Image Interpretation, part II

December 14, 2010

Chapter 1

Local invariant features

1.1 Introduction

Local invariant features are a powerful tool, that has been applied succesfully in a wide range of computer vision systems and applications. Apart from object recognition, they have proven useful for various applications in need for correspondences between images: registration of aerial images, image mosaicking, 3D reconstruction, image retrieval, vision-based robot navigation, symmetry detection, scene and action recognition, to name just a few.

It is worth noting in this context that the importance of local features has also been demonstrated in the context of visual object recognition in the human brain.More precisely, experiments have shown that removing the corners from images impedes human recognition, while removing most of the edge information does not.

The ideal local feature would be a point as defined in geometry: having a location in space but no spatial extent. In practice however, images are discrete with the smallest spatial unit being a pixel and discretization effects playing an important role. To localize features in images (e.g. using image intensities and derivatives thereof), a local neighbourhood of pixels needs to be analyzed, giving all local features some (implicit or explicit) spatial extent. For some applications (e.g. camera calibration or 3D reconstruction) this spatial extent is ignored in further processing, and only the location derived from the feature extraction process counts. In those cases, the local features are often termed *interest points*.

However, in most applications those features also need to be described, such that they can be identified and matched, and this again calls for a local neighbourhood of pixels. Often, this neighbourhood is taken equal to the neighbourhood used to localize the feature, but this need not be the case. In this context, one often uses the term *region* instead of interest point.

When a local neighbourhood of pixels is used to describe an interest point, the feature extraction process has to determine not only the location of the interest point, but also the size and possibly the shape of this local neighbourhood. Especially in case of geometric deformations, this significantly complicates the process, as the size and shape have to be determined in an invariant way, i.e. such that the local features cover the same part of the object surface even when the scale or viewpoint changes.

Since we assume the object surface is locally planar, the same set of image transformations derived in the previous chapter can be reused. We start with a detector that is only translation and rotation invariant, the Harris corner detector (see section 1.2). Next, we discuss how to deal with scale changes (see section 1.3). Finally, we discuss an example of an affine invariant interest point detector, namely MSER (see section 1.4). As we will see, the more invariance

is included, the more complex the feature extraction procedure becomes. This reduces the robustness of the detection process. For that reason, local feature detectors invariant to projective transformations are of little use. Moreover, on a local scale, the perspective effects can usually be ignored and the projective transformation can be approximated quite well by an affine transformation.

Various methods for local feature extraction have been proposed. Desired properties of a good method include

- *Repeatability:* The same feature (covering the same part of the object or scene) should be found in several images in spite of geometric and photometric transformations. This can be achieved through invariance.
- *Distinctiveness:* Each feature should have a distinctive description, that makes it possible to discriminate it from other features.
- *Compactness and efficiency:* Especially when dealing with realtime applications and/or large databases it is important that the needed computation time and memory consumption are not too high.
- *Locality:* A feature should occupy a relatively small area of the image, to make the method robust to clutter and occlusions. Locality also helps to satisfy the planarity constraint.

Locality and distinctiveness are somewhat contradicting properties, since smaller regions contain less information and as a result will also be less discriminative. So a compromise needs to be made, with features large enough to be distinctive yet small enough to avoid problems with non-planarities, occlusions and image clutter.

The same also holds for invariance versus distinctiveness: affine invariant features are typically less discriminative than scale-invariant features, which are in turn less discriminative than features that are only rotation-invariant. For instance, a rectangle and a square cannot be distinguished when using affine invariance, but are clearly distinct when only considering invariance under similarity transformations.

When using local features, the following three processing steps are usually required:

1. *Local feature extraction*: For each image, determine the position and possibly scale/shape of the local features. Note that this step is applied to each image separately.
2. *Local feature description*: For each of the extracted local features, describe the intensity pattern contained by the local feature in an invariant and robust way. This gives a *feature vector* for each local feature. Again, this step uses only information from a single image at a time.
3. *Local feature matching*: Finally, the feature vectors belonging to different local features are compared. *Correspondences* (or matches) between two images can be found by looking for local features with similar feature vectors (i.e. with a small distance between their feature vectors).

