

Chapter 3

Transformation-Search Based Methods

Abstract Another way of object representation is to utilize object models consisting of a finite set of points and their position. By the usage of point sets recognition can be performed as follows: First, a point set is extracted from a scene image. Subsequently, the parameters of a transformation which defines a mapping of the model point set to the point set derived from the scene image are estimated. To this end, the so-called transformation space, which comprises the set of all possible transform parameter combinations, is explored. By adopting this strategy occlusion (resulting in missing points in the scene image point set) and background clutter (resulting in additional points in the scene image point set) both lead to a reduction of the percentage of points that can be matched correctly between scene image and the model. Hence, occlusion and clutter can be controlled by the definition of a threshold for the portion of the point sets which has to be matched correctly. After introducing some typical transformations used in object recognition, some examples of algorithms exploring the transformation space including the so-called generalized Hough transform and the Hausdorff distance are presented.

3.1 Overview

Most of the global appearance-based methods presented so far suffer from their invariance with respect to occlusion and background clutter, because both of them can lead to a significant change in the global data representation resulting in a mismatch between model and scene image.

As far as most of the methods presented in this chapter are concerned, they utilize object models consisting of a finite set of points together with their position. In the recognition phase, a point set is extracted from a scene image first. Subsequently, transformation parameters are estimated by means of maximizing the similarity between the scene image point set and the transformed model point set (or minimizing their distance respectively). This is done by exploring the so-called transformation space, which comprises the set of all possible transform parameter combinations. Each parameter combination defines a transformation between the model data and the scene image. The aim is to find a combination which maximizes

the similarity (or minimizes a distance, respectively). Finally, it can be checked whether the similarities are high enough, i.e., the searched object is actually present at the position defined by the transformation parameters.

Occlusion (leading to missing points in the scene image point set) and background clutter (leading to additional points in the scene image point set) both result in a reduction of the percentage of points that can be matched correctly between scene image and the model. Hence, the amount of occlusion and clutter which still is acceptable can be controlled by the definition of a threshold for the portion of the point sets which has to be matched correctly.

The increased robustness with respect to occlusion and clutter is also due to the fact that, with the help of point sets, *local* information can be evaluated, i.e., it can be estimated how well a single point or a small fraction of the point set located in a small neighborhood fits to a specific object pose independent of the rest of the image data (in contrast to, e.g., global feature vectors where any discrepancy between model and scene image affects the global features). Additionally, it is possible to concentrate the point set on characteristic parts of the object (in contrast to gray value correlation, for example).

After a brief discussion of some transformation classes, some methods adopting a transformation-based search strategy are discussed in more detail. The degrees of freedom in algorithm design for this class of methods are

- *Detection method for the point set* (e.g., edge detection as proposed by Canny [3], see also Appendix A). The point set must be rich enough to provide discriminative information of the object. On the other hand, however, large point sets lead to infeasible computational complexity.
- *Distance metric* for measuring the degree of similarity between the model and the content of the scene image at a particular position.
- *Matching strategy* of searching the transformation space in order to detect the minimum of the distance metric. A brute force approach which exhaustively evaluates a densely sampled search space is usually not acceptable because the algorithm runtime is too long. As a consequence a more intelligent strategy is required.
- *Class of transformations* which is evaluated, e.g., affine or similarity transforms.

3.2 Transformation Classes

Before we take a closer look at some methods which search in the transformation space we have to clarify what kind of transformation is estimated. Commonly used transformation classes are similarity transformations and affine transformations. Both of them are linear transformations, a fact that simplifies calculations and therefore reduces the computational complexity significantly compared to the usage of non-linear transformations. In reality, however, if a 3D object is moved in 3D space, the appearance change of the object in a 2D image acquired by a camera at fixed position can only be modeled exactly by a perspective transformation,

which is non-linear. Fortunately, the perspective transformation can be approximated by an affine transformation with good accuracy if the “depth” of the object resulting from the third dimension is small compared to the distance to the camera and therefore the object can be regarded as *planar*. Affine transformations are given by

$$\mathbf{x}_{S,i} = \begin{bmatrix} x_{S,i} \\ y_{S,i} \end{bmatrix} = \mathbf{A} \cdot \mathbf{x}_{M,i} + \mathbf{t} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x_{M,i} \\ y_{M,i} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (3.1)$$

where $\mathbf{x}_{S,i}$ denotes the position of a point or feature (e.g. line segment) in the scene image and $\mathbf{x}_{M,i}$ its corresponding model position. The matrix \mathbf{A} and a translation vector \mathbf{t} parametrize the set of all allowed transformations. Altogether, affine transformations are specified by six parameters a_{11} , a_{12} , a_{21} , a_{22} , t_x , and t_y . A further simplification can be done if only movements of planar 2D objects perpendicular to the optical axis of the image acquisition system together with scaling have to be considered. In that case the affine transformation can be reduced to the similarity transform

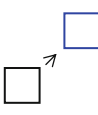
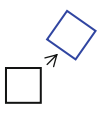
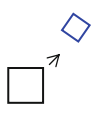
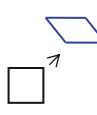
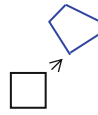
$$\mathbf{x}_{S,i} = \begin{bmatrix} x_{S,i} \\ y_{S,i} \end{bmatrix} = \mathbf{S} \cdot \mathbf{x}_{M,i} + \mathbf{t} = s \cdot \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix} \cdot \begin{bmatrix} x_{M,i} \\ y_{M,i} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (3.2)$$

characterized by four parameters s, φ, t_x , and t_y . s denotes a scaling factor, φ a rotation angle, and t_x and t_y a translation in the image plane. If s is explicitly set to 1, the transformation is called rigid transformation.

Some types of transformations are illustrated in Table 3.1: The rigid transform comprises translation and rotation; the similarity transform in addition contains scaling. The affine transform maps parallel lines to parallel lines again, whereas the perspective transform, which is nonlinear, maps a square to a quadrangle in the general case.

Perspective transformations can actually also be modeled linear if so-called homogeneous coordinates are used: a 2D point for example is then represented by the triple $[\lambda \cdot x, \lambda \cdot y, \lambda]$, where λ denotes a scaling factor. Points are regarded as equivalent if they have identical x and y values, regardless of the value of λ . Using homogeneous coordinates, the projective transformation of a point located in a plane to another plane can be described as

Table 3.1 Overview of some transformation classes

				
Translation	Rigid transform	Similarity transform	Affine transform	Perspective transform

$$\mathbf{x}_{S,i} = \begin{bmatrix} \lambda \cdot x_{S,i} \\ \lambda \cdot y_{S,i} \\ \lambda \end{bmatrix} = \mathbf{T} \cdot \mathbf{x}_{M,i} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{M,i} \\ y_{M,i} \\ 1 \end{bmatrix} \quad (3.3)$$

Hence, eight parameters are necessary for characterization of a perspective transform.

3.3 Generalized Hough Transform

3.3.1 Main Idea

The Hough Transform was originally developed for the detection of straight lines (cf. Hough [7] or Duda and Hart [4]), but can be generalized to the detection of arbitrarily shaped objects if the object shape is known in advance.

Now let's have a look at the basic idea of the Hough transform: given a set of points \mathbf{P} , every pixel $p = [x, y] \in \mathbf{p}$ could possibly be part of a line. In order to detect all lines contained in \mathbf{P} , each p "votes" for all lines which pass through that pixel. Considering the normal form

$$r = x \cdot \cos(\alpha) + y \cdot \sin(\alpha) \quad (3.4)$$

each of those lines can be characterized by two parameters r and α . A 2D accumulator space covering all possible $[r, \alpha]$, which is divided into cells, accounts for the votes. For a given point $[x, y]$, all parameter combinations $[r, \alpha]$ satisfying the normal form can be determined. Each of those $[r, \alpha]$ increases the corresponding accumulator cell by one. Taking all pixels of the point set into account, the local maxima of the accumulator reveal the parameters of the lines contained in the point set (if existent).

The principle of voting makes the method robust with respect to occlusion or data outliers, because even if a fraction of the line is missing, there should be enough points left for a "correct" vote. In general the Hough transform works on an edge-filtered image, e.g., all pixels with gradient magnitude above a certain threshold participate in the voting process.

