Table 2.4. Recording conditions for the example recognition applications.

Domain	Condition(s)
Image Content	Ideally, the only skin colored object in the image is the user's hand. Other skin colored objects may be visible, but they must be small compared to the hand. This also applies to the arm; it should be covered by a long-sleeved shirt if it is visible. In dynamic gestures the hand should have completely entered the image before recording starts, and not leave the image before recording ends, so that it is entirely visible in every recorded frame.
Lighting	Lighting is sufficiently diffuse so that no significant shadows are visible on the hand. Slight shadows, such as in Fig. 2.2 and Fig. 2.3, are acceptable.
Setup	The distance between hand and camera is chosen so that the hand fills approximately 10–25% of the image. The hand's exact position in the image is arbitrary, but no parts of the hand should be cropped. The camera is not rotated, i.e. its x axis is horizontal.
Camera	Resolution may vary, but should be at least 320×240. Aspect ratio remains constant. The camera is adjusted so that no overexposure and only minor color cast occurs. (Optimal performance is usually achieved when gain, brightness and shutter speed are set so that the image appears somewhat underexposed to the human observer.) For dynamic gestures, a frame rate of 25 per second is to be used, and shutter speed should be high enough to prevent motion blur. A consumer quality camera is sufficient.

such as those shown in Fig. 2.2 and 2.3. This process, which occurs within the feature extraction module in Fig. 2.1, is visualized in Fig. 2.4.

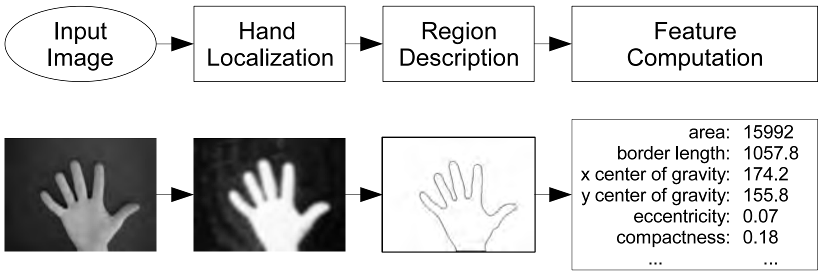


Fig. 2.4. Visualization of the feature extraction stage. The top row indicates processing steps, the bottom row shows examples for corresponding data.

2.1.2.1 Hand Localization

The identification of foreground or target regions constitutes an interpretation of the image based on knowledge which is usually specific to the application scenario. This knowledge can be encoded explicitly (as a set of rules) or implicitly (in a histogram, a neural network, etc.). Known properties of the target object, such as shape, size, or color, can be exploited. In gesture recognition, color is the most frequently used feature for hand localization since shape and size of the hand's projection in the two-dimensional image plane vary greatly. It is also the only feature explicitly stored in the image.

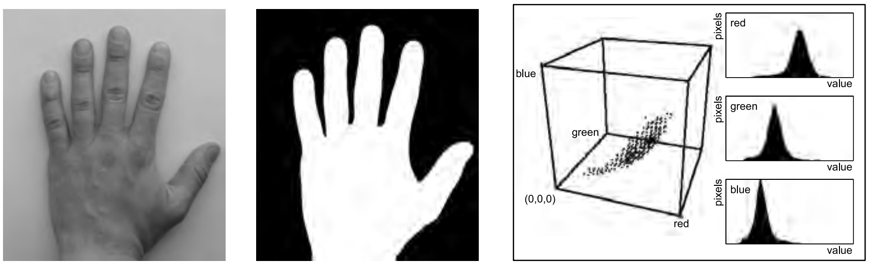
Using the color attribute to localize an object in the image requires a definition of the object's color or colors. In the RGB color model (and most others), even objects that one would call unicolored usually occupy a range of numerical values. This range can be described statistically using a three-dimensional discrete histogram $h_{\text{object}}(r, g, b)$, with the dimensions corresponding to the red, green, and blue components.

h_{object} is computed from a sufficiently large number of object pixels that are usually marked manually in a set of source images that cover all setups in which the system is intended to be used (e.g. multiple users, varying lighting conditions, etc.). Its value at (r, g, b) indicates the number of pixels with the corresponding color. The total sum of h_{object} over all colors is therefore equal to the number of considered object pixels n_{object}, i.e.

$$\sum_r \sum_g \sum_b h_{\text{object}}(r, g, b) = n_{\text{object}} \tag{2.5}$$

Because the encoded knowledge is central to the localization task, the creation of h_{object} is an important part of the system's development. It is exemplified below for a single frame.

Fig. 2.5 shows the source image of a hand (a) and a corresponding, manually generated binary mask (b) that indicates object pixels (white) and background pixels (black). In (c) the histogram computed from (a), considering only object pixels (those that are white in (b)), is shown. The three-dimensional view uses points to indicate colors that occurred with a certain minimum frequency. The three one-dimensional graphs to the right show the projection onto the red, green, and blue axis. Not surprisingly, red is the dominant skin color component.



(a) Source image (b) Object mask (c) Object color histogram

Fig. 2.5. From a source image (a) and a corresponding, manually generated object mask (b), an object color histogram (c) is computed.

On the basis of h_{object}, color-based object detection can now be performed in newly acquired images of the object. The aim is to compute, from a pixel's color, a probability or belief value indicating its likeliness of representing a part of the target object. This value is obtained for every pixel (x, y) and stored in a probability image $\mathbf{I}_{\text{object}}(x, y)$ with the same dimensions as the analyzed image. The following paragraphs derive the required stochastic equations.

