AVITRACK

Contract n° AST3-CT-2003-502818

Internal Technical Note

Tracking Report

Version 1.0 – Draft 2

IN_AVI_2_011

Contract Number : AST3-CT-2003-502818

Document Title : Tracking Report

Document version : 1.0

Document status : Draft 2

Date : 30-Nov-2004

Availability : Restricted

Authors : Mark Borg, David Thirde, Miguel Pinzolas

Abstract

This document describes the initial work performed by the University of Reading in object tracking for the AVITRACK project.

Keyword List

Tracking, Frame to Frame Tracking, Local Feature Tracker, Colour Tracker, Cluster Tracker.

DOCUMENT CHANGE LOG

Document Issue.	Date	Reasons for change
1.0 – Draft 1 1.0 – Draft 2	23-Nov-2004 30-Nov-2004	Initial release. Minor changes.

APPLICABLE AND REFERENCE DOCUMENTS (A/R)

A/R	Reference	Title
Please Refer to the Reference Section at the end of this document.		

Table of contents

1. INTRODUCTION

This document presents the initial work performed by the University of Reading in developing Tracking Algorithms as part of "Work Package 3 - Scene Tracking" of the AVITRACK project [1]. An initial evaluation of the algorithms using AVITRACK test sequences acquired at Toulouse airport is also presented. The work on WP3 is still ongoing.

2. SYSTEM OVERVIEW

The tracking algorithms presented in this document form part of the "Frame-to-Frame Tracking" module, which is responsible for performing motion detection (see Motion Detection Report [4]) and short-term tracking. This module is used to process video streams from 8 cameras independently. The tracking results of each camera are then sent in XML via CORBA communications to the "Data Fusion" module, which in turn combines the tracking results from all the 8 cameras into one output for behaviour analysis.

Two of the tracking algorithms have already been integrated into the first delivery of the Frame-to-Frame Tracking module, called *AvitrackFrameTracker* (v0.3) [14], while the third is in the process of being integrated. Having all the tracking algorithms integrated into one application makes it possible to evaluate the algorithms on an equal footing, as well as having the ability to test the algorithms as part of the whole system – together with motion detection, data fusion, etc.

The tracking algorithms were mostly tested with the following AVITRACK data sequences: S3-A320-CAM2, S3-A320-CAM3, and S3-A320-CAM8, some results of which are presented in this document. A more formal evaluation, using ground-truth information and pre-defined evaluation criteria, will be done in the near future, as part of "Work Package 6.1 – Scene Tracking Evaluation".

3. TRACKING – AN OVERVIEW

Real-time object tracking can be described as a correspondence problem, and involves finding which object in an image frame corresponds to which object in the next frame of a video stream – hence, the name: frame to frame tracking. Normally, the time interval between two successive frames is very small, meaning that the changes visible from one frame to the other should be limited. This allows the use of some form of temporal constraints and/or object features that can help to simplify the correspondence problem.

Several factors can make life very difficult for tracking algorithms, such as motion detection errors, and the interactions between the objects as viewed by a camera. Objects can appear to merge together, occlude each other, fragment, undergo rigid or non-rigid motion, etc. The tracking algorithms should be as robust as possible to these events.

There are basically two approaches to solving the tracking problem: (1) *top-down* or *model-based* tracking, and (2) *bottom-up* or *low-level tracking*. The first approach uses knowledge of what objects will appear in the scene, and given a 3D model of the object, hypothesis are generated and tested against the image for the presence of that object. The second approach starts with the image segmented into a number of regions and uses only low-level information available in the image to identify and track objects.

Several different tracking methods have been used in computer vision, amongst these one can find: blob tracking, model-based tracking, template matching methods, methods using optical flow, feature tracking, Kalman filter trackers, particle filters, active contour based methods, colour tracking etc. [15,16].

4. THE TRACKING ALGORITHMS

For the AVITRACK work done so far, it was decided to use a *bottom-up approach* for tracking, as these algorithms are normally more generic and require little context about the scene. Also, model-based methods are normally slow and do not run in real-time.

The following is the list of tracking algorithms that have so far been implemented and tested by the University of Reading team:

- **Local Feature Tracker**
- **Colour Tracker**
- **Cluster Tracker**

Each of these tracking algorithms is described in more detail in the following sections.

5. LOCAL FEATURE TRACKER

This algorithm tracks an object by selecting a set of sparse local features for that object and tracks the features from one frame to the next. It is based on the KLT (Kanade, Lucas, Tomasi) tracking algorithm, which is described in [2,3].

5.1. THE KLT ALGORITHM

A local feature is represented by a small window, which is assumed to undergo an affine motion displacement δ from one frame to the next, composed of a translational motion **d** and a window deformation *D*. The size of the window is specified by parameter WINDOW_SIZE and defaulted to 5x5 pixels in this implementation of the algorithm.

$$\delta = D\mathbf{x} + \mathbf{d} \qquad \text{where } D = \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix}, \mathbf{d} = \begin{bmatrix} dx \\ dy \end{bmatrix}, \mathbf{x} = (x, y)^T \tag{5.1}$$

Tracking of a feature then consists of determining the 6 parameters of the above equation by performing a search in the next frame and measuring the similarity in appearance of the feature between the current frame (I_t) and the next one (I_{t+1}).

For the search and tracking part, only the translational motion vector **d** is used, as this is normally more reliable. But for measuring the similarity between two feature windows, the full affine motion model is used – both **d** and matrix *D*. The dissimilarity measure uses the r.m.s. residue and is given by:

$$\varepsilon = \int\int_W \left[I_{t+1}(A\mathbf{x} + \mathbf{d}) - I_t(\mathbf{x})\right]^2 w(\mathbf{x})\, d\mathbf{x} \qquad \text{where } A = D + 1, (1 = \text{identity matrix}) \tag{5.2}$$

A Newton-Raphson method is used to minimise the dissimilarity measure. For this implementation, the weighting function $w(\mathbf{x})$ is simply set to 1, to obtain the same weight across the whole window, for computational efficiency.

The KLT method is normally suited for tracking small displacements of features between two consecutive frames. To allow for even larger displacements between frames, a multi-resolution approach is used, with a 2-level image pyramid selected for this implementation. Tracking is first performed on a version of the image sub-sampled by a factor of 8 (image size of 90x72), and then this coarse result is used to do the tracking on the full image (760x576).

For selecting good features to track, the KLT algorithm finds windows that contain a strong intensity variation in both horizontal and vertical directions, such as textured regions, corners, etc., and which can normally be reliably tracked. The algorithm computes the 2x2 gradient matrix Z at each candidate 5x5 window W within the object being tracked. Only pixels which are labelled as 'foreground' by the motion detector are considered. The gradient operator used is the derivative of a Gaussian G' with σ set to 1.0, and is convolved with the image to get the horizontal gradient g_x and vertical gradient g_y.

$$Z_W = \begin{bmatrix} \sum g_x^2 & \sum g_x g_y \\ \sum g_y g_x & \sum g_y^2 \end{bmatrix} \tag{5.3}$$

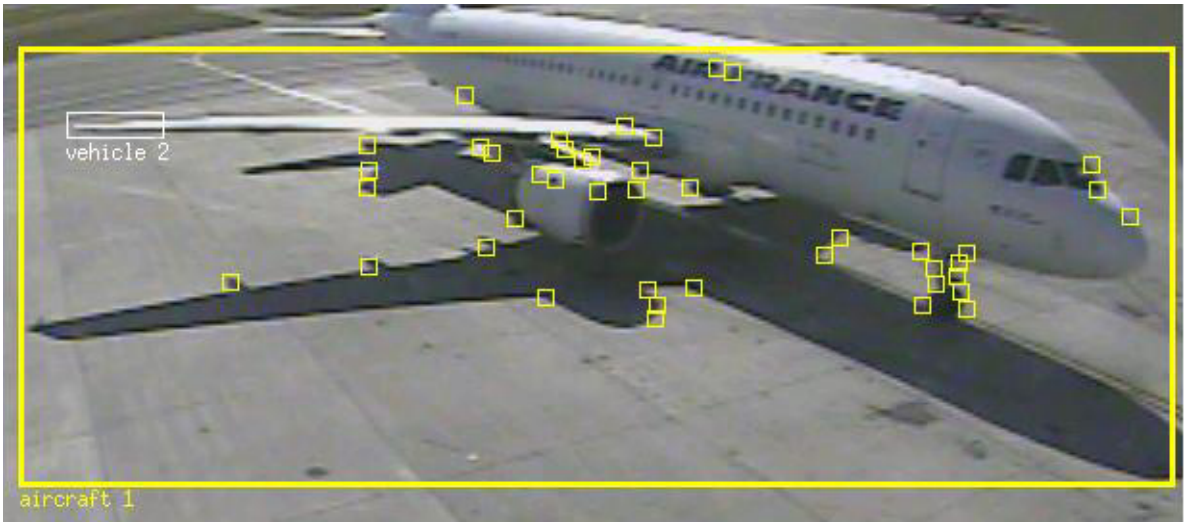
$$G'(r) = -\frac{r}{\sigma^2} e^{\left(\frac{-r}{\sigma^2}\right)} \tag{5.4}$$

The 2 eigenvalues λ_1 and λ_2 of matrix Z are then computed and a feature window W is considered to be a good feature to track if it satisfies the condition:

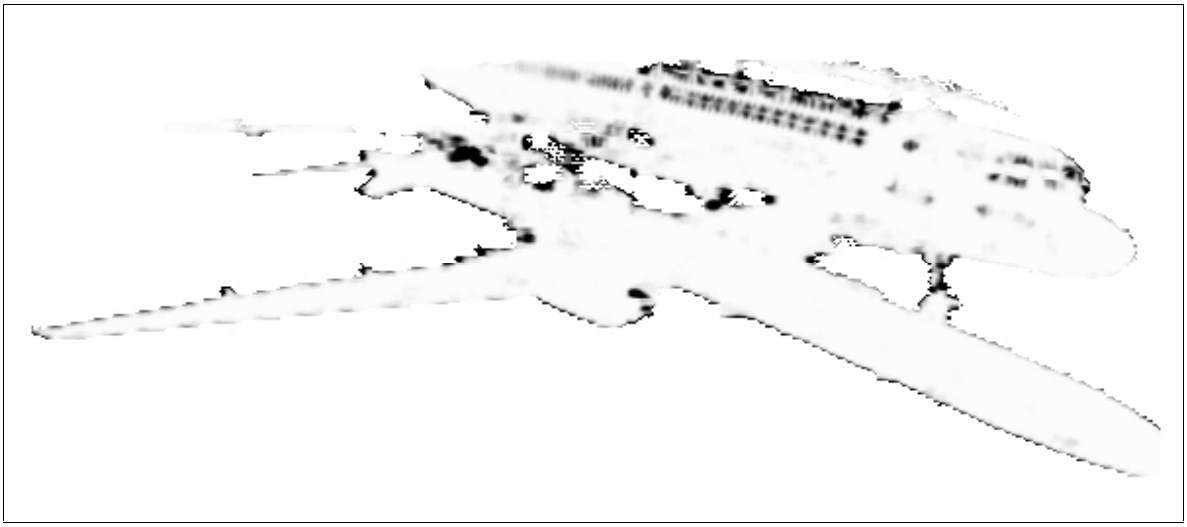
$$\lambda_m > \lambda_0 \qquad \text{where } \lambda_m = \min(\lambda_1, \lambda_2) \tag{5.5}$$

This condition is based on the idea that if both eigenvalues are large, then the intensity change in both directions is strong enough not to be caused by image noise. The global limit λ_0 is set to 1.0 for this implementation of the algorithm.

