to show little change. As mentioned in §9.1, the features should ideally:

- represent characteristics of the object that stay relatively constant in terms of the constraints mentioned above,
- can be used to discriminate between different objects in a reliable way,
- be robust to changes in the object's appearance or the environment.

The following features were chosen for the blob tracking method of this application:

Object Features	
Extent	The rectangular bounds of the object.
Size	Area of the object (number of pixels).
Centroid	The centre point of the object.
Colour	A representation of the object's colour.

The first feature gives an indication of the image area covered by an object and is represented by the object's bounding box. For the omnidirectional image, the bounding box is measured in polar coordinates (θ,r), where θ is the extent of the object along the azimuth angle and r is the radial extent[4]. (A separate bounding box,

[4] Distance r could be converted into the angle of elevation ϕ to get a true spherical coordinate representation (θ,ϕ) for the bounding box (as in §6.5). But this was deemed to be costly, and instead the radial distance from the image centre is used. This is defined as the distance along the line joining the

defined in the image *x*,*y* coordinate space is also maintained for each object. This bounding box is not used for matching or tracking purposes, but only serves to restrict the image area to be scanned when iterating through the object's pixels). The next feature, object size, is simply a count of the number of pixels, while the centroid is the average position (*x*,*y*) of the pixels. The fourth feature is the object's colour, which is represented as a histogram. The HSV colour space is used because of its separation of chromaticity from the brightness component (see §8.7.2.1), thus allowing the two to be treated separately and one given more attention than the other. The relationship of these object features to the underlying blob cluster[5] is illustrated in Figure 10.3 below.

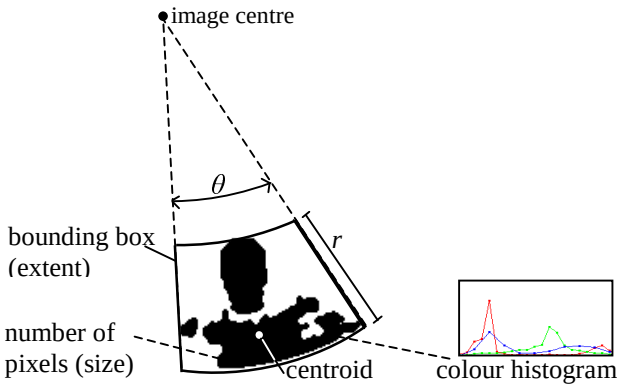

Figure 10.3: Object Features

One advantage of the choice of these features is that they can be extracted from the image in an efficient way. In the case of the colour histogram, the omnidirectional image has already been converted to HSV during the background subtraction phase and this is re-used here.

10.3 Similarity Measures

Using the object features described in the previous section, the following similarity measures are defined:

point to the image centre. The only disadvantage of using (*θ*,*r*) is the difference in units (degrees, pixels) – this is taken care of when using the bounding box for matching blobs.

[5] When discussing features, the words object and blob clusters will be used interchangeably. The features are used for both within the program, for example, when comparing new blob clusters in an image against the existing objects.

Similarity Measures	
Overlap	This measure gives the amount of overlap between two objects.
Centroid distance	The distance between the centroids of two objects.
Area ratio	The ratio of the size of two objects.
Colour Similarity	The difference between the colours of two objects.

The first two measures are used to determine how far an object has moved when comparing it to potential matches in the next frame. The temporal constraint says that the overlap should be high and the spatial distance should be small. The other two measures (area and colour) are used to compare the appearance of the object with that of the potential matches, which should be nearly equal according to the similarity constraint.

10.3.1 The Overlap Measure

The overlap measure is calculated using the bounding boxes of the two objects, as indicated in Figure 10.4. Given two blob clusters C_1, C_2 and their bounding boxes:

$$\text{overlap}(C_1, C_2) = \frac{2 \times \text{area}(\text{intersection}(box_{C1}, box_{C2}))}{\text{area}(box_{C1}) + \text{area}(box_{C2})} \qquad (10.5)$$

$$\text{where:} \quad \text{area}(box) = |\theta_1 - \theta_2| \tfrac{H}{90^\circ} \times |r_1 - r_2|$$

This measure produces a result within the range [0..1], where 0 means no overlap and 1 means full overlap, that is, the bounding boxes are identical. As mentioned in §10.2, the bounding box contains mixed units – the azimuth range $[\theta_1..\theta_2]$ is measured in degrees, while the radial range $[r_1..r_2]$ is measured in pixels. To get a meaningful area value, the azimuth range of the bounding box is converted to pixels as shown in (10.5), so that both have the same units.