We will first describe three different methods for local feature extraction, with different levels of invariance. Next, we will explain feature description and feature matching.

1.2 Harris corner detector

The Harris corner detector is designed to find distinctive local features that are invariant under translations and rotations only. The basic idea underlying this method is that a good feature (local image patch) should be clearly distinctive from its neighboring patches. In other words, shifting the window in *any* direction (u, v) should give a large change in the intensity pattern. Approximating this change in intensity pattern using a Taylor expansion up to first order derivatives, we get:

$$E(u,v) = \sum_{(x,y)\in W} (I(x+u, y+v) - I(x,y))^2 \tag{1.1}$$

$$\simeq [u \;\; v] \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{1.2}$$

$$\simeq [u \;\; v] M \begin{bmatrix} u \\ v \end{bmatrix} \tag{1.3}$$

The matrix M is the *second moment matrix*, also called the auto-correlation matrix, and is often used for feature detection and for describing local image structures. It describes the gradient distribution in a local neighborhood of a point.

Usually, the sum over a fixed window W is replaced by a convolution with a Gaussian, and also the derivatives are computed with a Gaussian kernel, to suppress noise. This gives:

$$M = \sigma_D^2 \, g(\sigma_I) * \begin{bmatrix} I_x^2(\mathbf{x}, \sigma_D) & I_x I_y(\mathbf{x}, \sigma_D) \\ I_x I_y(\mathbf{x}, \sigma_D) & I_y^2(\mathbf{x}, \sigma_D) \end{bmatrix} \tag{1.4}$$

$$with \qquad I_x(\mathbf{x}, \sigma_D) = \frac{\partial}{\partial x} g(\sigma_D) * I(\mathbf{x}) \tag{1.5}$$

$$g(\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.6}$$

σ_D is referred to as the *differentiation scale* and σ_I as the *integration scale.*

The eigenvalues of this matrix represent the principal signal changes in two orthogonal directions in a neighborhood around the point defined by σ_I. When both eigenvalues are small, the intensity pattern remains almost unchanged - in other words, the window lies in a more or less uniform, 'flat' image area. When one eigenvalue is small and one eigenvalue is large, this means the intensity pattern changes a lot in one direction, but remains almost unchanged in the direction perpendicular to it. This happens typically for windows lying on an edge in the image. Both eigenvalues are large at locations in the image for which the image signal varies significantly in both directions. These are points that are well localized and (at least on a local scale) distinctive, so these make good features.

In practice, Harris proposed to use the following measure for *cornerness*, which combines the two eigenvalues in a single measure and is computationally less expensive:

$$cornerness = det(M) - \lambda \, trace(M) \tag{1.7}$$

with $det(M)$ the determinant and $trace(M)$ the trace of the matrix M. A typical value for λ is 0.04. Since the determinant of a matrix is equal to the product of its eigenvalues and the trace corresponds to the sum, it is clear that high values of the cornerness measure correspond to both eigenvalues being large. Adding the second term with the trace reduces the response of the operator on strong straight contours. Moreover, computing this measure based