Out of all the possible feature candidates for an object, the N features with the largest λ_m are selected. The total number of features N tracked for an object O is determined from the object's area and the global parameter FEATURE_DENSITY, which lies in the range [0,1]. A density value of 1 is defined to represent the case where the object's area is completely covered by non-overlapping feature windows in a grid-like fashion. The number N is further restricted to be within a reasonable upper and lower limit defined by MIN_FEATURES_PER_OBJECT and MAX_FEATURES_PER_OBJECT.



(a)



(b)

Fig. 5.1(a) shows the features selected for the airplane object, while Fig. 5.1(b) shows the values of the minimum eigenvalues λ_m (shown inverted). A high λ_m indicates pixels in the image where the amount of bi-directional texture, such as corner-like areas, is high. Note how the features selected by the KLT algorithm occur at points with high λ_m.

$$N' = \left(\frac{\text{area}_O}{\text{WINDOW\_SIZE}^2} \right)_{\text{FEATURE\_DENSITY}}$$
$$N = \min(\text{MAX\_FEATURES\_PER\_OBJECT}, \max(N', \text{MIN\_FEATURES\_PER\_OBJECT})) \quad (5.6)$$

To ensure that features of an object are selected as evenly as possible across the object's area, a minimum distance between features is enforced – this distance is currently set to the window size.

Fig 5.1(a), on the previous page, shows the features selected for the aeroplane object in frame #396 of video sequence S3-A320-CAM2, while Fig 5.1(b) shows the values of λ_m for the object's pixels. A high λ_m indicates areas in the image where there is a strong bi-directional texture (high-frequency change in both horizontal and vertical directions), such as corner-like areas. In the case of the aeroplane object of Fig 5.1, there is a limited amount of these 'textured' areas and many of them occur near the object's boundary. In the selection process of the KLT algorithm, some of the features are chosen from the boundary – these may not prove to be good features for tracking. Other unreliable features may occur on 'artificial' areas caused by specular highlights, etc.

When tracking the features, two configuration parameters control the tracking process. If the residue ε of (5.2) becomes larger than MAX_RESIDUE, or if the number of iterations in the search process exceeds MAX_ITERATIONS, then the feature's state is set to 'lost'. As features of an object are lost, new features are selected to replace the lost ones, so as to try and maintain a constant number *N* of features for the object. Also, if the area of the object increases with time, then *N* is adjusted accordingly.

5.2. FROM FEATURES TO OBJECTS

The KLT algorithm described above and in [2,3], considers features to be independent entities and tracks each of them individually. Therefore, the KLT algorithm must be incorporated into a higher-level tracking process that is able to group features into objects, maintain associations between the features and the objects, and uses the individual tracking results of the features to track objects from one frame to the next, taking into account interactions between objects such as object merging situations, object splitting, stationary objects, etc.

5.3. THE OBJECT TRACKING ALGORITHM

For each object *O* being tracked, a set *S* of features is maintained, with the size of the set given by *N* in (5.6). Given a list of known objects $\{O_i\}$ at time *t-1*, the tracking process for time *t* can be summarised as:

1. Generate object predictions $\{P_i\}$ for time *t* from the list of known objects $\{O_i\}$ at *t-1*.
2. Run the KLT algorithm to track the features of $\{P_i\}$ individually.
3. Given a list of measurement objects $\{M_j\}$ detected by the motion detector at time *t*, match predictions $\{P_i\}$ to measurement objects $\{M_j\}$.
4. Any remaining unmatched predictions in $\{P_i\}$ are set to 'lost'. Any remaining unmatched measurement objects in $\{M_j\}$ are considered as potential new objects.

5. Update the state of those predictions in $\{P_i\}$ that were matched to measurement objects and replace lost features. The final result is a list of tracked objects $\{O_i\}$ at time *t*.
6. Let *t* = *t+1* and repeat step 1.

These steps are described in further detail in the following sections.

5.3.1. GENERATE OBJECT PREDICTIONS

This initial step generates the set of object predictions $\mathrm{P} = \{P_i\}$ for time *t* from the list of known objects $\{O_i\}$ obtained at the end of the previous frame (*t-1*). For each object O_i, the prediction P_i is simply a copy of O_i, with its state changed to PREDICTION.

$$\{P_i\}_t = \{O_i\}_{t-1} \tag{5.7}$$

5.3.2. RUN THE KLT ALGORITHM

For each prediction P_i, its features in set *S* are tracked individually using the KLT algorithm described earlier. Features marked as 'lost' by the KLT algorithm are removed from *S*.

5.3.3. MATCH PREDICTIONS TO MEASUREMENTS

This step uses the set of measurement objects $\mathrm{M} = \{M_j\}$ detected by the motion detector at time *t*. These consist of connected regions of pixels labelled as 'foreground' by the motion detector. For this implementation, the motion detector algorithm used is the Colour Mean and Variance algorithm described in [4] – but any one of the other motion detection algorithms in [4] can also be used.

Predictions $\{P_i\}$ are matched to measurement objects $\{M_j\}$ using the tracking results of the features of $\{P_i\}$ obtained in the previous step. A match score function f is defined, which returns the total number of features *W* of a prediction P_i that reside within the bounding box of a measurement M_j. An additional score of 1 is added, if the prediction's bounding box overlaps that of the measurement or if one is completely inside the other. This is required in case there happens to be no features in the overlapping region of a prediction and a measurement.

$$f(P_i, M_j) = \left| \{W : W \in S_{P_i}, W \in M_j\} \right| \tag{5.8}$$

The Matching Rules

In the ideal case, given a prediction P_i, the above score function f should return a non-zero value for one and only one of the measurements that matches P_i, and zero for the rest. But objects, as viewed by a camera, can interact with each other in complex ways – for example, objects can appear to merge, an object may split into parts, be occluded, etc. A number of matching rules are used to handle these cases.

Rule 1. *Ideal Match*:
The first matching rule handles the ideal matches mentioned above, i.e. a one-to-one match between a prediction and a measurement, and identified by condition (5.9). The prediction is marked as visible and its state is updated by the measurement's data. The object will later on also qualify for replacing lost features.

$$\begin{aligned} & f(P_i, M_j) > 0 \quad \text{and} \\ & f(P_k, M_j) = 0, \;\; f(P_i, M_l) = 0 \qquad \forall k \neq i, \; l \neq j \end{aligned} \tag{5.9}$$

Rule 2. *Splitting Object*:
The second matching rule handles the case when an object at time *t-1* splits into several objects when seen at time *t*. This occurs when a prediction P_i matches with several measurement objects, but no other prediction matches with any of these measurements – in other words, the set of meaurements is partitioned into two subsets: the subset $M1$ of measurements that match only with P_i and the subset of those that do not match with P_i :

$$\begin{aligned} & f(P_i, M_j) > 0 \quad M_j \in M1 \subseteq M, \; |M1| > 1 \qquad \text{and} \\ & f(P_k, M_j) = 0 \quad \forall M_j \in M1, \; k \neq i \qquad \text{and} \\ & f(P_i, M_l) = 0 \quad \forall M_l \notin M1 \end{aligned} \tag{5.10}$$

The prediction is then split into new objects, one for each of the matched measurements in $M1$. The features of the original prediction P_i are then assigned to the new objects depending on whether they reside within the new object's bounding boxes or not. In this way, features are maintained throughout an object splitting event. The object with the highest score is assigned the object ID of the original prediction.

Rule 3. *Merging Object*:
The third matching rule handles merging objects. This occurs when more than one prediction matches with a measurement region:

$$\begin{aligned} & f(P_i, M_j) > 0 \quad P_i \in P1 \subseteq P, \; |P1| > 1 \qquad \text{and} \\ & f(P_i, M_k) = 0 \quad \forall P_i \in P1, \; k \neq j \qquad \text{and} \\ & f(P_l, M_j) = 0 \quad \forall P_l \notin P1 \end{aligned} \tag{5.11}$$

In this case the state of the predictions (such as position and bounding box) cannot be obtained by a straightforward update from the measurement's state, since only one combined (merged) measurement is available from the motion detector. But an attempt is made to update the global states of the predictions from the known local state of their tracked features.

The prediction's new centre is estimated by taking the average relative motion of the prediction's features from the previous frame at time *t*-1 to the current one. This is based on the assumption that the average relative motion of the features is approximately equal to the object's global motion – this may not be necessarily true for non-rigid objects undergoing large motion, and may also be affected by the aperture problem due to the small size of the feature windows. Fig. 5.2 (next page) shows an example of how the tracking results of the features are used to estimate the tracking result of an object while in a merged state.

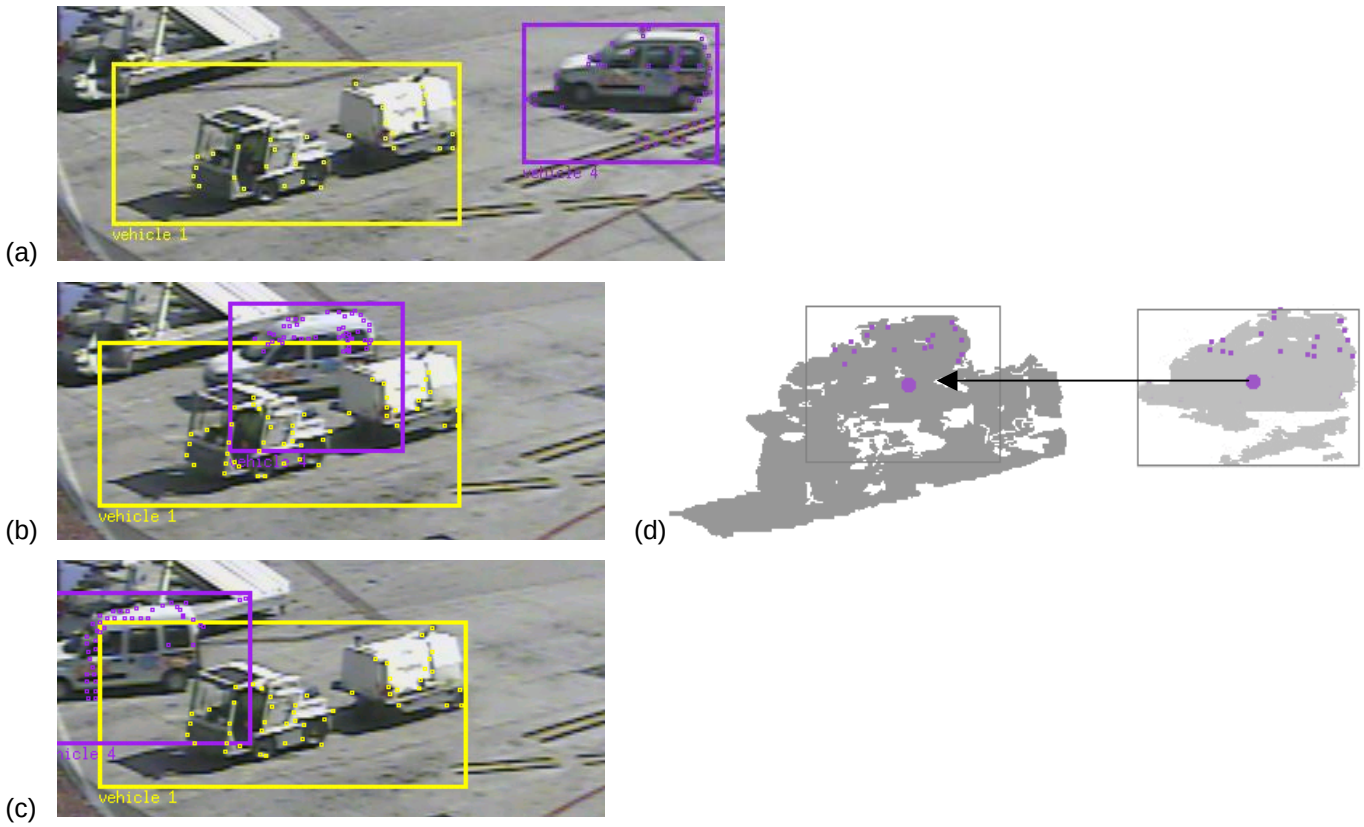


Fig. 5.2 shows two vehicles merging together. The position of vehicle 4 (magenta colour), is estimated from the average of the individual motion of the features that are tracked successfully from the first frame to the second one (shown in (d)).