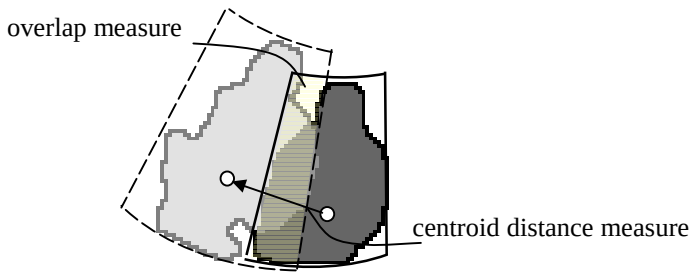

Figure 10.4: Overlap and Centroid Distance Measures

10.3.2 The Centroid Distance Measure

This measure indicates how far away the centroids (*x*,*y*) of two objects are and is calculated using the equation:

$$\text{centroid\_distance}(C_1,C_2) = 1 - \frac{d(C_1,C_2)}{2 \times \text{mirror\_boundary\_radius}} \qquad (10.6)$$

$$\text{where: } d(C_1,C_2) = \sqrt{(x_{C1} - x_{C2})^2 + (y_{C1} - y_{C2})^2}$$

where the result is in range [0..1], with 1 if the centroids of the two objects are exactly at the same position and 0 if they are the farthest possible distance apart (on opposite sides of the image).

10.3.3 The Area Ratio Measure

The area ratio is used to compare two objects by checking how similar they are in size:

$$\text{area\_ratio}(C_1,C_2) = \frac{\min(\text{size}_{C1},\text{size}_{C2})}{\max(\text{size}_{C1},\text{size}_{C2})} \qquad (10.7)$$

The output value if this measure is also within range [0..1], with value 1 meaning the two objects have identical size.

10.3.4 The Colour Similarity Measure

One way of checking how similar two objects are is to compare their colour histograms. Using colour as one of the object's feature provides several advantages such as robustness to changes in an object's size, orientation, rotation, etc. For this implementation, colours are expressed using the HSV colour space and the histogram consists of 32×32×16 bins for the H,S,V colour components respectively. The number of bins chosen reflects a balance between resolution and image noise. Also, a large histogram will mean more memory is needed for each object. The histogram for the brightness component contains half as many bins (lower resolution) as the other two, in order to reduce the sensitivity of the object's colour to lighting conditions.

A histogram can be viewed as a discrete probability distribution function and there are several methods available in the field of statistics for comparing distribution functions. One such measure is the *Bhattacharyya metric* and is the one selected for this application. This metric has many desirable properties, many of which are examined in [AHER97; THAC96], including self-consistency and having a fixed bias (compared to the well-known χ^2 test).

Given two discrete normalised histograms *H* and *G*, where *H(i)* is the number of colours in bin *i*, the *Bhattacharyya coefficient* ρ_B is defined as:

$$\rho_B(H,G) = \sum_{i=1}^{N} \sqrt{H(i)}\sqrt{G(i)} \qquad (10.8)$$

$$\text{provided: } \sum_i H(i) = 1, \sum_i G(i) = 1$$

The coefficient ρ_B is within the range [0..1], with $\rho_B = 1$ when the two histograms match perfectly. The colour similarity measure is then defined to be:

$$\text{colour\_similarity}(C_1, C_2) = \frac{1}{3}\left(\sum_{C \in H,S,V} \rho_B(H_1^C, H_2^C) \right) \qquad (10.9)$$

with the result within the range [0..1], where 1 indicates a perfect match.

The left-hand column of Figure 10.5 (next page) shows the histogram constructed for the person shown (labelled '*A1*') in frame #10815 of the PETS-ICVS dataset. The histogram is shown in red, green, blue for the hue, saturation, value respectively. The second column shows the histograms for the 5 objects (*C1*, *C2*, etc.) of Figure 10.1(d) and the value of the Bhattacharyya coefficient for these objects compared to object *A1* of the earlier frame. The value is highest for object *C2*.