Given an object pixel, the probability of it having a certain color (r, g, b) can be computed from h_{object} as

$$P(r, g, b|\text{object}) = \frac{h_{\text{object}}(r, g, b)}{n_{\text{object}}} \tag{2.6}$$

By creating a complementary histogram h_{bg} of the background colors from a total number of n_{bg} background pixels the accordant probability for a background pixel is obtained in the same way:

$$P(r, g, b|\text{bg}) = \frac{h_{\text{bg}}(r, g, b)}{n_{\text{bg}}} \tag{2.7}$$

Applying Bayes' rule, the probability of any pixel representing a part of the object can be computed from its color (r, g, b) using (2.6) and (2.7):

$$P(\text{object}|r, g, b) = \frac{P(r, g, b|\text{object}) \cdot P(\text{object})}{P(r, g, b|\text{object}) \cdot P(\text{object}) + P(r, g, b|\text{bg}) \cdot P(\text{bg})} \tag{2.8}$$

$P(\text{object})$ and $P(\text{bg})$ denote the a priori object and background probabilities, respectively, with $P(\text{object}) + P(\text{bg}) = 1$.

Using (2.8), the object probability image $\mathbf{I}_{\text{obj,prob}}$ is created from $\mathbf{I}$ as

$$\mathbf{I}_{\text{obj,prob}}(x, y) = P(\text{object}|\mathbf{I}(x, y)) \tag{2.9}$$

To classify each pixel as either background or target, an object probability threshold Θ is defined. Probabilities equal to or above Θ are considered target, while all others constitute the background. A data structure suitable for representing this classification is a binary mask

$$\mathbf{I}_{\text{obj,mask}}(x, y) = \begin{cases} 1 & \text{if } \mathbf{I}_{\text{obj,prob}}(x, y) \geq \Theta \quad \text{(target)} \\ 0 & \text{otherwise} \qquad\qquad\quad\; \text{(background)} \end{cases} \tag{2.10}$$

$\mathbf{I}_{\text{obj,mask}}$ constitutes a threshold segmentation of the source image because it partitions $\mathbf{I}$ into target and background regions.

Rewriting (2.8) as (2.11), it can be seen that varying $P(\text{object})$ and $P(\text{bg})$ is equivalent to a non-linear scaling of $P(\text{object}|r, g, b)$ and does not affect $\mathbf{I}_{\text{obj,mask}}$ when Θ is scaled accordingly (though it may improve contrast in the visualization of $\mathbf{I}_{\text{obj,prob}}$). In other words, for any prior probabilities $P(\text{object}), P(\text{bg}) \neq 0$, all possible segmentations can be obtained by varying Θ from 0 to 1. It is therefore practical to choose arbitrary values, e.g. $P(\mathrm{object}) = P(\mathrm{bg}) = 0.5$. Obviously, the value computed for $P(\mathrm{object}|r,g,b)$ will then no longer reflect an actual probability, but a belief value.[3] This is usually easier to handle than the computation of exact values for $P(\mathrm{object})$ and $P(\mathrm{bg})$.

$$P(\mathrm{object}|r,g,b) = \left(1 + \frac{P(r,g,b|\mathrm{bg})}{P(r,g,b|\mathrm{object})} \cdot \frac{P(\mathrm{bg})}{P(\mathrm{object})}\right)^{-1} \tag{2.11}$$

A suitable choice of Θ is crucial for an accurate discrimination between background and target. In case the recording conditions remain constant and are known in advance, Θ can be set manually, but for uncontrolled environments an automatic determination is desirable. Two different strategies are presented below.

If no knowledge of the target object's shape and/or position is available, the following iterative low-level algorithm presented in [24] produces good results in a variety of conditions. It assumes the histogram of $\mathbf{I}_{\mathrm{obj,prob}}$ to be bi-modal but can also be applied if this is not the case.

1. Arbitrarily define a set of background pixels (some usually have a high a priori background probability, e.g. the four corners of the image). All other pixels are defined as foreground. This constitutes an initial classification.
2. Compute the mean values for background and foreground, μ_{object} and μ_{bg}, based on the most recent classification. If the mean values are identical to those computed in the previous iteration, halt.
3. Compute a new threshold $\Theta = \frac{1}{2}\left(\mu_{\mathrm{object}} + \mu_{\mathrm{bg}}\right)$ and perform another classification of all pixels, then goto step 2.

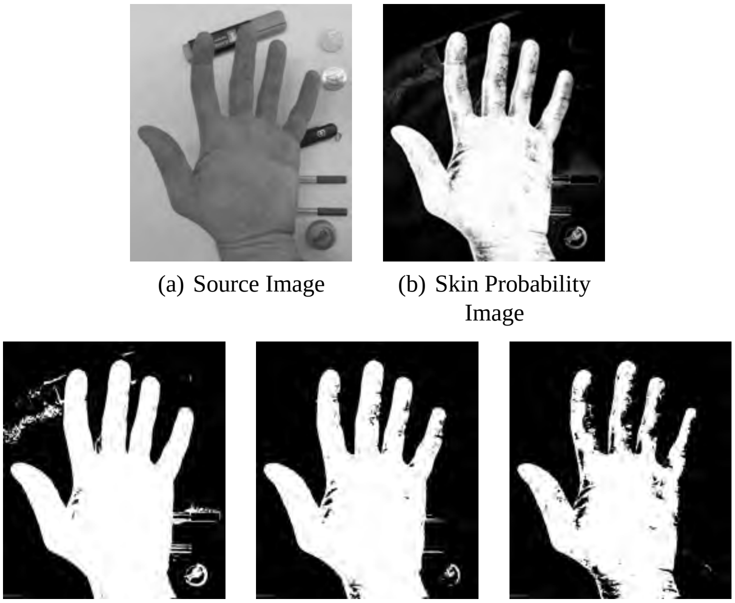
Fig. 2.6. Automatic computation of the object probability threshold Θ without the use of high-level knowledge (presented in [24]).

In case the target's approximate location and geometry are known, humans can usually identify a suitable threshold just by observation of the corresponding object mask. This approach can be implemented by defining an expected target shape, creating several object masks using different thresholds, and choosing the one which yields the most similar shape. This requires a feature extraction as described in Sect. 2.1.2 and a metric to quantify the deviation of a candidate shape's features from those of the expected target shape.

