

## Chapter 8

# Moving Object Detection

One of the main activities of visual perception is that of being aware of what is happening in the surrounding world. In most cases, complete knowledge of the environment is not required for a sensor to be able to achieve a basic level of awareness. And the problem can be reduced to finding out (detecting) those ‘objects’ that undergo changes, from the rest of the objects that do not appear to change – the latter can be considered to form part of a static ‘background scene’ that can be ignored. Within this context, objects that appear to change are labelled *foreground objects* and those that do not are labelled *background objects*, or collectively as *the background*.

Usually the main reason why objects appear to change is because they move, and so the process of detecting changes is also called *moving object detection*<sup>1</sup>. There are several different techniques available in computer vision for moving object detection.

The main ones are:

- Optical flow methods,
- frame difference methods, and
- background subtraction methods.

The detection method chosen for the OmniTracking program uses the *background subtraction* technique, since this, amongst others, is well suited for stationary cameras – which is the normal case for omnidirectional cameras, since with their large field-of-view they don’t have to move (rotate) to see the world.

---

<sup>1</sup> In some computer vision literature, a distinction is made between *motion detection* and *moving object detection*. Motion detection is defined as determining the changes due to motion (without doing any further organisational processing on the changes), while moving *object* detection involves determining the different objects that are moving [SONK93 §14]. For this thesis, the latter is being attempted, so explaining the name of the chapter.

This chapter first starts with an overview of the other detection methods. Then, the background subtraction technique is examined in more detail. This is followed by a description of how moving object detection was implemented for the OmniTracking program and ends with the results obtained when running the program on the datasets mentioned in §4.

## 8.1 Optical Flow

Optical flow methods use the apparent motion of the image brightness values in a sequence of images to estimate the relative motion of objects with respect to the camera. This apparent motion of pixels is used to construct a 2D vector field of velocities, called a *motion field*, which can be seen as the 2D projection (on the image plane) of the 3D velocity field of the objects in the world [TRUC98 §8.3].

The main advantage offered by the optical flow technique is that it works when both the objects and the camera are moving with respect to each other. But optical flow is a very computationally intensive and slow process. Another disadvantage is that if the camera is stationary, then objects are required to move – if they stop moving, objects will have a zero motion field and so are undetectable.

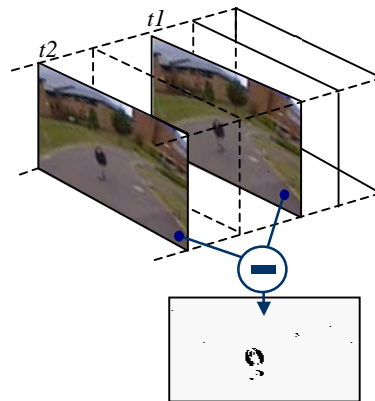
## 8.2 Frame Difference

These are the simplest methods and consist of comparing two adjacent frames from a video stream to find out those pixels that have changed. This is usually done by calculating the difference between the brightness values of the pixel in the two frames, based on the assumption that changes in brightness are due to real changes in the scene. Some threshold is then applied to the differences to eliminate small variations due to sensor acquisition noise. The result is a binary image where each pixel is labelled as either ‘moving pixel’ or ‘background’. This process can be expressed by the following equation:

$$\Delta_{t_1,t_2}(x,y) = \begin{cases} 1 & \text{if } |f_{t_1}(x,y) - f_{t_2}(x,y)| > \tau \\ 0 & \text{otherwise} \end{cases} \quad (8.1)$$

for some threshold  $\tau$  and using image frames at time  $t1$  and  $t2$ . The image frames can be either consecutive ( $t2 = t1+1$ ), or a certain number  $n$  of frames apart. Because of the use of adjacent frames, these methods are also called *temporal differencing* [JAIN95 §14.1].

The main advantage, apart from the simplicity, is that frame differencing is very adaptive to dynamic environments. This is because the gap between frames  $t1$  and  $t2$  is very short compared to any (usually slow) changes that might occur in the ‘static’ background. On the other hand, a disadvantage is that an object usually does not move much from frame  $t1$  to  $t2$  and only parts of the object (the outer parts) will appear as moving, as can be seen in Figure 8.1. This is called the *foreground aperture problem* [TOYA99].



**Figure 8.1:** Frame Differencing technique  
(The images are parts of frames 460 and 465 of the PETS2001 dataset).

### 8.3 Background Subtraction

Background subtraction methods can also be loosely classified into the category of *difference-based* methods (like frame differencing), but instead of using adjacent frames and finding the differences between the two, a *background model* is used. Each frame is ‘compared’ to this background model and the differences from the background are found. The main requirement for background subtraction methods is that the camera remains stationary; else the background model will become invalid<sup>2</sup>.

---

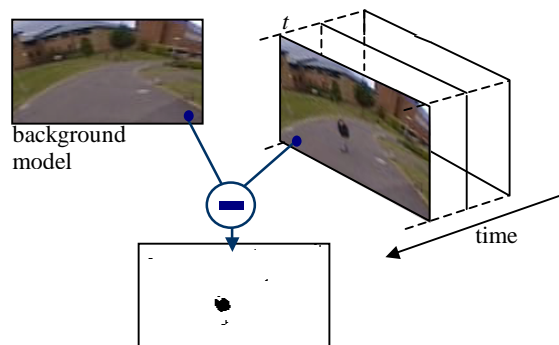
<sup>2</sup> Although there are some implementations where background subtraction methods have been adapted to be used for PTZ (pan-tilt-zoom) cameras. Such as the *Appearance Sphere* application, quoted in [YAMA02], which builds a background model for a PTZ camera. But these add complexity, require precise calibration and camera synchronisation and can be computationally expensive.

There are several different variants of the background subtraction technique, as described in the next section.

### 8.3.1 Simplest Form – Mean and Global Threshold Method

In its simplest form, the background model consists of an image of what the empty scene (that is, with no moving objects) is expected to look like. Each frame is then subtracted from the background image  $B$  and a threshold operation is applied (using the value  $\tau$  for all image pixels).

$$\Delta_t(x, y) = \begin{cases} 1 & \text{if } |B(x, y) - f_t(x, y)| > \tau \\ 0 & \text{otherwise} \end{cases} \quad (8.2)$$



**Figure 8.2:** Background Subtraction technique – the basic idea.

Compared to frame differencing, background subtraction methods have the advantage of detecting the whole object (see Figure 8.2). But their main disadvantage is that they are very sensitive to dynamic changes in the scene (that is, the background model can become out-of-date).

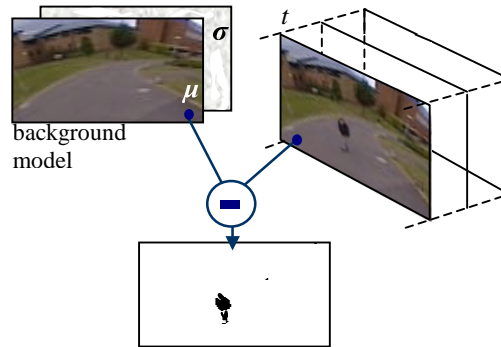
In this section it was mentioned that the background image consists of an image of the empty scene. But in most cases, it's not possible to get such an image beforehand. Especially for busy outdoor environments, like the one mentioned in §4.2 and being used for this project. Also, the ideal case is for the background model to be built automatically without human intervention. The simplest way of building a background image is to take the average of several frames:

$$B(x, y) = \mu_{(x,y)} = \frac{1}{n} \sum_{t=1}^n f_t(x, y) \quad (8.3)$$

The idea behind the use of averages has its origins in the 19<sup>th</sup> century, when it was discovered in photography that, by taking photos with very long exposures, moving objects are eliminated from the scene [FRIE97]. The time duration of  $n$  frames used to build the background image is usually called the *training phase* or *initialisation phase* of background subtraction.

### 8.3.2 Handling Noise – Normal Distribution Method

One of the limitations of the previous method is the use of the global threshold  $\tau$  for determining whether a pixel value is similar to the value expected from the background model or not. Assuming a constant scene, the variation observed for a background pixel over a period of time should only be caused by camera noise, which is usually modelled by a normal distribution with zero mean  $N_{noise}(0, \sigma^2)$  for that particular pixel [ELGA99].



**Figure 8.3:** Background Subtraction technique – normal distribution model.

Adding the camera noise to the pixel's mean value from the background model, the pixel's variation can then be modelled by  $N(\mu, \sigma^2)$ . So,  $\sigma$  together with  $\mu$  form the background model  $B$ , and  $\sigma$  is calculated during the training phase using the following equation ( $\mu$  is still calculated using (8.3)):

$$\sigma_{(x,y)} = \sqrt{\frac{1}{n} \sum_{t=1}^n f_t(x,y)^2 - (\mu_{(x,y)})^2} \quad (8.4)$$

$$B(x,y) = (\mu_{(x,y)}, \sigma_{(x,y)}) \quad (8.5)$$

A pixel is then labelled as foreground if  $|f(x,y) - \mu_{(x,y)}| > k\sigma_{(x,y)}$  for some  $k$ .