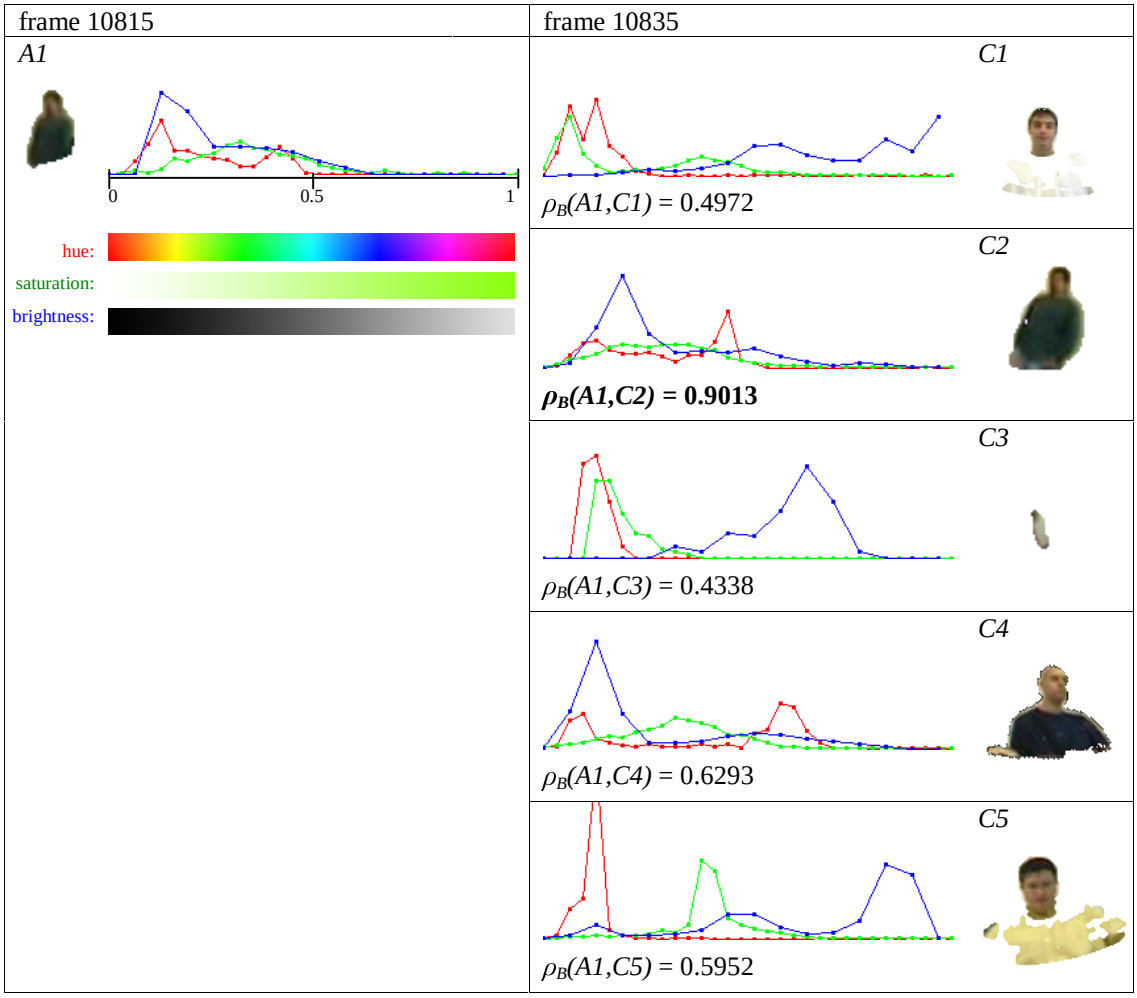

Figure 10.5: Colour Similarity measure. Comparison of object A1 (left column) taken from frame 10815 of the PETS-ICVS dataset, with the objects of frame 10835 (right column). The best match for object *A1* is blob cluster *C2*.

10.4 Object Tracking

Object tracking is done by solving the *correspondence problem* between blob clusters and objects on a frame-by-frame basis. Blob clusters are given by the list *C* that was generated at the end of §10.1 – these are extracted from each image frame *t* and only exist for the duration of that frame. On the other hand, objects are maintained in a global list *O* and last for the duration of the program. For each new frame, blob clusters are matched with the existing objects using the similarity measures defined beforehand. Any correct matches will result in the object being updated with the blob cluster's state (hence the object is tracked from its previous position at time *t-1* to the new position *t*). If an object has no corresponding blob cluster, then the object is considered 'lost'. And if there is a blob cluster with no corresponding object, then it is

potentially a new object. This process is then repeated for the next image frame *t+1*. The algorithm is shown in Figure 10.6 below:

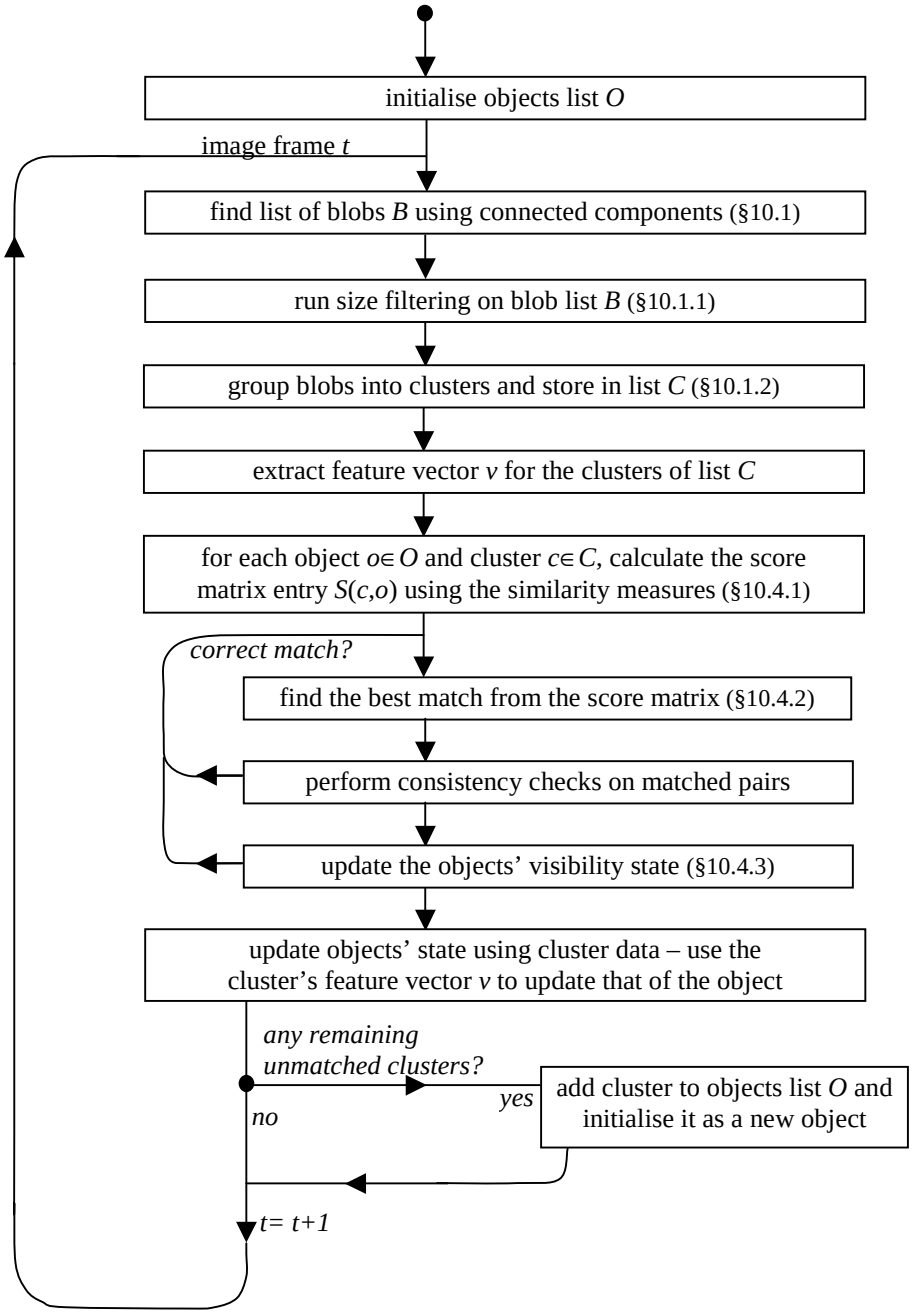

Figure 10.6: Blob Tracking Algorithm

Objects maintain the following information:

Object information	
Object State	Consists of the feature vector describing the object.
Visibility State	These indicate if the object is being tracked, age of the object, if it has been lost, etc.
History	Information maintained for historical purposes to be used for summarisation, showing path of object, etc.

The object state consists mainly of the set of features mentioned in §10.2. Objects get this feature information from the blob clusters they match with. The *visibility state* is made up of a set of values and flags that:

- indicate when (frame number) the object was born (first detected). This is used by some consistency checks to derive the age of the object to serve as a measure of confidence in the object.
- indicate if the object's status is currently 'lost', 'tracked', or 'merged'. An object is in a lost state if it does not match to any of the blob clusters in the current image frame. And an object is in a merged state if it matches with a blob cluster that also matches with another object – that is, both objects share the same blob cluster.
- show the 'number of lives left'. This is a running count that is incremented every time the object is successfully tracked from one frame to the next and is decremented every time the object is lost. When the count reaches zero, the object is deleted.
- keep a count of how long an object has been in a lost state. This value is reset once the object matches with a blob cluster (is found again). It is used in conjunction with the number of lives to determine when to delete an object.

