Universiteit
Leiden
The Netherlands

Content-based retrieval of visual information

Oerlemans, A.A.J.

Citation
Oerlemans, A. A. J. (2011, December 22). *Content-based retrieval of visual information*. Retrieved from https://hdl.handle.net/1887/18269

Version:	Corrected Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/18269

Note: To cite this publication please use the final published version (if applicable).

Chapter 9

Detecting and Identifying Moving Objects in Real-Time

For surveillance applications or for human-computer interaction, the automated real-time tracking of moving objects in images from a stationary camera can be very useful. In this chapter we describe a novel method for object tracking which works in real-time using standard hardware. The object tracking algorithm uses an improved adaptive version of an existing robust motion detection method and a set of decision-rules is used to handle difficult situations such as tracking multiple objects at the same time, crossing paths and partial occlusion. Experiments on several real complex test sets were performed and discussed.

9.1 Introduction

Tracking moving objects in images from a stationary camera, such as people walking or cars driving, is a very useful technique for surveillance applications. The same technique can also be used in different environments, such as human-computer interaction or the automated counting of objects. A well-known example of a real-time object tracking application used for human-computer interaction is the Sony eye-toy.

We have implemented a new method of object tracking by using decision rules for handling the fairly general problem of unlimited moving objects, occlusions by the environment and other objects, and indoor and outdoor lighting.

9.2 Related work

A lot of work has been done in the field of object tracking. [8] [10] [19] [20] [42] [49] [94] One of the main conferences for this research subject is Performance and Evaluation of Tracking and Surveillance (PETS). The articles clearly show that several common problems such as occlusions, multiple objects and variable illumination are largely unsolved. Many of the systems achieve good accuracy by making special assumptions such as a single person, two persons, only indoors lighting, etc. In fact, the conclusions of some of the advanced systems are simply that they are unsuited for outdoors [82]. In short, the general problem that we have looked at is very difficult.

The first step for every object tracking algorithm is motion detection. This can be done using a standard background-subtraction method based on color or intensity, but other more sophisticated methods like Gaussians [14] or optical flow [50] [47] are widely used. For an overview of background subtraction algorithms, see [5].

For the object tracking itself, a variety of methods is available. The choice of the right method depends on the application. Object tracking can be done with just simple nearest-neighbor matching of regions with motion [53]. An improvement of this method would be to use motion prediction, for example with Kalman filters [17] or Bayesian networks [20].

Other methods for object tracking include the kernel-based object tracker made by Comaniciu et al. [9]. Another interesting real-time object tracking system to look at is W4, by Haritaoglu et al. [21] [22].

9.3 Motion detection

Horprasert, Harwood and Davis have developed a method [27] [28] for robust motion detection using background subtraction with a statistical model of the background. We have chosen to adapt this method for our object tracking algorithm, because of its robustness to changes in lighting and the ability to adapt to deposited objects. The main idea of this method is to decompose the difference between the current color information and the modeled background for every pixel into a chromaticity (color) component and a brightness component.

After the background model is created (see below for details), new frames can be compared to this model to detect motion by comparing the expected RGB value for pixel i, $E_i = [E_R(i), E_G(i), E_B(i)]$, to the RGB value of that pixel in the current frame, $I_i = [I_R(i), I_G(i), I_B(i)]$ in the following way:

When seen in a three-dimensional RGB cube, the line between the origin and E_i is called the expected chromaticity line for pixel i. All colors on this line are perceived as the same color but with a different brightness. When comparing a pixel value from the current frame to the background model, there is a point $\alpha_i E_i$ for which the distance from I_i to $\alpha_i E_i$ is minimal. This distance is called

the chromaticity distortion, the difference in color. The value α_i is called the brightness distortion, the difference in brightness.

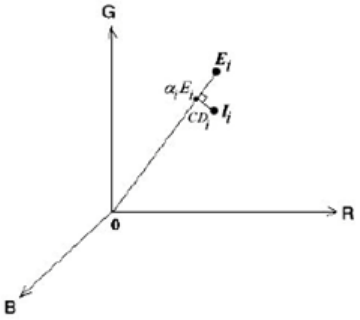

Figure 9.1: The RGB cube with points E and I.

