Part II

Segmentation Using Object Models

It is a mistake to think you can solve
any major problems just with potatoes.
(Douglas Adams)

Chapter 9

Object Detection based on Graph-Models I: Cartoons

The first part of this thesis has presented an automatic segmentation system that does not use any pre-knowledge about the foreground objects to be extracted. However, sometimes it is desired to specify more precisely which objects should be extracted. This allows to disambiguate between the important acting objects and unimportant objects that should not be extracted (like the audience in sport videos). Moreover, having a model about the objects can provide a better segmentation in cases where the scene background is not known or where it is very similar to the foreground. This and the following chapter present a graph-based model to describe video objects. This model is applied to extract a user-defined object from video sequences or still images. The model-detection algorithm is based on an inexact graph-matching between the user-defined object model and an automatically extracted graph describing the input image. This chapter discusses the model-generation process, and we present a matching algorithm tailored to the object detection in cartoon sequences. Cartoon sequences are especially difficult to process with ordinary segmentation algorithms, but they are fitting well to the graph-model based approach. The successive chapter further extends the object-detection algorithm to natural video sequences.

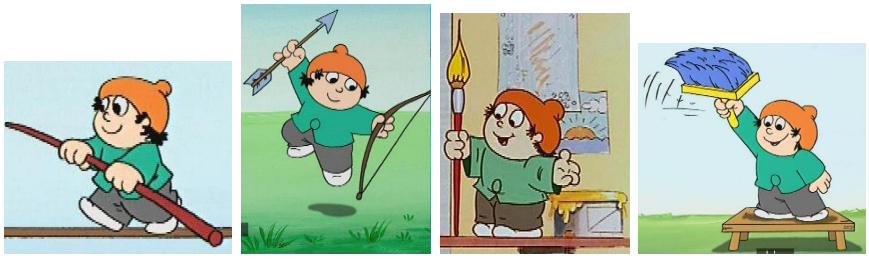
9.1 Introduction

In the first part of the thesis, an automatic segmentation system was described that is based on a background-subtraction technique. Background subtraction employs knowledge about the scene background, but it uses no pre-knowledge about the foreground objects. This has the advantage that no information about the foreground objects is required. On the other hand, a simple background subtraction cannot decide if the detected objects are important for the successive processing steps. In many cases, we are only interested in some of the visible objects, even though more objects are extracted. One example are sport videos, where the athletes are extracted, but also moving people in the audience. In contrast with this, we also experience problems when only part of the important object is moving, like in news broadcasts in which the anchorman usually only moves his head. In this case, the previously proposed segmentation system cannot know that the body of the anchorman is not simply part of the background. These problems can only be solved by providing the segmentation algorithm with pre-knowledge about the objects to be extracted.

A video object can generally appear in many different deformed or articulated appearances in an image (see Fig. 9.1). Moreover, it can be occluded or viewed from different sides. The crucial problem in defining an object model is to find a representation that is flexible enough to fit to all the different appearances of the object. On the other hand, the model should not be too general, since it will fit to incorrect places otherwise.

In this chapter, we introduce a system for video-object detection and extraction based on user-defined models. Our object models are described by "model graphs", in which nodes represent image regions and edges denote spatial proximity. Each node is attributed with color and shape information about the corresponding image region. Model graphs are specified manually based on a sample image of the object. Object recognition starts with automatic color segmentation of the input image. For each region, the same features are extracted as specified in the model graph. Recognition is based on finding a subgraph in the input graph that matches the model graph. Evidently, it is not sufficient to search for an isomorph subgraph, since node and edge attributes will not match exactly. Furthermore, the automatic segmentation step leads to an oversegmented image. For this reason, we employ *inexact* graph matching, where several nodes of the input graph may be mapped onto a single node in the model graph.

Graph matching is a well-known technique in computer vision and several efficient heuristics have been developed for the graph isomorphism problem. These include algorithms based on nonlinear optimization [76], quadratic programming [163, 145], relaxation labeling [184], or algorithms



(a) (b) (c) (d)

Figure 9.1: *The same object can appear in many different shapes.*

that are specialized for a specific class of graphs [43]. A completely different approach to region correspondence uses the *Earth Movers Distance* (EMD), which is a popular distance measure in the field of image retrieval [79]. Recently, region-based algorithms have also become popular in the context of searching in video databases [17, 112, 23]. In this case, characteristic regions are first extracted from a query image and subsequently, these regions are used to form a database query for images that contain similar regions. However, since the relevant regions are extracted automatically, no pre-knowledge about the spatial object structure is available. Consequently, the object structure is often neglected.

In this chapter, we concentrate on the object recognition in cartoon sequences. This class of sequences is difficult to handle with current automatic segmentation algorithms, because the motion estimation has difficulties arising from large homogeneous regions and because the object appearance is typically highly variable. In the next chapter, we extend the graph-model based object detection approach to a similar detection system for natural images.

9.2 Principle of region-based graph matching

Our approach for object detection is based on the assumption that objects can be described reliably by a set of attributed regions and their spatial relationship. The model structure and features are expressed by an object model graph $G_M = (V_M, E_M)$, where each node in V_M represents an image region with uniform color. Nodes have attributes describing region color, shape, and size. Edges in the model graph define spatial proximity. This means that if $(v_1, v_2) \in E_M$, region v_1 must be near region v_2. The model graph representation allows to recognize objects independent of their exact spatial layout as long as the characteristic spatial structure of the objects

remains. In particular, articulated object motion can be modeled in a straightforward way.

Model graphs are defined manually by the user in a graphical editor (top part of Figure 9.2). To ease the definition, a sample image of the object can be segmented semi-automatically. Subsequently, the region features are extracted automatically from the sample image. Finally, the spatial structure of the object is defined by connecting neighboring regions.

Object recognition starts with an automatic color segmentation of the input image. For each of the regions obtained from this segmentation, the same features are extracted as for the model graph (bottom part of Figure 9.2). A fully connected *input graph* is defined, generating nodes from the regions and attributing the edges with the distance between pairs of regions. Since the regions are generated by an automatic segmentation process covering the whole image, this input graph will be much larger than the model graph. Furthermore, due to oversegmentation, regions that belong together semantically may be split into separate regions.

The object recognition is based on the idea to find a subgraph in the input graph that matches our model graph. Obviously, it is not possible to find an isomorph subgraph, because node and edge attributes will not match exactly and model-graph regions are possibly split into oversegmented regions. Hence, we apply an *inexact* graph matching where several nodes of the input graph can be mapped onto a single node in the model graph ($1 : N$-matching). The quality of a match is described by judging the compatibility of the node and edge attributes.

In order to reduce the high computational complexity of graph matching, we employ a fast three-step matching algorithm.

- The first step reduces the search space by eliminating nodes in the input graph that are very unlikely to occur in the match.

- The second step performs a $1 : 1$-matching of the *skeleton tree* of the model graph. The skeleton tree is a sub-graph $T = (V_M, E_S)$ (with $E_S \subseteq E_M$) of the model graph that only contains a subset of the edges such that it forms a tree. This $1 : 1$-matching of the skeleton tree can be carried out very efficiently using a dynamic-programming approach.