### 8.3.3 Handling Background Motion – Mixture of Gaussians Method

The assumption so far has been that the background scene does not change. But it can happen that background elements show some movement, for example, trees moving in the wind in outdoor environments. This ‘uninteresting’ motion should still be classified as belonging to the background. The brightness variations exhibited by these pixels is multimodal and is best handled by using a *mixture of Gaussian* (normal) *distributions* [STAU99]. The mixture model  $M$  is defined as:

$$M = \sum_{i=1}^k \pi_i N_i(\mu, \sigma) \quad (8.6)$$

where  $k$  is the number of component distributions used (normally ranging from 3 to 5),  $N_i$  is the  $i^{\text{th}}$  individual normal component distribution and  $\pi_i$  is its (mixing) weight, with  $\sum_{i=1}^k \pi_i = 1$ .

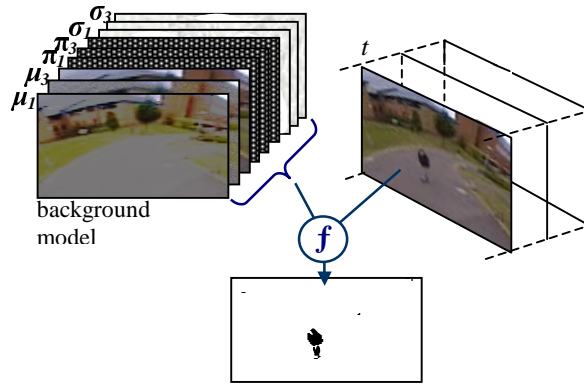
The basic idea of the mixture model is that the different distributions will each model a particular background element that the pixel happens to result from (in the case of a moving tree, a pixel can at one time be the light coming from the tree and at another moment it will be the sky, for example). A separate Gaussian mixture model is used for each pixel. The background model is then defined as:

$$B(x, y) = (\pi_1, \pi_2, \dots, \pi_k, \mu_1, \mu_2, \dots, \mu_k, \sigma_1, \sigma_2, \dots, \sigma_k) \quad (8.7)$$

In the above equation, the individual parameters are defined for each pixel, that is,  $\pi_i$  should read as  $\pi_{I(x,y)}$ , but the  $(x,y)$  subscript has been omitted to avoid clutter.

Fitting the mixture model to the pixel data, during the training phase of background subtraction, is usually done by maximising some likelihood function of the mixture model  $M$ . Methods such as the *Expectation-Maximisation (EM)* algorithm or the *K-means* algorithm are usually used for this process [FRIE97; STAU99].

A Gaussian mixture background subtraction is very robust to background scene motion. However, fitting the mixture model is computationally expensive, and maintaining a model for each pixel requires large amounts of memory.



**Figure 8.4:** Background Subtraction technique – Gaussian Mixture model.

### 8.3.4 Post-Processing

The result of background subtraction is a binary image in which each pixel is labelled as foreground or background. Normally, some sequence of post-processing operations is applied to the result to make the background subtraction algorithm more robust. Examples include: morphological operations, cleaning operators (to fill gaps in the result and remove noise), and shadow removal.

## 8.4 Background Adaptation

One of the disadvantages of background subtraction is that the scene can change over (normally long) periods of time and so the background model may get out of date (for example, illumination changes during the day). To counteract this, the background model can be modified at run-time to adapt to any such changes – this is called *background adaptation*, or sometimes *background maintenance* [TOYA99].

A common technique for performing background maintenance is to employ a *moving-window average*, where the background is re-calculated at time  $t$  from the previous  $n$  frames (from  $t-1$  to  $t-1-n$ ). This requires the previous  $n$  samples for each pixel to be stored.

A better alternative is to use the *temporal integration* approach, where the background model is updated using the following general equation:

$$B_t(x, y) = (1 - \alpha)B_{t-1}(x, y) + \alpha f_t(x, y) \quad (8.8)$$

where  $\alpha$  is called the *integration parameter* (or *blending parameter*). More specifically, for the single normal distribution method, (8.8) is implemented as follows:

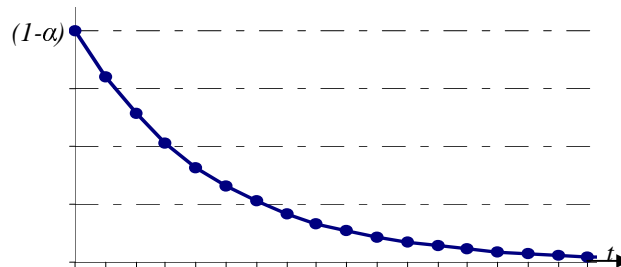
$$\begin{aligned}\mu_{t(x,y)} &= (1-\alpha)\mu_{t-1(x,y)} + \alpha f_t(x,y) \\ \sigma_{t(x,y)} &= (1-\alpha)\sigma_{t-1(x,y)} + \alpha\sqrt{f_t(x,y)^2 - \mu_{t(x,y)}^2} \\ B_t(x,y) &= (\mu_{t(x,y)}, \sigma_{t(x,y)})\end{aligned}\tag{8.9}$$

In the case of the Gaussian mixture model, the same blending function can be applied to the distribution component that best matches (supports) the pixel. Alternatively, the mixture model is updated using a completely different procedure, such as an incremental version of the EM algorithm [FRIE97].

The parameter  $\alpha$  ranges from 0 to 1 and determines how responsive is the background model to change. A large value means that a higher contribution from the current pixel value is added to the background model  $B_t$  and the contribution of the older values ( $B_{t-1}$ ) is reduced. At any time  $t$ , the reduction of the contributions of background  $B$  follows the sequence:

$$\begin{aligned}B_{t+1} &= (1-\alpha)B_t + \alpha f_{t+1} \\ B_{t+2} &= (1-\alpha)^2 B_t + (1-\alpha)\alpha f_{t+1} + \alpha f_{t+2} \\ B_{t+3} &= (1-\alpha)^3 B_t + (1-\alpha)^2 \alpha f_{t+1} + (1-\alpha)\alpha f_{t+2} + \alpha f_{t+3} \\ &\dots etc.\end{aligned}$$

It is clear that this sequence forms a weighted sum of previous pixel values with the weight  $(1-\alpha)^t$  being an exponential function. For this reason, the background update process is called an *exponential forgetting process* with  $1/\alpha$  being the time constant of the process [FRIE97].



**Figure 8.5:** Exponential forgetting for background update.



### 8.4.1 Types of Background Update

Normally, update of the background model is only performed for those pixels which at any time  $t$  are labelled as background by the detection process. This implies that background update is done as the last step of the background subtraction process, after the detection and labelling phase is finished. This *selective* type of background update helps to increase the accuracy of detection, since foreground objects will not corrupt the background model.

But problems may occur if the decision of whether a pixel is background or not is incorrect. If the background of a pixel changes and it is incorrectly labelled as foreground, this may lead to its background model never to be updated and hence causes persistently incorrect decisions to be taken by the detection algorithm. This is a *deadlock* situation [ELGA99].

One way of solving the deadlock problem is to do a *blind update* – updating all the pixels regardless of whether they are foreground or not. The disadvantage of this is the increase in detection errors.

A compromise between selective and blind update, is to use two blending parameters, one for background pixels and the other for foreground. Generally, the parameter for foreground pixels is set to a slower rate of integration [BOUL99].

$$B_t(x, y) = \begin{cases} (1 - \alpha_{bkg})B_{t-1}(x, y) + \alpha_{bkg} f_t(x, y) & \text{if } f_t(x, y) \text{ is 'background'} \\ (1 - \alpha_{frg})B_{t-1}(x, y) + \alpha_{frg} f_t(x, y) & \text{if } f_t(x, y) \text{ is 'foreground'} \end{cases} \quad (8.10)$$

## 8.5 Review of Applications using Background Subtraction

This section gives a brief review of some applications that use the background subtraction technique for moving object detection, with special attention to those using omnidirectional cameras. More applications can be found in [MCIV00; TOYA99].

- [BOUL99] use background subtraction for an omnidirectional camera-based surveillance application named *LOTS*. Background subtraction is performed on the raw omnidirectional image. Two separate background models are used,

in cascade fashion. The primary background consists of a single normal distribution,  $B_t^{1st} = (\mu_t, \sigma_t)$ , and is updated using separate foreground and background parameters  $\alpha$  (see §8.4.1). The second background is only applied to pixels labelled as foreground by the first background model, and it is updated using  $B_{t+1}^{2nd} = \alpha_{f_{fg}} f_t(x, y) + (1 - \alpha) B_t^{2nd}$ , if  $|B_t^{2nd} - f_t(x, y)|$  is below some threshold, or  $B_{t+1}^{2nd} = f_t(x, y)$  otherwise.

- Colour is used for background subtraction in the application described in [CUTL98]. Separate averages are kept for the colour components (RGB colour space) with the background model consisting of:  $B_t = (\mu_t^R, \mu_t^G, \mu_t^B)$ . A pixel is labelled as foreground if it satisfies the condition:  $\sum_{c \in (R, G, B)} |\mu_t^c(x, y) - f_t(x, y)| > k\sigma$ , where  $\sigma$  is estimated beforehand.
- The  $W^t$  application of [HARI98], uses a background model consisting of:  $B_t = (f_{\min}, f_{\max}, f_{\delta})$ , where the values are the maximum, minimum, and maximum difference found in the set of pixel values during the training phase. A pixel is labelled as foreground if either of the conditions  $|f_t(x, y) - f_{\max}(x, y)| > f_{\delta}(x, y)$  or  $|f_t(x, y) - f_{\min}(x, y)| > f_{\delta}(x, y)$  is true. The background is updated using the selective update approach (§8.4.1).
- A Gaussian mixture model is used by [STAU99] for modelling the background, with 3 to 5 component distributions and using an online K-means algorithm for fitting the mixture model to the training data. In this application, a variation of the blind background update type is used, and background update is done before the foreground is detected. If the pixel value matches a component distribution (is within  $2.5\sigma$ ), then that component's parameters  $\mu_i$ ,  $\sigma_i$ , and  $\pi_i$  are updated using temporal integration (8.8). The other components' parameters are left unchanged. If no component matches the pixel's value, the least probable component is deleted and replaced with a new component which has  $\mu_{i,t(x,y)} = f_t(x, y)$ ,  $\sigma_{i,t(x,y)}$  set to a large value, and low  $\pi_{i,t(x,y)}$ . For moving object detection, the component distributions in the mixture model are sorted in order of their probability of occurrence, and the first  $L$  of these are chosen to represent the background. If a pixel does not fall within  $2.5\sigma$  of any of these  $L$  distributions, it is labelled as foreground.