If the chromaticity distortion or the brightness distortion exceeds certain thresholds, the pixel is classified as a motion-pixel.

9.3.1 Building the background model

The background model is created using a set of N frames without motion. We have used $N = 50$ for our experiments, which equals to two seconds of video. First, for each pixel i the mean RGB value over N frames is calculated. The mean R value of pixel i for example, is calculated as follows:

$$\mu_R(i) = \sum_{j=1}^{N} I_R(i_j) \tag{9.1}$$

where i_j is pixel i in frame j. This results in the expected color value

$$E_i = [\mu_R(i), \mu_G(i), \mu_B(i)] \tag{9.2}$$

for each pixel i. Next, the standard deviation

$$s_i = [\sigma_R(i), \sigma_G(i), \sigma_B(i)] \tag{9.3}$$

is calculated, where $\sigma_R(i)$ is the standard deviation for the I_R values of pixel i over N frames. The value s_i is used for normalizing the mean RGB values.

As mentioned above, the brightness distortion α_i can be found by minimizing the distance between I_i and the line OE_i:

$$\phi(\alpha_i) = (I_i - \alpha_i E_I)^2 \tag{9.4}$$

In terms of RGB values and taking the normalization into account, the equation for α_i can be written as:

$$\begin{aligned} \alpha_i &= \underset{c}{\operatorname{argmin}} \left[\begin{array}{l} \left(\frac{I_R(i)-c\mu_R(i)}{\sigma_R(i)}\right)^2 + \\ \left(\frac{I_G(i)-c\mu_G(i)}{\sigma_G(i)}\right)^2 + \\ \left(\frac{I_B(i)-c\mu_B(i)}{\sigma_B(i)}\right)^2 \end{array} \right] \\ &= \frac{\left(\frac{I_R(i)\mu_R(i)}{\sigma^2_R(i)} + \frac{I_G(i)\mu_G(i)}{\sigma^2_G(i)} + \frac{I_B(i)\mu_B(i)}{\sigma^2_B(i)}\right)}{\left([\frac{\mu_R(i)}{\sigma_R(i)}]^2 + [\frac{\mu_G(i)}{\sigma_G(i)}]^2 + [\frac{\mu_B(i)}{\sigma_B(i)}]^2\right)} \end{aligned} \tag{9.5}$$

The chromaticity distortion CD for pixel i can be calculated by taking the absolute difference between I_i and $\alpha_i E_i$. In terms of RGB values this becomes:

$$CD_i = \sqrt{\frac{I_R(i) - \alpha_i\mu_R(i)}{\sigma^2_R(i)} + \frac{I_G(i) - \alpha_i\mu_G(i)}{\sigma^2_G(i)} + \frac{I_B(i) - \alpha_i\mu_B(i)}{\sigma^2_B(i)}} \tag{9.6}$$

Two other values are also calculated: the variation of the brightness distortion α_i and the variation of the chromaticity distortion b_i. These are defined as:

$$a_i = \sqrt{\frac{\sum_{i=1}^{N}(\alpha_i - 1)^2}{N}} \tag{9.7}$$

$$b_i = \sqrt{\frac{\sum_{i=1}^{N}(CD)^2}{N}} \tag{9.8}$$

Note that, although the name may suggest otherwise, these values are actually the standard deviations of the brightness and the chromaticity distortion. Using these two values, the normalized brightness distortion and the normalized chromaticity distortion can be calculated:

$$\overline{\alpha_i} = \frac{\alpha_i - 1}{\alpha_i} \tag{9.9}$$

$$\overline{CD_i} = \frac{CD_i}{b_i} \tag{9.10}$$

At this point, each pixel is modeled by a tuple of four values: (E_i, s_i, a_i, b_i). Now, for each new frame, pixel values can be compared to the model to detect motion. For each pixel the normalized chromaticity distortion and the normalized brightness distortion are calculated and compared to certain thresholds. Motion

is detected at a pixel if the normalized chromaticity distortion is higher than τ_{CD} or if the normalized brightness distortion is below τ_{α}.