- The third matching step considers the whole model graph and extends the matching to a $1 : N$-matching.

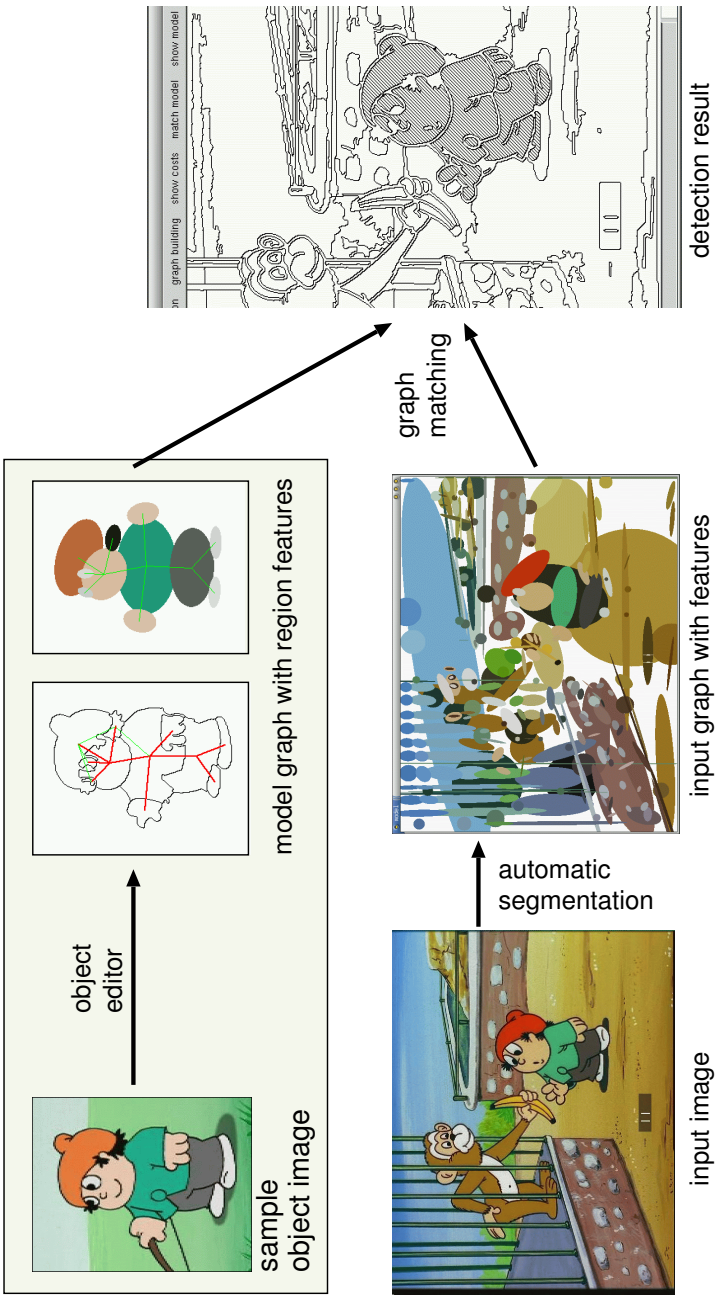
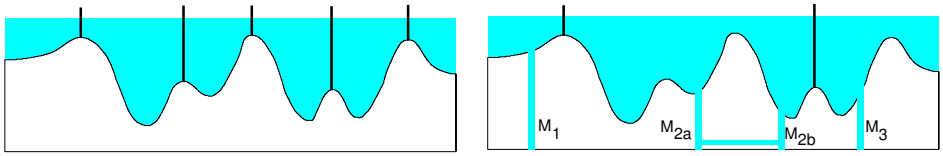


Figure 9.2: *Framework of the object-detection system. First, an object model is defined by the user in a graphical object-model editor (top part of drawing). The object detection starts with an automatic color segmentation of the input image (bottom part of drawing), which results in a large input graph. A matching algorithm localizes the position of the model graph in the input graph to identify the regions that belong to the object.*



(a) Standard watershed. (b) Marker driven watershed.

Figure 9.3: *Marker-driven manual watershed algorithm. Whereas in the standard algorithm each local minimum creates a new segment (a), the marker-driven watershed algorithm only builds watersheds between markers (b). Note that the markers M_{2a} and M_{2b} are assigned to the same region, so that no watershed is built between them.*

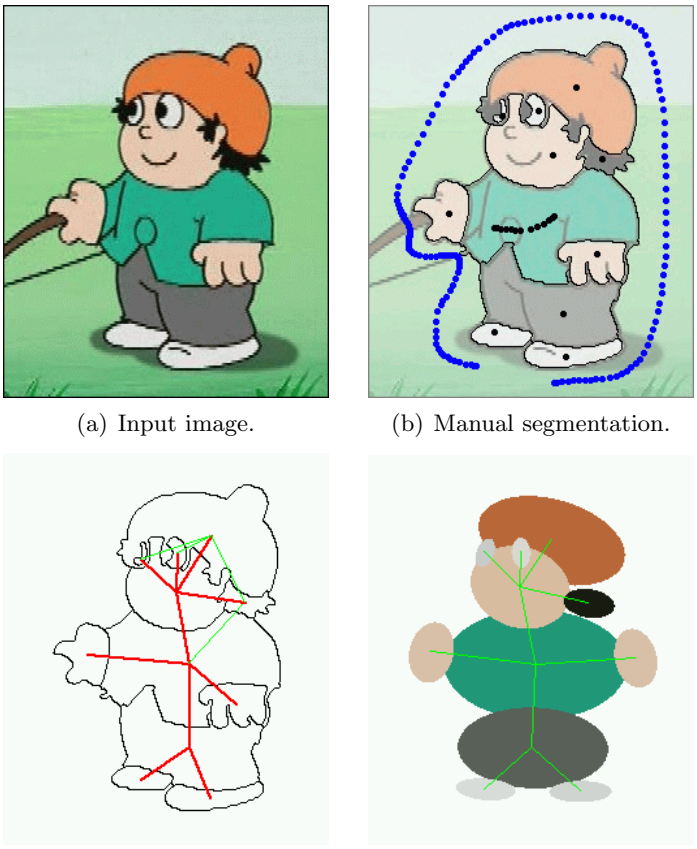
9.3 Model editor

This section describes the editor for generating the model graphs. The object models which are used during the object-detection process, are defined manually by the user in a graphical editor. Manual user interaction is required, because only the user knows exactly the semantic meaning of the object model and thus only he can specify the characteristic attributes of a particular object. Since the model specification is an easy task and the models can be saved into a database of frequently-used models, the required time for user interaction is low.

Segmentation of the object regions is based on a marker-driven watershed algorithm, which is applied on the gradients of a sample image. The difference to the standard watershed algorithm is that the water does not start flooding from the various local minima. Instead, the water commences to flow from the markers. Several markers can be grouped together such that water basins of these markers are attributed to the same region (i.e., no watershed is built between markers for the same region, see Figure 9.3(b)).