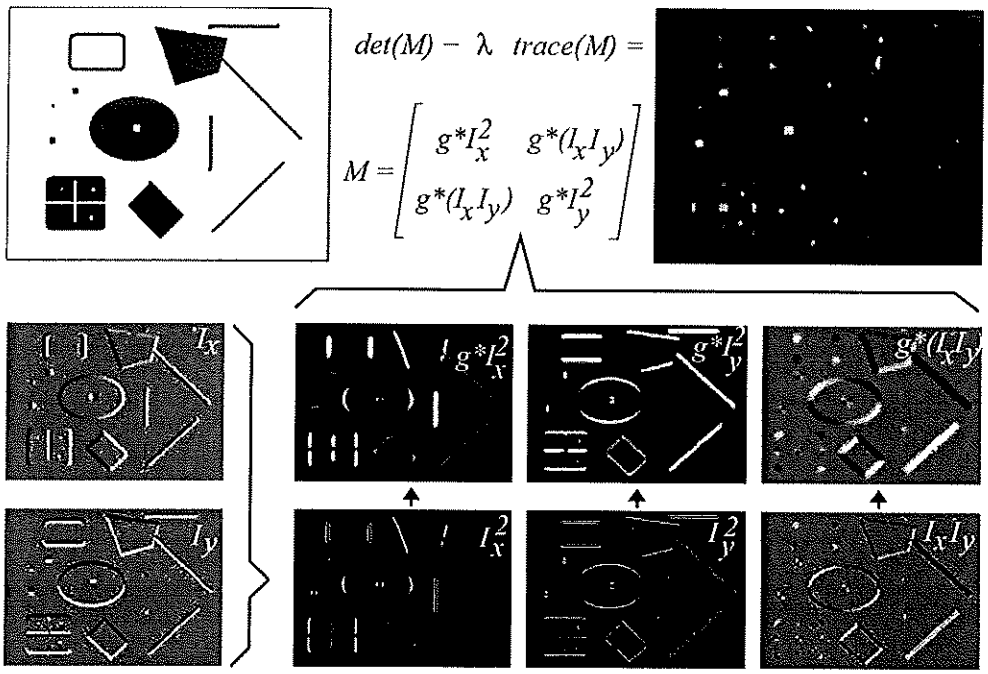


Figure 1.1: **Illustration of the components of the second moment matrix and Harris cornerness measure.**

on the determinant and the trace is computationally less demanding than actually computing the eigenvalues. This seems less relevant now, but it was important back in 1988 when the computational resources were still very limited.

All stages of the corner extraction process are illustrated in figure 1.1. Given the original image $I(x, y)$ (upper left), the first step consists of computing the first order derivatives I_x and I_y (lower left). Next, one takes the product of these gradient images (lower right). Then, the images are smoothed with a Gaussian kernel. These images contain the different elements of the second moment matrix, which are then in a final step combined into the cornerness measure, following equation 1.7 (upper right).

When used as an interest point detector, local maxima of the cornerness function are extracted, using non-maximum suppression. Such points are translation and rotation invariant. Moreover, they are stable under varying lighting conditions. In a comparative study of different interest point detectors, the Harris corner was proven to be the most repeatable and most informative. Additionally, they can be made very precise: sub-pixel precision can be achieved through quadratic approximation of the cornerness function in the neighborhood of a local maximum.

Figure 1.2 shows the corners detected with this measure for two example images related by a rotation. Note that the features found correspond to locations in the image showing two dimensional variations in the intensity pattern. These may correspond to real 'corners', but the detector also fires on other structures, such as T-junctions, points with high curvature, etc. As can be seen in the figure, many but not all of the features detected in the original image (left) have also been found in the rotated version (right). In other words, the *repeatability* of the Harris detector under rotations is high.

Additionally, features are typically found at locations which are *informative*, i.e. with a