Thresholds τ_{CD} and τ_{α} are selected by creating a histogram of the normalized chromaticity distortions and a histogram of the normalized brightness distortions of all pixels of all frames in the training set. Below is an example of such histograms:

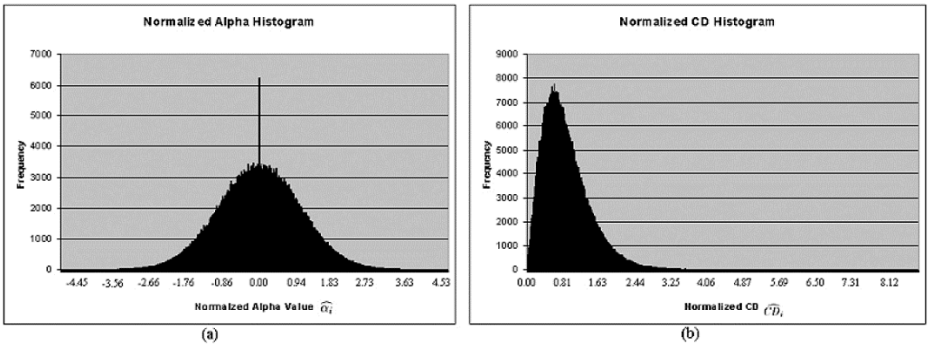

Figure 9.2: Histograms of the (a) normalized Alpha (brightness) and (b) CD (chromaticity) values

By choosing a percentage of all pixels of the training set which should be correctly classified as background pixels, a chromaticity distortion threshold can be selected. Note that in fact all pixels should be classified as background, but this would result in too high a threshold.

9.3.2 Adaptive background model

The model described above is static. Once all values are calculated, the model is not changed, although the scene for which the model is made could be changing due to different lighting conditions or deposited objects. To handle the situation of a changing background, we have altered the algorithm described above to get an adaptive version.

To accomplish this, the entire training set is kept in memory and for each new frame the oldest frame in the training set is replaced by the current frame, but only those pixels where no motion is detected are replaced. Ideally, the background model should be recalculated completely at this point, but for real-time applications this is too time-consuming. For this reason, only one scanline of the model is recalculated.

The global chromaticity distortion threshold cannot be used in this adaptive version of the model, so it is changed into a per-scanline threshold. Each time the

background model for a scanline is updated, the chromaticity distortion threshold for that line is recalculated.

Another difference between the motion detection method by Horprasert et al. and our method is the use of an upper bound for the brightness distortion, where the lower bound is denoted by $\tau_{\alpha 1}$ and the upper bound by $\tau_{\alpha 2}$. We classify a pixel as motion pixel when the brightness distortion is below $\tau_{\alpha 1}$ or above $\tau_{\alpha 2}$.

9.3.3 Post processing

The output of the motion detection algorithm contains noise. This is especially true when used with compressed video input, because the algorithm was designed to be used with uncompressed data. For clearing the noise from the images, a number of image filters has been used, such as removing salt-and-pepper noise, erosion, dilation and median filtering.

For object tracking, the output of the motion detection has to be converted to a list of regions with motion. Each motion-pixel is assigned a region ID, based on 8-connectivity.

9.4 Object tracking

For each input frame, the motion detection algorithm yields a list of regions, or blobs. A blob, short for 'binary large object', is a coherent group of pixels where motion was detected. Every object moving through the scene will create such a blob in a series of successive frames. The task of the object tracking algorithm is linking the positions of these blobs to a single object and to find out the path which has been followed by the object.

Figure 9.3 illustrates this. The object tracking algorithm should report one object for each frame and for each of these frames, the object has to be given the same identifier.

For each frame, the task is to link each blob to an object that was present in the previous frame. One possibility for finding a corresponding object for each blob is comparing the positions of the blobs with all positions of objects in the previous frame. An improved method would be to predict the position each object would have in the current frame and to compare those to the positions of the blobs. This method is of course very dependent on the method of motion prediction used.

For speed reasons, we chose to use yet another method for linking blobs to objects. We decided to use the bounding boxes of the blobs for tracking. Object positions and estimated positions are also stored as a bounding box.