An attempt is also made to update the sizes of the bounding boxes of the predictions in order to maximise the coverage of the measurement region by the combined predictions' bounding boxes. This handles cases where objects are moving towards the camera while in a merged state and hence their sizes increase. If not done, the result is parts of the measurement region that are not 'explained' by any of the predictions.

If two objects remain in a merged state for a long time (configured to 70 frames for these tests) and there has been little relative motion between them during the time they have been in a merged state, then the two objects are combined into one. The ID of the largest object is assigned to the final object.

Handling Stationary Objects

The motion detector used for this implementation allows objects that have become stationary to be integrated into the background model. A multi-layered background model is used in order to allow stationary objects to be re-activated once they start moving again. Each layer has an associated binary mask that indicates the pixels that belong to that layer, so the objects added to the background model can have arbitrary shape.

Detecting whether an object is stationary can either be done by the motion detector or the tracker itself – both methods have been implemented. The tracker checks whether an object has become stationary by checking for motion of the features of that object. If any of the features moves by more than 1 pixel, then the object is not stationary. Alternatively, the tracker can rely on the motion detector to determine if an object is stationary or not. When both methods were tested on the AVITRACK datasets, the second method was found to be more reliable. The reason is that the features are sparsely distributed across the region and allow the motion to be determined at only a limited number of points.

A count is maintained of the number of consecutive frames that the object has been stationary. When this count exceeds the parameter STATIC_COUNT_INCORPORATE, the object is integrated as a new background layer into the background model and using the pixels labelled as 'foreground' as the background layer mask.

In the AVITRACK sequences, there are several instances where an object is in the process of slowing down to become stationary, but at the same time, part of the object starts moving – for example, as a vehicle is coming to rest, a person emerges from the vehicle and starts moving in front of the vehicle (both vehicle and person remain merged and indistinguishable). Using a simple motion test for the whole 'object' (vehicle+person) will not handle these cases correctly – the person's motion keeps the vehicle from becoming stationary.

As a result, the motion test was relaxed so that if only a small and localised part of an object remains in motion, the whole object is integrated into the background model and a new object is created for the moving part. The bounding box of the part of the object still in motion is computed, and if the ratio of the area of this bounding box and the area of the object is below a certain threshold STATIC_OBJECT_REACTIVATION_MOTION_RATIO, then the object is considered to be stationary. Fig. 5.3 (next page) shows an example from the AVITRACK sequence. One side effect of this motion test relaxation is that 'ghosts' may occur when the part of the object in motion uncovers part of the original object – the static object handling algorithm was extended to try to detect and eliminate such ghosts.

Once a stationary object is integrated into the background model, the motion detector will still continue to return a measurement for that object at each frame. The measurement is labelled as STATIC, to differentiate it from other measurements arising from objects in motion. The tracker uses this to check if the object has started moving again and needs to be re-activated[1].

A search is made for other measurement objects that overlap the static measurement – these must be new measurements (newly detected motion) or else already-seen objects that were first seen within the bounds of the static measurement. If their total area compared to the static measurement's area is larger than the threshold STATIC_OBJECT_REACTIVATION_MOTION_RATIO, then the static object is re-activated and the other measurement objects deleted. If not, then it means that only small part(s) of the static object have started to move again and new object(s) are created for the moving part(s). If the area in motion continues to increase, once it exceeds the threshold the static object is completely re-activated and the object(s) representing the small moving part(s) deleted.

The Matching Rules for Stationary Objects

The set of matching rules mentioned earlier have been extended to include cases when stationary objects and normal objects interact with each other. This is required since predictions are still maintained by the tracker for stationary objects and the motion detector returns static measurements for objects integrated into its background model.

[1] By re-activation, it is meant that the background layer for that object is removed from the background model.

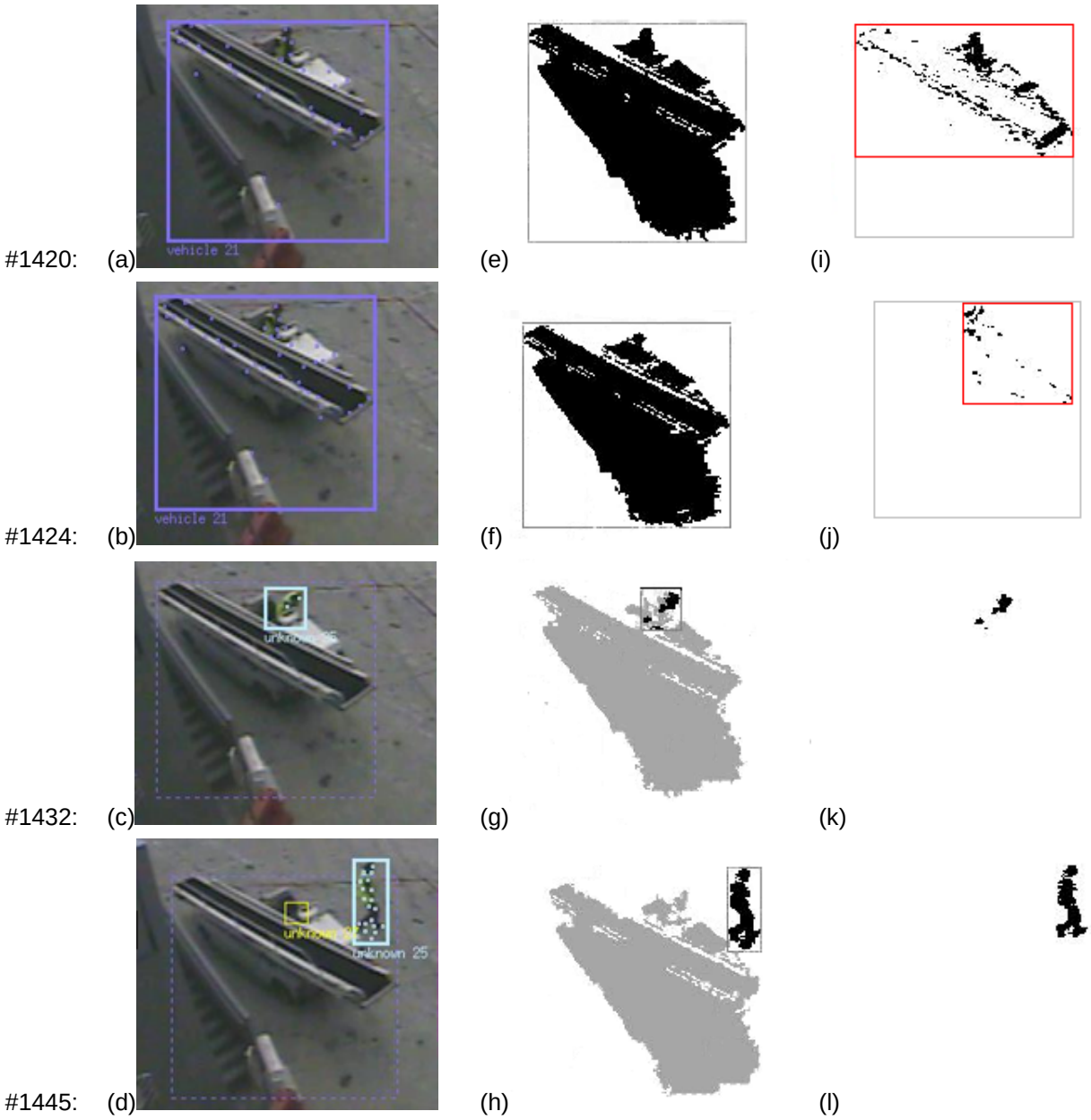


Fig. 5.3 Handling of a stationary object with partial motion within it.
These series of figures (a) to (d) show a conveyor belt vehicle coming to rest, and the driver leaving the vehicle. Figures (e) to (h) show the pixels labelled as foreground (black) and pixels integrated into the background model (light grey), while figures (i) to (l) show the pixels that are actually in motion. In frame 1424, the bounding box of the pixels actually in motion (red box in (j)) compared to the bounding box of the foreground pixels becomes less than the threshold STATIC_OBJECT_REACTIVATION_MOTION_RATIO, and therefore the object is integrated into the background in the next frame (light grey area in (g)). This allows the person to be separated from the vehicle, even though some parts of the vehicle were still in motion at that time. Note that in (d), a new object is detected (bounded by a yellow box) – this is the person's ghost and is a side-effect caused by background integration. These ghosts are eliminated later on by the Feature Tracker.

Rule 4. *Ideal Static Match:*
The first such rule handles ideal matches between a static prediction and a static measurement – a one-to-one match. This is the same as Rule 1 above, for normal objects, except that the prediction and measurement must have their state set to STATIC.

$$\begin{aligned} &f(P_{S1}, M_{S2}) > 0 \quad P_{S1}, M_{S2} \text{ static} \quad \text{and} \\ &f(P_k, M_{S2}) = 0, \quad f(P_{S1}, M_l) = 0 \quad \forall k \neq s1,\ l \neq s2 \end{aligned} \tag{5.12}$$

Rule 5. *New Motion within Static:*
If a static prediction matches a static measurement, as well as other new measurements, then this rule applies. The other new measurements must not match any other prediction apart from the static one. If the area of the new measurements is large enough, the static object is re-activated according to the method described previously.

$$\begin{aligned} &f(P_{S1}, M_{S2}) > 0 \quad P_{S1}, M_{S2} \text{ static} \quad \text{and} \\ &f(P_{S1}, M_j) > 0 \quad M_j \in M1 \subseteq M,\ |M1| > 1 \quad \text{and} \\ &f(P_k, M_j) = 0, \quad f(P_k, M_{S2}) = 0 \quad \forall M_j \notin M1,\ k \neq s1 \\ &f(P_{S1}, M_l) = 0 \quad \forall M_l \notin M1,\ l \neq s2 \end{aligned} \tag{5.13}$$

Rule 6. *Existing Motion within Static:*
If a static prediction matches a static measurement, as well as other measurements that in turn match other predictions, then this rule is used. The other measurements represent existing objects that are moving in front of the static object. The static object is not reactivated.

$$\begin{aligned} &f(P_{S1}, M_{S2}) > 0 \quad P_{S1}, M_{S2} \text{ static} \quad \text{and} \\ &f(P_{S1}, M_j) > 0 \quad M_j \in M1 \subseteq M,\ |M1| > 1 \quad \text{and} \\ &f(P_k, M_{S2}) = 0 \quad k \neq s1 \quad \text{and} \\ &f(P_{S1}, M_l) = 0 \quad \forall M_l \notin M1,\ l \neq s2 \end{aligned} \tag{5.14}$$

Rule 7. New and *Existing Motion within Static:*
This is a combination of Rules 5 and 6. The non-static measurements matching with the static prediction are divided into 'new motion' ones (those that do not match any other prediction) and 'existing motion' (match another prediction), and each is handled according to rule 5 and rule 6 respectively.