For a generalization of the Hough transform (cf. Ballard [1]) a model of the object contour has to be trained prior to recognition. The thus obtained information about the object shape is stored in a so-called R-Table. Subsequently, recognition is guided by this R-Table information. The R-Table generation proceeds as follows.

3.3.2 Training Phase

1. *Object contour point detection:* In the first step all contour points $\mathbf{x}_{T,i} = [x_{T,i}, y_{T,i}]$ of a sample image showing the object to be trained are located together with their gradient angle θ_i , e.g., with the help of the canny edge

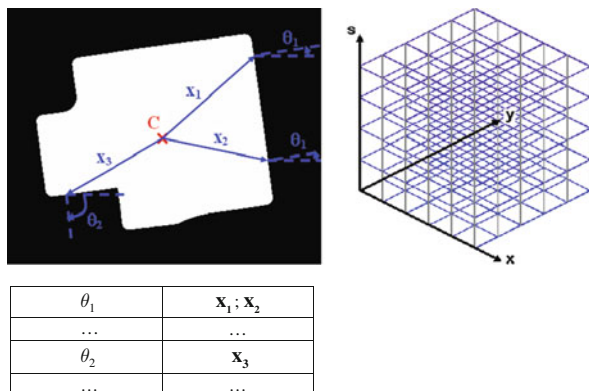


Fig. 3.1 Illustrative example of the generalized Hough transform. *Left*: R-table generation with three example points; *Right*: 3D accumulator for translation and scale estimation of size $5 \times 5 \times 4$

detector [3] including non-maximum suppression (cf. Appendix A). The underlying assumption is that the object contour is characterized by rapid gray value changes in its neighborhood due to the fact that the background usually differs from the object in terms of gray value appearance.

2. *Center point definition*: Specification of an arbitrary point $[x_C, y_C]$.
3. *R-table calculation*: For each detected contour point, remember its gradient angle θ_i and the distance vector to the center $[x_{R,i}, y_{R,i}] = [x_{T,i} - x_C, y_{T,i} - y_C]$. This model data can be stored in form of a table (often referred to as *R-Table* in the literature) where for each θ the corresponding distance vectors to the center are stored. The space of θ (usually ranging from -180° to 180°) is quantized into equally sized intervals. If, for example, each R-table entry covers an interval of 1° , it consists of 360 entries altogether. Note that multiple distance vectors can belong to a single gradient angle θ if multiple contour points exhibit the same θ . Figure 3.1 gives an example for three arbitrarily chosen contour points and the corresponding fraction of the R-table.

3.3.3 Recognition Phase

1. *Object contour point detection*: find all contour points $\mathbf{x}_{S,i} = [x_{S,i}, y_{S,i}]$ in a scene image and their gradient angle θ_i , in general by applying the same method as during training, e.g., the canny edge detector with non-maximum suppression.
2. *Derivation of assumed center points and voting*: For each detected contour point $\mathbf{x}_{S,i}$, assumed center points $\mathbf{x}_{CE,l} = [x_{CE,l}, y_{CE,l}]$ can be calculated considering the gradient angle θ_i and the model data:

$$\begin{bmatrix} x_{CE,l} \\ y_{CE,l} \end{bmatrix} = \begin{bmatrix} x_{S,i} \\ y_{S,i} \end{bmatrix} + s \cdot \begin{bmatrix} x_{R,l} \\ y_{R,l} \end{bmatrix} (\theta_i) \quad (3.5)$$

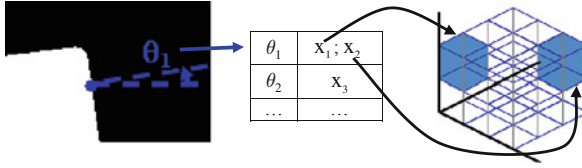


Fig. 3.2 Illustrating the voting process for one object contour point. In this example two entries can be found in the R-table for the current angle θ_l . Consequently, the content of two accumulator cells (marked *blue*) is increased

The distance vectors $\mathbf{x}_{R,l} = [x_{R,l}, y_{R,l}] (\theta)$ are obtained based on the R-table information: For each gradient angle θ a list \mathbf{r}_θ consisting of L distance vectors $\mathbf{x}_{R,l}$ with $l \in [1, \dots, L]$ can be retrieved by a look-up operation in the R-table. For each distance vector a specific $\mathbf{x}_{CE,l}$ is calculated and the corresponding cell in a 2D accumulator array is increased by one (cf. Fig. 3.2). Every cell represents a certain range of x and y position values. The selection of the cell size is a trade-off between accuracy and, on the other hand, memory demand as well as algorithm runtime. s denotes a scale factor. Typically, s is varied in a certain range and with a certain step size depending on the expected scale variation of the object and the desired accuracy in scale determination. For each s another assumed center point can be calculated. Hence, for each R-table entry multiple center points are calculated and therefore multiple accumulator cells are increased. This can be done in a 2D accumulator or alternatively in a 3D accumulator where the third dimension represents the scale (see right part of Fig. 3.1). In most of the cases, also the object angle ϕ is unknown. Therefore, again multiple center points can be determined according to

$$\begin{bmatrix} x_{CE,l} \\ y_{CE,l} \end{bmatrix} = \begin{bmatrix} x_{S,i} \\ y_{S,i} \end{bmatrix} + s \cdot \begin{bmatrix} x_{R,l} \\ y_{R,l} \end{bmatrix} (\theta_i + \phi) \quad (3.6)$$

where ϕ is varied within a certain range and with a certain step size. Please note that the distance vectors retrieved from the R-table have to be rotated by ϕ in that case. The object angle ϕ represents a fourth dimension of the accumulator.

3. *Maximum search in the accumulator*: All local maxima above a threshold t are found poses of the object.

3.3.4 Pseudocode

```
function findAllObjectLocationsGHT (in Image  $I$ , in R-Table
data  $\mathbf{R}$ , in threshold  $t$ , out position list  $\mathbf{p}$ )
```

```
// calculate edge point information
calculate  $x$  and  $y$  gradient of  $I$ :  $I_x$  and  $I_y$ 
detect all edge points  $\mathbf{x}_{S,i}$  based on  $I_x$  and  $I_y$ , e.g. by
```

```

non-maximum suppression and hysteresis thresholding(Canny)
for i=1 to number of edge points
     $\theta_i \leftarrow \arctan(I_y/I_x)$  // edge point orientation
next

//voting
init 4-dimensional accumulator accu with borders
 $[x_{\min}, x_{\max}, x_{\text{Step}}], [y_{\min}, y_{\max}, y_{\text{Step}}], [\phi_{\min}, \phi_{\max}, \phi_{\text{Step}}], [s_{\min}, s_{\max}, s_{\text{Step}}]$ 
for i=1 to number of edge points
    for  $\phi = \phi_{\min}$  to  $\phi_{\max}$  step  $\phi_{\text{Step}}$ 
        retrieve list  $\mathbf{r}_{\phi+\theta_i}$  of  $\mathbf{R}$  (entry at position  $\phi + \theta_i$ )
        for  $l=1$  to number of entries of list  $\mathbf{r}_{\phi+\theta_i}$ 
            for  $s = s_{\min}$  to  $s_{\max}$  step  $s_{\text{Step}}$ 
                 $\mathbf{x}_{R,l} \leftarrow$  distance vector (list entry  $r_{\phi+\theta_i,l}$ ),
                rotated by  $\phi$  and scaled by  $s$ 
                calculate assumed object center point
                 $\mathbf{x}_{CE,l}(\mathbf{x}_{R,l}, \mathbf{x}_{S,i}, s, \phi)$  according to Equation 3.6
                if  $\mathbf{x}_{CE,l}$  is inside  $[x,y]$  -pos. bounds of accu then
                    increment accu at position  $[x_{CE,l}, y_{CE,l}, \phi, s]$ 
                end if
            next
        next
    next
next
next

// determine all valid object positions
find all local maxima in accu
for i=1 to number of local maxima
    if  $accu(x_i, y_i, \phi_i, s_i) \geq t$  then
        append position  $[x_i, y_i, \phi_i, s_i]$  to  $\mathbf{p}$ 
    end if
next




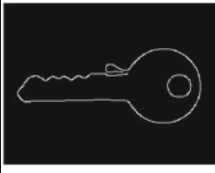
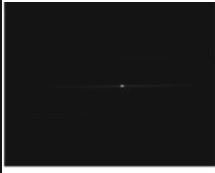
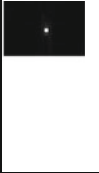

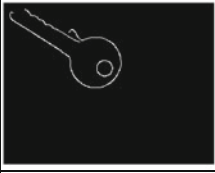



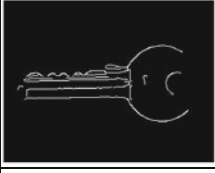

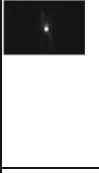
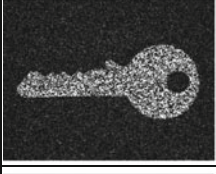
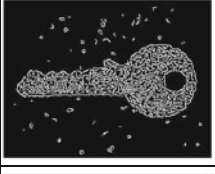
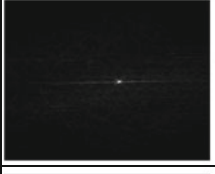




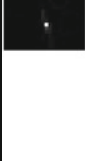
```