Relevant object regions are defined manually by placing markers in a sample image. The exact region boundaries are subsequently located by the watershed algorithm. Errors in the segmentation can be corrected by joining regions that have been separated by the watershed algorithm. Internally, this is realized by considering the markers of both regions equivalent (compare the markers M_{2a}, M_{2b} in Figure 9.3(b)). The region attributes are extracted from the sample image (see Figure 9.4(d)), but they can be modified by the user in case the sample image does not contain a typical view of the object.

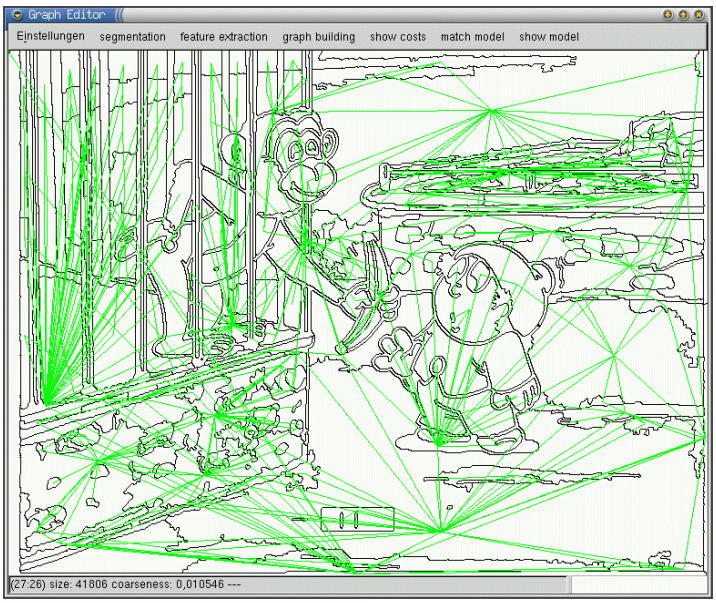
Finally, graph edges are added to define regions that should be close to each other, independent of a specific object view. Note that the model



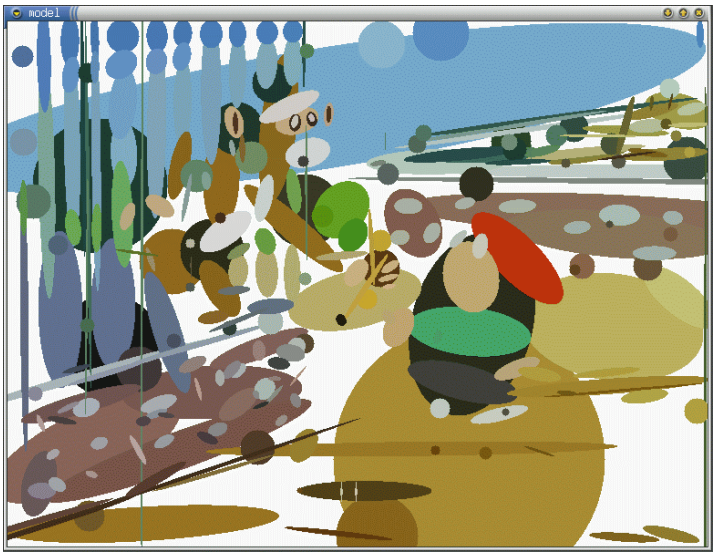
(a) Input image. (b) Manual segmentation.

(c) Model graph. (d) *Blobbed* model graph showing region features.

Figure 9.4: *The creation process of a model graph. Based on the sample input image (a), the user places markers into the image to separate the regions (b). Edges are introduced (c), where edges of the model skeleton tree are depicted with strong red lines, and the fine green lines denote the refinement edges used in the* $1:N$ *matching step. The region features can be visualized in an abstract presentation (d).*



(a)



(b)

Figure 9.5: *Result of the input graph obtained from the automatic color segmentation. Each color region established a node in the input graph, enriched with features like node color and size.*

should only contain those edges that are semantically necessary for the specific object. For a model of a human, for example, the head is directly connected to the body, but not to one of the arms even though this may be the case in a specific sample image.

As we will see later when considering the matching algorithm, matching becomes particularly efficient when the model graph has a tree topology. Therefore, we classify the model graph edges into two classes: *skeleton tree* edges, and *refinement* edges. The principal 1 : 1-matching step only uses the skeleton tree edges. This forms no severe restriction, since most natural objects can be described sufficiently using trees. The refinement edges are used in the $1 : N$-matching step when oversegmented regions are combined to cover the whole object.

9.4 Automatic color segmentation

Automatic color segmentation is carried out using a combination of watershed segmentation and region merging. The watershed algorithm provides a very fast pre-segmentation, but this is usually strongly oversegmented and thus not sufficient for our purpose. Hence, an additional region-merging algorithm is applied on the pre-segmentation result to further combine neighboring regions obtained from the watershed algorithm. Although the watershed pre-segmentation is not required, it considerably speeds up the segmentation process, because the region-merging algorithm can start with larger initial regions (more detail about the color segmentation can be found in Appendix E).

Region merging has proven to be a powerful segmentation algorithm, enabling the use of various merging criteria to control the merging process. We have chosen the Ward criterion, which results in a segmentation in which the region variance is minimized. The fundamental idea is to consider every neighboring pair of regions and calculate the increase of variance that a possible merge of the two regions would impose. Let σ_i^2 denote the variance of region r_i, μ_i the region mean brightness, and $|r_i|$ the region size. Then, the increase of variance when regions r_i and r_j are merged, can be calculated by the new variance σ_{ij}^2 minus the original individual variances, which gives

$$\Delta_{ij} = \sigma_{ij}^2 - \sigma_i^2 - \sigma_j^2 = \frac{|r_i| \cdot |r_j|}{|r_i| + |r_j|} \big(\mu_i - \mu_j\big)^2. \tag{9.1}$$

The region-merging algorithm now successively combines the two regions r_a, r_b for which Δ_{ab} is minimal until the minimum Δ_{ab} exceeds a threshold.

We denote the final set of regions as $R = \{r_i\}$. A more in-depth treatment of the color segmentation is provided in Chapter 10.

9.5 Feature extraction and matching criteria

To evaluate the similarity of image regions, a set of features is extracted for each region during model creation, as well as for each region generated by the automatic segmentation process. Based on these features, node and edge cost-functions are defined which serve as matching criteria in the graph matching step. The calculation of features that are not required for candidate selection (see below) can be postponed after the candidate selection step. Since only a smaller subset of the regions is actually used in the matching process, the computation time is reduced.