Combining the Matching Rules

The matching rules described above address cases such as object splitting, merging, etc., separately. In most cases, a combination of these cases will arise at any one time. The following algorithm indicates how these rules are called iteratively to handle most of the complex combination of cases that can arise.

1. For each remaining unmatched Static Prediction or unmatched Static Measurement:
   1.1. Run Rule 5: New Motion within Static
   1.2. Run Rule 6: Existing Motion within Static
   1.3. Run Rule 7: New and Existing Motion within Static
   1.4. Run Rule 4: Ideal Static Match
2. For any remaining unmatched Prediction:

2.1. Run Rule 1: Ideal Match

2.2. Run Rule 2: Object Splitting

2.3. Run Rule 3: Object Merging

3. Check Matched Static Predictions for potential re-activation
4. Handle Ghosts caused by incorporating Static Objects into the Background Model
5. Check if Matched Predictions have become stationary and include them into the Background Model.

5.3.4. HANDLE NEW AND LOST OBJECTS

After matching predictions $\{P_i\}$ to measurements $\{M_j\}$, any remaining unmatched Predictions are considered to be 'lost'. A count is kept of how many consecutive frames, a prediction has been lost. If this count exceeds the parameter MAX_KEEP_ALIVE, then the prediction is deleted permanently

Any remaining unmatched Measurements will be considered to be new objects that first appeared in the scene at time *t*. The KLT algorithm is used to select features for these new objects, using the pixels labelled foreground as a mask.

5.3.5. UPDATE OBJECT STATE

For each prediction that has been successfully matched with a measurement(s), if the number of remaining features is less than the allowed number *N*, as calculated by (5.6), then the KLT algorithm is used to select new features to replace the lost ones. In case of merged objects, the search for new features is performed only in the non-overlapping parts of the merged region.

The remaining features are also checked to make sure that they are within the bounding box of the prediction and have not become attached to some part of the background. Otherwise, the object can leave a trail of incorrectly-tracked features behind it as it moves across the scene.

5.4. EXPERIMENTAL RESULTS

This implementation of the Local Feature Tracker has been integrated into the *AvitrackFrameTracker* application [14] and was evaluated using several of the AVITRACK test sequences. Overall, the algorithm was found to have a good performance. More specifically:

- Objects can be tracked successfully with a low and sparse number of features (a feature density of 0.2, and a lower minimum of 20 features was found to give good results).
- The matching rules used by the Feature Tracker can explain most of the interactions in the scene.

The Feature Tracker also suffers from problems:

- Local features tend to have a short lifetime, so they have to be replaced when they are lost (no longer tracked). This can cause problems when objects remain merged or occluded for a long time, as the loss rate is normally faster than the replacement rate.

- For objects with little texture, the 'best' local features tend to be found near the boundary of the object. These windows include both part of the object and part of the background and are less reliable to track.
- The tracker has difficulty in initialising the tracking of new objects if these appear within a cluttered area.
- The algorithm is sensitive to some of the parameters and is also affected by noise from the motion detector, as it uses the regions returned by the motion detector as the basis for feature selection and during the matching phase.

5.5. CONCLUSIONS

This implementation of the Local Feature Tracker was found to give good results when applied to AVITRACK test sequences. The processing speed is also quite good (approx. 8 to 10 fps at present). With further optimisation, it should not be a problem to achieve run-time performance. Some problems were identified with the tracking results of the algorithm and these will be addressed to try and improve the tracking accuracy of the algorithm.

6. COLOUR TRACKER

This algorithm tracks objects by using their global colour information. In general, colour information is more robust to changes in an object's size and orientation, and should be more stable over longer periods of time than other non-colour based features. The object's colour histogram is used as the colour model.

This tracker uses a combination of two methods to track objects: a colour histogram similarity measure, based on the Bhattacharyya coefficient, is used for matching isolated objects directly [5,6]; and the CamShift algorithm (an extension of the Mean Shift algorithm) is used to track objects when they are in a merged or partially occluded state [7,8].

6.1. THE OBJECT COLOUR MODEL

The HSV colour space was selected for representing the object's colour histogram, since this colour space separates the chromaticity components (Hue and Saturation) from the brightness component (Value). The chromaticity components are more robust to illumination changes and to changes in the object's orientation.

However one disadvantage of the HSV colour space is the instability that occurs in the hue component when the saturation or the brightness value is very low. For this implementation, pixels with a saturation or value less than $0.05\mathrm{S}_{\max}$ and $0.05\mathrm{V}_{\max}$ respectively, are considered to be "colourless" pixels (hue undefined).

The object colour model consists of a 2D hue-saturation histogram, with the number of bins specified by the parameters NUM_HUE_BINS and NUM_SATURATION_BINS. Only pixels with a valid hue are used for building the hue-saturation histogram. From some initial tests with this tracker, it was found that adding a 1D histogram for representing the brightness of the "colourless" pixels improves the tracking of objects with very little colour. The number of bins for this 1D histogram is specified by parameter NUM_VALUE_BINS.

$$\begin{aligned} &f_{HS}(i,j) && i \in 0..\text{NUM\_HUE\_BINS} - 1,\ \ j \in 0..\text{NUM\_SATURATION\_BINS} - 1 \\ &f_{V}(i) && i \in 0..\text{NUM\_VALUE\_BINS} - 1 \end{aligned} \tag{6.1}$$

The histograms are stored in a normalised form, i.e. they must satisfy the following condition:

$$\left(\sum_{\forall i,j} f_{HS}(i,j)\right) + \left(\sum_{\forall i} f_{V}(i)\right) = 1 \tag{6.2}$$

6.2. THE BHATTACHARYYA COEFFICIENT

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*. This metric has many desirable properties, many of which are examined in [5,6], including self-consistency and having a fixed bias (compared to the well-known χ^2 test).

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

$$\rho_B(f,g) = \sum_{i=1}^{N} \sqrt{f(i)\,g(i)}$$
$$\text{provided: } \sum_i f(i) = 1, \sum_i g(i) = 1 \qquad (6.3)$$

The coefficient ρ_B is within the range [0..1], with ρ_B =1 when the two histograms match perfectly. Then the similarity measure M_{col} for the colour models of any two objects is defined to be equal to (6.4), returning a value within the range [0..1], with 1 for a perfect match.

$$M_{\text{col}}(f,g) = \rho_B(f_{HS}, g_{HS}) + \rho_B(f_V, g_V) \qquad (6.4)$$

This similarity measure is used for matching objects when they are isolated from each other. Their histograms are considered to match if the value M_{col} is larger than COLOUR_SIMILARITY_THRESHOLD.

6.3. THE CAMSHIFT ALGORITHM

For non-isolated objects, a direct matching of the colour histograms is not possible, as it cannot be determined directly which part of the merged region belongs to which object. Instead, the CamShift algorithm is used to track the objects through merging and partial occlusion. CamShift stands for "Continuously Adaptive Mean Shift" algorithm, described in [8], and it is a generalisation of the Mean Shift algorithm of [7].

Mean Shift Algorithm

Given an object *O* that has been successfully tracked at time *t-1*, the Mean Shift algorithm is used to find its new position at the current time *t*. From the position of the object at time *t-1*, and a prediction generated by the tracker of its new position at time *t*, a search area is defined. The size of this search area is controlled by parameter MAX_SEARCH_DISTANCE.

The object's colour histogram is used to generate a probability map M_{prob} of this search area. This is performed by taking the HSV colour *c* of each pixel (*x,y*) within the search area of the image, locating the bin in the histogram for this pixel colour value, and then filling the probability map with the histogram value. Since the histogram is normalised, it can be viewed as a discrete probability distribution function, and the histogram value indicates the probability that a pixel belonging to the object takes a colour value within the range of that particular histogram bin.

For this implementation, only pixels within the search area that have been labelled as foreground by the motion detector are considered; the rest are set to 0 in the probability map. Also, the histogram used for the back-projection depends on whether the pixel's colour *c* is 'colourless' or not. In (6.5) below, the function *b*(), maps the colour value into the corresponding bin of the histogram.

$$M_{\text{prob}}(x,y) = \begin{cases} 0 & \text{if } (x,y) \notin \text{search area} \\ 0 & \text{if } (x,y) \text{ is not a 'foreground' pixel} \\ f_{HS}\left(b(c_H), b(c_S)\right) & \text{if pixel is a 'coloured' pixel} \\ f_V\left(b(c_V)\right) & \text{if pixel is a 'colourless' pixel} \end{cases} \quad \text{where } c = \langle c_H, c_S, c_V \rangle \qquad (6.5)$$

Fig 6.1 (next page) shows the probability map that is generated for an object, together with its colour histogram model.