An example for hand localization can be seen in Fig. 2.7. Using the skin color histogram shown in Fig. 2.5 and a generic background histogram h_{bg}, a skin probability image (b) was computed from a source image (a) of the same hand (and some clutter) as shown in Fig. 2.5, using $P(\mathrm{object}) = P(\mathrm{bg}) = 0.5$. For the purpose of visualization, three different thresholds $\Theta_1 < \Theta_2 < \Theta_3$ were then applied, resulting in three different binary masks (c, d, e). In this example none of the binary masks

[3]For simplicity, the term "probability" will still be used in either case.

is entirely correct: For Θ_1, numerous background regions (such as the bottle cap in the bottom right corner) are classified as foreground, while Θ_3 leads to holes in the object, especially at its borders. Θ_2, which was computed automatically using the algorithm described in Fig. 2.6, is a compromise that might be considered optimal for many applications, such as the example systems to be developed in this section.


(a) Source Image (b) Skin Probability Image

(c) Skin/Background classification using Θ_1 (d) Skin/Background classification using Θ_2 (e) Skin/Background classification using Θ_3

Fig. 2.7. A source image (a) and the corresponding skin probability image (b). An object/background classification was performed for three different thresholds $\Theta_1 < \Theta_2 < \Theta_3$ (c-e).

While the human perception of an object's color is largely independent of the current illumination (an effect called *color constancy*), colors recorded by a camera are strongly influenced by illumination and hardware characteristics. This restricts the use of histograms to the recording conditions under which their source data was created, or necessitates the application of color constancy algorithms [7, 9].[4]

In [12], a skin and a non-skin color histogram are presented that were created from several thousand images found on the WWW, covering a multitude of skin

[4]In many digital cameras, color constancy can be achieved by performing a white balance: The camera is pointed to a white object (such as a blank sheet of paper), and a transformation is performed so that this object appears colorless ($r = g = b$) in the image as well. The transformation parameters are then stored for further exposures under the same illumination conditions.

hues, lighting conditions, and camera hardware. Thereby, these histograms implicitly provide user, illumination, and camera independence, at the cost of a comparably high probability for non-skin objects being classified as skin (*false alarm*). Fig. 2.8 shows the skin probability image (a) and binary masks (b, c, d) for the source image shown in Fig. 2.7a, computed using the histograms from [12] and three different thresholds $\Theta_4 < \Theta_5 < \Theta_6$. As described above, $P(\text{object}) = P(\text{bg}) = 0.5$ was chosen. Compared to 2.7b, the coins, the pens, the bottle cap, and the shadow around the text marker have significantly higher skin probability, with the pens even exceeding parts of the fingers and the thumb (d). This is a fundamental problem that cannot be solved by a simple threshold classification because there is no threshold Θ that would achieve a correct result. Unless the histograms can be modified to reduce the number of false alarms, the subsequent processing stages must therefore be designed to handle this problem.

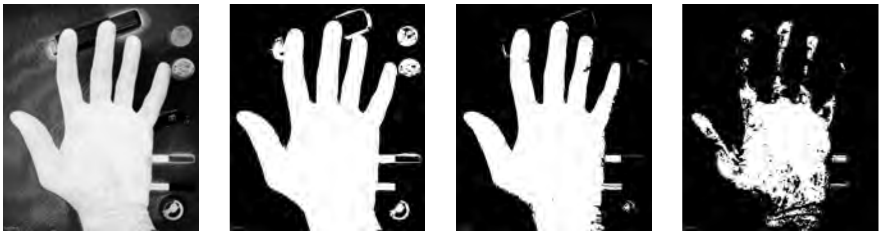

(a) Skin Probability Image (b) Skin/Background classification using Θ_4 (c) Skin/Background classification using Θ_5 (d) Skin/Background classification using Θ_6

Fig. 2.8. Skin probability image (b) for Fig. 2.8a and object/background classification for three thresholds $\Theta_4 < \Theta_5 < \Theta_6$ (b, c, d).

Implementational Aspects

In practical applications the colors r, g, and b commonly have a resolution of 8 bit each, i.e.

$$r, g, b \in \{0, 1, \ldots, 255\} \tag{2.12}$$

A corresponding histogram would have to have $256^3 = 16,777,216$ cells, each storing a floating point number of typically 4 bytes in size, totaling 64 MB. To reduce these memory requirements, the color values are downsampled to e.g. 5 bit through a simple integer division by $2^3 = 8$:

$$r' = \left\lfloor \frac{r}{8} \right\rfloor \quad g' = \left\lfloor \frac{g}{8} \right\rfloor \quad b' = \left\lfloor \frac{b}{8} \right\rfloor \tag{2.13}$$

This results in a histogram $h'(r', g', b')$ with $32^3 = 32,768$ cells that requires only 128 kB of memory. The entailed loss in accuracy is negligible in most applications, especially when consumer cameras with high noise levels are employed. Usually the downsampling is even advantageous because it constitutes a generalization and therefore reduces the amount of data required to create a representative histogram.

The LTI-LIB class `skinProbabilityMap` comes with the histograms presented in [12]. The algorithm described in Fig. 2.6 is implemented in the class `optimalThresholding`.

2.1.2.2 Region Description

On the basis of $\mathbf{I}_{\mathrm{obj,mask}}$ the source image $\mathbf{I}$ can be partitioned into regions. A region R is a contiguous set of pixels p for which $\mathbf{I}_{\mathrm{obj,mask}}$ has the same value. The concept of contiguity requires a definition of pixel adjacency. As depicted in Fig. 2.9, adjacency may be based on either a 4-neighborhood or an 8-neighborhood. In this section the 8-neighborhood will be used. In general, regions may contain other regions and/or holes, but this shall not be considered here because it is of minor importance in most gesture recognition applications.

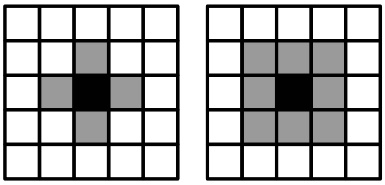


Fig. 2.9. Adjacent pixels (gray) in the 4-neighborhood (a) and 8-neighborhood (b) of a reference pixel (black).