3.3.5 Example

Table 3.2 illustrates the performance of the generalized Hough transform: as an example application, the pose of a key (x , y , scale, and rotation) has to be determined. The image shown in the left column of the top row served as training image for R-table construction in all cases.

The left column shows scene images where the key has to be detected, whereas the contour points extracted by a Canny edge detector with non-maximum suppression are depicted right to it. In the rightmost two columns two cuts through the 4D accumulator at the found position are shown; one cut revealing the $[x, y]$ -subspace of the accumulator at the found $[s, \phi]$ -position and another revealing the $[s, \phi]$ -subspace of the accumulator at the found $[x, y]$ -position.

Table 3.2 Performance of the generalized Hough transform in the case of the unaffected training image, when the object undergoes a similarity transform, in the presence of illumination change, noise, and occlusion (from top to bottom)

Scene image	Edge image extracted from scene image with the Canny edge detector	XY accu 	RS accu 
			
			
			
			
			

The x and y accumulator size are the image dimensions (cell size 2 pixels), the scale ranges from 0.6 to 1.4 (cell size 0.05) and the rotation from -36° to 36° (cell size 3°). The following examples are shown: same image for training and recognition, a similarity-transformed object, illumination change, strong noise, and finally occlusion (from top to bottom). In all examples the accumulator maximum is sharp and distinctive, which indicates good recognition reliability.

Please note that the accumulator maximum position remains stable in case of illumination change, noise, and occlusion despite a considerable appearance change of the object caused by these effects. As far as the similarity transform example is concerned, the accumulator maximum moves to the correct position (left and upward in the XY accu; to the extreme top and right position in the RS accu) and remains sharp. However, runtime of the algorithm is rather high, even for relative small images of size 300×230 pixels (in the order of 1 s on a 3.2 GHz Pentium 4).

3.3.6 Rating

The main advantage of the generalized Hough transform is that it can compensate for occlusion and data outliers (as demonstrated by the key example) as there should be enough contour pixels left which vote for the correct object pose. On the other hand, however, the accumulator size strongly depends on the dimensionality of the search space and the envisaged accuracy.

Let's consider an example with typical tolerances and resolutions: x -/ y -translation tolerance 200 pixel, cell resolution 1 pixel, rotation tolerance 360° , cell resolution 1° , scale tolerance 50–200%, cell resolution 1%. As a consequence, the 4D accumulator size amounts to $200 \times 200 \times 360 \times 150 = 2.16 \times 10^9$ cells, leading to probably infeasible memory demand as well as long execution times due to the time-consuming maximum search within the accumulator. Therefore, modifications of the scheme exist which try to optimize the maximum search in the accumulator.

Another disadvantage is that the rather rigid object representation does only allow for a limited amount of local object deformations. If the deformation is restricted to minor parts of the object contour, the method is robust to these outliers, but if large parts of the shape show minor deformations the accumulator maximum might be split up into multiple maxima at similar poses. Noise can be another reason for such a split-up. This fact can be alleviated by choosing a coarse accumulator resolution. Then every accumulator cell covers a larger parameter range, and therefore a boundary point at a slightly different location often still contributes to the same accumulator cells. But keep in mind that the price we must pay is a reduction of accuracy!

There exist numerous applets in the Internet which are very suitable for experimenting with and learning more about the Hough transform, its performance, and limitations. The interested reader is encouraged to check it out.¹

¹See e.g. <http://d0server1.fnal.gov/users/ngraf/Talks/UTeV/Java/Circles.html> or <http://homepages.inf.ed.ac.uk/rbf/HIPR2/houghdemo.htm> (links active January 13th 2010)

3.3.7 Modifications

Even if the generalized Hough transform suffers from its high memory demand and complexity, due to its robustness with respect to occlusion and large outliers the usage of the Hough transform as a pre-processing step providing input for other schemes which actually determine the final pose is an interesting combination. In that case rather large accumulator cell sizes are chosen as only coarse pose estimates are necessary. This involves low or at least moderate memory and time demand as well as considerable tolerance with respect to local deformations.

A possible combination might consist of the GHT and so-called active contour models (see Chapter 6): contrary to the Hough transform, approaches aiming at compensating local deformations by finding exact object contours with the help of local information (e.g., like Snakes) only have a limited convergence area and therefore demand a reliable rough estimate of the object pose as input. Hence, advantages of both approaches can be combined (see, e.g., the method proposed by Ecabert and Thiran [5]).

In order to overcome the memory demand as well as speed limitations of the generalized Hough transform, Ulrich et al. [11] suggest a hierarchical approach utilizing image pyramids for determining the x , y and ϕ position of an object. According to their approach, the pyramids are built for the training as well as the scene image. On the top pyramid level, a conventional GHT is performed yielding coarse positions which are refined or rejected at lower levels. Therefore, a scan of the complete transformation space has only to be performed at top level, where quantization can be chosen very coarse which is beneficial in terms of memory demand. The knowledge obtained in this step helps to speed up the computation as well. It can be exploited as follows:

- *Accumulator size reduction*: as only parts of the transformation space close to the coarse positions have to be examined, the size of the accumulator can be kept small despite of the finer quantization.
- *Limitation of image region*: based on the coarse position and its estimated uncertainties, the image region for gradient orientation calculation can be restricted efficiently.
- *Accelerated voting*: as the object rotation ϕ is already approximately known, look-up in the R-table can be restricted to very few rotation steps.

Ulrich et al. implemented a strategy incorporating separate R-tables for each pyramid level and possible object rotation. This involves an increased memory demand for the model, but they showed that this is overcompensated by the reduction of accumulator size as well as runtime. Both can be reduced by several orders of magnitude compared to the standard scheme. In a comparative study Ulrich and Steger [10] showed that a GHT modified in such a way can compete with other recognition schemes which are widely used in industrial applications.

3.4 The Hausdorff Distance

3.4.1 Basic Approach

3.4.1.1 Main Idea

The Hausdorff distance H is a nonlinear metric for the proximity of the points between two point sets. When applied in object recognition, one point set M represents the model whereas the second, I , describes the content of a scene image region. H can be used as a measure of similarity between the image content in the vicinity of a given position and the model. If H is calculated for multiple positions it is possible to determine the location of an object. The absolute value of H indicates whether the object is present at all. H is defined by

$$H(M, I) = \max(h(M, I), h(I, M)) \text{ with} \tag{3.7}$$

$$h(M, I) = \max_{m \in M} \left(\min_{i \in I} \|m - i\| \right) \text{ and } h(I, M) = \max_{i \in I} \left(\min_{m \in M} \|i - m\| \right) \tag{3.8}$$

where $\|\cdot\|$ denotes some kind of distance norm between a model point m and an image point i , e.g., the Euclidean distance norm. $h(M, I)$ is called forward distance and can be determined by calculating the distance to the nearest point of I for each point of M and taking the maximum of these distances. $h(M, I)$ is small exactly when every point of M is located in the vicinity of some point of I .