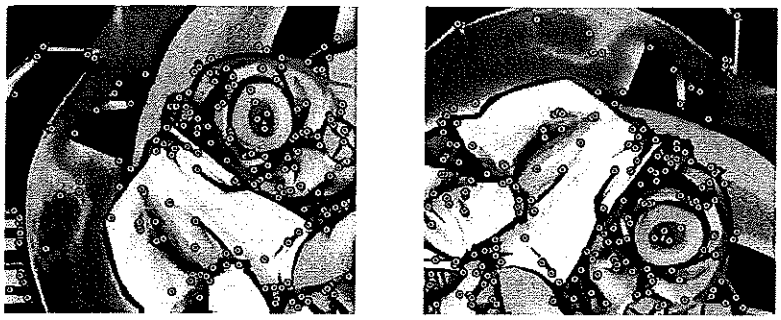


Figure 1.2: **Harris corners detected on rotated image examples.**

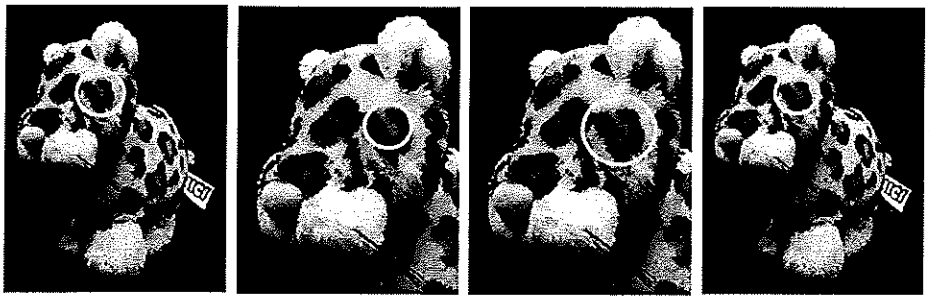


Figure 1.3: **Scale-invariant features (2 rightmost images) correspond in spite a change in scale. This does not hold for features of fixed size (2 leftmost images).**

high variability in the intensity pattern. This makes them more discriminative and easier to bring into correspondence.

1.3 Dealing with scale changes

The approach of the previous section gives points that are invariant to translations and rotations. In this section we describe a method that takes the exploitation of invariance one step further and allows the same features to be found also for images at different resolution, zoom, or distance to the object.

The basic idea is illustrated in figure 1.3. The features in the two leftmost images have the same size, but do not correspond. The features in the two rightmost images, on the other hand, have different sizes yet cover the same physical structure. The challenge of scale-invariant local feature detection is how to determine the position and size of the features in such a way that similar features are found again after applying a scale change.

Suppose we have an image I_1 and extract local features $\{f_{1i}\}$ using a scale invariant local feature extraction method. Suppose we now rescale this image to generate a new image I_2 (e.g. by zooming in on a detail, or changing the resolution of the image) and also extract local features for this second image, giving us a set $\{f_{2i}\}$. Two corresponding features f_{1a} and f_{2b} will typically *not* have the same image coordinates. Likewise, they will typically *not* have the same size (scale). Their image coordinates (position) and scale have been selected based on the algorithm for scale-invariant local feature detection, and this algorithm has been conceived in such a way that features are typically found at corresponding locations and corresponding scales, in spite of the geometric deformations. How this can be done is explained in the remain-

der of this section.

It is important to note that the local features have been extracted from each image *independently*. This is important, since we do not know the correspondences beforehand, and searching over the entire image to find a corresponding location and scale would be computationally too expensive. Local features should be selected in a given image, without any a prior knowledge about the corresponding patches in the model image(s). Putting the same remark differently, the problem is not finding physically corresponding patches given a pair of images, but coming up with patches in an image and then, given another view of the same scene, coming up with physically corresponding patches solely based on this second image.

1.3.1 A multiscale approach

The Harris corner detector described in the previous section extracts features at a single scale (determined by the integration and derivation scales). To deal with scale changes, a straightforward approach consists of extracting features over a range of scales and using *all* these features together to represent the image. This is referred to as a *multi-scale* approach.

In practice, this is implemented as follows. First, a scale space pyramid is constructed (see also chapter ?? on scale space representations), generating several lower resolution versions of the image by convolving the image using low pass filters (typically Gaussians) followed by subsampling/interpolation. Next, features are extracted from all the different scale levels.

While simple, this approach obviously also has its shortcomings. It cannot cope well with the case where a local image structure is present over a range of scales, resulting in multiple interest points being detected at each scale within this range. As a consequence, there are typically many overlapping points, representing the same structure, but with slightly different localization and scale. The high number of points increases the ambiguity and the computational complexity of matching and recognition.