- [HUAN02] use background subtraction for an omnidirectional-based application called *NOVA*, used for tracking people in a room. Due to the simplified nature of the indoor environment, background subtraction is performed on the panoramic image obtained from unwrapping the raw omnidirectional image (unlike the *LOTS* application mentioned further above). The single normal distribution method is used, with the addition that the background model is augmented with the brightness distortion  $\alpha$  and chrominance distortion  $CD$  values (background model  $B_t = (\mu_t, \sigma_t, \alpha_t, CD_t)$ ). The last two are used for doing shadow detection as a post-processing step to background subtraction. The differences between the image and the background panoramic image are collapsed to a *1-D profile* – by accumulating a histogram of pixel differences in each column of the panorama. A global threshold is then applied to each column to see if enough pixels within that column were labelled as foreground.
- [ZHU99] also use background subtraction for an indoor environment-based application captured with an omnidirectional camera. Similarly to the previous application, background subtraction is performed on the dewarped panoramic image. A combination of background image subtraction and frame differencing is used. The results from background subtraction  $s(x,y)$  and frame differencing  $d(x,y)$  are combined at a region level rather than at a pixel level, with the pixel  $(x,y)$  of region  $R$  being accepted as foreground if the condition  $s(x, y) \geq \min_{\forall (x_i, y_i) \in R} [d(x_i, y_i)]$  is satisfied.
- [YAMA02] uses background subtraction for omnidirectional images, in which the background is modelled with  $B_t = (\mu_t, \sigma \sin(2\pi\omega t) + k \times \text{noise})$ , where the term  $\sigma \sin(2\pi\omega t)$  models the flicker of fluorescent light and CRT screens, and the  $k \times \text{noise}$  factor represents the camera sensor noise. If the condition:  $\mu_t - \sigma \sin(2\pi\omega t) - k \times \text{noise} \leq f_i(x,y) \leq \mu_t + \sigma \sin(2\pi\omega t) + k \times \text{noise}$  is satisfied, then the pixel is labelled as background. Temporal integration is used to update the background model.
- [JABR00] implement a background subtraction method that combines colour and edge information. The background is modelled by:  $B_t = (\mu^c, \sigma^c, H^c, V^c)$  for  $c \in (R,G,B)$ , where  $H$  is the horizontal edge map and  $V$  the vertical one

obtained from the Sobel edge detector. The edge maps of the frame at time  $t$  are subtracted from the edge maps of the background model and the resulting edges are classified as: occluding edge, occluded edge or background edge. The edge information is combined with the thresholded colour differences to get a final measure of change. The idea of using edge information is that this usually leads to a more accurate extraction of the boundaries of objects.

## 8.6 Types and Sources of Detection Errors

Moving object detection is usually the first processing step performed by computer vision applications. Any detection errors that occur during this step will propagate to later processing phases and affect their results in a negative way. Therefore it is important that the algorithms used for moving object detection are as accurate as possible and that detection errors are eliminated or minimised.

The accuracy of these algorithms is usually measured by their *detection rate* – that is, how many moving objects in the world they are able to find. And also by the number of detection errors they generate – normally expressed in terms of the number of false positives and false negatives. A *false positive* occurs when an algorithm says that a certain set of pixels belong to a moving object, when in reality there is no object at the indicated position. A *false negative* occurs when there is a moving object in the scene, but the algorithm misses the object and labels its pixels as background [TRUC98 §A.1].

In the case of the background subtraction technique, several conditions can potentially give rise to detection errors, and these are briefly mentioned below. The chosen background subtraction algorithm should be implemented with these conditions in mind, with the aim of trying to achieve a certain degree of robustness against them. But, because of their low-level nature (that is, independent pixel-based processing), background subtraction techniques may not be able to solve all of the problems, and the solution to some of them (if at all possible), may require domain knowledge and higher-level processing [TOYA99].

Camera-related problems:

- **Noise objects:** Camera noise usually manifests itself as random fluctuations in the intensity values of pixels and these may cause the background subtraction

algorithm to mistakenly label them as foreground pixels – generating false positives. The inclusion of a camera noise model in the background (see §8.3.2), helps to reduce this problem. Applying a *size filter* to the background subtraction result is also effective, because noise objects tend to be small [COLL00].

- **Incomplete extraction:** Camera noise can also cause an object not to be fully differentiated from the background, leaving gaps especially in the boundary of the object. In the worst case, an object may become fragmented. Applying morphological operations on the background result can alleviate this problem.
- **Registration errors:** The basic assumption for background subtraction techniques is that the camera is stationary. But in some cases the camera may suffer from vibrations, for example, when objects come very close to the camera or due to wind in outdoor scenes. Camera motion can cause the background to get misaligned with respect to an image and the wrong background values to be used. One way of minimising this problem is to implement some form of automatic registration that brings the background and image back in alignment with each other.

Lighting problems:

- **Gradual light changes:** The appearance of outdoor background scenes is affected by the slow change of illumination caused by daylight<sup>3</sup>. This may also affect indoor scenes, where windows are present. This is normally solved through background adaptation (§8.4).
- **Sudden light changes:** Examples include turning the lights on and off (in indoor scenes) and the sun moving behind clouds (outdoor scenes). In this case, background adaptation might not help, as its rate of update is much slower. Sudden light changes affect the whole scene and cause many false positives to appear – one can detect them by checking for any abnormal increase in the number of foreground objects, and then change or reconstruct the background [XU01].

---

<sup>3</sup> See [FORS02 §4.1.3] for a graph showing variations of daylight measured at different times of the day and under different atmospheric conditions.

Shadow-related problems:

- **Object distortion:** Shadows can give rise to a number of different detection errors. If shadows are labelled as part of the object, then the object's shape will appear distorted, and this distortion can affect later processing phases that use geometrical properties for object classification, location estimation, etc. [CUCC01]. The effect of shadows can be suppressed by using shadow detection algorithms or by using information that is invariant to shadows.
- **Object loss:** Shadows can also give rise to object loss when an object's shadow is cast upon another object in the scene.
- **Object under-segmentation:** When two objects are close to each other, their shadow might cause them to appear to be connected. And the two objects will be merged together by the algorithm causing under-segmentation – that is, the algorithm reports less objects than in fact there are in the scene [PRAT01]. Shadow suppression reduces this problem.

Object-related problems:

- **Camouflage:** The objects themselves, or their behaviour, may be the source of errors for background subtraction. Camouflage occurs when parts of an object's colour match the background, so making them indistinguishable from the background. This results in object fragmentation. Using clustering or morphological operations may help to reduce this problem [COLL00].
- **Slow-moving objects:** If objects move at a very slow rate, comparable to the background adaptation rate, there is the possibility for them to be partially 'absorbed' into the background. This can cause the objects to be lost or may give rise to false positives when they eventually move away from the current position. Selective background update solves this problem (§8.4.1). Another way is through the use of multiple backgrounds [BOUL99].
- **Foreground aperture problem:** When a uniformly-coloured object moves, it can happen that the interior pixels of the object are not detected [TOYA99]. This problem is more common for frame differencing methods, but it may also occur during background subtraction if an object has been stationary for some time. Motion will only be perceived at the edges of the object and not in the centre. This causes the object to be fragmented.

Background-related problems:

- **Illegitimate motion:** Motion by elements of the background scene (example, trees swaying in the wind) causes false positives to be detected. Modelling the background with multiple distributions, as described in §8.3.3, helps to reduce this problem.
- **Moved background objects:** Another source for false positives is when objects change elements of the scene or remove and introduce new elements (for example, a person enters a room and moves a chair from one place to another) [STAU99]. Background adaptation can help to reduce this problem, as the affected elements will eventually be absorbed into the background.
- **Ghosts:** When an object that has been stationary for a long time (and hence absorbed into the background) moves away, two objects are detected – the moving object and another false positive where the object was originally located. This ‘negative object’ is called a *ghost* [COLL00]. Use of background update (except for the selective update type – see §8.4.1) will eventually remove the ghost object. It is also possible to use some post-processing logic to patch the ‘hole’ left in the background by the object. This problem might give rise to deadlock situations if selective background update is used [CUCC01].
- **Deadlock problem:** As mentioned in the previous paragraph, if selective background update is used (that is, the background is not update for pixels labelled as foreground), false positives tend to persist indefinitely as the background of these pixels will not be updated. This problem can be reduced by using the other update types mentioned in §8.4.1.
- **Quiet training phase:** The background model is normally built automatically from the first few image frames captured by the camera. The ideal condition is for the scene to be empty of any moving objects to get an accurate background representation, but in real-life this is not always possible. The presence of moving objects during the training phase of the algorithm will cause some corruption of the background, which in turn might give rise to detection errors. This problem is also called the *bootstrapping problem* by [TOYA99].