Figure 9.3: Some frames from a sequence where one person walks through the scene.

9.4.1 Data structure

First, we define the data structures used by the object tracking algorithm. The inputs for the algorithm are the blobs detected by the motion detection algorithm. These blobs have the following properties:

- Position of the bounding box
- A list of pixels belonging to the blob
- Color model

The color model contains the average color for 32 parts of the blob. We chose to divide the blobs vertically into 32 equal parts. Figure 9.4 is a schematic view of such color models.

Figure 9.4: Output of the motion detection algorithm: blobs and their color models.

The object tracking algorithm keeps an internal list of tracked objects. For each frame, these objects are compared with the detected blobs and if they correspond, they are updated with information of these blobs. Objects have the following properties:

- Object ID
- A history of a maximum of 25 positions of this object in previous frames
- The predicted position of this object for the next 25 frames
- Color model
- Type

Object positions are stored for the prediction of movement. The object type can be either be simple or compound, which will be explained later.

Before explaining each step of the object tracking algorithm, we show a typical situation in figure 9.5: the motion detection algorithm has detected two blobs and the object tracking algorithm was tracking two objects. The blobs correspond with the predicted positions of the objects, so the objects will be updated with the position and color information of the blobs. The third image shows all previous and predicted positions of the tracked objects.

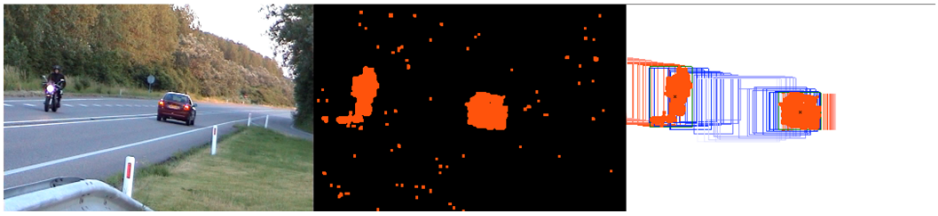

Figure 9.5: An input frame, the detected blobs and the previous and estimated object positions.

9.4.2 Object motion prediction

Using the coordinates of the bounding boxes of the previous positions of an object, a prediction can be made for the position of these bounding boxes in future frames. A bounding box of an object position at a certain time is actually a tuple $(t_k, X_1, Y_1, X_2, Y_2)$. A prediction has to be made for these tuples with values t_{k+1} and higher, where t_k is the value of t for the current frame. We chose to use a quadratic approximation of $(t_k, (X_1 + X_2)/2)$ and $(t_k, (Y_1 + Y_2))/2)$.

With this approach, the size of the predicted bounding box will be same as the current size. (Figure 9.5 illustrates this.)

A quadratic approximation of a series of tuples $(t_1, X_1), \ldots, (t_n, X_n)$ can be calculated by a least-squares fit of a quadratic function. The quadratic function can be described by the function $y = ax^2 + bx + c$. We try to find the values for a, b and c so that the error-value, defined as

$$\epsilon(a, b, c) = \sum_{i=1}^{n} (at_i^2 + bt_i + c - X_i)^2 \tag{9.11}$$

has the smallest value.

It is known that if the partial derivatives of the error function with respect to a, b and c are all equal to zero, the error function has the least value. The partial derivative of the error function with respect to a is

$$\frac{\partial \epsilon}{\partial a} = \sum_{i=1}^{n} 2(at_i^2 + bt_i + c - X_i) \tag{9.12}$$

Setting this equal to zero and rearranging gives the following formula:

$$a[\sum_{i=1}^{n} t_i^2] + b[\sum_{i=1}^{n} t_i] + cn = \sum_{i=1}^{n} X_i \tag{9.13}$$

The same can be done for the partial derivatives with respect to b and c, which yields these formulas:

$$a[\sum_{i=1}^{n} t_i^3] + b[\sum_{i=1}^{n} t_i^2] + c[\sum_{i=1}^{n} t_i] = \sum_{i=1}^{n} X_i \tag{9.14}$$

$$a[\sum_{i=1}^{n} t_i^4] + b[\sum_{i=1}^{n} t_i^3] + c[\sum_{i=1}^{n} t_i^2] = \sum_{i=1}^{n} X_i \tag{9.15}$$

These three formulas can be solved and the resulting a, b and c values can be used for the quadratic prediction function.