$h(I, M)$ (the reverse distance) is calculated by evaluating the distance to the nearest point of M for each point of I and taking the maximum again. $h(I, M)$ is small exactly when every point of I is located in the vicinity of some point of M . Finally, H is calculated by taking the maximum of these two values.

Figure 3.3 should make things clear. The proceeding of calculating the forward distance is shown in the top row: at first, for each model point (marked green)

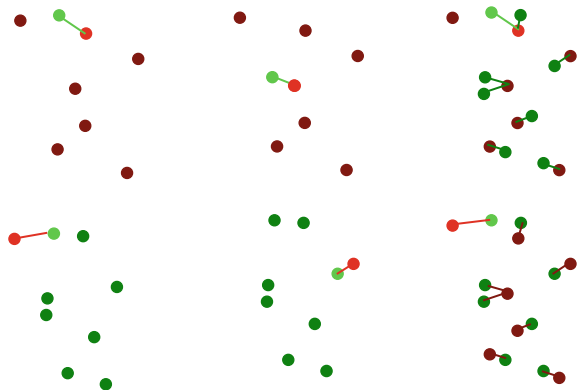


Fig. 3.3 Illustrating the process of calculating the Hausdorff distance (model points are marked green, image points red)

the nearest image point (marked red) is searched. This is explicitly shown for two model points in the two leftmost point sets (the thus established correspondences are marked by bright colors). After that, the forward distance is set to the maximum of these distances (shown in the right part; marked light green). In the bottom row the calculation of the inverse distance can be seen: for each image point the nearest model point is detected (illustrated for two example image points marked light in the leftmost two columns). Subsequently, the inverse distance is set to the maximum of these distances (marked light red). Finally, the Hausdorff distance is set to the maximum of the forward and reverse distance.

Please note that, in general, forward and inverse distance are not equal: $h(M, I) \neq h(I, M)$. In fact, this is also true for our example as one of the correspondences established during calculation of the forward distance differs from the correspondence of the reverse distance (see the upper left areas of the point sets depicted in the right part of Fig. 3.3, where correspondences are indicated by red and green lines).

The Hausdorff distance has the desirable property that the total number of model points and the total number of image points don't have to be identical, because multiple image points i can be matched to a single model point m and vice versa. Hence, a reliable calculation of the metric is still possible if the number of model points differs from the number of image points, which usually is the case in real-world applications.

For the purpose of object detection the directed Hausdorff distances have to be adjusted to the so-called partial distances $h^{f_F}(M, I)$ and $h^{f_R}(I, M)$. Imagine an image point set where one point, which is located far away from the other points, is caused by clutter. This would result in a large value of $h(I, M)$, which is obviously not intended. Respectively, an isolated point of M would produce large values of $h(M, I)$ if it is not visible in the image due to occlusion (see Table 3.3 for an illustration).

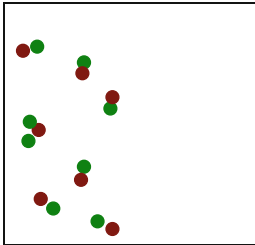
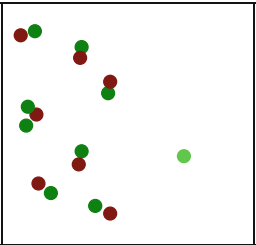
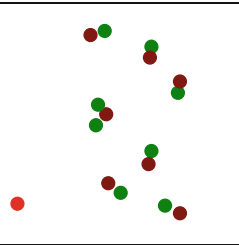
Such a behavior can be circumvented by taking the k -largest value instead of the maximum during the calculation of the directed distances $h(I, M)$ and $h(M, I)$. We can define f_F as the fraction of model points which need to have a nearest distance below the value of $h^{f_F}(M, I)$ which is finally reported. $h^{f_F}(M, I)$ is called the partial directed forward distance. If for example $f_F = 0.7$ and the model consists of 10 points, their minimum distances to the image point set can be sorted in ascending order and $h^{f_F}(M, I)$ is set to the distance value of the seventh model point. For $f_F = 1$ the partial distance $h^{f_F}(M, I)$ becomes equal to $h(M, I)$. A respective definition of f_R exists for $h^{f_R}(I, M)$.

As a consequence, it is possible to control the amount of occlusion which should be tolerated by the recognition system with the choice of f_F . The parameter f_R controls the amount of clutter to be tolerated, respectively.

3.4.1.2 Recognition Phase

Rucklidge [9] proposes to utilize the Hausdorff distance as a metric which indicates the presence of searched objects. He suggests to scan a 6D transformation space in

Table 3.3 Illustrating the problems due to occlusion and clutter which can be solved by the introduction of the partial Hausdorff distance measures

		
<p>Model point set (dark green) and image point set (dark red) that match well. H is small. Please note that the number of model points is not equal to the number of image points</p>	<p>The same situation, but with an extra model point (light green) which is not detected in the image, e.g., due to occlusion. H is large because of the forward distance.</p>	<p>Now the image point set contains an extra point (light red), e.g., due to clutter. H is large because of the reverse distance.</p>

order to determine the parameters of an affine transformation. To this end, the transformation space is sampled and for every sampled position, which consists of six specific parameter values and defines a specific transformation t , the partial forward and reverse Hausdorff distances $h_t^{fF}(t(M), I)$ and $h_t^{fR}(I, t(M))$ with respect to the transformed model points $t(M)$ are calculated. Valid object positions are reported for transformation parameters where the Hausdorff distance reaches local minima. Additionally, the distances have to remain below user defined thresholds τ_F and τ_R :

$$h_t^{fF}(t(M), I) < \tau_F \wedge h_t^{fR}(I, t(M)) < \tau_R \quad (3.9)$$

Hence the search can be controlled with the four parameters τ_F , τ_R , f_F , and f_R .

Each dimension of the transformation space (defined by one of the six transformation parameters) is sampled equally spaced with a step size such that the resulting position difference of each transformed model point between two adjacent parameter values t_k and t_{k+1} does not exceed the size of one pixel: $|t_k(m) - t_{k+1}(m)| \leq 1 \text{ pixel} \forall m \in M$. Additionally, for each transformation t the transformed model points $t(m)$ are rounded to integer positions for speed reasons (see below). As a result, no sub-pixel accuracy can be achieved, but a finer sampling and/or the abdication of rounding would be prohibitive in terms of runtime. However, there is still much demand for an acceleration of the search, which is until now still brute-force, in order to reduce runtime. To this end, Rucklidge [9] suggests the following modifications:

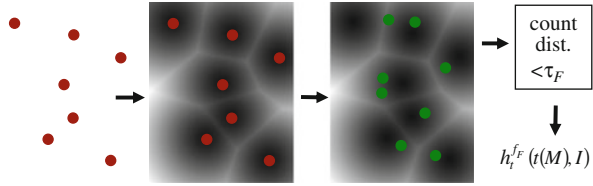
- *Size restriction of the search space:* The space of transformations which are reasonable can be restricted by applying constraints. First, all transformed model

points $t(m)$ have to be located within the borders of the scene image under investigation. Additionally, in many applications a priori knowledge can be exploited, e.g., the scale and/or rotation of the object to be found have to remain within some rather narrow tolerances.