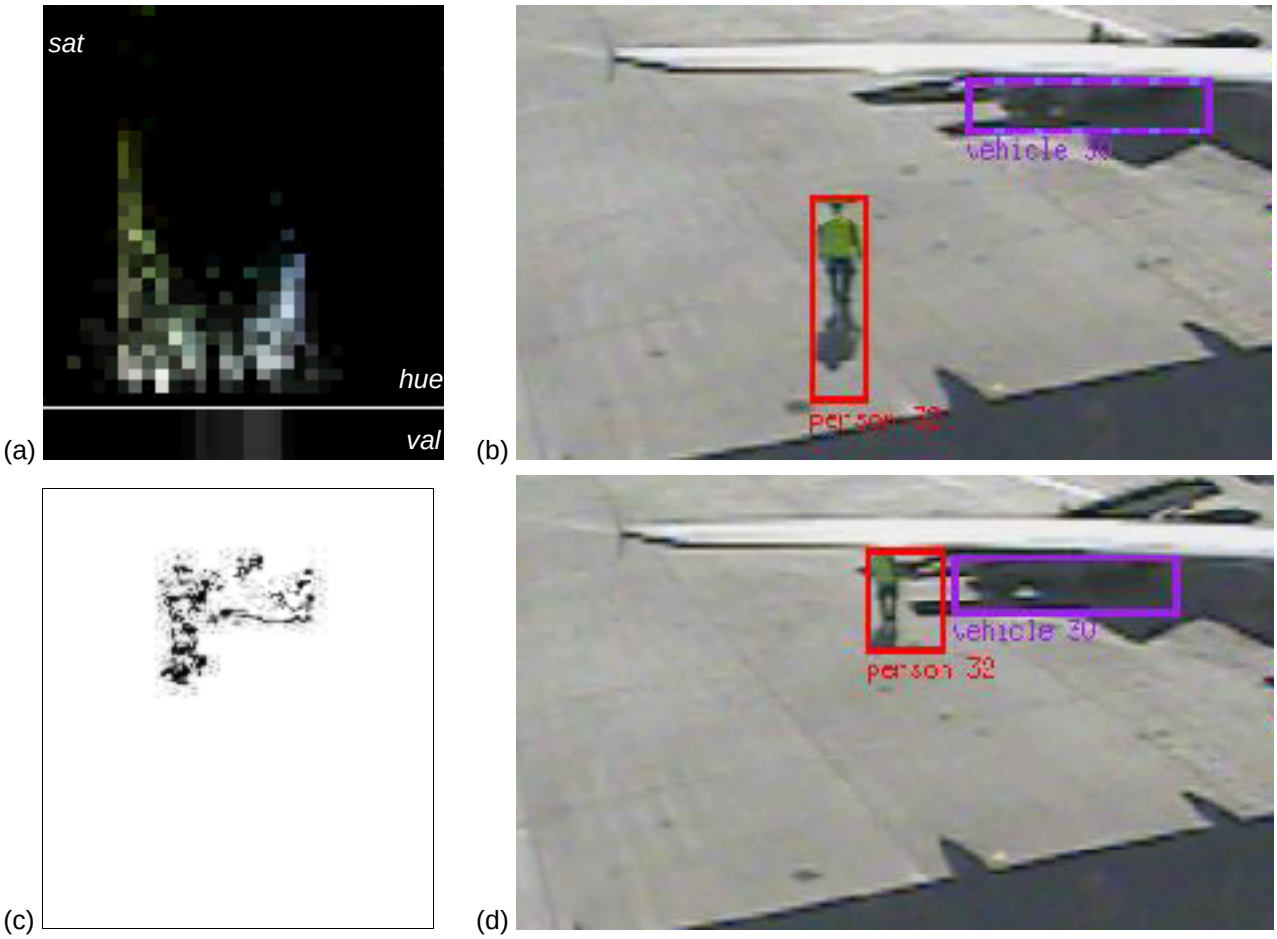


Fig. 6.1 (a) shows the 2D hue-saturation and 1D brightness histograms for the person shown in (b). This colour model is back-projected onto image (d) and the probability map within the search area is shown in (c). Only pixels labelled as foreground by the motion detector are considered. The new object's position as tracked by the CamShift algorithm is shown in red in (d).

Given the probability map M_{prob}, the Mean Shift algorithm finds the centre of mass (the mean location) within the search area, by computing the zeroth and first moments:

$$\begin{aligned} &\text{zeroth moment:} && M_{00} = \sum_x \sum_y M_{\text{prob}}(x,y) \\ &\text{first moments:} && M_{10} = \sum_x \sum_y x\, M_{\text{prob}}(x,y) \\ & && M_{01} = \sum_x \sum_y y\, M_{\text{prob}}(x,y) \\ &x_c = \frac{M_{10}}{M_{00}}, && y_c = \frac{M_{01}}{M_{00}} \end{aligned} \tag{6.6}$$

This is then used to change the centre of the search window by taking the average of the previous window centre (x_{c0}, y_{c0}) and the new mean location (x_c, y_c).

$$(x_c, y_c) \leftarrow \frac{1}{2}\left[(x_c, y_c) + (x_{c0}, y_{c0})\right] \tag{6.7}$$

The process of finding the mean (6.6) and shifting (re-centering) the window (6.7) is repeated until the process converges to 1 pixel, or until the maximum number of iterations is reached, in which case a tracking failure occurs. Tracking can also fail if the zeroth moment becomes 0 – this happens when the entire search area has zero probability.

CamShift Algorithm

One problem with the Mean Shift algorithm is the in-built assumption that the colour distribution remains unchanged. This is certainly not the case when tracking objects, as the colour histogram will undergo changes from one frame to the next. The CamShift algorithm generalises the Mean Shift algorithm by taking into account the fact that colour distributions change dynamically. CamShift uses the zeroth moment to continuously adapt the size of the search window when tracking an object [8].

6.4. THE OBJECT TRACKING ALGORITHM

Given a list of known objects $\{O_i\}$ at time *t-1*, the tracking process for time *t* can be summarised as follows:

1. Generate object predictions $\{P_i\}$ for time *t* from the list of known objects $\{O_i\}$ at *t-1*.
2. Given a list of measurement objects $\{M_j\}$ detected by the motion detector at time *t*, build the colour models for the measurement objects.
3. The list of predictions $\{P_i\}$ is matched with the measurement objects $\{M_j\}$.
4. Any remaining unmatched predictions in $\{P_i\}$ are set to 'lost'. Any remaining unmatched measurement objects in $\{M_j\}$ are considered as possible new objects.
5. Update the state of those predictions in $\{P_i\}$ that were matched to measurement objects. If the predictions are not in a merged state, then their colour model is updated. The final result is a list of tracked objects $\{O_i\}$ at time *t*.
6. Let *t* = *t+1* and repeat step 1.

These steps are described in further detail in the following sections.

6.4.1. GENERATE OBJECT PREDICTIONS

Given a list of objects $\{O_i\}$ that have been tracked successfully in the previous frame (*t-1*), we generate predictions $\mathrm{P} = \{P_i\}$ of where these objects should be located at time *t*. These predictions are based on the assumption that an object continues moving with approximately the same speed and direction as when last seen in the previous few frames. For each object, a history *H* of the last known positions of the object is kept, and from this the predicted motion is generated. The length of this motion history is set to 5 for this implementation to reduce the effects of noise, and the history is weighted to give more evidence to the most recent entries.

$$H = \{\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \mathbf{x}_{t-3}, \mathbf{x}_{t-4}, \mathbf{x}_{t-5}\}$$
$$\mathbf{x}_{\Delta} = 0.35\mathbf{x}_{t-1} + 0.25\mathbf{x}_{t-2} + 0.2\mathbf{x}_{t-3} + 0.1\mathbf{x}_{t-4} + 0.1\mathbf{x}_{t-5} \quad (6.8)$$
$$\mathbf{x}'_t = \mathbf{x}_{t-1} + \mathbf{x}_{\Delta}$$

6.4.2. BUILD COLOUR MODELS

The motion detector returns a list of measurement objects $\mathrm{M} = \{M_j\}$ detected by the motion detector at time *t*. The colour models for these measurements are constructed at this stage, according to (6.1) and (6.2), and then these are used in the next step for matching and tracking.

6.4.3. MATCH PREDICTIONS TO MEASUREMENTS

Taking all possible pairs of predictions and measurement objects (P_i, M_j) that are within MAX_SEARCH_DISTANCE of each other, the histogram similarity measure M_{col}, defined by (6.4), is used to generate similarity scores. These are placed in a match score table *S*. If a score is below the parameter COLOUR_SIMILARITY_THRESHOLD, then the score is set to 0. If a prediction and a measurement overlap, then an additional score is added to ensure that it is not 0.

The Matching Rules

For matching predictions to measurements, this tracker uses a similar approach to that of the Local Feature Tracker – together with the score table, a set of matching rules are used to handle interactions between objects, such as object splitting, merging situations, etc. Much of this similar functionality is in fact implemented as common code available to both trackers. The matching rules used by the Colour Tracker are listed below. Only differences specific to the colour tracking algorithm are described in detail below – for the common functionality, please refer to the more detailed description of the previous tracker (§5.3.3).

Rule 1. *Ideal Match*:
A one-to-one match between a prediction and a measurement. The same condition applies except for using the score table *S* instead of the matching function f.

Rule 2. *Splitting Object*:
A prediction matches with several measurements – a one-to-many match. The prediction is split into new objects, one for each of the matched measurements. The new prediction objects are assigned the colour models of their corresponding measurements with which they are associated (these are the colour models created in §6.4.2)

Rule 3. *Merging Objects with CamShift Tracking*:
The previous rules made use of the Bhattacharyya similarity measures directly from the score table, as the predictions and measurements are distinct from each other, and so a histogram comparison is sufficient. But this third matching rule applies to cases where predictions and measurements are merged together and no longer distinct. Instead, the CamShift algorithm is used to track the predictions while in a merged state. The condition below identifies such cases:

$$S(P_i, M_j) > 0 \quad \text{and}$$
$$S(P_k, M_j) > 0 \quad \text{for some } P_k \neq P_i \quad (6.9)$$

Using the predicted position $\mathbf{x}'_t$ of P_i given by (6.8), the search area is defined as the bounding box of the prediction centred at $\mathbf{x}'_t$ and enlarged by MAX_SEARCH_AREA. The probability map of this search area is generated by histogram back-projection as described in (6.5). Only those pixels of the search area labelled 'foreground' are considered – for the rest the probability map is set to 0.

The CamShift algorithm is then performed on this search area and using the probability map. The output of the CamShift algorithm can be one of the following 3 outcomes:

(1) Tracking fails because the maximum number of iterations is exceeded while performing the Mean Shift algorithm,

(2) Tracking fails because the zeroth moment is 0,

(3) The prediction is tracked successfully, and CamShift returns the new position $\mathbf{x}_t$ of the prediction at time *t*.

In case of the first two outcomes, the state of the prediction P_i is set to 'lost'. If P_i is tracked successfully, an extra verification step is performed. This consists of building the colour model at the new position $\mathbf{x}_t$ returned by the CamShift algorithm and comparing this to the prediction's existing colour model using the histogram similarity measure (6.4). If the two colour models do not match, then tracking is considered to have failed; else a success is reported. This extra verification step is needed because the CamShift tracker can sometimes become distracted and drift away from its real target.

Handling Stationary Objects

The handling of stationary objects is identical to that of the Feature Tracker, except that the Colour Tracker checks for stationary objects by comparing the motion of the centre of the object.

Combining the Matching Rules

The following algorithm indicates the order in which the matching rules are called:

1. For each remaining unmatched Static Prediction or unmatched Static Measurement:
   1.1. Run Rule 5: New Motion within Static
   1.2. Run Rule 6: Existing Motion within Static
   1.3. Run Rule 7: New and Existing Motion within Static
   1.4. Run Rule 4: Ideal Static Match
2. Run Rule 1: Ideal Match
3. Run Rule 2: Object Splitting
4. For any remaining unmatched Prediction:
   4.1. Run Rule 3: Object Merging with CamShift Tracking
5. Check Matched Static Predictions for potential re-activation
6. Handle Ghosts caused by incorporating Static Objects into the Background Model

7. Check if Matched Predictions have become stationary and include them into the Background Model.

6.4.4. HANDLE NEW AND LOST OBJECTS

After matching predictions $\{P_i\}$ to measurements $\{M_j\}$, any remaining unmatched Predictions are considered to be 'lost'. A count is kept of how many consecutive frames, a prediction has been lost. If this count exceeds the parameter MAX_KEEP_ALIVE, then the prediction is deleted permanently

Any remaining unmatched Measurements will be considered to be new objects that first appeared in the scene at time *t*.