9.4.3 Rule-based object tracking

For object tracking, the task is to find corresponding blobs in the current frame for the objects being tracked. We define this correspondence in two different ways:

- A blob belongs to a tracked object, if the blob covers more than 50% of the predicted bounding box of the object
- An object contains a blob, if more than 50% of the area of the blob is within the area of the predicted bounding box of the object

Note that this means that an object can have no more than one blob belonging to it and that a blob can be contained by no more than one object.

To illustrate the concepts of belongs to and contains, a few schematic examples will be given. In these examples, a red circle stands for a blob and a green square stands for the predicted position of an object. Figure 9.6 shows the ideal situation: A blob belongs to an object (it covers more than 50% of the bounding box) and the object also contains the blob (at least 50% of the blob is inside the bounding box).

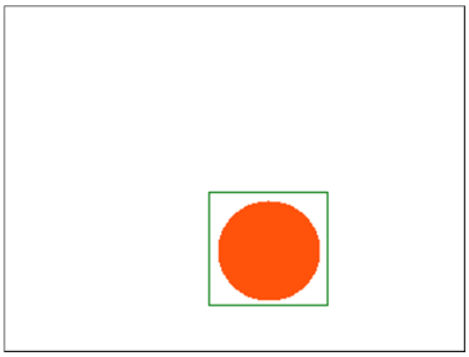

Figure 9.6: A blob belonging to an object, which also contains the blob.

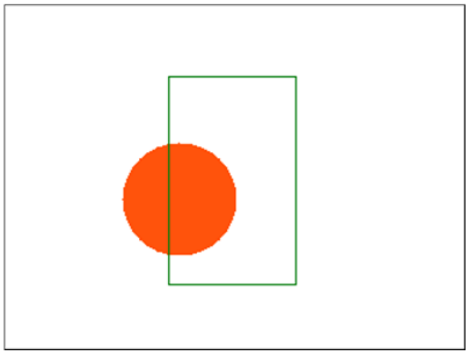

Figure 9.7: An object containing a blob, which does not belong to it.

Figure 9.7 shows that a blob can be contained by an object, but that the blob does not have to belong to it.

We need to find not only one object which a blob belongs to, but every object a blob belongs to. Also, we want to know every blob that is contained by an object. At this point, we introduce the Type property of an object: The Type property of an object can be either simple or compound. Simple objects are all objects which can be tracked by the straightforward version of the object tracking algorithm in which every blob corresponds to no more than one object.

Compound objects are virtual objects which consist of two or more objects which are currently occluding or occluded. The object tracking algorithm will track these objects just as simple objects, but will also track the actual simple objects which are involved.

The object tracking starts with an empty list of tracked objects. For each new input frame the following is done:

- For each blob, all objects it belongs to are searched for. A matrix B is created, where the value at $B_{i,j}$ equals 1 if blob i belongs to object j and it equals 0 otherwise.
- For each tracked object, the blobs it contains are determined. A matrix C is created, where the value at $C_{i,j}$ equals 1 if object i contains blob j, otherwise this value is 0.
- For each blob k, calculate the number of objects the blob belongs to:

$$b_k = \sum_i B_{k,i} \tag{9.16}$$

- For each object l not being part of a compound object, calculate the number of blobs contained by that object:

$$c_l = \sum_j C_{l,j} \tag{9.17}$$

- For each blob k, follow these rules to determine which objects need to be updated:
  - $b_k = 0$
    There is no object which this blob belongs to. This situation will be handled later.
  - $b_k = 1$
    This blob belongs to one object l. Check the c value for that object and do the following:
    - $c_l = 0$
      The object does not contain any blob. Apparently, the predicted position of the object and the actual position are differing. Update the object with information of the blob.
    - $c_l = 1$
      The object contains a blob. We assume that this blob is the blob we are currently checking. Update the object with information of the blob.
    - $c_l > 1$
      The object contains more than one blob. There are two possible cases:
      - The object is a simple object. We assume that the object contains the current blob and several other blobs and that all these blobs actually belong to one object. The object is updated with information of all blobs.
      - The object is a compound object. We assume that the compound object has split up into one or more simple objects and

at most one compound object. For all blobs contained by the object, try to find a corresponding simple object by comparing their predicted positions to the blob positions. In other words, try to find the simple objects which contain the blobs.