Under laboratory recording conditions, $\mathbf{I}_{\mathrm{obj,mask}}$ will contain exactly one target region. However, in many real world applications, other skin colored objects may be visible as well, so that $\mathbf{I}_{\mathrm{obj,mask}}$ typically contains multiple target regions (see e.g. Fig. 2.8). Unless advanced reasoning strategies (such as described in [23, 28]) are used, the feature extraction stage is responsible for identifying the region that represents the user's hand, possibly among a multitude of candidates. This is done on the basis of the regions' geometric features.

The calculation of these features is performed by first creating an explicit description of each region contained in $\mathbf{I}_{\mathrm{obj,mask}}$. Various algorithms exist that generate, for every region, a list of all of its pixels. A more compact and computationally more efficient representation of a region can be obtained by storing only its border points (which are, in general, significantly fewer). A border point is conveniently defined as a pixel $p \in R$ that has at least one pixel $q \notin R$ within its 4-neighborhood. An example region (a) and its border points (b) are shown in Fig. 2.10.

In a counterclockwise traversal of the object's border, every border point has a predecessor and a successor within its 8-neighborhood. An efficient data structure for the representation of a region is a sorted list of its border points. This can be

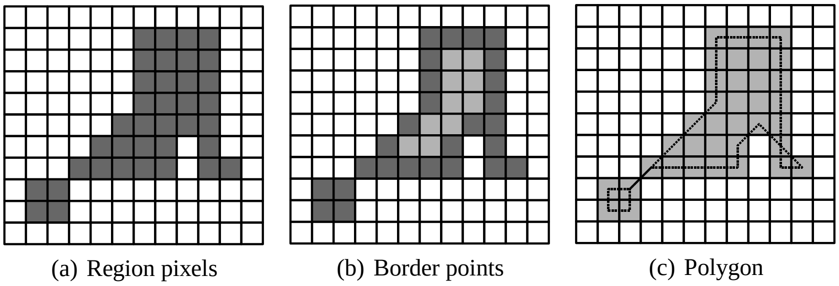


Fig. 2.10. Description of an image region by the set of all of its pixels (a), a list of its border pixels (b), and a closed polygon (c). Light gray pixels in (b) and (c) are not part of the respective representation, but are shown for comparison with (a).

interpreted as a closed polygon whose vertices are the centers of the border points (Fig. 2.10c). In the following, the object's border is defined to be this polygon. This definition has the advantages of being sub-pixel accurate and facilitating efficient computation of various shape-based geometric features (as described in the next section).[5]

Finding the border points of a region is not as straightforward as identifying its pixels. Fig. 2.11 shows an algorithm that processes $\mathbf{I}_{\mathrm{obj,mask}}$ and computes, for every region R, a list of its border points

$$B_R = \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\} \tag{2.14}$$

Fig. 2.12 (a, b, c) shows the borders computed by this algorithm from the masks shown in Fig. 2.8 (b, c, d). In areas where the skin probability approaches the threshold Θ, the borders become jagged due to the random nature of the input data, as can be seen especially in (b) and (c). This effect causes an increase in border length that is random as well, which is undesirable because it reduces the information content of the computed border length value (similar shapes may have substantially different border lengths, rendering this feature less useful for recognition). Preceding the segmentation (2.10) by a convolution of $\mathbf{I}_{\mathrm{obj,prob}}$ with a Gaussian kernel alleviates this problem by dampening high frequencies in the input data, providing smooth borders as in Fig. 2.12 (d, e, f).

The LTI-LIB uses the classes `borderPoints` and `polygonPoints` (among others) to represent image regions. The class `objectsFromMask` contains an algorithm that is comparable to the one presented in Fig. 2.11, but can also detect holes within regions, and further regions within these holes.

[5]It should be noted that the polygon border definition leads to the border pixels no longer being considered completely part of the object. Compared to other, non-sub-pixel accurate interpretations, this may lead to slight differences in object area that are noticeable only for very small objects.

[6]$\mathbf{m}(x, y) = 1$ indicates touching the border of a region that was already processed.

[7]$\mathbf{m}(x, y) = 2$ indicates crossing the border of a region that was already processed.

1. Create a helper matrix $\mathbf{m}$ with the same dimensions as $\mathbf{I}_{\text{obj,mask}}$ and initialize all entries to 0. Define (x, y) as the current coordinates and initialize them to $(0, 0)$. Define (x', y') and (x'', y'') as temporary coordinates.
2. Iterate from left to right through all image rows successively, starting at $y = 0$. If $\mathbf{m}(x, y) = 0 \wedge \mathbf{I}_{\text{obj,mask}}(x, y) = 1$, goto step 3. If $\mathbf{m}(x, y) = 1$, ignore this pixel and continue with the next pixel.[6] If $\mathbf{m}(x, y) = 2$, increase x until $\mathbf{m}(x, y) = 2$ again, ignoring the whole range of pixels.[7]
3. Create a list B of border points and store (x, y) as the first element.
4. Set $(x', y') = (x, y)$.
5. Scan the 8-neighborhood of (x', y') using (x'', y''), starting at the pixel that follows the butlast pixel stored in B in a counterclockwise orientation, or at $(x' - 1, y' - 1)$ if B contains only one pixel. Proceed counterclockwise, skipping coordinates that lie outside of the image, until $\mathbf{I}_{\text{obj,mask}}(x'', y'') = 1$. If (x'', y'') is identical with the first element of B, goto step 6. Else store (x'', y'') in B. Set $(x', y') = (x'', y'')$ and goto step 5.
6. Iterate through B, considering, for every element (x_i, y_i) its predecessor (x_{i-1}, y_{i-1}) and its successor (x_{i+1}, y_{i+1}). The first element is the successor of the last, which is the predecessor of the first. If $y_{i-1} = y_{i+1} \neq y_i$, set $\mathbf{m}(x_i, y_i) = 1$ to indicate that the border touches the line $y = y_i$ at x_i. Otherwise, if $y_{i-1} \neq y_i \vee y_i \neq y_{i+1}$, set $\mathbf{m}(x_i, y_i) = 2$ to indicates that the border intersects with the line $y = y_i$ at x_i.
7. Add B to the list of computed borders and proceed with step 2.