6.5. EXPERIMENTAL RESULTS

This tracker has been integrated into the *AvitrackFrameTracker* application [14]. When it was tested with AVITRACK test sequences, it was found to perform quite badly, with a lot of tracking errors. The main reason is that the majority of objects in AVITRACK are colourless and similar to each other (in terms of their brightness histogram). This then causes the Mean Shift algorithm to lose track of the object and it starts drifting around the image, especially in areas of clutter.

If reliable colour information is available in the scene, such as when tested against the PETS 2001 dataset 1, the results of the Colour Tracker were found to be good. This implies that the implementation of the algorithm appears not to be at fault.

6.6. CONCLUSIONS

This implementation of the Colour Tracker performs quite badly on the AVITRACK test sequences. The main reason is due to the lack of colour information in most of the objects being tracked. Further work needs to be done to try and improve the tracking results.

7. CLUSTER TRACKER

The tracking of moving objects inside a sequence taken from a fixed camera usually involves image differencing and posterior thresholding. After this processing, a cloud of points is obtained that later, somehow, must be grouped and assigned to clusters that coarsely represent moving objects. Performing this process without performing object identification is a difficult task. However, accurate object identification is a time-consuming task, so that if the tracking must be carried out in real-time this option is not available.

In [9-11], a tracking algorithm based on grouping clusters of moving points without object recognition is developed. This algorithm is based on a probabilistic generative model for the clusters, which allows maximum-likelihood estimation of the cluster parameters by the EM algorithm. The generative model is non-Gaussian in grey-level values, thus providing a better fit to the statistics of natural images. Also, the number of clusters is not fixed a priori, but dynamically updated on the basis of evidence from the image. In this respect, this method is similar to approaches such as layers of support [12].

Along [9-11], different aspects of the algorithm have been changed, but a common structure can be extracted. Based upon this structure, a new implementation of the algorithm has been proposed and tested. This new implementation introduces some changes on the original algorithm which make it faster without losing performance.

7.1. DESCRIPTION OF THE ORIGINAL ALGORITHM

As stated in §7, although different versions of the algorithm have been described, its essential characteristics remain unchanged and can be summarized as follows:

1. Model is a mixture of two kinds of clusters:
   a. Target clusters.
   b. Background cluster.
2. Every model *m* is characterized by the probability $f_m(\vec{u}, I(\vec{u}))$ it has of generating a pixel with co-ordinates $\vec{u}$ and gray-level value $I(\vec{u})$.
3. This probability is a separable function of the pixel co-ordinates and of the pixel gray level, so that: $f_m(\vec{u}, I(\vec{u})) = g_m(\vec{u}) \cdot h_m(I(\vec{u}))$.
4. Model parameters are determined by the Expectation-Maximization (EM) algorithm.

In the following subsections, an outline of these characteristics is described.

Target clusters

The probability $f_m(\vec{u}, I(\vec{u}))$ that a pixel with coordinates $\vec{u}$ and gray level $I(\vec{u})$ is generated for target cluster *m* is given by:

$$f_m(\vec{u}, I(\vec{u})) = g_m(\vec{u}) \cdot h_m(I(\vec{u})) \tag{7.1}$$

where:

$$g_m(\vec{u}) = \frac{1}{2\pi\sqrt{|\Sigma_m|}} \exp\left(-\frac{1}{2} \Delta u_m^T \cdot \Sigma_m^{-1} \cdot \Delta u_m \right) \tag{7.2}$$

$$h_m(I(\vec{u})) = 1/q \tag{7.3}$$

In (7.2) $\Delta u_m^T = (\vec{u} - \vec{c}_m)^T$ is the vector difference between the pixel's co-ordinates and the cluster center $\vec{c}_m$ (one of the cluster parameters). The other cluster parameters are the elements of $\sum_m^{-1}$, the covariance matrix of the cluster. This equation leads to a geometrical interpretation of $f_m(\vec{u}, I(\vec{u}))$ as a exponentially-decreasing function of the pixel co-ordinates, with ellipsoidal-like contour lines. To see this, the $\sum_m$ can be decomposed by means of Singular Value Decomposition into the following product:

$$\sum_j = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} L_{MAX} & 0 \\ 0 & L_{MIN} \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}^{-1} \tag{7.4}$$

In Fig. 7.1, a picture of the geometrical interpretation of the model parameters can be observed.

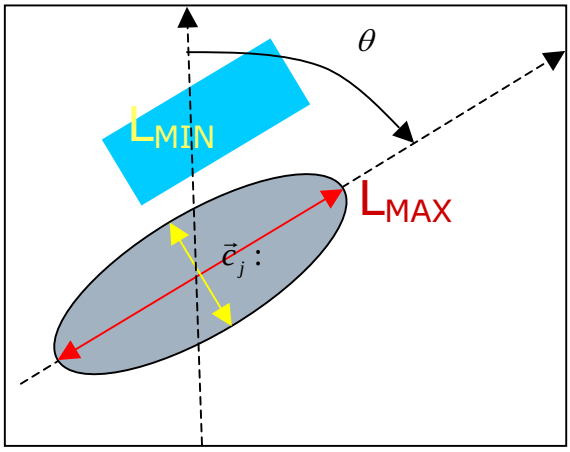


Fig. 7.1. Geometrical interpretation of $f_m(\vec{u}, I(\vec{u}))$.

Equation (7.3) means that the probability $f_m(\vec{u}, I(\vec{u}))$ is independent of the gray level of the pixel, as *q* is constant and equal to the number of different grey levels that can be encountered in the image (typically, *q* = 256 for 8-bit grey level images).

Background cluster

Equation (7.1) also describes the probability distribution for the background cluster. However, the expression for the $g_m(\vec{u})$ and $h_m(I(\vec{u}))$ functions changes:

$$g_j(\vec{u}) = 1/m \tag{7.5}$$

$$h_0(I(\vec{u})) = \frac{1}{2\mu_0} \exp\left(-\left| \frac{\delta(\vec{u})}{\mu_0} \right| \right) \tag{7.6}$$

As can be seen in (7.5), the probability that a pixel belongs to (or is generated by) the background cluster does not depend on the pixel co-ordinates, as the term m in (7.5) is constant and equal to the total number

of pixels of the image. However, as shown in (7.6), it depends on $\delta(\vec{u}) = I(\vec{u}) - R(\vec{u})$, the difference between the grey level of the pixel and the corresponding one in the reference image. The only parameter of the background cluster is μ_0, calculated as the mean of the absolute value of the grey-level difference of pixels in the background.

7.2. THE CLUSTER TRACKING ALGORITHM

The algorithm can be summarized in the following steps:

1. Detection of new isolated clusters.
2. Optimisation (EM algorithm).
3. Cluster splitting.
4. New optimization (if any cluster has split).
5. Cluster merging and elimination.
6. Updating of the reference image and background cluster parameters.

In the following subsections, each of these steps will be briefly explained.

Detection of new isolated clusters.

New clusters are detected by locating maxima in the difference image, after accounting for the clusters already known to be present in the previous frame of the image sequence. Specifically, a “corrected” image difference is computed, in which each grey-level difference is weighted by the probability of its belonging to the background cluster:

$$\delta_c(\vec{u}) = \delta(\vec{u}) \cdot p_0^0(\vec{u}, I(\vec{u})) \tag{7.7}$$

where p_0^0 is the *a posteriori* probability of the background cluster calculated in the previous frame. The a posteriori probabilities of every cluster are computed as:

$$p_j(\vec{u}) = \frac{w_j \cdot f_j(\vec{u}, I)}{\sum_{k=1}^{N} w_k \cdot f_k(\vec{u}, I)} \tag{7.8}$$

where all the calculus are performed with the cluster parameters carried over from the previous frame.

This corrected image difference in (7.7) is smoothed and downsampled by a factor of to obtain the coarse-grained difference image. Local maxima of are assumed to arise from new targets if the corresponding value of is greater than a threshold. This threshold is deduced in [11] as:

$$\mu_0 \left(1 + \log \frac{q}{2\mu_0} \right) \tag{7.9}$$

Optimisation

The cluster parameters are iteratively updated by means of the EM algorithm [13], using the *a posteriori* probabilities given by (7.8). In [9, 10], the EM algorithm is employed, while in [11] the MAP (maximum *a posteriori*) version of the EM algorithm is used, to avoid large variations of the cluster parameters.

Cluster splitting

Clusters are considered for splitting if their shape is very different from the ellipsoid described by the top-hat distribution:

$$g_j^{TH}(\vec{u}) = \begin{cases} \left(4\pi\sqrt{|\Sigma_j|}\right)^{-1} & \text{if } \Delta\vec{u}_j^T \cdot \Sigma_j^{-1} \cdot \Delta\vec{u}_j \le 4 \\ 0 & \text{otherwise} \end{cases} \quad (7.10)$$

using the same centroid and covariance as estimated with the Gaussian model. To test whether the observed density is significantly different from this expectation, the ellipsoid is divided into 9 sections orthogonally to its main axis and the squared deviation of the observed density from the expected density is computed for each section, and normalized by the expected density. If the largest *negative* deviation from the expected density is below a threshold, the cluster is split at the location of the bin with this largest negative deviation.

New optimization (if any cluster has split).

Only if some cluster has been split, a new pass of the EM algorithm is performed in order to adjust the new cluster parameters.

Cluster merging and elimination.

Two clusters are *merged* if:

- Their relative Mahalanobis distances are less than a certain threshold, and
- The merging cost is under a certain quantity.

One cluster is *eliminated* if the cost of merging it with the background is under a certain threshold.

Cluster merging and elimination is a key step in the algorithm. The functionals that measure the cost of merging two clusters change depending on the algorithm version, but all of them share the idea that a cluster with low density (few moving points inside a certain Mahalanobis distance of the cluster centre) is more likely to be merged.

One important question that arises when two clusters are going to be merged is the identity preservation, i.e. what identity must be assigned to the new resultant cluster. When a target cluster is split into two, the larger of the new clusters is assigned the identity of the parent cluster. When two target clusters are merged, the new cluster is assigned the identity of the larger of the two merged clusters, unless the smaller cluster is the parent of the larger one, in which case the identity of the smaller cluster is assigned to the new one.

Updating of the reference image and background cluster parameters.

Lastly, the reference image for frame *t+1* is calculated by updating the reference image for frame *t* with the values of the difference image in the same frame. The update of each pixel of the reference image is weighted by the posterior probabilities of its belonging to the background cluster as stated in (7.11).

$$r^{t+1}(\vec{u}) = r^t(\vec{u}) + \frac{1}{\tau_r} p_0^t \cdot \delta^t(\vec{u}) \quad (7.11)$$

where τ_r is a parameter that measures the velocity of adaptation to slow changes in the background. Also, the background parameter is updated as:

$$\mu_0^{t+1}(\vec{u}) = \frac{\sum_{\vec{u}} p_0^t(\vec{u}) \cdot \left|\delta^t(\vec{u})\right|}{\sum_{\vec{u}} p_0^t(\vec{u})} \tag{7.12}$$

7.3. THE NEW IMPLEMENTATION

In order to speed up the algorithm without sacrificing performance, the algorithm has been reduced to three main steps, and some of the procedures have been changed. The new proposed structure is:

1. Detection of new isolated clusters (different from the original).
2. Optimisation, including cluster merging and elimination.
3. Updating of the reference image and background cluster parameters.