## 8.7 Implementation

The OmniTracking application uses background subtraction for detecting moving objects. Background subtraction is performed directly on the omnidirectional image. It was decided to model each pixel with a single normal distribution and to perform background model adaptation using two different rates, for the foreground and background pixels respectively. Background subtraction is performed using colour information, as this is a better discriminant than just greyscale. To solve the over-segmentation issue, shadow detection and removal was implemented by working in the HSV colour space. Finally, thresholding-with-hysteresis is performed on the background subtraction result to get the classification of pixels into foreground and background.

### 8.7.1 *Choosing a Background Model*

The first decision that had to be taken was which background model to use. The three main types of background models were described in §8.3. The Gaussian mixture model is the most accurate of the three because it models each pixel with multiple components and it appears that it should be the method of choice. But the problem with this background model is its computational cost and large memory requirements.

#### 8.7.1.1 *A Test using Mixture Models*

To check whether it was worthwhile using this method or not, a test program was written that modelled each pixel by a mixture of 3 Gaussians. The EM algorithm was used to fit the mixture model to an initial set of 100 frames from the PETS2001 dataset, selected because this dataset shows evidence of motion in the background – swaying trees, moving clouds, slight camera jitter, etc.

Each pixel value  $f(x,y)$  was defined by the colour vector  $(H,S)$ , where  $H$  and  $S$  are its hue and saturation values (as defined in the HSV colour space). Therefore, each 2D Gaussian component  $N_i$  takes the following parameters:  $(\mu_H, \mu_S), (\sigma_H^2, \sigma_S^2), \sigma_{HS})$ , where  $\sigma_H^2, \sigma_S^2$  are the individual variances and  $\sigma_{HS}$  is the covariance.  $N_i$  is defined as:



$$N_i = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2}[(H,S)-(\mu_H,\mu_S)]^T \Sigma^{-1} [(H,S)-(\mu_H,\mu_S)]} \quad \text{where } \Sigma = \begin{bmatrix} \sigma_H^2 & \sigma_{HS} \\ \sigma_{HS} & \sigma_S^2 \end{bmatrix} \quad (8.11)$$

The component mixture  $M$  of each pixel then consists of  $(N_1, N_2, N_3, \pi_1, \pi_2, \pi_3)$ . This results in a memory requirement of 72 bytes per pixel<sup>4</sup>. Since the size of each frame of the PETS2001 dataset is of 768×576 pixels, the total memory requirement for the model is of ~31Mb – this not counting additional values such as thresholds and pre-calculated values that can be used to make processing run faster.

In addition, running the EM algorithm for each mixture model of every pixel is quite expensive, even when a fast version of the EM algorithm is used, for example, the *Incremental EM algorithm* [NG03]. From some initial tests, the standalone background subtraction algorithm with a mixture model was running at a rate of between 2 and 3 frames per second<sup>5</sup>, and the initial model from the first 100 frames took about 4 minutes to be built, using the standard EM algorithm.

One way of reducing the runtime costs is to reduce the size of the image. But this is not feasible for omnidirectional images, with their already low-resolution. Other improvements include assuming zero covariance (independence) between the colour components ( $\sigma_{HS}=0$ ) and using look-up-tables (at the cost of more memory) [STAU99].

At this point it is useful to see how many of these Gaussian components are actually used. Figure 8.6 shows the variations of the  $(H,S)$  values for six pixels and the number of Gaussian components that were required to explain these variations. For example, point 1 is a pixel that is sometimes a tree (blue component) and sometimes the sky (red component) with weights 0.57 and 0.43 respectively. Figure 8.6(b) shows that for most of the pixels, one component is enough. As expected, the pixels needing all 3 components are the trees and some parts of the cars with their metallic surfaces. Most of the road and the grass do not change much. There are some variations in the road surface, which are arranged in linear-like structures – these most probably are due to JPEG noise (pixels that happen to fall on the ‘edges’ of the 8×8 compression blocks).

---

<sup>4</sup> The values are stored as floating-point variables, which in C++ are represented by 4 bytes. Therefore the total consists of 20 bytes for each of the 3 components and the rest for the mixture weights  $\pi_i$ .

<sup>5</sup> This without background update.

Most of the background movement seen in this dataset is quite small (a few pixels). This is because the background elements in the outdoor scene are at a certain distance from the camera, and also due to the low resolution of the omnidirectional camera. In general, one may assume that this also holds for most environments in which omnidirectional-based surveillance applications are used. The background movements manifest themselves mostly as thin linear-like groups of pixels (mostly due to pixels that are on the edges between two background elements) – these can be eliminated using techniques such as size filtering or thresholding.

Using only hue and saturation for modelling pixels misses objects that happen to have the same colour as the background, but different brightness. To detect these objects, the brightness  $V$  needs to be added to  $H$  and  $S$ , meaning more memory would be required for the mixture model.

For these reasons it was decided not to use the Gaussian mixture model for background subtraction, but the single Gaussian distribution model (see §8.3.2). This method should give reasonable results while at the same time being fast. But given more time, and if the mixture model program is optimised enough<sup>6</sup>, it could be used instead.

### 8.7.1.2 *The Chosen Model*

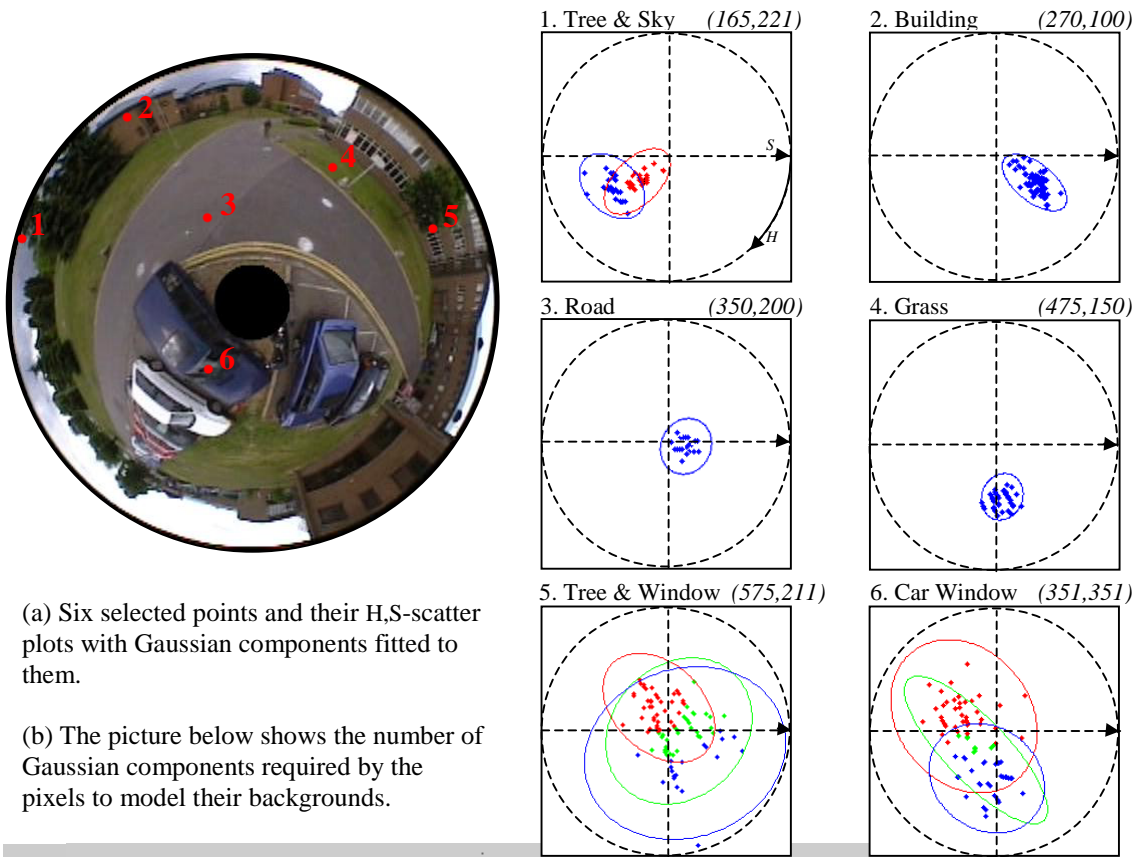
The chosen background subtraction method for the OmniTracking application, models each pixel by a normal distribution as follows:

$$B(x, y) = N_{(x,y)}(\mu, \sigma) = N_{(x,y)}\left((\mu_H, \mu_S, \mu_V), (\sigma_H^2, \sigma_S^2, \sigma_V^2)\right) \quad (8.12)$$

And the  $H$ ,  $S$ ,  $V$  colour components are considered to be independent, that is, correlation between them is assumed to be zero. Colour is used as this gives a better detection rate. The choice of working in the HSV colour space is due to the presence of shadows, as will be mentioned in the next section.