- *Box-reverse distance*: Usually, the objects to be located only cover a rather small fraction of the search image. Therefore only points located in a box $[x_{\min} \dots x_{\max}, y_{\min} \dots y_{\max}]$ have to be considered when calculating the reverse distance at a given position.
- *Optimization of calculation order*: A speedup due to rearranging the calculations can be achieved in two ways:
 - For most of the positions, the distances will not meet the threshold criteria. Therefore it is worthwhile to calculate only the forward distance at first and then to check whether its value is below τ_F . Only if this criterion is met, the reverse distance has to be calculated, because otherwise the current transformation has to be rejected anyway regardless of the reverse distance.
 - Furthermore, with a modification of the partial distance calculation it is often possible to stop the calculation with only a fraction of the points being examined. Let's consider the forward distance: instead of calculating the minimum distance to the image point set of every transformed model point $t(m)$ and then evaluating whether the distance at the f_F -quantile is below τ_F , it is better to count the number of model points which have a minimum distance to the image point set that remains below τ_F . The final check is then whether this number reaches the fraction f_F of the total number of model points. This enables us to stop the evaluation at a point where it has become clear that f_F cannot be reached any longer: this is the case when the number of model points, which are already checked and have a distance above τ_F , exceeds the fraction $1 - f_F$ with respect to the number of model points.
- *Usage of the so-called distance transform*: If the transformed model points are rounded to integer positions, it is likely that the same position results for different model points and transformations, i.e., $t_1(m_a) = t_2(m_b)$. Therefore – when evaluating the forward distance – it might be worthwhile to remember the minimum distance to the image point set at a specific position $t(m)$: provided that the already calculated distance for $t_1(m_a)$ has been stored, the distance for $t_2(m_b)$ can be set to the stored distance of $t_1(m_a)$ immediately. It can be shown that an even more efficient way is to perform a calculation of the minimum distance to the image point set for every pixel of the scene image prior to the Hausdorff distance calculation, because then information can be re-used for adjacent pixels. This is done by calculating the so-called *distance transform* $\Delta[x, y]$, which specifies the minimum distance of position $[x, y]$ to the image point set, and is defined by

$$\Delta[x, y] = \min_{i \in I} \|[x, y] - i\| \quad (3.10)$$

Fig. 3.4 Illustrating the usage of the distance transform (depicted as grayscale image) in order to speedup calculation of the forward distance



As a consequence, each $t(m)$ only probes $\Delta[x, y]$ during the calculation of $h_t^{fF}(t(M), I)$. Figure 3.4 illustrates the proceeding: $\Delta[x, y]$ can be derived from the image point set (red points) in a pre-processing step. Dark values indicate low distances to the nearest image point. A model point set (green points) is superimposed according to the transformation t currently under investigation and the forward distance can be calculated very fast. Besides, this speedup is the reason why the forward distance is calculated first: a similar distance transform for the reverse distance would depend on the transformation t , which complicates its calculation prior to the search.

- *Recursive scan order*: Probably the most important modification is to apply a recursive coarse-to-fine approach, which allows for a significant reduction of the number of transformations that have to be checked in most cases. Roughly speaking, the transformation space is processed by dividing it recursively into equally spaced cells. In the first recursion step the cell size s is large. For each cell it is evaluated with the help of a quick check whether the cell possibly contains transformations with a Hausdorff distance below the thresholds. In that case the cell is labeled as “interesting.” Only those cells are recursively split into sub-cells, which are again evaluated, and so on. The recursion stops either if the cell cannot contain Hausdorff distances meeting the criteria for a valid object location or if the cell reaches pixel-size. The quick evaluation of a cell containing many transformations is based on the observation that the distance transform $\Delta[x, y]$ decreases at most by 1 between adjacent pixels. Hence, many transformations with similar parameters can be ruled out if $\Delta[x, y]$ is large for a certain parameter setting. For details the interested reader is referred to [9].

3.4.1.3 Pseudocode

```
function findAllObjectLocationsHausdorffDist (in scene image
S, in model point set M, in thresholds  $f_F$ ,  $f_R$ ,  $\tau_F$ ,  $\tau_R$ , out position list p)

// calculate edge point information
detect all edge points e based on image gradients, e.g. by
non-maximum suppression and hysteresis thresholding (Canny)
set scene image point set I to locations of edge points e
```

```

// preprocessing: calculate distance transform
for y = 1 to height (S)
  for x = 1 to width (S)
    calculate  $\Delta[x,y]$  (Equation 3.10)
  next
next

// scanning of transformation space
s  $\leftarrow$  sstart // set cell size to start size (coarse level)
while s  $\geq$  1 do
  set the step sizes of the six transformation parameters
  such that one step causes at most s pixels position diff.
  // sampling-loop (actually six nested for-loops)
  // use step sizes just derived
  for each possible transformation cell (t's within bounds)
    if current cell of transformation space has not been
    rejected already then
      // evaluation of forward distance
      r  $\leftarrow$  0 // initialize number of rejected points
      while unprocessed model points exist do
        m  $\leftarrow$  next unprocessed point of M
        if  $\Delta[t(m)] > \tau_F$  then
          r  $\leftarrow$  r + 1 // reject current point
          if  $r/N_M > 1 - f_F$  then
            r  $\leftarrow$  NM // forward distance too high
            break // abort while-loop through all m
          end if
        end if
      end while
      if  $r/N_M > 1 - f_F$  then
        mark current cell defined by t as "rejected "
      else
        // forward-dist. ok -> evaluate reverse dist.
        calculate reverse distance  $h_t^{FR}(I, t(M))$ 
        if  $h_t^{FR}(I, t(M)) > \tau_R$  then
          mark current cell defined by t as "rejected"
        else if s == 1 then
          // finest sampling level-> object found
          append position defined by t to p
        end if
      end if
    end if
  end if
  adjust cell size s // e.g. multiplication with 0.5
end while

```



```
// post-processing
merge all adjacent positions in p such that only local
minima of the hausdorff distance are reported
```

3.4.1.4 Example

Two examples where the Hausdorff distance is used for object recognition are shown in Figs. 3.5 and 3.6. In each case the objects to be found have been undergone a perspective transformation. For each example the model point set is shown in part (a), followed by the scene image where the object has been undergone a projective transformation (b), the point set extracted from the scene image by edge detection (c) and, finally, all recognized instances in the scene image overlaid on the scene image point set in bold black (d). The point sets are set to edge pixels, which can be extracted, e.g., with the Canny detector including non-maximum suppression (cf. [3]).

It can be seen that all instances are recognized correctly, even in the presence of clutter and partial occlusion (second example). The computational complexity is very high, because of the rather high dimensionality of the transformation space (six dimensions, as the parameters of an affine transformation are estimated) as well as the cardinality of the sets (in the order of magnitude of 1,000 points for the model

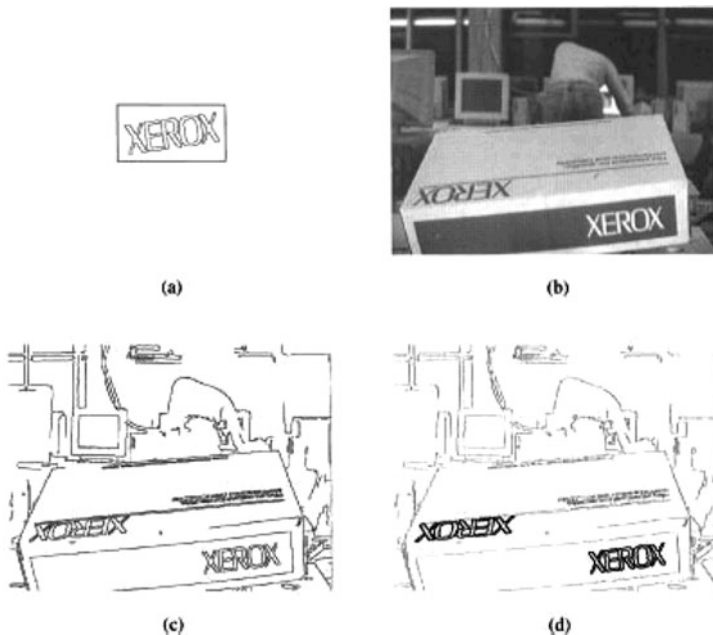


Fig. 3.5 Taken from Rucklidge [9]² showing an example of the detection of a logo

² With kind permission from Springer Science+Business Media: Rucklidge [9], Fig. 1, © 1997 Springer.