Information of a historical nature includes the path of the object, the starting position when the object was first detected, the maximum size the object has attained during its tracking, etc. Most of this information is used by higher-level algorithms, for example, a node that summarises what has been seen by the tracking application.

10.4.1 Match Scoring

Given the list of blob clusters *C* extracted from the current frame and the list of objects *O*, a *match score matrix S* is constructed. The score is calculated for each cluster-object pair and consists of a weighted sum of the four similarity measures of §10.3, defined as:

$$S = \{s(c,o) : c \in C, o \in O\} \qquad (10.10)$$

$$s(c,o) = \frac{1}{\mathrm{W}}\begin{bmatrix} \mathrm{W}_{\text{colour}} \times \text{colour\_similarity}(c,o) + \\ \mathrm{W}_{\text{area}} \times \text{area\_ratio}(c,o) + \\ \mathrm{W}_{\text{distance}} \times \text{centroid\_distance}(c,o) + \\ \mathrm{W}_{\text{overlap}} \times \text{overlap}(c,o) \end{bmatrix}$$

$$\text{where}: \quad \mathrm{W} = \mathrm{W}_{\text{colour}} + \mathrm{W}_{\text{area}} + \mathrm{W}_{\text{distance}} + \mathrm{W}_{\text{overlap}}$$

The score takes a value in the range [0..1], with 1 for a perfect match. The weights are used to give different importance to the similarity measures. For the OmniTracking program, the following weights were chosen:

$$\left(\mathrm{W}_{\text{colour}} = 2.0,\ \mathrm{W}_{\text{area}} = 0.5,\ \mathrm{W}_{\text{distance}} = 1.0,\ \mathrm{W}_{\text{overlap}} = 1.0\right) \qquad (10.11)$$

A higher weight was given to the colour similarity measure as this was found to be a good discriminator, while the area ratio measure provides only a marginal benefit, so its weight was reduced.

While calculating the scores for all possible cluster-object pair, outright mismatches are eliminated. For example, if the centroid distance measure indicates a distance between the two objects that is larger than the maximum allowed movement per frame, then the score is set to 0. The same is done if the area ratio indicates a too large area change. This can be viewed as applying thresholds on the individual distance measures.

10.4.2 Matching Blob Clusters to Objects

Once the match scoring process is finished, the matching process tries to find the globally optimal match from the score matrix *S*, by minimising the error:

$$\varepsilon = \sum_{i,j} \left(1.0 - s(c_i, o_j)\right) \qquad (10.12)$$

If a match solution leaves some blob clusters unmatched, then a penalty is added to the total error ε. Blob clusters that are left unmatched will be considered to be new objects. The idea behind the use of the penalty is that the algorithm tries to conservatively match clusters to established objects and then reluctantly creates new objects from the unmatched clusters left. This is similar in principle to the idea of using 'null' matches as described in [STAU99] – an unmatched blob cluster matches to

the null object which always gives a constant error. Although in the case of [STAU99], the matching is between connected components and Kalman models. The penalty value used in this case is ⅓. An example of the optimum match is shown in bold in Table 10.1

Table 10.1: Match Score Matrix for frame #10835 of PETS-ICVS dataset (see Figure 10.1(d)).

	O0	*O1*	*O2*	*O3*	*O4*
C1	0.3542	0.5036	**0.9741**	0.3543	0
C2	0	0	0	0.5311	**0.4278**
C3	0	0.4392	0	**0.7031**	0.4664
C4	**0.9684**	0.5189	0.4467	0.4035	0
C5	0.4386	**0.9314**	0.5066	0.4062	0

Finding the global optimal solution by iterating through all the cluster-object pairs has order O(*n!*) which can cause combinatorial explosion. The set of objects *O* and blob clusters *C* can be considered to be a *directed bipartite graph G* = (*O*,*C*), with the match scores forming the edges linking elements of *O* to those of *C*[6]. There are several greedy algorithms that can be used to find a near-optimal solution for a bipartite graph, such as the well-known *Hungarian assignment method*. But unfortunately, due to the addition of the penalty for unmatched clusters, the author could not find a greedy algorithm or any other algorithm that improves on the order O(*n!*).

But using rules like those mentioned at the end of the previous section, can at least help to reduce the possibility of combinatorial explosions. In particular, the maximum-allowed movement threshold uses the centroid distance measure to spatially split the graph into several smaller sub-graphs, as shown in Figure 10.7 below. Generally, each sub-graph will contain only a few objects, which can then be matched using an exhaustive search.