9.5.1 Color

The color of each region is described by its coefficients in the Hue, Value, Saturation (HVS) color space. This color space allows an easy definition of a distance metric having a close relationship with the human perception. HVS space can be visualized as a cone with black at the tip and the rainbow colors around the base. After transforming the HVS color (h, v, s) with $h \in [0; 2\pi]$, $v, s \in [0; 1]$ into cartesian coordinates using $x = v \cdot s \cdot \cos h$, $y = v \cdot s \cdot \sin h$, $z = v$, we use the Euclidean distance between the two colors as color matching cost. We denote the matching cost for assigning region $r_i \in R$ to model node $m_i \in V_M$ as $C^C_{m_i}(r_i)$.

9.5.2 Size

The shape feature is simply the size of a region in pixels. During the matching process, two cost measures are used for region sizes: one based on the absolute region size and one based on relative region sizes. The absolute region size measure is applied during the candidate selection step to sort out regions that are much larger than the object model. The absolute size feature is computed as the ratio of the input region size $|r_i|$ with respect to the model region size $|m_i|$: $f^S_{m_i}(r_i) = |r_i|/|m_i|$.

For the computation of the relative size measure, let $|r_i|, |r_j|$ be the sizes of two connected regions and $|m_i|, |m_j|$ the sizes of the corresponding model regions. Since the size of the object in the image may vary, we do not compare the absolute region sizes to the model in the actual matching step. In fact, only the relative sizes of connected regions are compared to the model. Following this approach, we define the matching cost $C^{RelS}_{m_i,m_j}(r_i, r_j)$ by the piecewise linear function depicted in Figure 9.6. This measure does

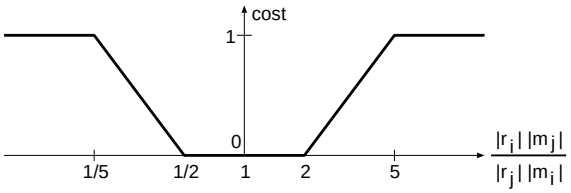
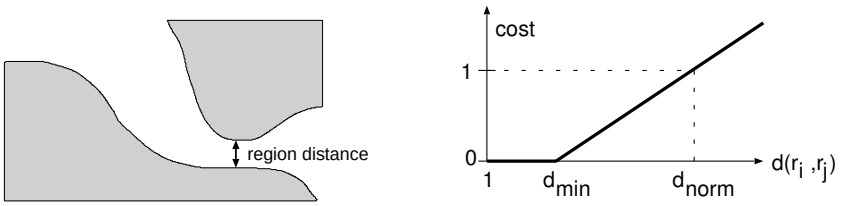


Figure 9.6: *Relative size cost $C^{RelS}_{m_i,m_j}(r_i, r_j)$ for matching a pair of connected model nodes to a pair of input regions.*



(a) Region distance. (b) Cost measure $C^D(r_i, r_j)$.

Figure 9.7: *(a) Region distances are calculated as the minimum Euclidean distance between the region borders. (b) The cost function increases linearly with the distance, but tolerates small distances between regions due to inaccurate segmentation.*

not penalize variations of region sizes up to a factor of two. This is to be robust for varying region sizes because of many factors like occlusions, differing viewing position, deformable objects, or inaccurate segmentation.

9.5.3 Distance

Connected model regions are assumed to have zero distance. However, the distance between a pair of input regions r_i, r_j is measured as the minimum pixel distance $d(r_i, r_j)$ between both region borders (see Fig. 9.7(a)). The region distance-cost is defined as (see Fig. 9.7(b))

$$C^D(r_i, r_j) = \begin{cases} 0 & \text{for } d(r_i, r_j) < d_{min}, \\ \frac{d(r_i,r_j)-d_{min}}{d_{norm}-d_{min}} & \text{else.} \end{cases} \tag{9.2}$$

Truncating the error for small distances has been introduced to tolerate small region distances between input regions. These small distances can be caused by an inaccurate segmentation. We have chosen $d_{min} = 5$ and $d_{norm} = 30$ pixels in our experiments, in which the image size was 720×576 pixels.

9.5.4 Shape

Automatic segmentation usually generates some regions having a "fuzzy" shape, being thin and having many concavities. These regions almost never belong to any object, but rather appear in background regions between objects. Since the regions are often located near object boundaries, they are close to all regions in the object and thus, for the matching algorithm, they seem to be part of the object. Hence, it is preferable to early identify these regions and exclude them from the matching.

To find such "misleading" regions, we make use of a shape feature that describes the *shape complexity* of a region. It is computed as

$$f^{Sh}(r_i) = 4\pi \frac{|r_i|}{border(r_i)^2}, \tag{9.3}$$

where $border(r_i)$ is the length of the region border and $|r_i|$ is the region area. Clearly, the shape feature is maximal ($f^{Sh} = 1$) when the region boundary is a circle and approaches zero when the region is long and thin. Note that f^{Sh} is invariant to scaling.

9.5.5 Orientation

Edge orientation is an optional matching criterion and can be activated manually for each individual edge. When matching symmetric objects, the orientation of the matched graph is ambiguous. To break this symmetry, edges can be declared as *oriented* edges. These edges remember which of the two regions is left of the other (or above the other). The relative orientation of two regions is determined by comparing the coordinates of their centers of gravity. If the model edge $e = (m_i, m_j)$ is an oriented edge and the orientation of the assigned input regions r_i, r_j differs, the costs is set to $C^O_{m_i,m_j}(r_i, r_j) = 1$; otherwise we set $C^O = 0$.

9.5.6 Node and edge costs

The above-mentioned costs are combined into node-cost and edge-cost functions, which are computed by

$$C^N_{m_i}(r_i) = \alpha C^C_{m_i}(r_i), \tag{9.4}$$

for the node cost and

$$C^E_{m_i,m_j}(r_i, r_j) = \beta C^D(r_i, r_j) + \gamma C^{RelS}_{m_i,m_j}(r_i, r_j) + \theta C^O_{m_i,m_j}(r_i, r_j) \tag{9.5}$$

for the edge cost, respectively, where the parameters α, β, γ, θ are weighting factors which we have set to 1. They can be increased or decreased

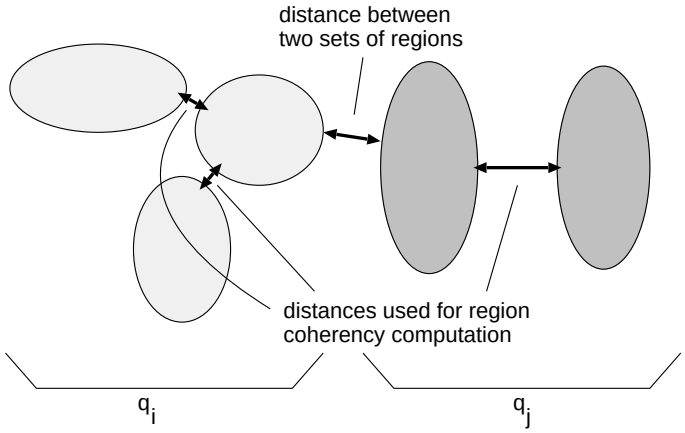


Figure 9.8: *Visualization of computing the distance cost between two sets of regions. The distance is defined as the minimum distance cost of all regions pairs ($r_a \in q_i, r_b \in q_j$) plus the minimum distance costs between the regions in each set.*

depending on the application. For example, when it is *a-priori* known that the color may vary because of differing lighting conditions, the weight of the color cost α should be decreased.