For all blobs for which no object is found, try to use the color model to find the corresponding object. The object with the closest matching color model will be updated with the information of a blob.

- $b_k > 1$
  This blob belongs to more than one object. We assume this is the result of two or more objects occluding each other. All objects this blob belongs to are combined into a new compound object. Note that an existing compound object can be one of these objects.

  The c value for each of the objects is not checked at this point. We assume that objects do not split up into several parts because of failing motion detection at the same time that it starts occluding an object.

- For each object l not updated yet, check the c value of the object. We then have the following possibilities:
  - $c = 0$
    The object does not contain a blob. It is possible that this is a simple object which is part of a compound object. This will be handled later.
  - $c = 1$
    The object contains one blob. The object is updated with information of that blob.
  - $c > 1$
    The object contains more than one blob. If the object is a simple object, we assume that all blobs are part of the object. The object will be updated with the combined information of these blobs.

    If the object is a compound object, we assume the object has split into one or more simple objects and at most one compound object. Use the previous rule for the case $b_k = 1$, $c_l > 1$ and with a compound object.
- Each object that is still part of a compound object is updated with its predicted position.
- For each blob that does not belong to an object and that is not contained by an object, create a new object.
- Each object that has not been updated at this point, is deleted from the list of tracked objects.

There are some assumptions made in the above algorithm, which might not always be true for some situations. Consequently, we expect the algorithm to fail for those situations. For example, we assume that objects can only be detected as two separate blobs because of failing motion detection, but we do not take into

account the possibility of an object actually splitting up. This can happen when someone steps out of a car, or when someone leaves a suitcase behind.

9.5 Results

We have tested the output of the object tracking algorithm on a set of six videos for which we created a ground truth. Table 9.1 gives a description for each video (see also Figure 9.8).

Number	Description
1	one person walking
2	a car driving
3	two persons walking
4	two persons walking, crossing paths
5	one person walking, partial occlusion
6	PETS2000 test video

Table 9.1: Test videos used in our experiments.

For our experiments, we define the following: An object is detected if the area of the object in the ground truth covers at least 50% of the area of an object in the program output, or if the area of the object in the program output covers at least 50% of the area of the object in the ground truth.

For each frame of the input video, the following values are calculated:

- G - # of objects in the ground truth for this frame
- P - # of objects in the program output for this frame
- C - # of correctly detected objects for this frame
- I - # of false detections for this frame

Note that $(I + C) = P$.

With these values, we can evaluate the object tracking algorithm. We have used the following benchmarks:

- The detection rate: For all frames, the percentage of objects correctly detected, C/P.
- The misdetection rate: For all frames, the percentage objects detected when no object was present, I/P.
- The mean error for the position of detected objects. The error-value for one object-position is:

$$\epsilon = (G_{x1} - P_{x1})^2 + (G_{y1} - P_{y1})^2 + (G_{x2} - P_{x2})^2 + (G_{x2} - P_{x2})^2 \quad (9.18)$$

where the corners of the bounding box are $(x1, y1)$ and $(x2, y2)$.

- The standard deviation of the error.
- The probability of an object ID change between two frames.

The results that we obtained can be seen in table 9.2.

	Video number					
	1	2	3	4	5	6
Detection	100.0	91.76	100.0	100.0	94.74	44.80
Misdetection	0.00	21.43	0.00	5.88	0.00	2.74
Mean error	2.75	4.94	9.48	3.85	4.80	7.84
Error std.dev.	3.47	4.29	4.99	3.21	4.15	4.32
P(id switch)	0.06	0.00	0.09	0.00	0.00	0.03

Table 9.2: Results from our object tracking experiments.