[6] The elements of a bipartite graph consist of two disjoint sets, with no edges defined between elements of the same set.

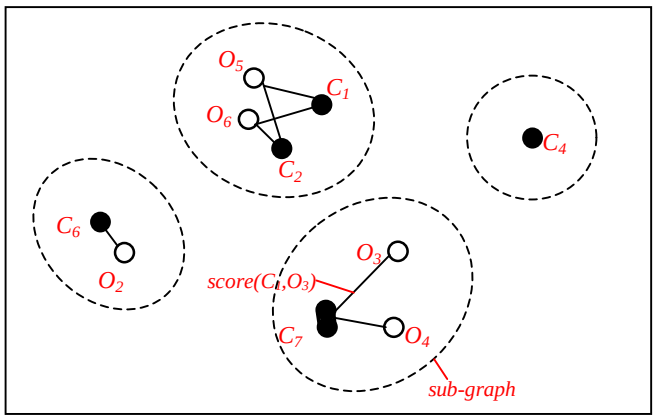

Figure 10.7: Matching as a graph search problem. The blob cluster's centroids are shown as black circles, while the object's centroids are the white-filled circles. The graph is split up into 4 sub-graphs, shown with dashed outline, using the maximum allowed movement threshold.

10.4.3 Updating the Object

The output of the matching process is a set of correspondences (O_i,C_j) between blob clusters and objects. Based on these correspondences, the object's visibility state is updated as follows:

if O_i has no matching C_j *then* : *(10.13)*

object is 'lost' → number_of_lives – 1
lost = true
lost_count + 1

if O_i matches with C_j *then* :

number_of_lives + 1
lost = false
lost_count = 0

if C_j matches with O_k, $O_k \neq O_i$ *then* :

object has 'merged' → merged = true

if O_i is lost *and* number_of_lives ≤ 0 *then*

delete O_i

if O_i is lost *and* lost_count > T_{lost} *and* size < $size_{max\_prev}$ *then*:

delete O_i

The last two conditions define the object deletion rules. The second rule is used for the cases where an object moves away from the camera until it gets very small and disappears altogether. The value $size_{max\_prev}$ in this case stores the maximum size that

the object had in previous frames – that is, to check if the size of the object is really getting smaller and is not a new object coming into view.

The object's feature vector is updated by using the features from the cluster blob it matches to. For the first three (centroid, size and bounding box) a full update is done, and the cluster's information overwrites that of the object. This ensures that changes in the position and size of the object come into effect immediately. But for the colour histogram, a partial update is done using the exponential forgetting equation defined by (8.8), with α having a default value of 0.4. When an object is in a merged state, then its feature vector is not updated and the object is thus *frozen* until it is no longer merged. The reason for freezing the object is because the tracker is not able to tell which part of the blob belongs to which object.

The final task performed by the tracking algorithm is to search the list of blob clusters *C* for any cluster that has no matching object. These are considered to be new objects and are moved into the object list *O*, and their visibility state is initialised.

10.5 Results

The first observation that can be made is that the blob clustering algorithm used by the program produces good results, especially in the case of objects having the same colour as the background, which are fragmented by the background subtraction algorithm. This is particularly evident in the PETS-ICVS dataset, where objects have a large apparent size due to their closeness to the camera and one of the persons in particular has a colour indistinguishable from the white walls. The algorithm groups the fragments correctly throughout the whole sequence. Some examples are shown in Figure 10.8 below.

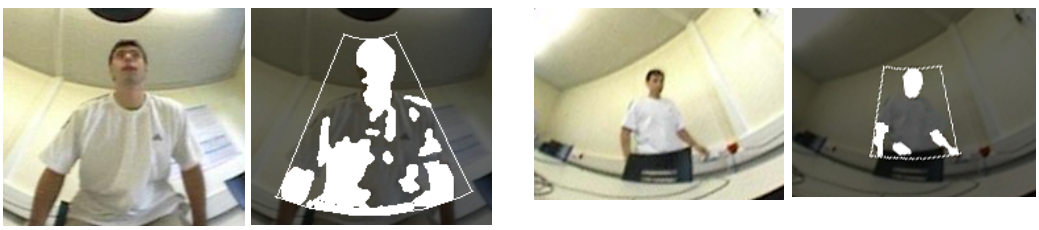

Figure 10.8: Results: Blob Clustering

Blob tracking works by using the blobs extracted from the image, based on the assumption that real world objects project themselves into blobs in the image. If two world objects occlude each other or appear to be touching in the image, only one blob will be detected. This causes the tracker not to be able to identify the two objects while they are merged together and in the worst cases to lose the objects completely. This is a serious drawback of blob tracking methods – they are not robust to occlusion and object merging.