9.5.7 Generalization of costs for $1:N$-matching

For the $1:N$-matching, we generalize the cost measures to handle mappings from several input regions to a single model region. The measures are defined such that the cost measures for $1:1$ matching result as a special case. We denote the generalized cost definitions with hats on the variable names. We define the $1:N$-matching cost for model nodes with $q_i \subset R$ as

$$\hat{C}^N_{m_i}(q_i) = \frac{1}{\sum_{r \in q_i} |r|} \sum_{r \in q_i} |r| \cdot C^C_{m_i}(r), \tag{9.6}$$

which is the sum of all node costs for the region in R, weighted with the region size.

The generalized distance measure has to capture the distance between two sets of regions. At the same time, it should also prevent that the regions within one set are itself distributed over the image with large distance. Hence, we introduce the *coherency* of a set of regions as the pairwise spatial proximity of the regions within both sets. This leads to our definition of

the generalized distance measure, which we compute as (see Fig. 9.8)

$$\hat{C}^D(q_i,q_j) = \underbrace{\min_{r_a\in q_i, r_b\in q_j} C^D(r_a,r_b)}_{\substack{\text{minimum distance between}\\ \text{both sets of regions}}} + \underbrace{\frac{1}{2}\sum_{r_a\in q_i}\min_{r_b\in q_i, r_b\neq r_a} C^D(r_a,r_b)}_{\text{coherence of region } q_i} + \underbrace{\frac{1}{2}\sum_{r_b\in q_j}\min_{r_a\in q_j, r_a\neq r_b} C^D(r_a,r_b)}_{\text{coherence of region } q_j}. \tag{9.7}$$

Note that this definition of coherency would assign two distant clusters of regions a high coherency, which is not desired. However, in the $1:N$-matching algorithm, regions are added one by one. Consequently, the set of regions cannot grow easily into two clusters of regions, because the cost for the first distant regions would be high.

The generalized cost functions for relative region size and orientation $\hat{C}^{RelS}$ and $\hat{C}^O$ are computed by determining the sum of region sizes and the center of gravity for the set of regions. The generalized total edge cost $\hat{C}^E$ is defined as the weighted sum of the individual costs similar to the definition of C^E in Eq. (9.5), replacing all cost components C with the generalized counterpart $\hat{C}$.

9.6 Matching algorithm

Graph matching is carried out in a three-step process.

- **Step 1** Since the color or size of many of the regions generated by the automatic segmentation will strongly deviate from the model regions, they can be excluded from the matching process to decrease the computational time. The first matching step determines for each model region a subset of input regions and performs the previously mentioned exclusion of unsuitable regions.

- **Step 2** The second matching step involves only the regions of the selected subset as candidates for a model node. The second matching step computes a $1:1$ matching of the model graph skeleton tree using a dynamic programming approach.

- **Step 3** This $1:1$ matching acts also as the initialization for the third matching step, where the $1:1$ matching is enriched to form a $1:N$ matching. *Enriching* means that additional input regions can be assigned to a single model region to decrease overall cost.

The three matching steps can also be viewed as incrementally imposing additional structural information. While the first step (candidate selection) is completely free from any structural constraints, the $1:1$-matching obeys the structure of the model skeleton tree, and the final $1:N$-matching step considers the full model-graph structure.

9.6.1 Candidate-region selection

The candidate input regions for a model region are selected based on the color, the region size and the shape feature. The idea of the candidate-selection step is to sort out regions that have the wrong color, a clearly wrong size, or a non-compact shape. Two selection strategies are possible: we can fix the number of candidates N_C for each model region and select the N_C best input regions as candidates, or we can set a threshold on the region similarity and consider all input regions with higher similarity as candidates. The choice of selection strategy is not critical when the number of candidates is sufficiently high and the thresholds are set high enough to ensure that the correct matches are not sorted out. We adopted a strategy with fixed number of candidates for each model region and observed that about $10-20$ candidates for each model region are sufficient.

At first, we filter out regions that are a factor θ larger than the corresponding model region or that have a significantly different shape. We apply two thresholds

$$C_v^S(c(v,i)) > \theta \quad \text{and} \quad f^{Sh}(c(v,i)) > \nu, \tag{9.8}$$

which were chosen as $\theta = 3$ and $\nu = 0.15$ in our experiments. These values can be adjusted or even selected individually for each model region, depending on the amount this region can change its size in different input images and on the application or model. Since θ and ν are only used to sort out clearly non-matching regions, they can be set arbitrary large or even can be omitted at all (but more candidates would have to be considered in this case). Since the automatically segmented regions are possibly only part of a single model-graph region and the total size of the object to be found in the input image is not known yet, regions that are too small should not be excluded. The remaining regions are sorted with a mapping

$$c : V_M \times \{1, 2, \ldots, |R|\} \rightarrow R, \tag{9.9}$$

such that

$$C_v^C(c(v,i)) < C_v^C(c(v,j)) \;\rightarrow\; i < j. \tag{9.10}$$

Hence, c sorts the input regions according to increasing matching costs, with the best matching region $c(v,1)$ for model node v, and the worst matching region that is still considered $c(v,|R|)$.

Note that the same input region can appear as candidate region for several model regions. The constraint that the same input region must not be assigned twice to different model nodes must be satisfied by the following algorithmic step.

9.6.2 Matching algorithm

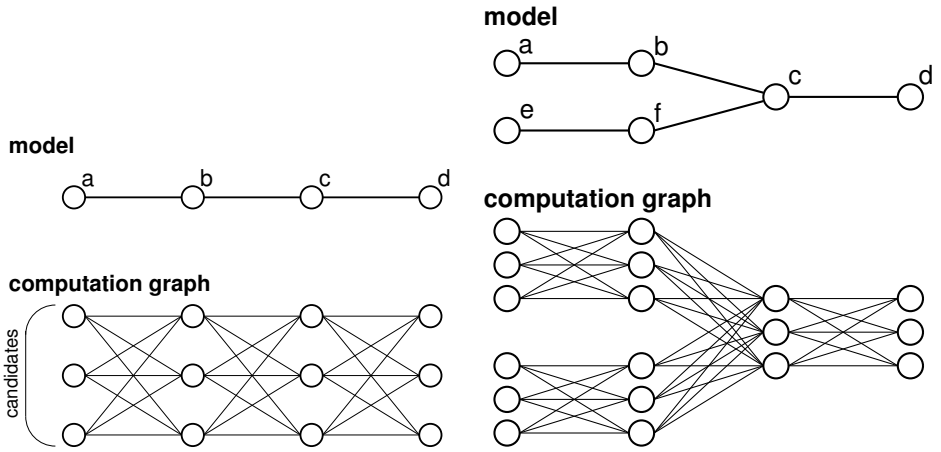
The 1 : 1-matching step is the most important step, since it does the primary localization of the model regions. The found matches are the seed for the 1 : N-matching step, where they are further extended with additional input regions.