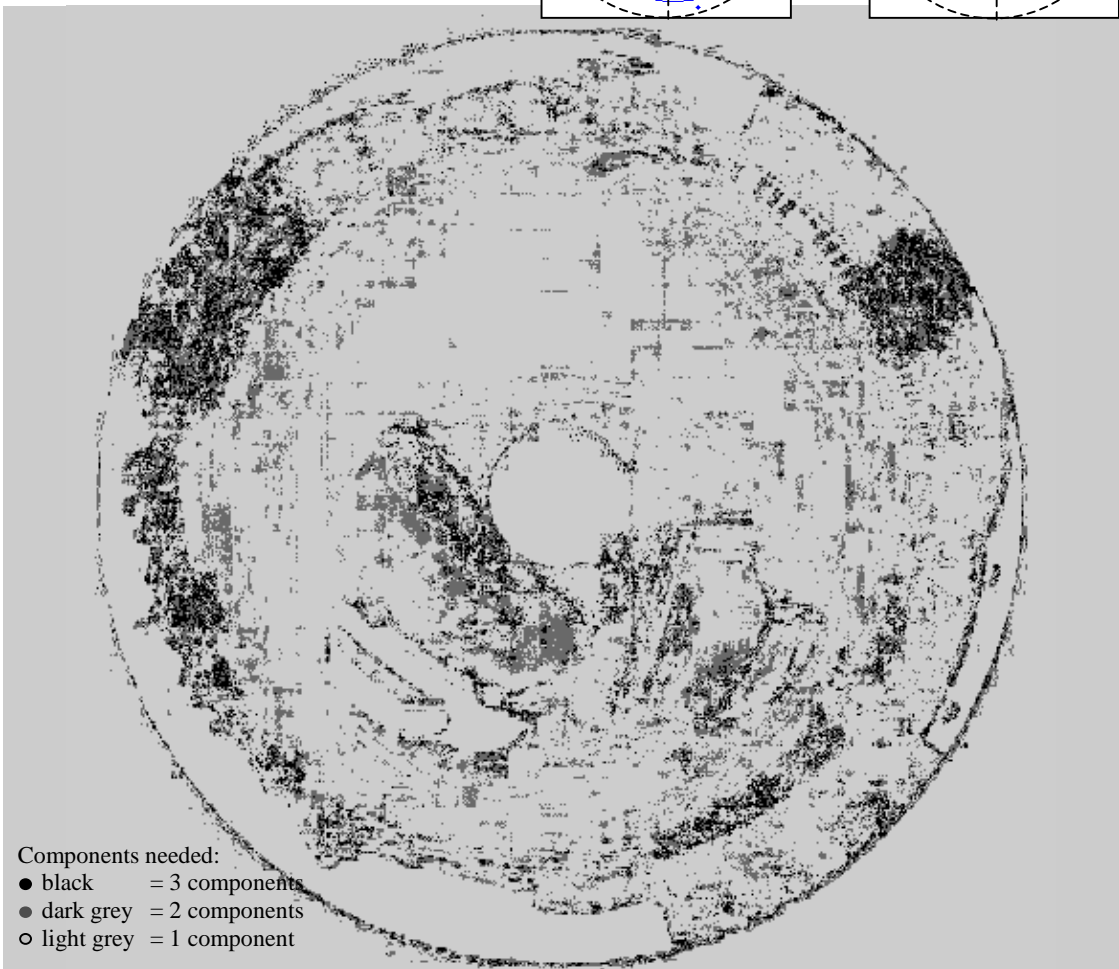
---

<sup>6</sup> For example, memory-wise there's no need to allocate storage for all 3 components, as few pixels will require all 3 of them (as can be seen in Figure 8.6(b)). But then, some form of pointer-based memory structure will be required, which might add some processing and memory overhead.



(a) Six selected points and their H,S-scatter plots with Gaussian components fitted to them.

(b) The picture below shows the number of Gaussian components required by the pixels to model their backgrounds.

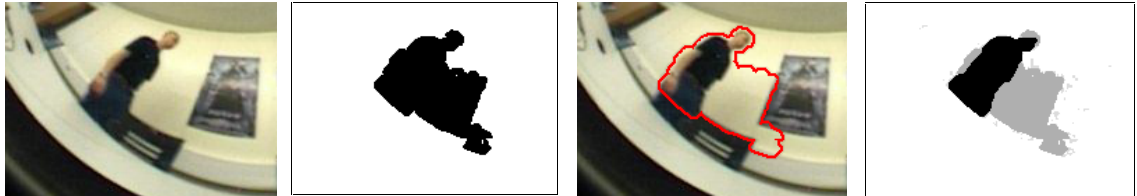


**Figure 8.6:** Mixture Models for pixels of the PETS2001 dataset.

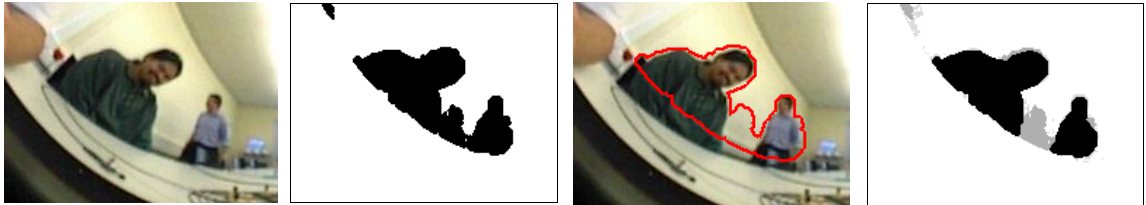
### 8.7.2 Robustness to Shadow

Shadow is a major source of problems for object detection, as mentioned in §8.6. This is especially true within the confines of indoor environments – for example, objects tend to be quite close to each other and shadow might cause the objects to appear to merge. Figure 8.7 shows some examples taken from the PETS-ICVS dataset.

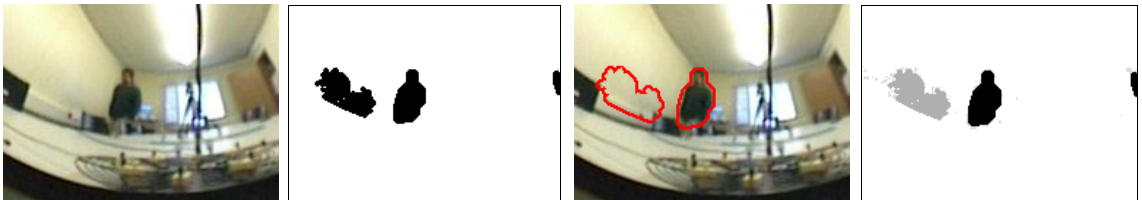
(a) Object Distortion due to shadow (part of frame 10310).



(b) Under-Segmentation (merging) due to shadow (frame 11205).



(c) False positive (shadow object) (frame 10820).



**Figure 8.7:** Detection Errors caused by Shadow (columns 1 to 3). Last column is with shadow detection enabled (grey). (Frames from PETS-ICVS dataset).

For this reason, it was decided that shadow detection should form an essential part of the background subtraction process.

But what is shadow? First, it can be seen that when an object is in shadow, its appearance (colour) does not depend on the object casting the shadow (blocking the light) [FRIE97]. Furthermore, when the object is in shadow, it is not completely dark, but is lit by the *ambient light* generated by the surrounding scene – that is, light reflected by the other objects in the scene. Normally, this ambient light is nearly grey (colourless), as it is the average of the light reflected by many of the surrounding objects [XU01].

Combining these two observations together: when an element in the background scene is in the shadow cast by a moving object, its colour will not change, but its brightness will be reduced. Therefore, if background subtraction is to be robust to shadows, it must be able to ignore brightness changes when the colour of the background remains the same – this is called *illumination invariance* [XU01]. To do this, the algorithm must be able to separate the *illumination component* of the pixel’s light from the *chromatic component*. This idea is also related to an ability of the human vision called *colour constancy*, where humans are able to assign the same colour to an object under different levels of illumination [HORP99].

There are many different ways of expressing the pixel’s brightness that achieve illumination invariance. For example:

- [ELGA02; XU01] use the values  $(r,g)$  where  $r = R/(R+G+B)$ ,  $g = G/(R+G+B)$  in the standard RGB colour space. These are invariant to brightness because of the normalisation factor in the denominator.
- [XU01] mentions also the log chromaticity differences  $\ln(R/G)$  and  $\ln(B/G)$ .
- For the YUV colour space, the values  $U/Y$ ,  $V/Y$  can be used.
- The hue  $H$  and saturation  $S$  values of the HSI or HSV colour spaces.
- [HORP00] consider colour values as vectors in the RGB colour space and derive the brightness differences  $\alpha$  and chromaticity distortions  $CD$  in terms of vector geometry.

More methods can be found in [PRAT01]<sup>7</sup>.

### 8.7.2.1 *HSV Colour Space and Shadow Detection*

For this implementation, it was decided to use the *HSV (hue-saturation-value) colour space*. This space separates the chromaticity values, expressed in terms of hue and saturation, from the illumination value  $V$ . The values  $H$  and  $S$  are invariant to illumination changes and can be used for suppressing shadows.

A pixel  $f(x,y) = (H,S,V)$  is considered to be a shadow pixel, if it satisfies the following condition, when compared to its background value  $(H_B, S_B, V_B)$ :

---

<sup>7</sup> In the case of background subtraction and many other computer vision applications, shadow is considered to be a nuisance and a source of problems. But shadow can be useful in its own right, for example, in *shape from shading* applications [SONK93 §9.3.2].

$$\begin{aligned}
H &\approx H_B, \quad S \approx S_B \\
\alpha V_B &< V < V_B \quad \text{where } 0 < \alpha \leq 1
\end{aligned}
\tag{8.13}$$

The last column in Figure 8.7 shows shadow detection applied to the PETS-ICVS dataset. The global parameter  $\alpha$  in (8.13) defines the maximum amount of shadow darkening that can be expected to be present in the scene, that is shadow strength is expected to be  $(1-\alpha)$ . This usually depends on the strength of the illumination source(s) and the amount of ambient light present. The reason for including this lower limit on brightness reduction is to avoid missing objects that happen to be coloured like the background but are darker. For example, the person in Figure 8.7(a) would be classified incorrectly as shadow (because the black top appears to be a darker version of the white-colour of the wall).