In what follows, a detailed explanation of the motivations and advantages achieved with the proposed changes will be carried out.

Detection of new isolated clusters (different from the original).

The new procedure proposed for the detection and initialization of the new isolated clusters is as follows:

1. Detecting local maxima in the corrected difference image.
2. Grouping those maxima belonging to the same connected blob.
3. Calculating the centre and correlation matrix from the blob data

In the former versions of the algorithm, each point from (7.7) above the threshold (7.9) was considered as the center of a new cluster. The covariance matrices of these new clusters were initialised with default values. This usually generated a large quantity of new clusters that slowed down the algorithm, as the duration of the EM phase greatly depends on the number of clusters considered. By grouping together the points belonging to the same connected blob, two advantages are achieved:

- First, less new models are generated, with the subsequent savings in time in all the posterior processing.
- Second, accurate estimations can be made for the initial values of the cluster parameters.

These values can be obtained from the moments of order 0, 1, and 2 of the blobs. The 0^{th}-order moment (area) divided by the total number of pixels on the image gives a good approximation for the *a priori* possibility that a pixel belongs to the cluster. The first-order moments can be used for the center of the cluster, and the covariance matrix is obtained directly from the central second-order moments m_{ij} as:

$$\Sigma = \begin{pmatrix} m_{20} & m_{11} \\ m_{11} & m_{02} \end{pmatrix} \tag{7.13}$$

Optionally, a dilate operation can be performed on the binary image of the thresholded image before blob labeling. This usually reduces the number of new clusters found, without causing a loss of performance.

Optimisation, including cluster merging and elimination.

As a difference with the last implementation in [11], cluster merging and elimination is carried out in each iteration of the EM algorithm. This might give the impression that it is more computationally expensive than if performed after the EM has finished, but in fact it proves to speed up the processing. The reason is that usually, and especially when a large number of new clusters have been detected in the first step of the algorithm, a great deal of the clusters either merge or are eliminated in the first iterations of the EM adjust, so that the rest of the iterations are performed more quickly. Also, the final EM fine-tuning in the case of cluster splitting is avoided.

Another difference is that in this implementation, cluster splitting is not considered. It has been found that, at least in the image sequences employed in the tests, old clusters naturally split when new clusters are detected in the new frame, and the corresponding objects are far enough.

Updating of the reference image and background cluster parameters.

This last step of the algorithm is maintained as in [9-11] and therefore, equations (7.11) and (7.12) are used for the update.

7.4. EXPERIMENTAL RESULTS

The proposed modification has been implemented both in Matlab and C++ as a standalone application. It is also currently in the process of being integrated into the *AvitrackFrameTracker* application [14].

Several tests were carried out using image sequences from the PETS database and from the AVITRACK project. The algorithm showed a good performance, although it is very sensitive to some of the parameters, like the minimum thresholds used for cluster merging and elimination.

Also, in this stage it is not able of dealing with extreme changes of illumination, and in the tracking there appear some broken trajectories due to failures in identity preservation when two clusters are merged. However, this is not a serious drawback, as this algorithm is supposed to be the lowest layer of tracking, and therefore the identity preservation and tracking coherence must be performed by higher-level processes.

7.5. CONCLUSIONS

In this work, a new implementation of the algorithm described in [9-11] has been detailed and tested. This implementation includes some changes that make the algorithm faster while maintaining its properties and performance.

The tests have shown that the algorithm is able of making reasonable tracking without carrying out object recognition. However, to avoid losing the coherence of the tracking when moving objects overlap, the use of some higher-level algorithms which process the outputs of the tracker is required.

8. CONFIGURATION PARAMETERS

Table 8.1 below lists the parameters that need to be configured for each of the tracking algorithms.

Algorithm	Parameter	Description	Value used
Local Feature Tracker	WINDOW_SIZE	Determines the size of the local feature window.	5x5
	FEATURE_DENSITY	Used to determine the maximum number of features that are selected and tracked per object. This density value is used together with the object's area and the WINDOW_SIZE parameter.	0.2
	MIN_FEATURES_PER_OBJECT	Places a lower limit on the number of features per object.	20
	MAX_FEATURES_PER_OBJECT	Places a maximum limit on the number of features per object.	50
	MAX_ITERATIONS	The maximum of iterations that are performed when tracking a local feature, before setting the tracking state of the feature to 'lost'.	10
	MAX_RESIDUE	The maximum residue amount that is allowed before setting the tracking state of the feature to 'lost'.	10
	NEW_REGION_AREA_MATCH_THRESHOLD	A threshold on the maximum area difference allowed when trying to recover lost objects.	60
	MAX_OBJECTS	The maximum number of objects that can be tracked at any one time. Used in the allocation of program data structures.	500
	STATIC_COUNT_INCORPORATE	If an object is stationary for this number of frames, then it is added to the background model of the Motion Detector.	6
	STATIC_OBJECT_REACTIVATION_MOTION_RATIO	When checking if an object is stationary or not, if the area of motion within the object over the object's full area is less than this value, then the object is considered to be stationary.	0.25
	MAX_KEEP_ALIVE	If an object is lost during tracking, it is maintained for this number of frames to try and recover it, before deleting it.	32

Colour based Tracker	NUM_HUE_BINS	Number of bins allocated to the hue colour component in the object's colour histogram.	32
	NUM_SATURATION_BINS	Number of bins allocated to the saturation colour component in the object's colour histogram.	32
	NUM_VALUE_BINS	Number of bins allocated to the value brightness component in the object's colour histogram.	32
	MAX_SEARCH_DISTANCE	The maximum spatial distance used in tracking an object from one frame to the next. Expressed as a ratio of the image width.	0.05
	COLOUR_SIMILARITY_THRESHOLD	The lower threshold used for determining colour similarity when using the Bhattacharrya coefficient.	0.2
	MAX_OBJECTS	See previous algorithm.	500
	STATIC_COUNT_INCORPORATE	See previous algorithm.	6
	STATIC_OBJECT_REACTIVATION_MOTION_RATIO	See previous algorithm.	0.25
	MAX_KEEP_ALIVE	See previous algorithm.	32
Cluster Tracker	N_MAX_ITERATIONS	Maximum number of iterations for the EM algorithm.	30
	MIN_CENTROID_CHANGE	Minimum change on the centroids of the clusters to keep on iterating with the EM algorithm.	0.1
	MIXING_FACTOR	If this factor is increased, clusters will not merge easily.	0.0
	ALPHA_M	Minimum Mahanalobis distance for cluster merging (in pixels).	10
	T_r	Inverse of the velocity of change for the background (in number of frames).	1
	MIN_DENSITY_FOR_ELIMINATION	Minimum value of the density operator to eliminate a cluster by merging it with the background.	12

9. PERFORMANCE

Table 9.1 below lists the processing speeds for the algorithms, when tested on sequence S3-A320-CAM2. These measurements reflect the speed of the algorithms at the current time – further optimisations will be implemented in the near future.

Algorithm	**Max**		**Min**		**Average**	
Local Feature Tracker	33 ms	(30.0 fps)	417 ms	(2.4 fps)	91 ms	(10.9 fps)
Colour Tracker	28 ms	(35.7 fps)	263 ms	(3.8 fps)	83 ms	(12.0 fps)
Cluster Tracker						

10. SOME RESULTS

The following pages show a sample of the tracking results obtained from the 3 tracking algorithms described in this document when run against a part of the AVITRACK sequence S3-A320-CAM2, from frame 300 to 3350. This is only intended to give an approximate idea of how the trackers perform at present. A more formal evaluation, using ground-truth information and pre-defined evaluation criteria, is to be undertaken in the near future, as part of "Work Package 6.1 – Scene Tracking Evaluation" [1]. The figures on the following pages are depicted in a 'film-strip' format, showing the results for frames 300 to 3350, every 50. The range of frames 650 to 1100 have been skipped, since no action occurs within this period of time (after the aircraft has stopped moving and just before the servicing operation begins).

In the results of the Local Feature Tracker and the Colour Tracker, tracked objects are shown enclosed by their bounding boxes. A solid bounding box indicates a moving object, while a dashed bounding box indicates a stationary object that has been integrated into the background. Each object being tracked from one frame to the next is assigned a unique identity number. This number is displayed just beneath the lower edge of the bounding box – this might be too small to be visible in the images shown on the following pages. Alternatively, one can follow the colour of the bounding boxes – each object is assigned a colour (from a pool of 16 pre-defined colours) that remains fixed throughout its lifetime.

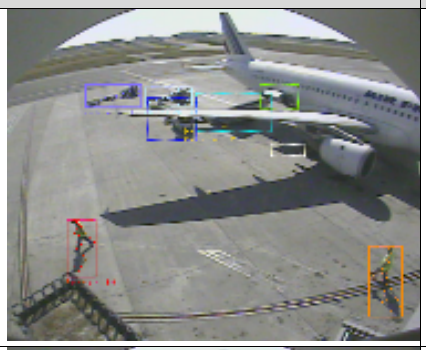
As mentioned in §2, the Cluster Tracker is currently running as a standalone application and has not yet been fully integrated into the AvitrackFrameTracker module. Hence, the format used for displaying the results is different than that of the other two. Tracked objects are indicated by an ellipsoid that indicates the limits of the top hat distribution used to model that object.

10.1. RESULTS – LOCAL FEATURE TRACKER

Some object fragmentation occurs for the aircraft (see result image 2) when the wing first comes into view (enters the detection region). The aircraft becomes stationary in image 4 and is integrated into the background – this allows new objects appearing in front of the aircraft to be detected and tracked (the person in image 5). The bounding box of the aircraft is not shown after image 10. In image 15, the shadow cast by the opened aircraft door (shown in green) is mistakenly detected as a new object (white object). In image 20, the person (shown in yellow) is eventually lost when it becomes partially occluded by the vehicle (cyan) – the object becomes too small in size and it becomes no longer possible to replace lost features. This person is reacquired as a new object in image 26 (shown in red). In images 35 to 36, the scene becomes cluttered and persons leaving the 2 vehicles (blue and cyan) are too close to each other to be tracked successfully. In image 36, note how the KLT algorithm manages to successfully find and track one or two features in the cone left on the apron (violet colour), even though this presents a very small area when viewed by camera 2. But the object is eventually lost in image 38, when the vehicle (large blue object) passes behind the cone – the shadow cast by the vehicle on the cone results in a large change in illumination of the cone's area and causes the loss of the features. In image 40, a stationary vehicle (blue object in image 36) starts moving at the same time that another vehicle is passing in front of it – the stationary object is successfully re-activated, but its ID is lost just after and it is detected as a new object (green vehicle in image 40).