Weighted graph-matching can be described as finding the maximum weight clique in the corresponding association graph [144], which is known to be *NP*-hard. However, for special classes of graphs, such as e.g. trees, efficient algorithms exist. Since almost all real-world objects can be accurately described by trees and because efficient algorithms for trees exist, we restrict our 1 : 1-matching step to finding the best matching tree for the skeleton tree of a model graph.

Our algorithm is based on a dynamic-programming approach. The objective is to find the mapping $\mathcal{M}_{1:1} : V_M \rightarrow \{1, 2, \ldots, |N_C|\}$ that minimizes the sum of node costs and edge costs in the tree:

$$\min_{\mathcal{M}_{1:1}} \Bigg\{ \sum_{v \in V_M} C_v^N\big(c(v, \mathcal{M}_{1:1}(v))\big) + \sum_{(v_1,v_2) \in E_S} C_{v_1,v_2}^E\big(c(v_1, \mathcal{M}_{1:1}(v_1)) \, , \, c(v_2, \mathcal{M}_{1:1}(v_2))\big) \Bigg\}. \tag{9.11}$$

Let us introduce the concept of computing the minimum cost mapping with a simple example. Assume that the model tree is e.g. a simple linear chain (Fig. 9.9(a)). We construct a computation graph by duplicating each model node to N_C nodes, each representing the decision that the model region is mapped to a specific candidate node. The node costs C^N are assigned to the nodes, i.e., the first column of nodes get costs $C_a^N(c(a,1))$, $C_a^N(c(a,2))$, ... , $C_a^N(c(a,N_C))$. Similarly, the edge costs C^E are assigned to the edges. Now minimizing the sum (9.11) is equivalent to computing the minimum cost path through the resulting computation graph. To compute the minimum cost path, we proceed column by column from left to right and determine for each node the predecessor node that gives the minimum total cost so far. More specifically, we assign attributes *mincost* and *last* to each node in the computation graph. The nodes in the left column are initialized with *mincost* equal to their respective node



(a) Model graph is a chain. (b) Model graph with a single join.

Figure 9.9: *Object-model skeleton trees with their respective computation graphs.*

costs C^N and $last = \text{nil}$. Continuing with the next column to the right, we calculate for each node the total cost that results from each choice for the predecessor candidate. This cost consists of the *mincost* of the predecessor node, the edge cost C^E linking the predecessor node to the current node and the current node cost C^N. The predecessor node that gave the least cost is stored into *last* and the corresponding minimum cost in *mincost*. When we arrive at the rightmost column, the candidate with the minimum cost is selected and the minimum cost path is traced back using the *last* attributes.

If the model tree contains junctions like shown in Figure 9.9(b), the algorithm above has to be extended. Since model node c has multiple incoming edges, the best predecessor candidate has to be selected from both, model node b and model node f. Consequently, *mincost* is now obtained by minimizing over the sum of all previous nodes and incoming edge costs. The computation time required therefore increases from N_C^2 steps for a column to $indegree \cdot N_C^2$ computations ($indegree = 2$ in our example). However, the total computation time does not increase, because the total number of edges in the computation tree remains constant. Hence, the complexity is $O(N_C^2 \cdot |V_M|)$. The complete matching algorithm is described in Algorithm 1 and 2. Algorithm 1 initializes the *pred* attributes that define the order in which the model nodes have to be considered in the calculation. The set $pred(v)$ of a node v is the set of adjacent nodes that have to be processed prior to node v. This attribute defines the depth-first recursion

Algorithm 1 Initialization of the computation graph for the subsequent dynamic-programming algorithm.

Require: the model tree $V_M = \{v_1, \ldots, v_N\}, E_S \subset V_M \times V_M$

$pred(v_1 \in V_M) \leftarrow \emptyset$
$l \leftarrow \{v_1\}$
$r \leftarrow V_M \setminus \{v_1\}$
while $l \neq \emptyset$ **do**
  select an arbitrary $v \in l$ and set $l \leftarrow l \setminus \{v\}$
  for all $(v, w) \in E_S \wedge w \in r$ **do**
    $pred(v) \leftarrow pred(v) \cup \{w\}$
    $l \leftarrow l \cup \{w\}$
  end for
end while

Algorithm 2 Compute the minimum cost assignment.

procedure $calccolumn(v \in V_M)$
for all $w \in pred(v)$ **do**
  call $calccolumn(w)$
end for
for $n = 1$ to N_C **do**
  if $pred(v) = \emptyset$ **then**
    $mincost(v, n) \leftarrow C_v^N(v, c(v, n))$
  else
    $cost \leftarrow C_v^N(v, c(v, n))$
    for $w \in pred(v)$ **do**
      $cost \leftarrow cost + \min_i\big(mincost(w, i) + C_{w,v}^E(c(w, i), c(v, n))\big)$
      $last(v, n, w)$ is set to the i that minimized the above sum
    end for
    $mincost(v, n) \leftarrow cost$
  end if
end for

order in which the costs for the nodes are calculated. Algorithm 2 performs the actual 1 : 1 matching. It processes all nodes in the depth-first order defined by $pred()$. The algorithm is initialized to start at the tree root: $calccolumn(v_1)$.

In each junction node, instead of only storing a single predecessor, we haev to store the best candidate node for each incoming model-tree edge. When tracing back the minimum cost path, we obtain a minimum-cost tree instead of a linear chain.

The algorithm described thus far has still one drawback. When the same input region occurs as candidate for different model nodes, the algorithm may use the same input region more than once. This is not desirable. For example, consider searching for a human with equal left and right arm (see Figure 9.10). The model nodes for both arms are the same and both subtrees are connected to the same body node. Since either the left or right arm in the input graph will match better to the model, the algorithm will assign the best one to both arms of the model.

This problem can be alleviated using two techniques. First, it is possible to make the edges connecting the arm and the body *oriented* edges (see Section 9.5.5) inducing extra cost when the left arm in the input graph is mapped to the right arm in the model graph and vice versa. However, this does not work in all situations and we have to extend the algorithm described previously to prevent double assignments. This can be done by introducing a *blocked* attribute to each computation graph node. This attribute stores the set of input regions that are used so far. In each junction node v of the computation graph ($|pred(v)| > 1$), combinations of previous node candidates k_1, k_2 that collide ($blocked(k_1) \cap blocked(k_2) \neq \emptyset$) cannot be selected. Note that since this is a combinatorial problem, the best candidate node for all preceding nodes cannot be determined independently. In fact, all combinations are enumerated and checked for validity. The valid combination with the minimum cost defines the best candidates for the preceding model nodes.