Fig. 2.11. Algorithm to find the border points of all regions in an image.

2.1.2.3 Geometric Features

From the multitude of features that can be computed for a closed polygon, only a subset is suitable for a concrete recognition task. A feature is suitable if it has a high ***inter-gesture variance*** (varies significantly between different gestures) and a low ***intra-gesture variance*** (varies little between multiple productions of the same gesture). The first property means that the feature carries much information, while the second indicates that it is not significantly affected by noise or unintentional variations that will inevitably occur. Additionally, every feature should be ***stable***, meaning that small changes in input data never result in large changes in the feature. Finally the feature must be computable with sufficient accuracy and speed. A certain feature's suitability thus depends on the constellation of the vocabulary (because this affects inter- and intra-gesture variance) and on the actual application scenario in terms of recording conditions, hardware etc. It is therefore a common approach to first compute as many features as possible and then examine each one with respect to its suitability for the specific recognition task (see also Sect. 2.1.3.3).

The choice of features is of central importance since it affects system design throughout the processing chain. The final system's performance depends significantly on the chosen features and their properties.

This section presents a selection of frequently used features and the equations to calculate them. Additionally, the way each feature is affected by the camera's perspective, resolution, and distance to the object is discussed, for these are often

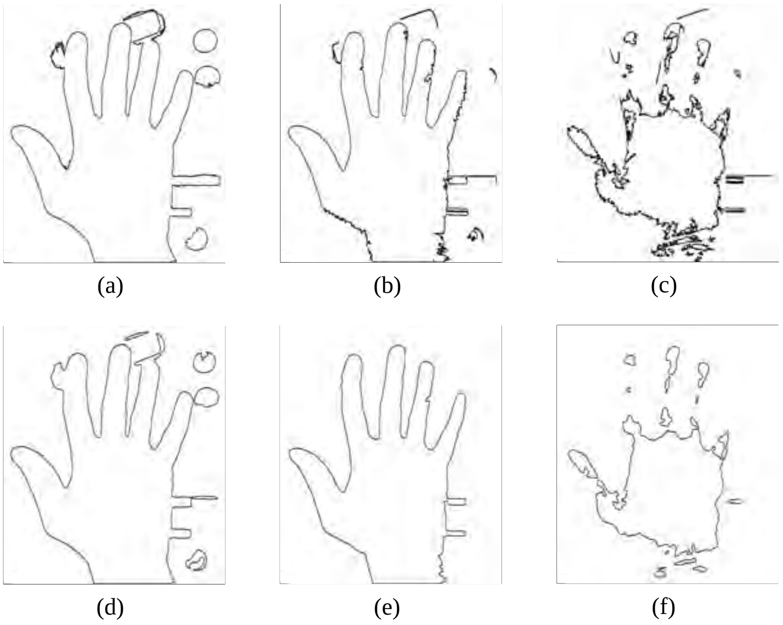


Fig. 2.12. a, b, c: Borders of the regions shown in Fig. 2.8 (d, e, f), computed by the algorithm described in Fig. 2.11. d, e, f: Additional smoothing with a Gaussian kernel before segmentation.

important factors in practical applications. In the image domain, changing the camera's perspective results in rotation and/or translation of the object, while resolution and object distance affect the object's scale (in terms of pixels). Since the error introduced in the shape representation by the discretization of the image (the discretization noise) is itself affected by image resolution, rotation, translation, and scaling of the shape, features can, in a strict sense, be invariant of these transformations only in continuous space. All statements regarding transformation invariance therefore refer to continuous space. In discretized images, features declared "invariant" may still show small variance. This can usually be ignored unless the shape's size is on the order of several pixels.

Border Length

The border length l can trivially be computed from B, considering that the distance between two successive border points is 1 if either their x or y coordinates are equal, and $\sqrt{2}$ otherwise. l depends on scale/resolution, and is translation and rotation invariant.

Area, Center of Gravity, and Second Order Moments

In [25] an efficient algorithm for the computation of arbitrary moments $\nu_{p,q}$ of polygons is presented. The area $a = \nu_{0,0}$, as well as the normalized moments $\alpha_{p,q}$

and central moments $\mu_{p,q}$ up to order 2, including the center of gravity (COG) $(x_{\text{cog}}, y_{\text{cog}}) = (\alpha_{1,0}, \alpha_{0,1})$, are obtained as shown in (2.15) to (2.23). x_i and y_i with $i = 0, 1, \ldots, n-1$ refer to the elements of B_R as defined in (2.14). Since these equations require the polygon to be closed, $x_n = x_0$ and $y_n = y_0$ is defined for $i = n$.

$$\nu_{0,0} = a = \frac{1}{2} \sum_{i=1}^{n} x_{i-1} y_i - x_i y_{i-1} \tag{2.15}$$

$$\alpha_{1,0} = x_{\text{cog}} = \frac{1}{6a} \sum_{i=1}^{n} (x_{i-1} y_i - x_i y_{i-1})(x_{i-1} + x_i) \tag{2.16}$$

$$\alpha_{0,1} = y_{\text{cog}} = \frac{1}{6a} \sum_{i=1}^{n} (x_{i-1} y_i - x_i y_{i-1})(y_{i-1} + y_i) \tag{2.17}$$

$$\alpha_{2,0} = \frac{1}{12a} \sum_{i=1}^{n} (x_{i-1} y_i - x_i y_{i-1})(x_{i-1}^2 + x_{i-1} x_i + x_i^2) \tag{2.18}$$

$$\alpha_{1,1} = \frac{1}{24a} \sum_{i=1}^{n} (x_{i-1} y_i - x_i y_{i-1})(2x_{i-1} y_{i-1} + x_{i-1} y_i + x_i y_{i-1} + 2x_i y_i) \tag{2.19}$$