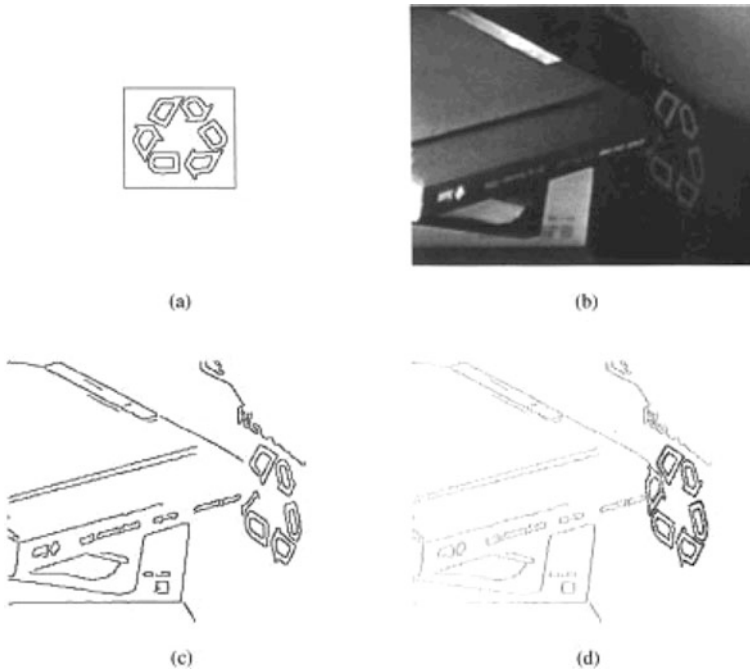


Fig. 3.6 Taken from Rucklidge [9]³ showing another example of the detection of a logo with partial occlusion

and 10,000 points for the scene image). As a consequence, Rucklidge reported execution times in the order of several minutes for both examples. Even if the used hardware nowadays is out of date, the method seems infeasible for industrial applications. But the method is not restricted to low-level features like edge pixels which occur in large numbers. When high-level features like line segments are used, the number of points in the point sets can be reduced significantly.

3.4.1.5 Rating

Due to the flexible object model consisting of an arbitrary point set a large range of objects, including complex shaped objects, can be handled with this method. The model generation process, which, e.g., extracts all points with gradient above a certain threshold from a training image, imposes no a priori constraints about the object appearance. Moreover, because of the usage of partial distances the method is robust to occlusion and clutter.

On the other hand, the gradient threshold also implies a dependency on the illumination: if the image contrast is reduced in the scene image, some points might be missed which are contained in the model. Additionally, the method tends to be

³With kind permission from Springer Science+Business Media: Rucklidge [9], Fig. 2, © 1997 Springer.

sensitive to erroneously detect object instances in image regions which are densely populated with pixels featuring a high gradient (“false alarms”). Therefore, Ulrich and Steger [10] reported the robustness inferior to other methods in a comparative study. In spite of the significant speedup when searching the transformation space, the scheme still is very slow. One reason is that no hierarchical search in the image domain is applied. Finally, the method doesn’t achieve sub-pixel accuracy.

3.4.2 Variants

3.4.2.1 Variant 1: Generalized Hausdorff Distance

In addition to the gradient magnitude, most edge detection schemes determine gradient orientation information as well. As the object recognition scheme utilizing the Hausdorff distance discussed so far doesn’t use this information, Olson and Huttenlocher [8] suggested a modification of the Hausdorff distance which is applicable to sets of edge pixels and also considers orientation information of the edge points. For example, the thus obtained generalized forward Hausdorff distance $h_a(M, I)$ is defined as

$$h_a(M, I) = \max_{m \in M} \min_{i \in I} \max \left(\left\| \begin{bmatrix} m_x - i_x \\ m_y - i_y \end{bmatrix} \right\|, \frac{|m_\phi - i_\phi|}{a} \right) \quad (3.11)$$

The original $h(M, I)$ serves as a lower bound for the new measure, i.e., the gradient orientation difference $m_\phi - i_\phi$ between a model and an image point is considered as a second measure and the maximum of these two values is taken for the calculation of $h_a(M, I)$. The parameter a acts as a regularization term which enables us to compare location and orientation differences directly. For a robust detection the f_F -quantile instead of the maximum distance of all points $m \in M$ can also be used.

Please note that the distance transform $\Delta[x, y]$ now becomes a 3D function, with the additional dimension characterizing the distance measure evolving from orientation differences. An elegant way of considering this fact is to use separate models for each possible rotation step of the object to be found, e.g., by successively rotating the object model with a certain step size.

According to Olson and Huttenlocher [8], the additional consideration of the orientation information leads to a significantly decreased false alarm rate, especially in densely populated image regions. Interestingly enough, they also reported a considerable acceleration of the method as fewer transformations have to be checked during the search.

3.4.2.2 Variant 2: 3D Hausdorff Distance

Another suggestion made by Olson and Huttenlocher is to extend the method to a recognition scheme for 3D objects undergoing projective transformation. To this end, each object is represented by multiple models characterizing its shape from a specific viewpoint. Each model is obtained by rotating a sample of the object in 3D space with a certain step size. The recognition phase is done by calculation of the Hausdorff distance to each model.

An observation which can be exploited for accelerating the scheme is that at least a portion of the models should be very similar with respect to each other. To this end, the models are clustered hierarchically in a tree structure during the training phase. Each model is represented by a leaf at the bottom tree level. In higher levels, the most similar models/leaves (or, alternatively, nodes already containing grouped leaves) are grouped to nodes, with each node containing the portion of the edge points identical in the two sub-nodes/leaves. The congruence incorporated in this structure can be exploited in the recognition phase in a top-down approach: if the point (sub-)set assigned to a certain node suffices for a rejection of the current transformation, this holds for every leaf belonging to this node.

3.4.2.3 Variant 3: Chamfer Matching

A closely related approach, which is called “hierarchical chamfer matching”, has been reported by Borgefors [2]. It utilizes the average distance of all transformed model points to their nearest image point as a distance measure instead of a quantile. For a rapid evaluation a distance transform is used, too. There exists a fast sequential way of calculating the distance transform of a scene image point set by passing the image only twice. Sequential distance transforms are known as “chamfer distances” explaining the name of the method. The search of the transformation space is not done by brute force; instead, the algorithm relies on reasonable initial position estimates which are refined by iterative optimization.

Speedup is achieved by employing a hierarchical search strategy, where an image pyramid of the edge image of the scene (edge pixels are used as point sets) is built. A distance transform can be computed for each level of the pyramid. As the start image is a binary image, averaging or smoothing operations for calculating the higher levels of the pyramid obviously don’t work. Therefore a logical “OR” operation is used when adjacent pixels are summarized for higher levels.

Compared to the Hausdorff distance, chamfer matching has the property that due to averaging occluded model points still contribute to the reported distance value if a minor part of the object is not visible. Another point is the lack of a measure comparable to the reverse Hausdorff distance: this results in an increased sensitivity to false alarms in densely populated image regions which contain many edge points.

3.5 Speedup by Rectangular Filters and Integral Images

3.5.1 Main Idea

In their article “Robust Real-time Object Detection,” Viola and Jones [12] proposed a transformation-search-based method which is optimized for computation speed. They showed that their scheme is capable to do real-time processing when applied to the task of detection of upright, frontal faces.

The method localizes instances of a single object class by applying a set of rectangular-structured filters to a query image instead of using a point set. The

filter kernels are reminiscent of Haar wavelet filters, as they can be represented by a combination of step-functions and consist of piecewise constant intervals in 2D. The input image is convolved with a set of filters at various positions and scales. Subsequently, a decision whether an object instance is present or not can be made at each position. These decisions are based on weighted combinations of the filter outputs. In other words, the $[x, y, s]$ -space is searched.

The search can be done very fast, because the specific shape of the rectangular filters allows for an extremely efficient implementation of the convolutions with the help of so-called *integral images*. Additionally, the outputs of different filters are combined in a smart way such that most of the time only a fraction of the filter set has to be calculated at a particular position. Overall, three major contributions are to be mentioned:

- *Integral images*: prior to recognition, a so-called integral image F is derived from the input image I . Roughly speaking, F contains the integrated intensities of I (details will follow). This pre-processing allows for a very rapid calculation of the filter responses, as we will see below.
- *Learning of weights*: as there are many possible instances of rectangular-shaped filter kernels, it has to be decided which ones to use and how to weight the individual outputs of the chosen filters. These questions are answered by a modified version of the AdaBoost algorithm proposed by Freund and Schapire [6], which learns the weights of the filters from a set of sample images in a training phase. The weighting favors filters that perform best if a single filter is utilized for object detection.
- *Cascaded classifying*: for speed reasons, not the complete filter set is applied to every position and scale. Instead, only a small subset of the filters searches the complete transformation space. Just promising areas, where a simple classifier based on these few filter responses reports possible object locations, are examined further by larger subsets, which are used to refine the initial estimate in those areas, and so on. This proceeding enables us to sort out large regions of I , which are very likely to be background, very quickly.