As an example, consider Figure 9.10. Note that Node 3 selects input regions a and b as its left arm. Under the assumption that the right arm looks identical to the left arm in the model, the dynamic-programming algorithm without *blocking* attribute would select the same input regions for the right arm. However, since a and b are contained in the *blocked* set of Node 3b and Node 5b, Node 2 has to choose Nodes d and e for the right arm. Unfortunately, it cannot ensure that both arms are assigned to the correct side, because the orientation is lost in the graph description. This orientation ambiguity can be resolved by defining the edges connecting the arms with the body as *oriented* edges.

9.6.3 $1:N$-matching

Starting with the $1:1$-matching result, the $1:N$-matching algorithm assigns additional input regions to model nodes if this decreases the total cost. We define the $1:N$ matching through a mapping $\mathcal{M}_{1:N} : V_M \rightarrow 2^R$. It is initialized with the result of the preceding $1:1$ matching by

$$\mathcal{M}_{1:N}(m_i) := \big\{c(m_i, \mathcal{M}_{1:1}(m_i))\big\}. \tag{9.12}$$

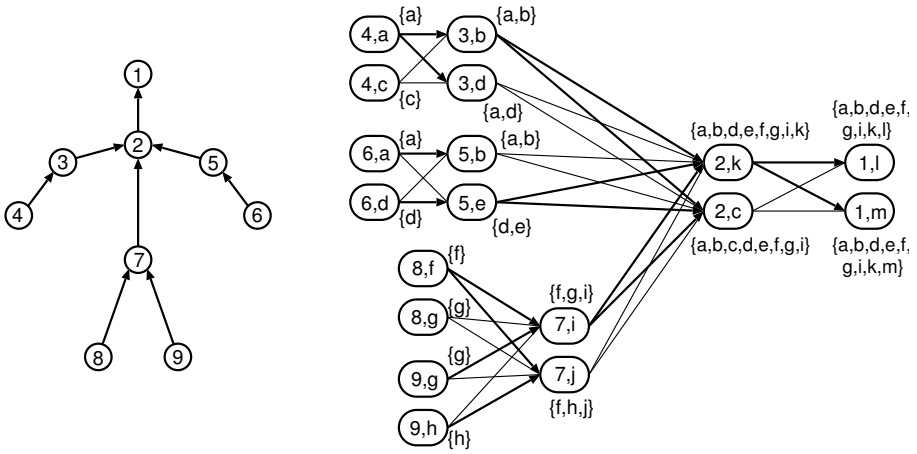


Figure 9.10: *Example calculation for a model graph (left) describing a human. The arrows denote the order of calculation as induced by the pred attributes. For simplicity, the computation graph on the right has been constructed with only two candidate nodes for each model node. The model nodes are denoted by numbers and the input nodes by letters. Selected edges are drawn with thick arrows and the corresponding blocked attribute is shown at each node. Calculation proceeds from left to right.*

More input regions are added with a greedy algorithm. In each iteration, a cost difference $\delta_{m_i}(q_i, r_k)$ is computed that equals the cost difference induced from adding input region r_k to region set q_i. As long as the cost difference is below zero, the input region with the largest decrease of cost is added. Otherwise, the algorithm ends. We define the cost difference as

$$\begin{aligned}\delta_{m_i}(q_i, r_k) = & \underbrace{\hat{C}^N_{m_i}(q_i) + \sum_{(m_i,m_j)\in E_M} \hat{C}^E_{m_i,m_j}(q_i, \mathcal{M}_{1:N}(m_j))}_{\text{old node and edge costs for region set } q_i} \\ & \underbrace{-\hat{C}^N_{m_i}(q_i \cup \{r_k\}) - \sum_{(m_i,m_j)\in E_M} \hat{C}^E_{m_i,m_j}(q_i \cup \{r_k\}, \mathcal{M}_{1:N}(m_j))}_{\text{new node and edges costs for regions set } q_i \text{ plus region } r_k} \\ & \underbrace{-\epsilon \hat{C}^B(q_i, r_k)}_{\begin{array}{l}\text{cost reduction be-}\\ \text{cause of common}\\ \text{boundary}\end{array}} ,\end{aligned} \tag{9.13}$$

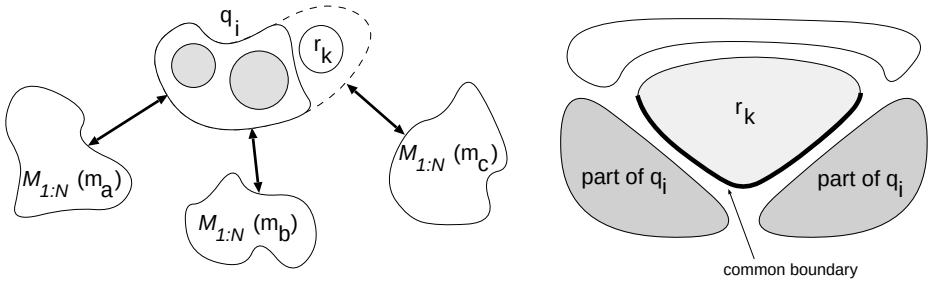
where the first terms are the generalized node and edge costs as defined in Section 9.5.6. The last term $\hat{C}^B$ decreases the cost for region r_k if it shares a common boundary with the regions in q_i (see Fig. 9.11). The weighting factor ϵ was set to 0.5 in our experiments.

To calculate $\hat{C}^B(q_i, r_k)$, we consider each pixel on the boundary of r_k and search for the region r' that is nearest to the pixel among all candidate regions for all model nodes. $\hat{C}^B(q_i, r_k)$ is set to the fraction of pixels for which the nearest region $r' \in q_i$. Clearly, if r_k is completely surrounded by regions in q_i, then $\hat{C}^B(q_i, r_k) = 1$.

Finally, we can summarize the effects of the $1:N$ matching step with a simple rule: a region will be added to the set of assigned regions of a model node if

- the region is mostly surrounded by other regions mapped to the same model node,
- the region bridges the space between two regions that should be neighboring, or
- the combined region size (or color) matches better to the model node size.

Figure 9.12 portrays an example showing that $1:N$ matching improves the accuracy of the object-model detection. Since model regions have been split



(a) Testing region r_k for inclusion into region set q_i.

(b) Common boundary between r_k and q_i.

Figure 9.11: *(a) In* $1:N$ *matching, the hypothetical cost after adding input region* r_k *is computed. If the cost difference is lower,* r_k *is attributed to the model node. (b) Definition of the common boundary of a region, i.e., the part of a boundary that is* inside *the region set.*

into several parts by the occlusion of a foreground object, $1:N$ matching is required to cover the whole model region.

9.7 Results

An example matching result for a scene with several objects having similar characteristics is shown in Figure 9.13. The object defined by the model is detected correctly. Small errors occur at the left hand of the object since the algorithm cannot decide whether the fingers are part of the hand or not. As the size of the hand without fingers matches better to the size of the jacket, the fingers are discarded. Figure 9.14 shows more detection results for the same object model.

Another example result is depicted in Figures 9.15 and 9.16. For these examples, a model of the Goofy cartoon character was generated. Note that the $1:1$ matching result that is presented in Figure 9.16(a) does not cover the complete object, since some regions have remained as several independent regions in the color segmentation. In the $1:N$ matching step, most of these regions are added to the object.