Note that for all videos 1-5, the detection rates are high and the misdetection rates are low. For the PETS2000 video which we chose for its complexity, it clearly shows an area for future improvement in our decision rules. In the current implementation, no rules exist in our algorithm for a situation which occurs in this video: A car being parked and someone stepping out of it, which results in 1 object becoming 2 objects. This problem will be addressed with an update to our rules.

9.6 Conclusions and future work

We have shown that using a rule-based algorithm for object tracking, we can reliably track multiple objects under several complex situations such as crossing paths, occlusion, and variable lighting. Our current system needs to be refined toward handling the situation where a single object becomes multiple objects.

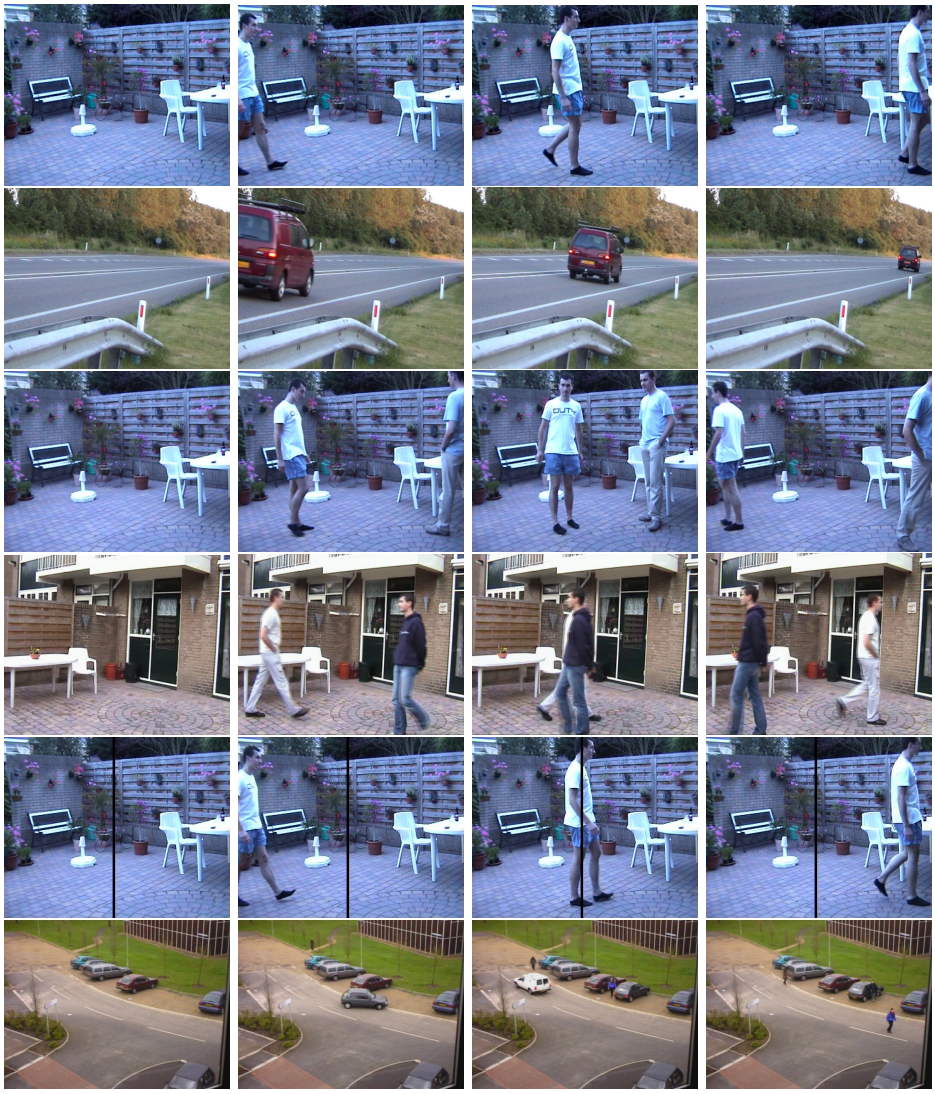

Figure 9.8: Some frames from the six video sequences used for testing the object tracking algorithm.