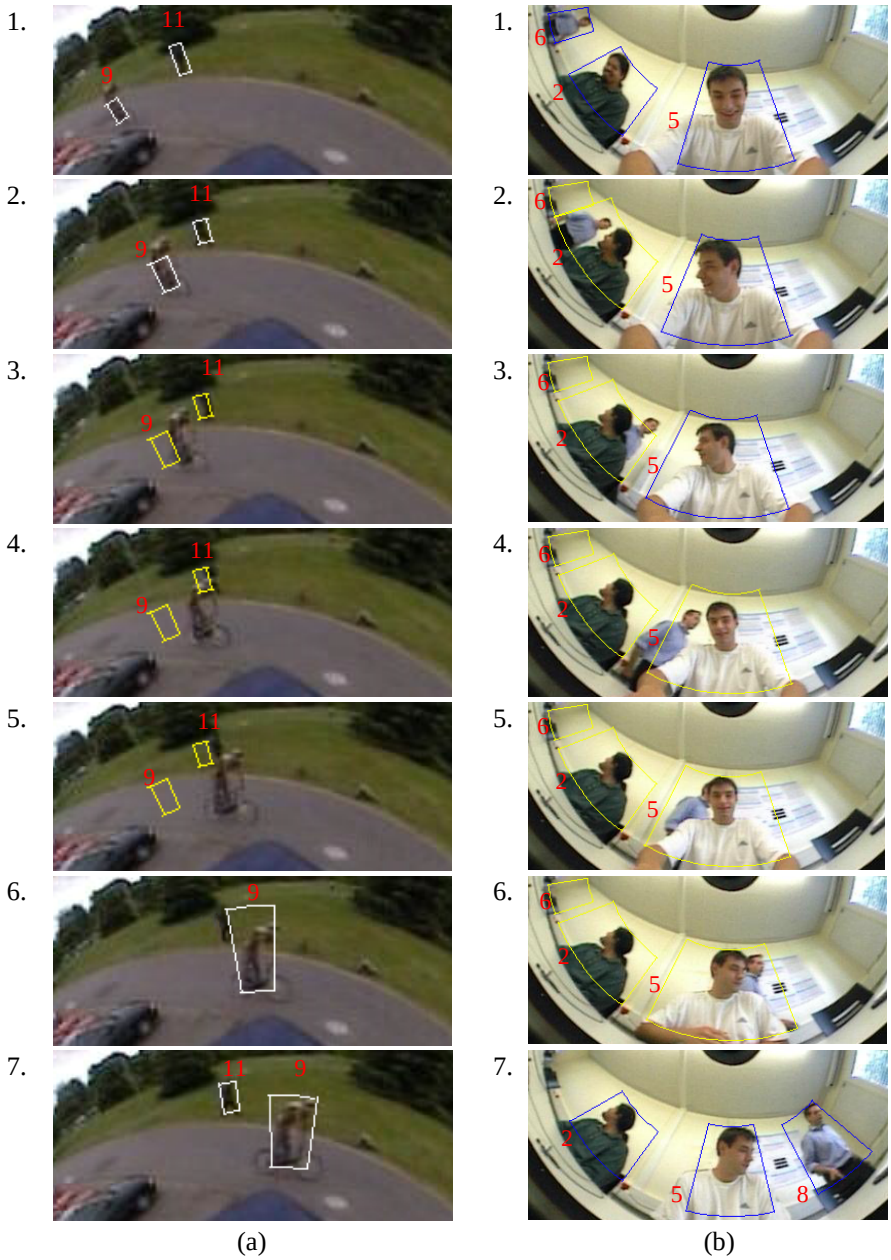

Figure 10.9: Results: Occlusion and Merging. (A yellow bounding box indicates the object is in a merged state and therefore its state is frozen, while a white/blue bounding box represents a normal state).

Figure 10.9(a) shows an example where an object occludes another in the PETS-2001 dataset. Although the tracker recovers object #11 successfully after it has been lost in frame 6, there is also the problem that the two objects merge just before occlusion. Starting from frame 3 till 5, the objects appear as one blob, and as mentioned in the previous section, the two objects match to the same blob, are labelled as 'merged' and their state is frozen (indicated by a yellow bounding box in the diagram). Not doing so, will result in both objects adapting to the same blob as one pollutes the feature vector of the other. But on the other hand, if the two objects were to merge and move at the same speed, it is very likely that both objects will be lost.