$$\alpha_{2,0} = \frac{1}{12a} \sum_{i=1}^{n} (x_{i-1} y_i - x_i y_{i-1})(y_{i-1}^2 + y_{i-1} y_i + y_i^2) \tag{2.20}$$

$$\mu_{2,0} = \alpha_{2,0} - \alpha_{1,0}^2 \tag{2.21}$$

$$\mu_{1,1} = \alpha_{1,1} - \alpha_{1,0}\alpha_{0,1} \tag{2.22}$$

$$\mu_{0,2} = \alpha_{0,2} - \alpha_{0,1}^2 \tag{2.23}$$

The area a depends on scale/resolution, and is independent of translation and rotation. The center of gravity $(x_{\text{cog}}, y_{\text{cog}})$ is obviously translation variant and depends on resolution. It is rotation invariant only if the rotation occurs around $(x_{\text{cog}}, y_{\text{cog}})$ itself, and affected by scaling (i.e. changing the object's distance to the camera) unless its angle to the optical axis remains constant.

The second order moments ($p+q = 2$) are primarily used to compute other shape descriptors, as described below.

Eccentricity

One possible measure for eccentricity e that is based on central moments is given in [10]:

$$e = \frac{(\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}}{a} \tag{2.24}$$

(2.24) is zero for circular shapes and increases for elongated shapes. It is rotation and translation invariant, but variant with respect to scale/resolution.

Another intuitive measure is based on the object's inertia along and perpendicular to its main axis, j_1 and j_2, respectively:

$$e = \frac{j_1}{j_2} \tag{2.25}$$

with

$$j_1 = \frac{\mu_{2,0} + \mu_{0,2}}{2a} + \sqrt{\left(\frac{\mu_{2,0} - \mu_{0,2}}{2a}\right)^2 + \left(\frac{\mu_{1,1}}{a}\right)^2} \tag{2.26}$$

$$j_2 = \frac{\mu_{2,0} + \mu_{0,2}}{2a} - \sqrt{\left(\frac{\mu_{2,0} - \mu_{0,2}}{2a}\right)^2 + \left(\frac{\mu_{1,1}}{a}\right)^2} \tag{2.27}$$

Here e starts at 1 for circular shapes, increases for elongated shapes, and is invariant of translation, rotation, and scale/resolution.

Orientation

The region's main axis is defined as the axis of the least moment of inertia. Its orientation α is given by

$$\alpha = \frac{1}{2} \arctan\left(\frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}}\right) \tag{2.28}$$

and corresponds to what one would intuitively call "the object's orientation". Orientation is translation and scale/resolution invariant. α carries most information for elongated shapes and becomes increasingly random for circular shapes. It is $180°$-periodic, which necessitates special handling in most cases. For example, the angle by which an object with $\alpha_1 = 10°$ has to be rotated to align with an object with $\alpha_2 = 170°$ is $20°$ rather than $|\alpha_1 - \alpha_2| = 160°$. When using orientation as a feature for classification, a simple workaround to this problem is to ensure $0 \leq \alpha < 180°$ and, rather than use α directly as a feature, compute two new features $f_1(\alpha) = \sin \alpha$ and $f_2(\alpha) = \cos 2\alpha$ instead, so that $f_1(0) = f_1(180°)$ and $f_2(0) = f_2(180°)$.

Compactness

A shape's compactness $c \in [0, 1]$ is defined as

$$c = \frac{4\pi a}{l^2} \tag{2.29}$$

Compact shapes ($c \rightarrow 1$) have short borders l that contain a large area a. The most compact shape is a circle ($c = 1$), while for elongated or frayed shapes, $c \rightarrow 0$. Compactness is rotation, translation, and scale/resolution invariant.

Border Features

In addition to the border length l, the minimum and maximum pixel coordinates of the object, $x_{\min}$, $x_{\max}$, $y_{\min}$ and $y_{\max}$, as well as the minimum and maximum distance from the center of gravity to the border, $r_{\min}$ and $r_{\max}$, can be calculated from B. Minimum and maximum coordinates are not invariant to any transformation. $r_{\min}$ and $r_{\max}$ are invariant to translation and rotation, and variant to scale/resolution.

Normalization

Some of the above features can only be used in a real-life application if a suitable normalization is performed to eliminate translation and scale variance. For example, the user might perform gestures at different locations in the image and at different distances from the camera.

Which normalization strategy yields the best results depends on the actual application and recording conditions, and is commonly found empirically. Normalization is frequently neglected, even though it is an essential part of feature extraction. The following paragraphs present several suggestions. In the remainder of this section, the result of the normalization of a feature f is designated by f'.

For both static and dynamic gestures, the user's face can serve as a reference for hand size and location if it is visible and a sufficiently reliable face detection is available.

In case different resolutions with identical aspect ratio are to be supported, resolution independence can be achieved by specifying lengths and coordinates relative to the image width N (x and y must be normalized by the same value to preserve image geometry) and area relative to N^2.

If a direct normalization is not feasible, invariance can also be achieved by computing a new feature from two or more unnormalized features. For example, from x_{cog}, $x_{\min}$, and $x_{\max}$, a resolution and translation invariant feature x_p can be computed as

$$x_p = \frac{x_{\max} - x_{cog}}{x_{cog} - x_{\min}} \tag{2.30}$$

x_p specifies the ratio of the longest horizontal protrusion, measured from the center of gravity, to the right and to the left.