By default, the value for parameter  $\alpha$  is set to 0.7 (typical of indoor scenes), and is user-configurable<sup>8</sup>. Setting  $\alpha$  to 1 in (8.13) disables shadow detection. For now, this value remains fixed while the program is running. This is a limitation and ideally, this value should be adapted depending on the lighting conditions in the scene (shadows are stronger at noon).

### 8.7.2.2 Low Chromaticity Conditions

The HSV colour space is cylindrical in nature and the hue component (the main value determining the chromaticity) is derived from the RGB space using [JAIN95 §10.4]:

$$\cos H = \frac{2R - G - B}{2\sqrt{(R - G)^2 + (R - B)(G - B)}}
\tag{8.14}$$

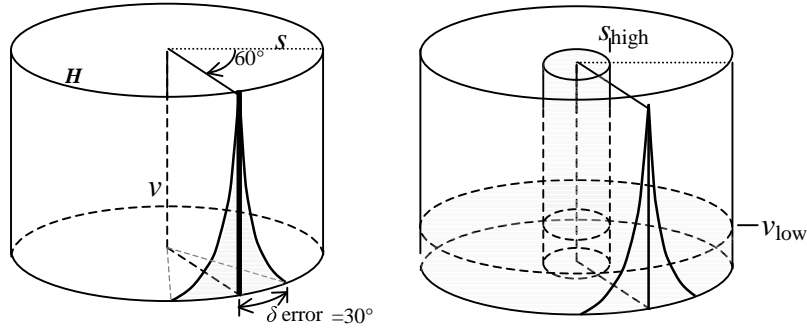
From (8.14), it is evident that hue is undefined when  $R = G = B = 0$ . In addition, hue is unstable when the colour is near the origin of the RGB space, giving rise to wide variations in hue for small changes in RGB.

In Figure 8.8, the range of ‘yellow’ colours, defined in RGB as  $(c, c, 0)$  for  $c \in [0..255]$ , map to the range of colours  $(60^\circ, 1, c)$  in the HSV colour space (the thick black vertical line in the diagram). Adding an error of  $(\pm\epsilon, 0, 0)$  to the RGB colours,

---

<sup>8</sup> The default value of 0.7 worked well for both of the PETS datasets.

results in the error shown in the figure. For example, if  $\varepsilon$  is 1, then the RGB colour  $(c \pm 1, c, 0)$  maps to HSV colour  $(60^\circ \pm \delta, 1, c)$ , where  $\delta$  is found to be<sup>9</sup>:  $\cos \delta = \frac{2c+1}{2\sqrt{c^2+c+1}}$ . The hue error  $\delta$  has a maximum value of  $30^\circ$  when  $c = 1 (=V)$ , showing the instability of hue for low RGB values.



**Figure 8.8:** Low chromaticity thresholds for HSV colour space

So, during background subtraction, shadow detection as defined in (8.13), is only done if the pixel's colour satisfies the following thresholds:

$$V > V_{low} \quad \text{and} \quad S < S_{high} \quad (8.15)$$

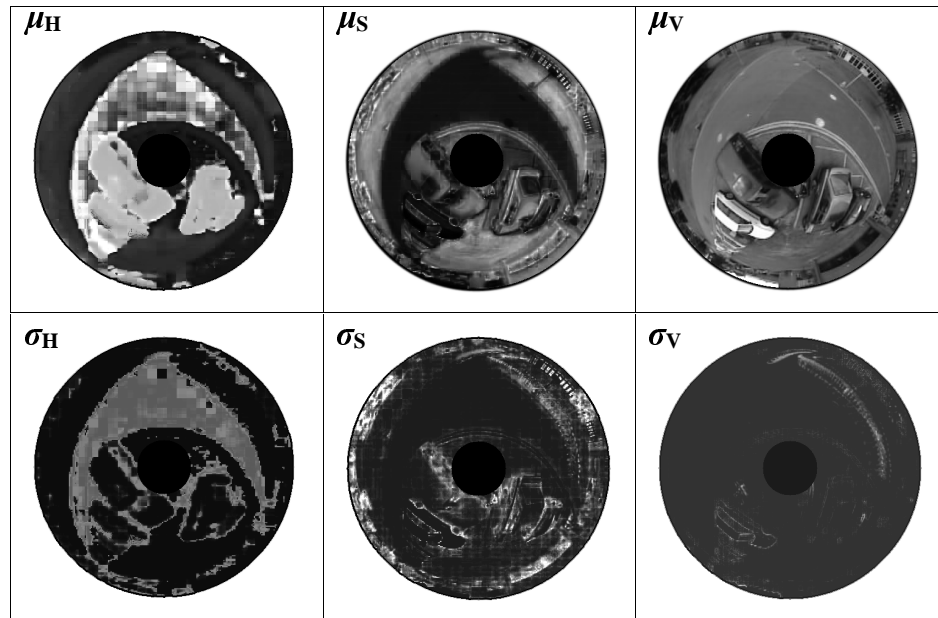
### 8.7.3 Background Initialisation

The background model  $B(x, y) = (\mu_{HSV}, \sigma_{HSV})$  is initialised from the first  $n$  frames of the video stream, using (8.3) and (8.4). The default duration is set to 32 frames, but this value is user configurable. Background initialisation may suffer from the problem of having moving objects within the scene while the background is being accumulated – this is the case of the PETS2001 dataset, where activity starts from frame 1. But no attempt was made at solving this problem as it wasn't considered critical – to compensate for this, one could increase the duration of background initialisation, and hope that there are not much slow-moving objects at the time.

The  $\sigma$  values obtained from background initialisation, are checked to make sure that they are above a certain global threshold  $\sigma_{min}$ . This threshold represents the estimate for the camera noise and also ensures that the variances of the background model are

<sup>9</sup> Derived by substituting the RGB colours  $(c+1, c, 0)$ ,  $(c, c, 0)$  into (8.14) and computing the difference between the two.

non-zero. One way of seeing this is as if applying a global threshold representing the camera noise, in addition to the per-pixel threshold that represents scene variability. A value of  $\sigma_{\min} = 5$  was chosen as the default for the HSV colour components.



**Figure 8.9:** Background mean and standard deviation maps

Figure 8.9 shows the background mean and standard deviation maps constructed during the initialisation phase. For speed reasons, the floating-point values  $\mu$  and  $\sigma$  are stored as integers in 16-bit image maps, with a fixed precision of 3 decimal places, and calculations are done using integer arithmetic. The mask generated by the calibration process (see §7.4.4) is used to reduce the workload per image. The OpenCV and IPL libraries both have conversion functions to and from HSV. But these use 8-bit images, reduce the hue to a range from 0 to 180, and do the work using floating-point arithmetic. Instead, the OmniTracking application uses its own conversion functions based on integer arithmetic.

#### **8.7.4 Background Subtraction Algorithm**

Using the background model maps mentioned in the previous section, and combining background subtraction with shadow detection, gives the following algorithm. This is combined later on with a hysteresis thresholding algorithm to get the final result.



The output of the algorithm will be the map  $l(x,y)$  where each pixel is labelled with one of the following values:

- background*..... this pixel represents a background element.
- foreground*..... this pixel represents a moving object.
- shadow pixel*..... this is a background pixel under the effect of shadow.
- probable pixel*..... this may be a moving object or a background shadow.  
Its final classification will be determined by the thresholding algorithm.

Let the colour of pixel  $(x,y)$  be  $(H,S,V)$ . The first step is obtaining the arithmetic difference for each colour component from the background value. In the case of hue, modular arithmetic is required:

$$\Delta_H = H \ominus \mu_H = \begin{cases} |H - \mu_H| & \text{if } |H - \mu_H| < \pi \\ \pi - |H - \mu_H| & \text{otherwise} \end{cases} \quad (8.16)$$

$$\Delta_S = |S - \mu_S| \quad (8.17)$$

$$\Delta_V = |V - \mu_V| \quad (8.18)$$

Then the pixel is labelled according to these conditions:

$$\text{if } \Delta_V > 3\sigma_V \text{ then:} \quad (8.19)$$

$$\text{if } \alpha\mu_V < V < \mu_V \text{ and } \Delta_H < 3\sigma_H \text{ and } \Delta_S < 3\sigma_S \text{ then:} \quad (8.19a)$$

$$l(x,y) = \text{shadow pixel}$$

$$\text{if } \Delta_V > 4\sigma_V \text{ then:} \quad (8.19b)$$

$$l(x,y) = \text{foreground pixel}$$

$$\text{else:} \quad (8.19c)$$

$$l(x,y) = \text{probable pixel}$$

$$\text{if } \Delta_H > 3\sigma_H \text{ and } V > V_{low} \text{ and } S < S_{high} \text{ then:} \quad (8.20)$$

$$\text{if } \Delta_H > 4\sigma_H \text{ then:} \quad (8.20a)$$

$$l(x,y) = \text{foreground pixel}$$

$$\text{else:} \quad (8.20b)$$

$$l(x,y) = \text{probable pixel}$$

$$\text{otherwise:} \quad (8.21)$$

$$l(x,y) = \text{background pixel}$$

Basically, (8.19) checks if a pixel had a large luminance variation and (8.20) checks if the pixel had a large hue variation. Inside the luminance condition, an extra check for shadow is done (8.19a), while the hue condition is only done if the pixel satisfies the low chromaticity thresholds ( $V_{low}, S_{high}$ ). For both luminance and hue, if the variation is above  $4\sigma$  (100% confidence) then the pixel is labelled foreground; if less than  $4\sigma$ , but above  $3\sigma$  (99.7% confidence), then it is labelled as ‘probable’. These probable pixels will then be examined later during the thresholding step, and either set to foreground or shadow or just discarded.