Occlusion and merging is more of a problem in the PETS-ICVS dataset, because of the proximity and large size of the objects. For example, Figure 10.9(b) shows how object #6 is merged and later occluded by object #2. When object #6 reappears from behind #2, it immediately merges with object #5. The large distance travelled by object #2 while merged and occluded, causes it to be lost and eventually the object dies. The temporal constraints used by the program could be relaxed to accommodate such large movements and allow blobs to match over a larger part of the image – but then other problems will appear and may defeat the purpose of such constraints. Object #2 is occluded only for a short period from frame 2 to 3 and then from frame 5 to 6, so it could potentially be tracked in the rest of the frames. But from the point of view of image blobs (and therefore blob tracking), the object is not visible as a separate object over a longer period of time.

As regards to the object features, it was found that the use of the HSV colour histogram together with the similarity measure based on the Bhattacharyya coefficient provides a good discrimination between objects. For example Figure 10.10 shows the histograms for the two objects of Figure 10.9(a). Even though the objects have a low contrast, there are several differences between their histograms, and this allows the two objects to be differentiated from each other. The difference between their histograms is given in the right-hand column. The figure also shows how the colour histogram of each object changes over time (three frames in this case) – it remains relatively stable over time.

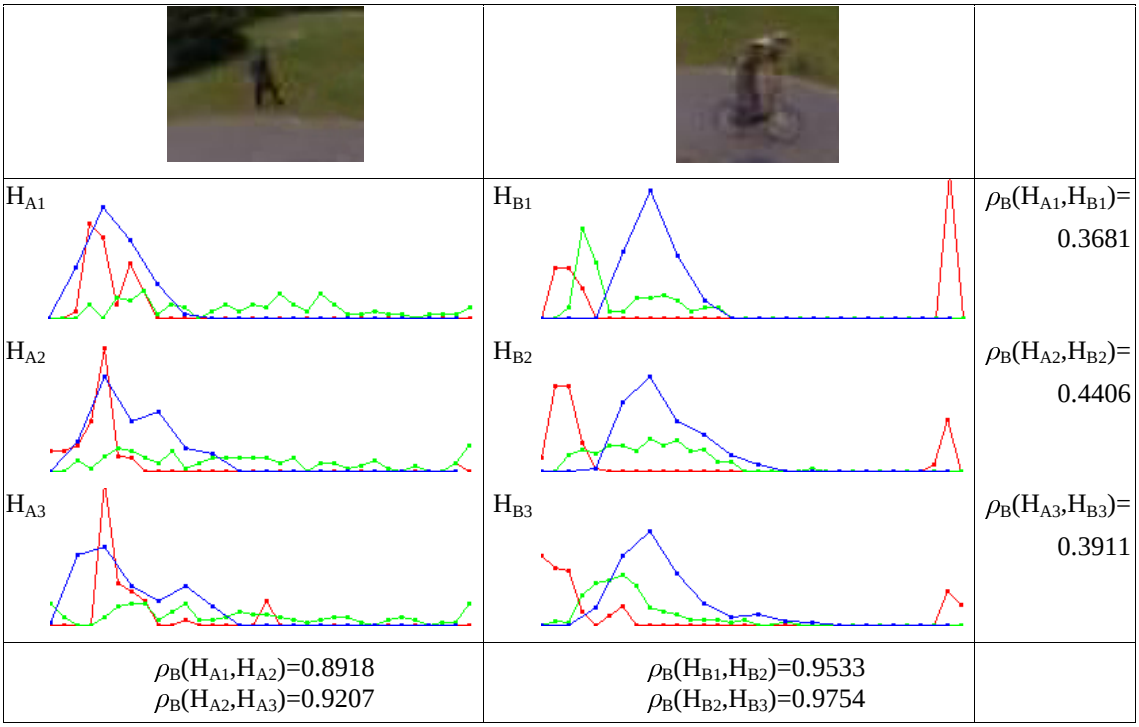

Figure 10.10: Results: Colour Similarity.

10.6 Conclusion

This chapter described how the blob tracking technique was implemented by the OmniTracking program; how the foreground pixels are grouped into blobs, how blob clustering is performed and how the blobs are matched with existing objects on a frame-to-frame basis. The object features used for the matching process are also described, in particular the use of the HSV colour histogram, which is found to be a good feature to match with. As a conclusion, the blob tracking method can lose track of objects in the presence of occlusion and object merging. Even when not lost, the exact position of objects may be unknown when in a merged state. This reduces the usefulness of this method. Blob tracking can be used when speed is very important or in sparse environments where the chances of occlusion and object merging are minimal. In other cases, more robust tracking methods should be used.