For dynamic gestures, several additional methods can be used to compute a normalized feature $f'(t)$ from an unnormalized feature $f(t)$ without any additional information:

- Subtracting $f(1)$ so that $f'(1) = 0$:

$$f'(t) = f(t) - f(1) \tag{2.31}$$

- Subtracting the arithmetic mean $\overline{f}$ or median f_{median} of f:

$$f'(t) = f(t) - \overline{f} \tag{2.32}$$

$$f'(t) = f(t) - f_{\text{median}} \tag{2.33}$$

- Mapping f to a fixed interval, e.g. $[0, 1]$:

$$f'(t) = \frac{f(t) - \min f(t)}{\max f(t) - \min f(t)} \tag{2.34}$$

- Dividing f by $\max |f(t)|$ so that $|f'(t)| \leq 1$:

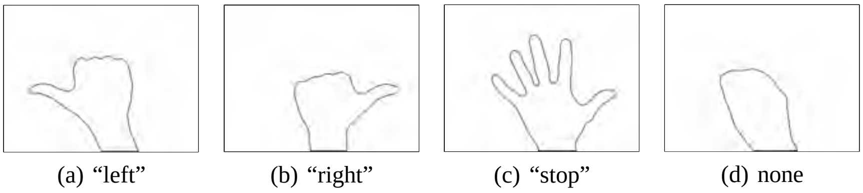
$$f'(t) = \frac{f(t)}{\max |f(t)|} \tag{2.35}$$

Derivatives

In features computed for dynamic gestures, invariance of a constant offset may also be achieved by derivation. Computing the derivative $\dot{f}(t)$ of a feature $f(t)$ and using it as an additional element in the feature vector to emphasize changes in $f(t)$ can sometimes be a simple yet effective method to improve classification performance.

2.1.2.4 Example

To illustrate the use of the algorithms described above, the static example gestures introduced in Fig. 2.2 were segmented as shown in Fig. 2.13, using the automatic algorithms presented in Fig. 2.6 and 2.11, and features were computed as shown in Tab. 2.5, using the LTI-Lib class `geometricFeatures`. For resolution independence, lengths and coordinates are normalized by the image width N, and area is normalized by N^2. α specifies the angle by which the object would have to be rotated to align with the x axis.



(a) “left” (b) “right” (c) “stop” (d) none

Fig. 2.13. Boundaries of the static hand gestures shown in Fig. 2.2. Corresponding features are shown in Tab. 2.5.

For the dynamic example gestures “clockwise”, “open”, and “grab”, the features area a, compactness c, and x coordinate of the center of gravity x_{cog} were computed. Since a depends on the hand’s anatomy and its distance to the camera, it was normalized by its maximum (see (2.35)) to eliminate this dependency. x_{cog} was divided by N to eliminate resolution dependency, and additionally normalized according to (2.32) so that $\overline{x'}_{\text{cog}} = 0$. This allows the gestures to be performed anywhere in the image. The resulting plots for normalized area a', compactness c (which does not require normalization), and normalized x coordinate of the center of gravity x'_{cog} are visualized in Fig. 2.14.

The plots are scaled equally for each feature to allow a direct comparison. Since “counterclockwise”, “close”, and “drop” are backwards executions of the above gestures, their feature plots are simply temporally mirrored versions of these plots.

Table 2.5. Features computed from the boundaries shown in Fig. 2.13.

		Gesture			
Feature	**Symbol**	"left" (2.13a)	"right" (2.13b)	"stop" (2.13c)	none (2.13d)
Normalized Border Length	l'	1.958	1.705	3.306	1.405
Normalized Area	a'	0.138	0.115	0.156	0.114
Normalized Center of Gravity	x'_{cog}	0.491	0.560	0.544	0.479
	y'_{cog}	0.486	0.527	0.487	0.537
Eccentricity	e	1.758	1.434	1.722	2.908
Orientation	α	57.4°	147.4°	61.7°	58.7°
Compactness	c	0.451	0.498	0.180	0.724
Normalized Min./Max. Coordinates	x'_{min}	0.128	0.359	0.241	0.284
	x'_{max}	0.691	0.894	0.881	0.681
	y'_{min}	0.256	0.341	0.153	0.325
	y'_{max}	0.747	0.747	0.747	0.747
Protrusion Ratio	x_p	0.550	1.664	1.110	1.036

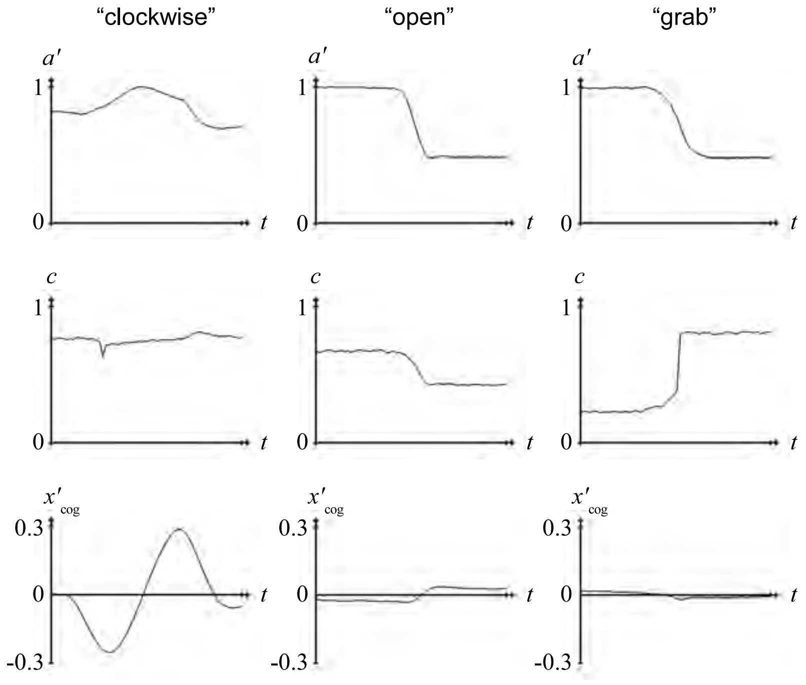


Fig. 2.14. Normalized area a', compactness c, and normalized x coordinate of the center of gravity x'_{cog} plotted over time $t = 1, 2, \ldots, 60$ for the dynamic gestures "clockwise" (left column), "open" (middle column), and "grab" (right column) from the example vocabulary shown in Fig. 2.3.