From initial test runs, it was found that because the algorithm checks for both luminance and chromaticity (hue) variations, better detection rates are obtained. This is mostly due to the camouflage problem, where objects might have the same colour or the same intensity as the background.

### 8.7.5 Thresholding with Hysteresis

The background subtraction algorithm labels some of the pixels as ‘probable’. The idea is that these pixels show a large difference from the background model ( $>3\sigma$ ), so most probably are foreground or shadow pixels, but their probability is not high enough to guarantee this on their own – they must be supported by neighbouring foreground or shadow pixels. This is what the hysteresis thresholding algorithm of the program does. In addition, it also acts as a noise filter by suppressing any isolated foreground or shadow pixels. The algorithm is described below:

For each pixel  $(x,y)$  in the label map  $l$ , consider its 8-neighbours and let:

$N_F$  = number of 8-neighbours labelled ‘foreground’, and

$N_S$  = number of 8-neighbours labelled ‘shadow’ pixels.

Then apply the following conditions to  $l(x,y)$  using threshold  $T$ , to demote any isolated pixels:

$$\text{if } l(x,y) = \text{foreground} \quad \text{and} \quad N_F < T \quad \text{then :} \quad (8.22)$$

$$l(x,y) = \text{probable}$$

$$\text{if } l(x,y) = \text{shadow} \quad \text{and} \quad N_S < T \quad \text{then :} \quad (8.23)$$

$$l(x,y) = \text{probable}$$

Finally apply the following conditions to  $l(x,y)$ , to promote any probable pixels that are supported by their neighbours:

$$\text{if } l(x,y) = \text{probable} \quad \text{and} \quad N_F \geq T \quad \text{and} \quad N_F > N_S \quad \text{then :} \quad (8.24)$$

$$l(x,y) = \text{foreground}$$

$$\text{if } l(x,y) = \text{probable} \quad \text{and} \quad N_S \geq T \quad \text{and} \quad N_S > N_F \quad \text{then :} \quad (8.25)$$

$$l(x,y) = \text{shadow}$$

Threshold  $T$  above is set to 4 – that is, half the neighbouring pixels. This process is repeated until either no pixel labels are changed in an iteration, or the maximum number of iterations are reached. Generally, thresholding with hysteresis converges quite rapidly and only a few iterations are required [TRUC98 §4.2.2]. In this case, the maximum number of iterations was set to 4. After thresholding is finished, any remaining probable pixels are set to background.

Other methods of achieving similar results are mentioned in §8.3.4, but compared with morphological operations, hysteresis thresholding was found to give good results and to be quite fast – it is faster than OpenCV’s morphological functions that are optimised for MMX processors.

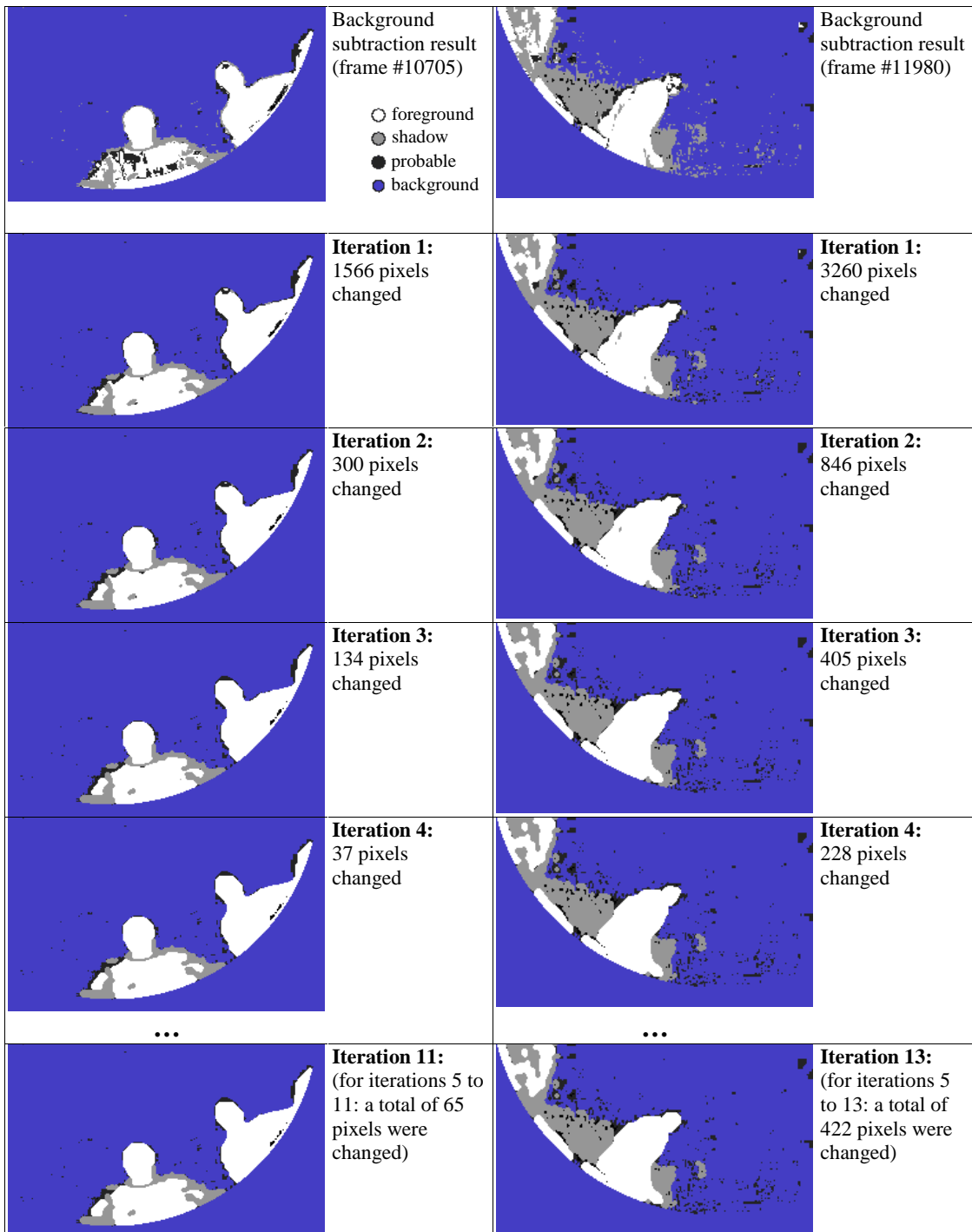
Figure 8.10 shows hysteresis thresholding applied to parts of two frames from the PETS-ICVS dataset. Note how the bulk of the changes happen in the first iteration for both frames. This was observed consistently throughout the rest of the video stream. The figure also shows what happens if the maximum number iterations is not limited to 4 – the further refinement in the labelling is minimal and not really worth the extra processing.

### 8.7.6 Background Model Adaptation

For the background model to remain useful, the motion detection module of the program uses a background adaptation algorithm based on the two blending parameter method (see §8.4.1). Parameter  $\alpha_{bkg}$  is used to update those pixels that in the label map  $l$  have been set to background or shadow, while  $\alpha_{frg}$  is used for the foreground-labelled pixels. Normally,  $\alpha_{frg} \leq \alpha_{bkg}$ , and the parameters are in range  $[0..1]$ . Setting  $\alpha_{frg}$  to 0, one gets the *selective update* method, while  $\alpha_{frg} = \alpha_{bkg}$  is the *blind update*

method. These two parameters are user-configurable and the following values were used for the PETS datasets:

dataset	$\alpha_{bkg}$	$\alpha_{frg}$
PETS2001 dataset 4	0.03000	0.00150
PETS-ICVS datasets C and B	0.00010	0.00001



**Figure 8.10:** Thresholding with Hysteresis – some results (PETS-ICVS dataset)

For the standard deviation map, the update (8.9), was modified from:

$$\begin{aligned}\mu_{t(x,y)} &= (1-\alpha)\mu_{t-1(x,y)} + \alpha f_t(x,y) & \alpha &= \alpha_{frg} \text{ or } \alpha_{bkg} \\ \sigma_{t(x,y)} &= (1-\alpha)\sigma_{t-1(x,y)} + \alpha\sqrt{f_t(x,y)^2 - \mu_{t(x,y)}^2}\end{aligned}$$

to:

$$\begin{aligned}\mu_{t(x,y)} &= (1-\alpha)\mu_{t-1(x,y)} + \alpha f_t(x,y) \\ \sigma_{t(x,y)} &= (1-\alpha)\sigma_{t-1(x,y)} + \alpha|f_t(x,y) - \mu_{t(x,y)}|\end{aligned}\quad (8.26)$$

to get rid of the expensive square-root.