9.8 Conclusions

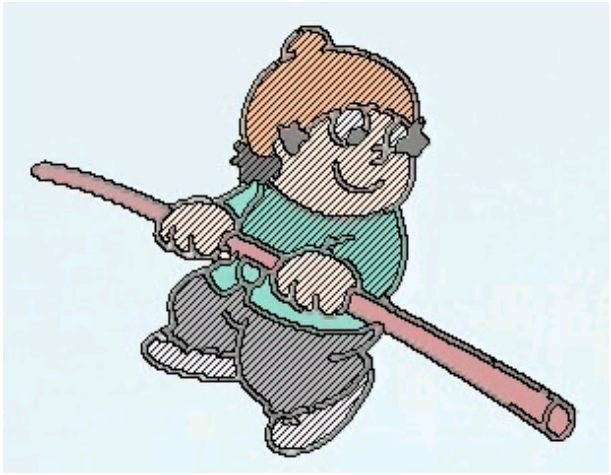
This chapter has presented a new algorithm for the detection of video objects that are described by manually-defined object models. A graph-based

modeling is used to describe region features and spatial relationships between regions. Thereby, the model allows for enough flexibility to find objects, even when they are deformed by articulated motion. The central matching step is carried out using a fast dynamic-programming algorithm, which is enabled by restricting the object-graphs to be trees. The computation time is currently about 1 second for a 720×576 video frame on a 550 MHz Pentium-III processor. Most of the time is used for the color segmentation step. Hence, it is possible to test the same input image for several object models even faster, because the initial color segmentation only has to be computed once.

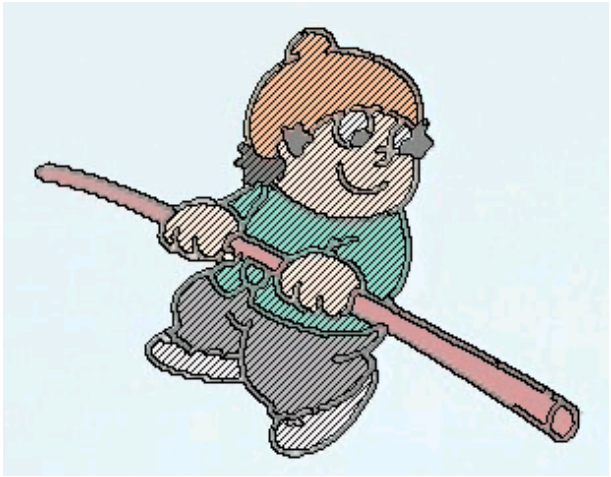
As an alternative to the dynamic-programming based object-detection algorithm, we have also implemented a direct $1 : N$-matching using a genetic programming approach [10]. However, the description of this approach was omitted here, since the results were inferior compared with the presented algorithm. Even though the cost function was defined in the same way, the genetic algorithm did not show robust convergence.

Our experiments for various sequences revealed that the described matching algorithm is robust if the visible object does not deviate too much from the model and if there is no significant occlusion. Possible errors are mostly introduced due to an erroneous color segmentation. Sometimes, regions having the same color are combined into the same region when they are close to another. Since the matching algorithm can map several input regions to a single model region, but not several model regions to a single input region, the matching algorithm searches for another region instead of assigning the undersegmented input region twice. Another major source of errors are pictures in which parts of the object are occluded, since the algorithm cannot find matching regions for the occluded parts. The consequence is that either wrong regions from the background are added to the object, or that the object is not detected at all, since the missing regions increase the total matching error too much.

Our overall impression is that the proposed approach is applicable for objects that only move within a plane. When the 3-D view onto the object is changed, many parts of the object become occluded or change their size. This leads to high matching errors and the object regions are not assigned correctly. To resolve this problem, it may be required to use a 3-D model of the object, such that the computer can reflect the changes from the differing viewing direction with the model. However, this would also require that the object pose is estimated, which complicates the matching process.



(a) $1:1$-matching.



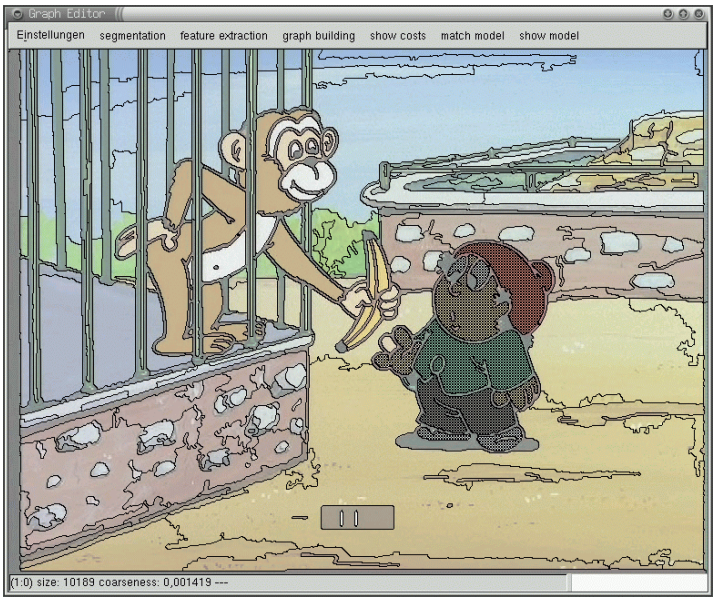
(b) $1:N$-matching.

Figure 9.12: *Matching the object model from Fig. 9.4. The image shown is part of a larger input image with several other objects. (a) Result after matching the object skeleton tree. Matched regions are marked. Note that the jacket is not covered completely since it has been oversegmented into several regions. (b) Matching results after the* $1:N$*-matching step. After the* $1:N$ *matching, the jacket is completely covered.*

(a) Input image.

(b) Found object model (second from the right).

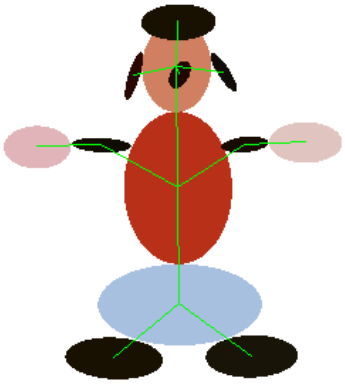
Figure 9.13: *The object model shown in Figure 9.4 is searched for in an input image with several similar objects.*



(a)

(b)

Figure 9.14: *Detection results for two images with the object model from Figure 9.4.*



(a) Goofy model.

(b) Detection result.

Figure 9.15: *(a) Model of the Goofy cartoon character. (b) Detection result of a* $1:N$ *matching with several similar objects.*

(a) 1 : 1-matching.

(b) 1 : N-matching.

Figure 9.16: *Detection result of Goofy for* 1 : 1 *matching (a) and* 1 : N *matching (b). It can be observed that the* 1 : 1 *does not include all regions. The* 1 : N *matching is almost complete.*