3.5.2 Filters and Integral Images

The filter kernels used by Viola and Jones [12] exhibit a rectangular structure and consist of two to four sub-rectangles. Some examples can be seen in Fig. 3.7. The filter output f_i of the convolution of an input image I with such a kernel k_i is defined by the sum of the intensities of I which are covered by the white areas minus the sum of intensities covered by the black areas.



Fig. 3.7 Examples of filter kernels utilized by Viola and Jones

Hence, the filter is well suited for rectangular-structured objects and yields high responses for object areas with a partitioning similar to the filter kernel. Different scales during search can be covered by different kernel sizes.

Overall, a great multitude of combinations of two to four sub-rectangles are possible. Note that the kernel center position, which is set arbitrarily by definition, can be shifted with respect to the actual center of the filter structure. Therefore multiple kernels with identical configurations of sub-rectangles exist.

The learning algorithm presented below has to choose the most promising filter configurations for the recognition phase. In order to make this task feasible the variety of kernels can be restricted, e.g., by considering only sub-rectangles of equal size, limiting the number of overall kernel sizes, or considering only small shifts or shifts spaced at a rather large step size.

The so-called integral image F is specified as follows: its value at position $[x, y]$ is defined by the sum of intensities of I considering all pixels located inside the rectangular area ranging from $[0, 0]$ up to and including $[x, y]$:

$$F(x, y) = \sum_{a=0}^x \sum_{b=0}^y I(a, b) \quad (3.12)$$

An example can be seen in Fig. 3.8, where the integral image is calculated for a simple cross-shaped object.

The integral image F can be calculated in a pre-processing stage prior to recognition in a recursive manner in just one pass over the original image I as follows:

$$R(x, y) = R(x, y - 1) + I(x, y) \quad (3.13a)$$

$$F(x, y) = F(x - 1, y) + R(x, y) \quad (3.13b)$$

where $R(x, y)$ denotes the cumulative row sum. R and F are initialized by $R(x, -1) = 0$ and $F(-1, y) = 0$.

By usage of F a very fast calculation of the convolution of I with one of the rectangular filter kernels is possible, because now the sum of intensities of a rectangular area ranging from $[x_0, y_0]$ to $[x_1, y_1]$ can be calculated by just considering the values

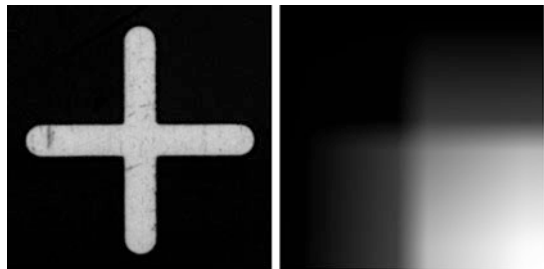


Fig. 3.8 Example of an integral image (*right*) of a cross-shaped object (*left*)

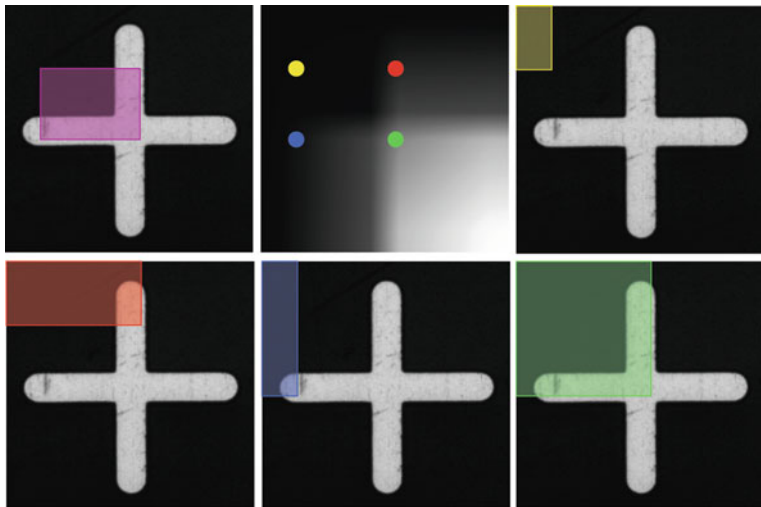


Fig. 3.9 Exemplifying the calculation of the sum of intensities in a rectangular region with the help of integral images

of F at the four corner points of the region instead of summing up the intensities of all pixels inside:

$$\sum_{a=x_0}^{x_1} \sum_{b=y_0}^{y_1} I(a, b) = F(x_1, y_1) - F(x_0, y_1) - F(x_1, y_0) + F(x_0, y_0) \quad (3.14)$$

Figure 3.9 illustrates the proceeding: In order to calculate the intensity sum of the purple region shown in the top left image, just four values of F have to be considered (as stated in Equation 3.14). This is shown in the top middle image, where the four corner points of the region are overlaid in color upon the integral image. The value of $F(x_0, y_0)$ defines the sum of intensities of the area marked yellow (as indicated in the top right image), $F(x_1, y_0)$ the intensity sum of the red, $F(x_0, y_1)$ the intensity sum of the blue, and $F(x_1, y_1)$ the intensity sum of the green area, respectively (cf. the images in the bottom row). As a consequence, the intensity sum of any rectangular-shaped area can be calculated by considering as few as four values of F , regardless of its size. This allows for an extremely fast implementation of a convolution with one of the rectangular-shaped filter kernels describe above.

3.5.3 Classification

If multiple filter kernels are applied at a specific position, the question is how to combine their outputs in order to decide whether an instance of the object is present at this particular position or not. To this end, a so-called linear classifier cl is chosen:

its output is set to 1 (indicating that an instance is present) if a weighted combination of binarized filter outputs b_t (which is a classification in itself by thresholding the “original” filter outputs f_t) is larger than a threshold, otherwise the output is set to 0:

$$cl(x, y, s) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t \cdot b_t(x, y, s) \geq 1/2 \cdot \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases} \quad (3.15)$$

where the α_t denotes the weights of the particular filters. Details of linear classification can be found in Appendix B.

Now it is also clear why shifted kernels with identical sub-rectangle configuration are used: it’s because filters responding to different parts of the object should contribute to the same object position.

In order to formulate such a classifier, the two tasks of selecting the filter kernels k_t and determining their weights α_t are solved in a training step with the help of a set of positive as well as negative training samples (i.e., images where the object is not present). To this end, Viola and Jones [12] suggest an adapted version of the AdaBoost algorithm (see Freund and Schapire [6]), where so-called “weak classifiers” (which show relatively high error rates, but are simple and fast) are combined to a so-called “strong classifier”. This combination, which can be a weighted sum, enables the strong classifier to perform much better (“boost” its performance) compared to each of the weak classifiers.

The learning of the weights α_t and the selection of the kernels k_t is done in T rounds of learning. At each round $t = 1, \dots, T$ one kernel k_t is selected. To this end, a classification of the training images is done for each filter kernel k_t based on its binarized output b_t . As the correct classification is known for every training image, an error rate ε_t can be calculated for each b_t . The kernel with the lowest error rate is chosen as k_t and its weight α_t is adjusted to this error rate (low ε_t lead to high α_t).

As training proceeds, the training images themselves are also weighted: if the weak classifier based on k_t misclassifies an image, its weight is increased; otherwise it is decreased. The error rates in the next round of training are calculated based on this weighting. This helps to find kernels that perform well for “critical images” in later rounds. Overall, all terms/factors which are necessary for applying the linear classifier as defined by Equation (3.15) are determined at the end of training.

Viola and Jones report good detection rates for classifiers consisting of approximately 200 filters for their example of detection of upright, frontal faces. With an implementation on a 700 MHz Pentium desktop computer processing a 384×288 image took approximately 0.7 s, which, however, is still too much for real-time processing.