LOCAL FEATURE TRACKER		Frames 300–1800
1 	2 	3 
4 	5 	6 
7 	8 	9 
10 	11 	12 
13 	14 	15 

LOCAL FEATURE TRACKER		Frames 1850–2550
16 	17 	18 
19 	20 	21 
22 	23 	24 
25 	26 	27 
28 	29 	30 

LOCAL FEATURE TRACKER		Frames 2600–3350
31 	32 	33 
34 	35 	36 
37 	38 	39 
40 	41 	42 
43 	44 	45 

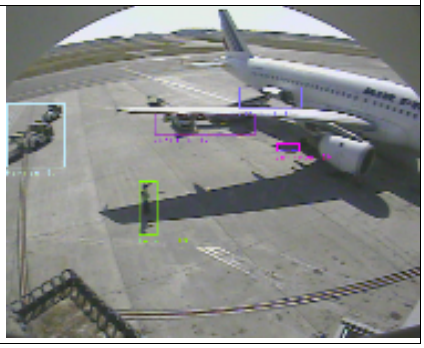
10.2. RESULTS – COLOUR TRACKER

The Colour Tracker suffers from the same fragmentation of the wing when the aircraft enters the camera's detection area (result image 2), similar to the Feature Tracker, as both are using the same motion detection algorithm as their input. But the behaviour of the Colour Tracker is worse under fragmentation, because normally the fragments have the same (or very similar) colour models and they tend to migrate towards the object's centre. The fragment in image 2 (shown in cyan), migrates towards the aircraft's centre and is absorbed by the larger fragment (yellow). The Colour Tracker is also susceptible to illumination changes – the use of the HSV colour space mitigates this to some extent. For example, the person shown in image 18 (red) is lost when it moves into the shadow area cast by the aircraft's wing, and is detected as a new object in image 19 (shown in green). Notice also how in image 20, part of the vehicle (violet) appears from behind the aircraft's wing, as the conveyor belt is elevated. This part is assigned to the object representing the aircraft door (blue), rather than to the vehicle object. Both object have little colour and their brightness histogram is very similar, so creating an error in the tracking results. Similarly, the vehicle in image 26 (shown in cyan) is lost when it approaches a vehicle of a similar colour model (violet) and it gets 'absorbed' by the other vehicle. This problem re-occurs in later frames – in image 38, the vehicle shown in blue is lost and is absorbed by the vehicle in violet; in image 40, and finally in image 42, where several vehicles of the same colour are identified as one large object by the Colour Tracker.

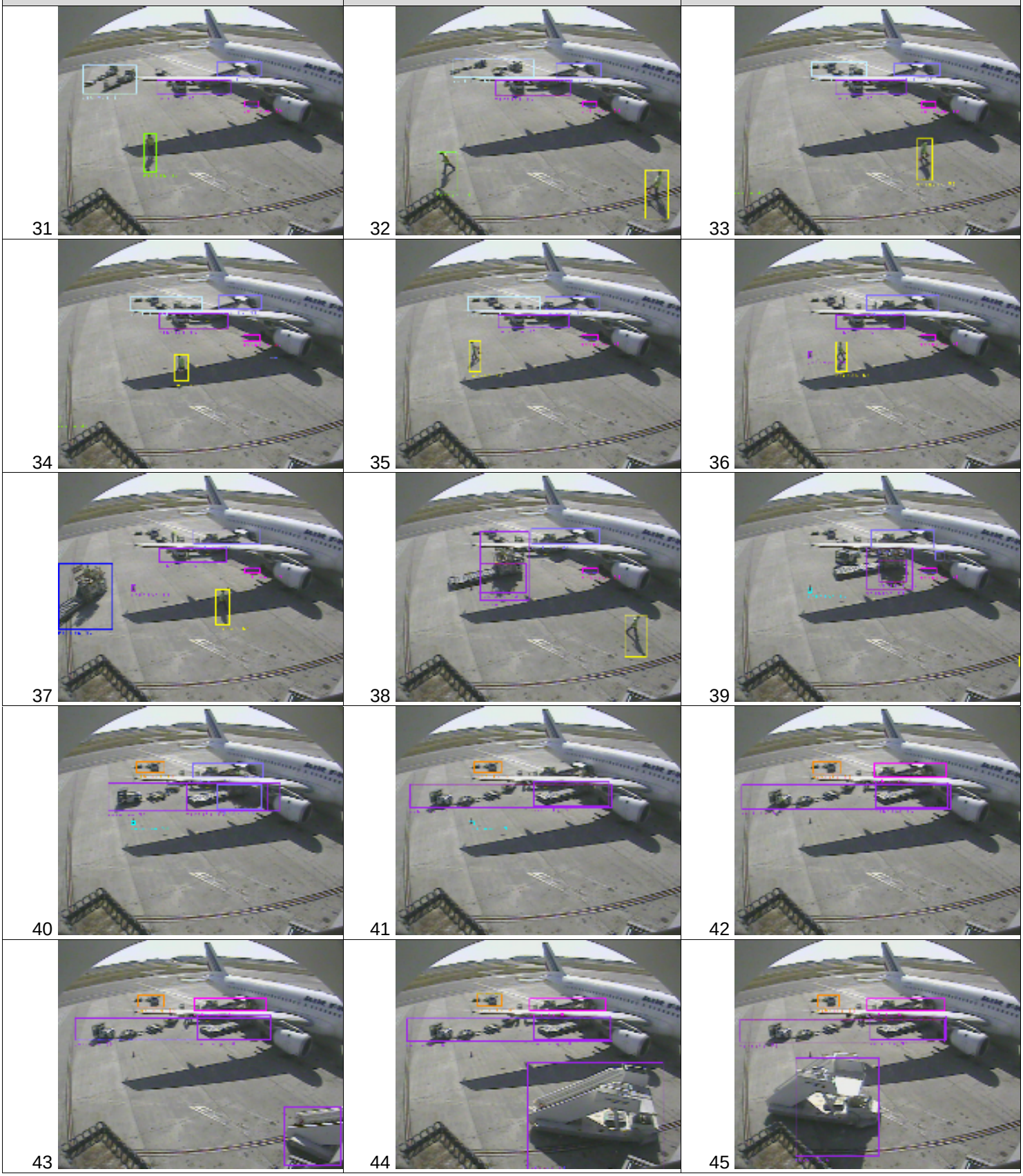
10.3. RESULTS – CLUSTER TRACKER

As stated in §7.4, the Cluster Tracking algorithm can be used as a low-level frame-to-frame tracker and other high level processes are required for maintaining the identity of objects when they become merged or occluded. Because of its probabilistic nature, this algorithm does not suffer from the fragmentation problem described for the previous two trackers. The aircraft is detected as one object in result images 1 to 3. It is then integrated into the background in image 4, so allowing new objects to be detected (the person in image 5). This tracker loses objects when they are affected by illumination changes – for example, the person shown in image14 is lost when it passes through the shadow cast by the aircraft's wing in image 15. Also, object identity is lost, when two or more objects become merged or partially occlude each other – see result images 17 to 18, for example. The cluster representing the merged objects eventually separates into two or more clusters when the separation between the objects reaches a certain threshold – for example, the vehicle and person in images 27 and 28.

COLOUR TRACKER		Frames 300–1800
1 	2 	3 
4 	5 	6 
7 	8 	9 
10 	11 	12 
13 	14 	15 

COLOUR TRACKER		Frames 1850–2550
16 	17 	18 
19 	20 	21 
22 	23 	24 
25 	26 	27 
28 	29 	30 

COLOUR TRACKER **Frames 2600–3350**



CLUSTER TRACKER		Frames 300–1800
1 	2 	3 
4 	5 	6 
7 	8 	9 
10 	11 	12 
13 	14 	15 

CLUSTER TRACKER		Frames 1850–2550
16 	17 	18 
19 	20 	21 
22 	23 	24 
25 	26 	27 
28 	29 	30 

CLUSTER TRACKER		Frames 2600–3050
31 	32 	33 
34 	35 	36 
37 	38 	39 
40 		

11. CONCLUSION

Three tracking algorithms have been implemented and tested using AVITRACK data sets. From initial evaluation, both the Local Feature Tracker and the Cluster Tracker give good tracking results.

The Local Feature Tracker can suffer from loss of object identity if the local features are lost during merged or occluded states and cannot be replenished fast enough. It also requires objects to be textured in order for good features to be selected.

The Cluster Tracker can lose object identity when clusters merge together, hence requiring a higher-level algorithm to preserve identities. The algorithm is also not able to deal with extreme illumination changes.

The Colour Tracker performs quite badly and generates a lot of tracking errors. The main reason for this is due to the lack of colour information in most of the objects being tracked in the AVITRACK sequences and also most of the objects have similar brightness distributions. A lot of issues were identified with this tracker and further work need to be done to improve its results.

12. FUTURE WORK

The following are some issues or approaches that will be addressed in future tracking work done for the AVITRACK project:

- Shadows are still a major issue, resulting in incorrect object positions being reported. Several attempts have been made to remove shadows. Some new techniques such as multi-view shadow elimination will be investigated in the future.
- The use of occlusion information by background elements and objects themselves can help to improve the tracking results. This is currently being studied.
- The use of feedback from data fusion may be investigated.
- The use of high-level scene context information may help to resolve certain tracking errors / failures.

REFERENCES

[1] "AVITRACK Technical Annex 1", AST3-CT-3002-502818.

[2] Shi J., and Tomasi C., "Good Features to Track". In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp 593-600, 1994.

[3] Birchfield S., "Derivation of Kanade-Lucas-Tomasi Tracking Equation". Unpubl. Available at http://www.vision.stanford.edu/~birch/klt.

[4] "Deliverable D3.1 - Motion Detection". AVITRACK, DL_AVI_2_006, Sept 2004.

[5] Thacker N.A., Prendergast D., and Rockett P.I., "B-Fitting: An Estimation Technique With Automatic Parameter Selection". In *Proc. British Machine Vision Conference (BMVC'96)*, Edinburgh, vol. 1, pp. 283-292, 1996.

[6] Aherne F.J., Thacker N.A., and Rockett, P.I., "The Bhattacharyya Metric as an Absolute Similarity Measure for Frequency Coded Data". *Kybernetika*, vol. 32, no. 4, pp. 1-7, 1997.

[7] Comaniciu D., Ramesh V., and Meer P., "Real-Time Tracking of Non-Rigid Objects Using Mean Shift". In *IEEE CVPR*, pp. 142-151, 2000.

[8] Bradski G., "Computer Vision Face Tracking For Use in a Perceptual User Interface". In *Intel Technology Journal*, Q2, 1998.

[9] Pece E.C., "Tracking by cluster analysis of image differences". In *PETS 2000*, pp. 295-303, 2000.

[10] Pece E.C., "Tracking of non-Gaussian clusters in the PETS2001 image sequences". In *PETS 2001*.

[11] Pece E.C., "From cluster tracking to people counting". In *PETS 2002*, pp. 9-17, 2002.

[12] Darrell T., and Pentland A.P., "Cooperative robust estimation using layers of support". In *IEEE Trans. P.A.M.I*. 17(5): 474-487, 1991.

[13] McLachlan G.J., and Krishnan T., "The EM Algorithm and Extensions". Wiley: New York 1997.

[14] "Software Delivery Report – AvitrackFrameTracker v0_3". AVITRACK, SD_AVI_2_009, Oct 2004.

[15] Sonka M., Hlavac V., and Boyle R., "*Image Processing, Analysis and Machine Vision*". Chapman & Hall, London, UK, 1993.

[16] Forsyth D., and Ponce J., "*Computer Vision: A Modern Approach*". Prentice Hall, NJ, USA, 2002.