The update parameters  $\alpha$  tend to be small fractions (like the values shown above). This can give rise to loss of numerical accuracy for floating-point arithmetic. In the case of this implementation, fixed-point integer math is used which can mean that some of the values will be truncated to 0. To avoid these problems, it was decided to do the background update only every number of  $N$  frames (set to 6 for this application).

To keep the same rate of exponential forgetting (of the old background values), (8.26) was modified slightly to take into account the fact that no update is done during the in-between  $N-1$  frames.

$$\begin{aligned}\alpha' &= \sum_{i=0}^{N-1} \alpha(1-\alpha)^i & (8.27) \\ \mu_{t(x,y)} &= (1-\alpha')\mu_{t-1(x,y)} + \alpha' f_t(x,y) \\ \sigma_{t(x,y)} &= (1-\alpha')\sigma_{t-1(x,y)} + \alpha'|f_t(x,y) - \mu_{t(x,y)}|\end{aligned}\quad \left( \begin{array}{l} \text{where } \alpha = \alpha_{frg} \text{ or } \alpha_{bkg} \\ \text{and } \alpha' = \alpha'_{frg} \text{ or } \alpha'_{bkg} \end{array} \right)$$

$\alpha'$  is only calculated once in the beginning, so no extra cost is added to the background update process (see §8.4 for how the exponential weighted sum is derived).

### 8.7.6.1 Background Model Failure

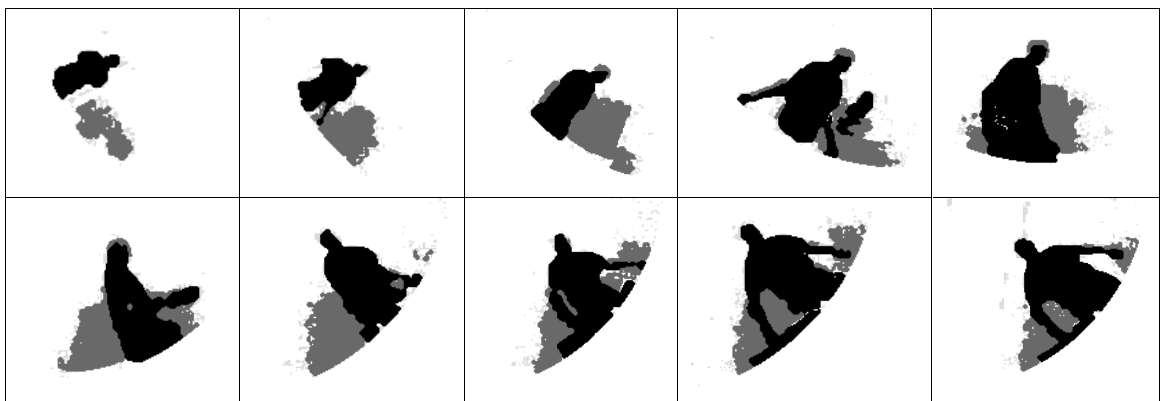
Even though it is adapted regularly, the background model may fail if there is a sudden and large illumination change, large camera movement, etc. The application should detect such situations, instead of just inundating later processing phases with detection errors and false positives.

The program keeps track of how many pixels are labelled as foreground. If this number exceeds 50% of the potential pixels (that is, taking into account the mask built by the calibration process to eliminate empty areas), an error message is reported on screen and process terminates.

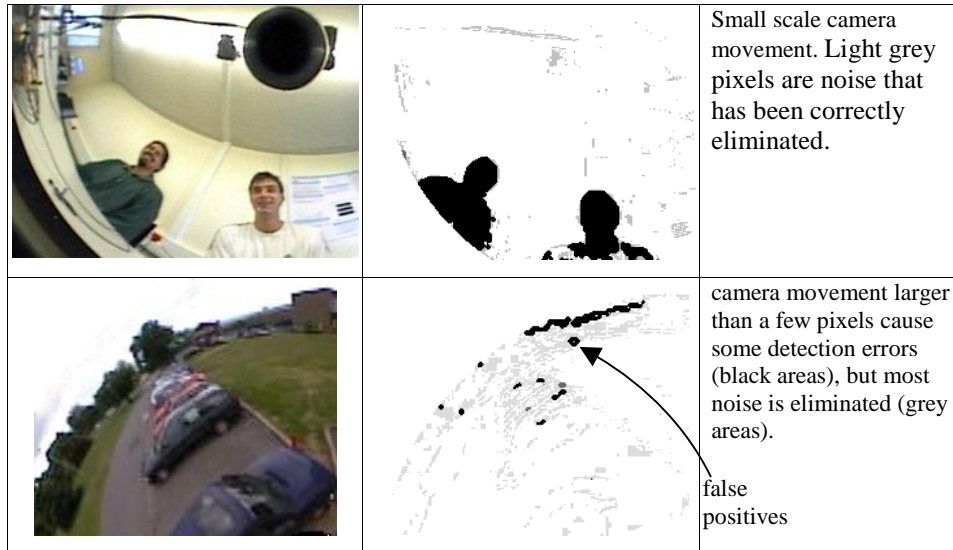
## 8.8 Results

In general, the background subtraction algorithm appears to perform quite well (at least on the PETS datasets). The set of figures below contain selected images that give some indication of where the algorithm works well and where it fails, in terms of the problems mentioned in §8.6.

The algorithm detects shadows quite accurately, as can be seen from Figure 8.11, where a person moves across the room and its outline is extracted quite well. Without shadow detection, the object's appearance would have been heavily distorted. The algorithm is also robust to small-scale noise and small camera motion ( $\sim \leq 3$  pixels). Where the background subtraction algorithm fails to suppress noise, later on, the tracking algorithm filters these out by using temporal constraints.

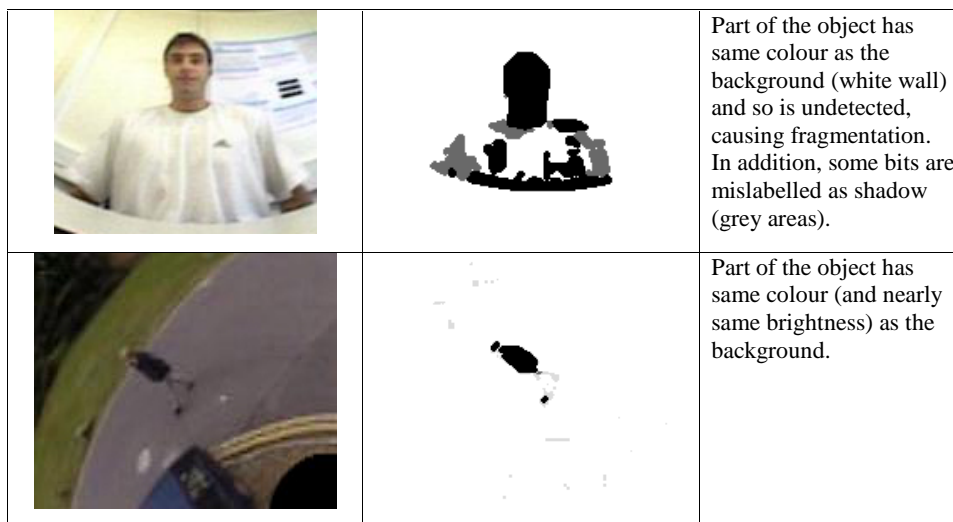


**Figure 8.11:** Results: detection of shadows (shadow shown in grey; PETS-ICVS dataset)






**Figure 8.12:** Results: noise due to camera movements

One problem that occurs mostly in the indoor scene is object fragmentation, when objects have the same colour (and brightness) as the background (camouflage problem). Some of these errors can be seen in Figure 8.13. This is solved later during the tracking phase by means of region clustering.

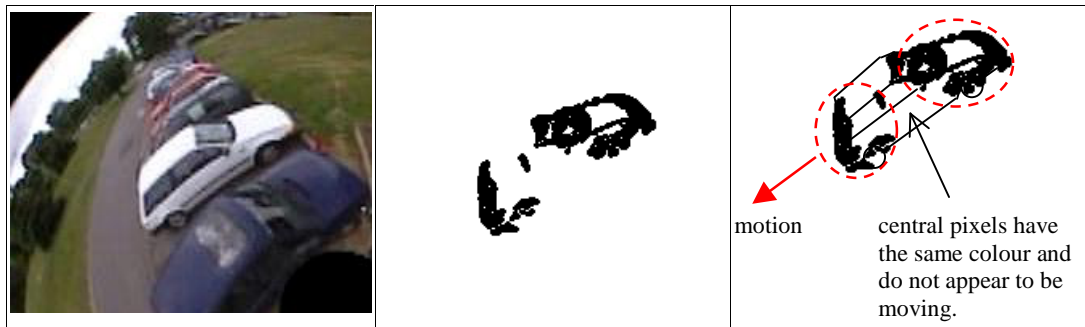


**Figure 8.13:** Results: Camouflage and object fragmentation

Other detection errors were generated because of the foreground aperture problem, presence of ghosts, and background elements that were displaced by the objects. Examples of these are shown in the remaining figures. These errors are eventually eliminated when the background adapts fully to them.

	<p>The original image with no motion in it.</p>
	<p>The blue car moves away, exposing the ground below it.</p>
	<p>Two objects detected: the car (on the left) and its ghost on the right (exposed ground).</p>

**Figure 8.14:** Results: The Ghost problem.



**Figure 8.15:** Results: Foreground Aperture problem



**Figure 8.16:** Results: Moved background elements