In order to achieve a speedup, they altered the classification to a cascaded application of multiple classifiers. In the first stage, a classifier consisting of just a few filters is applied for the whole transformation space. Search in the scale space is implemented by changing the filter size. In the next stage, a second classifier, which is a bit more complex, is applied only at those $[x, y, s]$ -positions where the first one detected an instance of the object. This proceeding goes on for a fixed number

of stages, where the number of filters contained in the classifiers increases progressively. At each stage, positions, which are highly likely to be background, are sorted out. In the end, only those positions remain which are classified to contain an instance of the object to be searched.

Each classifier has to be trained separately by the boosting procedure just described. Please note that, compared to the threshold used in Equation (3.15), the decision threshold has to be set much lower as we don't want the classifier to erroneously sort out positions where an object instance is actually present. Nevertheless, much of the background can be sorted out very quickly in the first stages. Experiments by Viola and Jones revealed that a speedup of a factor of about 10 could be achieved for a 10-stage cascaded classifier with 20 filters at each stage compared to a monolithic classifier of 200 filters at comparable detection rates for the example application of face detection.

3.5.4 Pseudocode

```

function detectObjectsCascadedClassify (in Image  $I$ , in list of
linear classifiers  $cl$ , out position list  $p$ )

// calculate integral image
for  $y = 1$  to height ( $I$ )
  for  $x = 1$  to width ( $I$ )
    calculate  $F(x,y)$  according to Equation 3.13
  next
next

// cascaded classification
init 3D array  $map$  with 1's // 1: obj. present; 0: no obj.
for  $i = 1$  to number of stages
  for  $y = 1$  to height ( $I$ ) step  $yStep$ 
    for  $x = 1$  to width ( $I$ ) step  $xStep$ 
      for  $s = 1$  to  $s_{max}$  step  $sStep$ 
        if  $map(x,y,s) == 1$  then // current pos still valid
          for  $t = 1$  to nbr of kernels of current stage  $i$ 
            scale kernel  $k_{i,t}$  acc. to current scale  $s$ 
            convolve  $I$  with filter kernel  $k_{i,t}$ , use  $F$ 
          next
          // classification according to  $cl_i$ 
          if  $cl_i(k_{i,1}, \dots, k_{i,T}) == 0$  then
             $map(x,y,s) \leftarrow 0$  // mark as background
          end if
        end if
      next
    next
  next
next

```

```

    next
next

// determine all valid object positions
for y = 1 to height(I) step yStep
  for x = 1 to width(I) step xStep
    for s = 1 to smax step sStep
      if map(x,y,s)==1 then
        append position [x,y,s] to p
      end if
    next
  next
next
next

```

3.5.5 Example

Viola and Jones report experimental results for the detection of upright, frontal faces. In the training images, the faces were approximately 24×24 pixel in size. Their cascaded detector consists of 32 stages with approximately 4,300 filters used in total. As far as the detection rates are concerned, this classifier performs comparable to other state-of-the-art detectors for that task, but takes much less time.

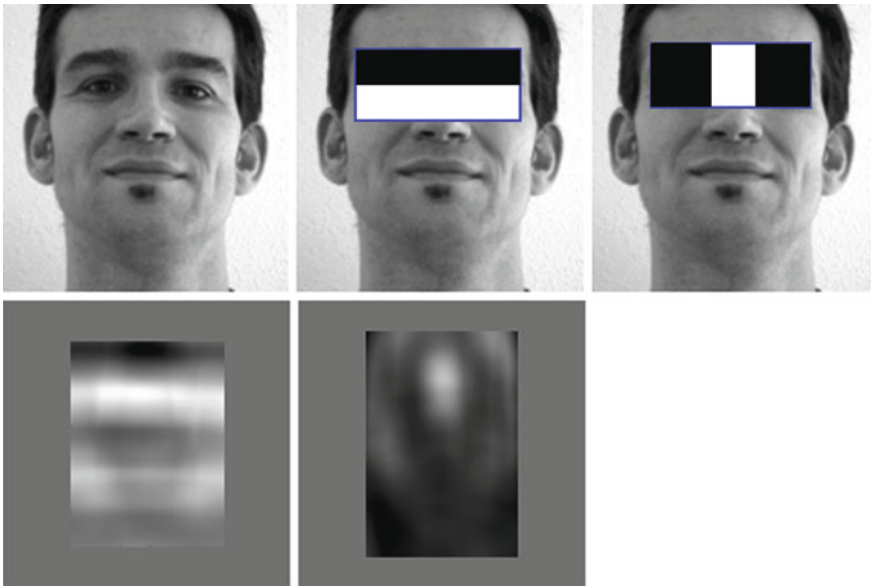


Fig. 3.10 Showing an example of the detection of an *upright*, frontal face with the filter kernels used in the first stage of the cascaded classification proposed by Viola and Jones [12]

They report processing times of about 70 ms for a 384×288 image on a 700 MHz Pentium processor.

Apparently, the combination of extremely fast filtering by integral images with the speedup through cascading works very well. In fact, the classifier used in first stage consists of as few as two filters and discards approximately 60% of the background region while almost 100% of the objects are retained at the same time.

An example can be seen in Fig. 3.10: the two filters selected by AdaBoost for the first stage relate to the facts that the eye regions typically are darker than the upper cheeks (first filter) and usually also darker than the bridge over the nose (second filter). The results of the convolution of these two filters with an example image are shown in the second row (bright areas indicate high convolution results).

3.5.6 Rating

On the positive side, in contrast to many other transformation-based schemes the method proposed by Viola and Jones is extremely fast. Real-time processing of video sequences of medium sized image frames seems possible with this method when using up-to-date hardware. Additionally, detection results for the example application of upright, frontal face recognition are comparable to state-of-the-art methods.

On the other hand, the extremely fast filtering is only possible for-rectangular-shaped filter kernels. Such a kernel structure might not be suited for some object classes. Clearly, the kernels fit best to objects showing a rectangular structure themselves. However, the authors argue that due to the extremely fast filtering a large number of filters can be applied (much larger compared to other methods using filter banks), which should contribute to alleviate such a misfitting. Another disadvantage which has to be mentioned is that the method does not explicitly account for differences of object rotation between training and recognition.

References

1. Ballard, D.H., "Generalizing the Hough Transform to Detect Arbitrary Shapes", *Pattern Recognition*, 13(2):111–122, 1981
2. Borgefors, G., "Hierarchical Chamfer Matching: A Parametric Edge Matching Algorithm", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, 1988
3. Canny, J.F., "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986
4. Duda, R.O. and Hart, P.E., "Use of the Hough Transform to Detect Lines and Curves in Pictures", *Communications of the ACM*, 1:11–15, 1972
5. Ecabert, O. and Thiran, J., "Adaptive Hough Transform for the Detection of Natural Shapes Under Weak Affine Transformations", *Pattern Recognition Letters*, 25(12):1411–1419, 2004
6. Freund, Y. and Schapire, R., "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", *Journal of Computer and System Sciences*, 55:119–139, 1997

7. Hough, P.V.C., “*Method and Means for Recognizing Complex Patterns*”, U.S. Patent No. 3069654, 1962
8. Olson, C. and Huttenlocher, D., “Automatic Target Recognition by Matching Oriented Edge Pixels”. *IEEE Transactions on Signal Processing*, 6(1):103–113, 1997
9. Rucklidge, W.J., “Efficiently locating objects using the Hausdorff distance”, *International Journal of Computer Vision*, 24(3):251–270, 1997
10. Ulrich, M. and Steger, C., “Performance Comparison of 2D Object Recognition Techniques”, *International Archives of Photogrammetry and Remote Sensing*, XXXIV(5):99–104, 2002
11. Ulrich, M., Steger, C., Baumgartner, A. and Ebner H., “Real-Time Object Recognition Using a Modified Generalized Hough Transform”, *Pattern Recognition*, 26(11):2557–2570, 2003
12. Viola, P. and Jones, M., “Robust Real-time Object Detection”, *2nd International Workshop on Statistical and Computational Theories of Vision – Modelling, Learning, Computing and Sampling*, Vancouver, 1–20, 2001