2.1.3 Feature Classification

The task of feature classification occurs in all pattern recognition systems and has been subject of considerable research effort. A large number of algorithms is available to build classifiers for various requirements. The classifiers considered in this section operate in two phases: In the *training phase* the classifier "learns" the vocabulary from a sufficiently large number of representative examples (the *training samples*). This "knowledge" is then applied in the following *classification phase*. Classifiers that continue to learn even in the classification phase, e.g. to automatically adapt to the current user, shall not be considered here.

2.1.3.1 Classification Concepts

This section introduces several basic concepts and terminology commonly used in pattern classification, with the particular application to gesture recognition.

Mathematical Description of the Classification Task

From a mathematical point of view, the task of classification is that of identifying an unknown event ω, given a finite set Ω of n mutually exclusive possible events

$$\omega \in \Omega = \{\omega_1, \omega_2, \ldots, \omega_n\} \tag{2.36}$$

The elements of Ω are called *classes*. In gesture recognition Ω is the vocabulary, and each class ω_i $(i = 1, 2, \ldots, n)$ represents a gesture. Note that (2.36) restricts the allowed inputs to elements of Ω. This means that the system is never presented with an unknown gesture, which influences the classifier's design and algorithms.

The classifier receives, from the feature extraction stage, an observation **O**, which, for example, might be a single feature vector for static gestures, or a sequence of feature vectors for dynamic gestures. Based on this observation it outputs a result

$$\hat{\omega} \in \Omega \quad \text{where} \quad \hat{\omega} = \omega_k, \quad k \in \{1, 2, \ldots, n\} \tag{2.37}$$

k denotes the index of the class of the event ω assumed to be the source of the observation **O**. If $\hat{\omega} = \omega$ then the classification result is correct, otherwise it is wrong.

To account for the case that **O** does not bear sufficient similarity to any element in Ω, one may wish to allow a *rejection* of **O**. This is accomplished by introducing a pseudo-class ω_0 and defining a set of classifier outputs $\hat{\Omega}$ as

$$\hat{\Omega} = \Omega \cup \{\omega_0\} \tag{2.38}$$

Thus, (2.37) becomes

$$\hat{\omega} \in \hat{\Omega} \quad \text{where} \quad \hat{\omega} = \omega_k, \quad k \in \{0, 1, \ldots, n\} \tag{2.39}$$

Modeling Classification Effects

Several types of classifiers can be designed to consider a *cost* or *loss* value L for classification errors, including rejection. In general, the loss value depends on the actual input class ω and the classifier's output $\hat{\omega}$:

$$L(\omega, \hat{\omega}) = \text{cost incurred for classifying event of class } \omega \text{ as } \hat{\omega} \tag{2.40}$$

The cost of a correct result is set to zero:

$$L(\omega, \omega) = 0 \tag{2.41}$$

This allows to model applications where certain misclassifications are more serious than others. For example, let us consider a gesture recognition application that performs navigation of a menu structure. Misclassifying the gesture for "return to main menu" as "move cursor down" requires the user to perform "return to main menu" again. Misclassifying "move cursor down" as "return to main menu", however, discards the currently selected menu item, which is a more serious error because the user will have to navigate to this menu item all over again (assuming that there is no "undo" or "back" functionality).

Simplifications

Classifiers that consider rejection and loss values are discussed in [22]. For the purpose of this introduction, we will assume that the classifier never rejects its input (i.e. (2.37) holds) and that the loss incurred by a classification error is a constant value L:

$$L(\omega, \hat{\omega}) = L \quad \text{for} \quad \omega \neq \hat{\omega} \tag{2.42}$$

This simplifies the following equations and allows us to focus on the basic classification principles.

Garbage Classes

An alternative to rejecting the input is to explicitly include *garbage classes* in Ω that represent events to which the system should not react. The classifier treats these classes just like regular classes, but the subsequent stages do not perform any action when the classification result $\hat{\omega}$ is a garbage class. For example, a gesture recognition system may be constantly observing the user's hand holding a steering wheel, but react only to specific gestures different from steering motions. Garbage classes for this application would contain movements of the hand along the circular shape of the steering wheel.

Modes of Classification

We distinguish between *supervised classification* and *unsupervised classification*:

- In supervised classification the training samples are labeled, i.e. the class of each training sample is known. Classification of a new observation is performed by a comparison with these samples (or models created from them) and yields the class that best matches the observation, according to a matching criterion.
- In unsupervised classification the training samples are unlabeled. Clustering algorithms are used to group similar samples before classification is performed. Parameters such as the number of clusters to create or the average cluster size can be specified to influence the clustering process. The task of labeling the samples is thus performed by the classifier, which of course may introduce errors. Classification itself is then performed as above, but returns a cluster index instead of a label.

For gesture recognition, supervised classification is by far the most common method since the labeling of the training samples can easily be done in the recording process.

Overfitting

With many classification algorithms, optimizing performance for the training data at hand bears the risk of reducing performance for other (generic) input data. This is called *overfitting* and presents a common problem in machine learning, especially for small sets of training samples. A classifier with a set of parameters p is said to overfit the training samples T if there exists another set of parameters p' that yields lower performance on T, but higher performance in the actual “real-world” application [15].

A strategy for avoiding overfitting is to use disjunct sets of samples for training and testing. This explicitly measures the classifier’s ability of generalization and allows to include it in the optimization process. Obviously, the test samples need to be sufficiently distinct from the training samples for this approach to be effective.

2.1.3.2 Classification Algorithms

From the numerous strategies that exist for finding the best-matching class for an observation, three will be presented here:

- A simple, easy-to-implement *rule-based* approach suitable for small vocabularies of static gestures.
- The concept of *maximum likelihood classification*, which can be applied to a multitude of problems.
- *Hidden Markov Models* (HMMs) for dynamic gestures. HMMs are frequently used for the classification of various dynamic processes, including speech and sign language.

Further algorithms, such as artificial neural networks, can be found in the literature on pattern recognition and machine learning. A good starting point is [15].