1.3.2 Automatic scale selection

To overcome the problem of many overlapping detections, typical of multiscale approaches, *scale invariant* methods have been introduced. These automatically determine both the location and scale of the local features. Features are typically circular regions, in that case.

This can be achieved by searching for maxima of some function over scales. This is known as *automatic scale selection*. The idea is to select the *characteristic* scale of a local structure, for which a given function attains an extremum over scales (see figure 1.4). The selected scale is characteristic in the quantitative sense, since it measures the scale at which there is maximum similarity between the feature detection operator and the local image structures. The size of the region is therefore selected independently of the image resolution for each point.

A popular choice for the function to be maximized is the normalized Laplacian-of-Gaussian (LoG)

$$LoG(\mathbf{x}, \sigma) = I_{xx}(\mathbf{x}, \sigma) + I_{yy}(\mathbf{x}, \sigma) \tag{1.8}$$

$$with \quad I_{xx}(\mathbf{x}, \sigma_D) = \frac{\partial^2}{\partial x} g(\sigma_D) * I(\mathbf{x}) \tag{1.9}$$

$$g(\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.10}$$

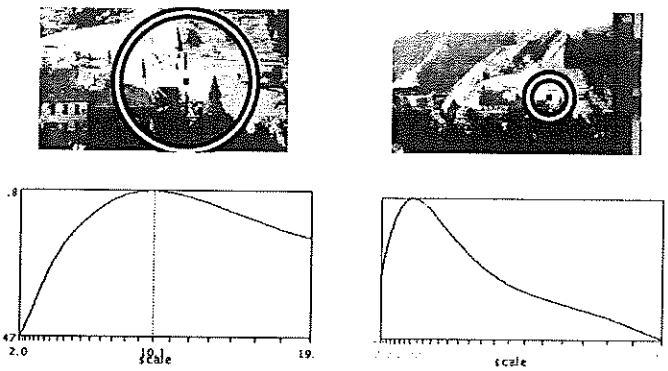


Figure 1.4: **Example of characteristic scales. The top row shows images taken with different zoom. The bottom row shows the responses of the Laplacian over scales for two corresponding points. The characteristic scales are 10.1 and 3.9 for the left and right image, respectively. The ratio of scales corresponds to the scale factor (2.5) between the two images. The radius of displayed regions in the top row is equal to 3 times the selected scales. (Image courtesy of K. Mikolajczyk)**

The LoG operator is circularly symmetric. It is therefore naturally invariant to rotation. It is also well adapted for detecting blob-like structures.

Scale and position can then be determined simultaneously, by looking for local maxima in the 3D scale space (with spatial dimensions x and y and scale as a third dimension). This procedure is schematically visualized in figure 1.5.

In practical implementations, the interest point criterion that is being optimized for localization is not always the same as the one used for optimizing the scale. For instance, the *Harris-Laplace* detector combines the measure used in the Harris detector to localize the feature with the Laplacian-of-Gaussian operator for scale selection. In these methods, position and scale are iteratively updated until convergence.

Figure 1.6 shows the scale-invariant local features obtained by applying the Harris-Laplace detector, for two images of the same scene related by a scale change. In order not to overload the images, only some of the corresponding regions that were detected in both images are shown.

1.4 Affine invariant local features

To deal with larger viewpoint changes, invariance to scale changes is not enough. That is why also methods for *affine invariant* local feature extraction have been proposed.

Since the patches are considered to be approximately planar and sufficiently small, affine deformation is a good model for the way in which such patches transform when viewed from different directions. Using affine features then allows the same reference view to be used for a wider set of viewpoints.

The crux of the matter is again the selection of the patches. Simple circular shapes can no longer be used, as a patch that is viewed as a circular patch when viewed head-on has to be deformed to an elliptical shape when viewed obliquely. Hence, elliptical shapes would be a better choice. Parallelograms are also possible. The problem now is to come up with ellipse- or parallelogram-shaped patches that change shape with changing viewpoint so as to ensure that they cover the same, physical part of the scene.

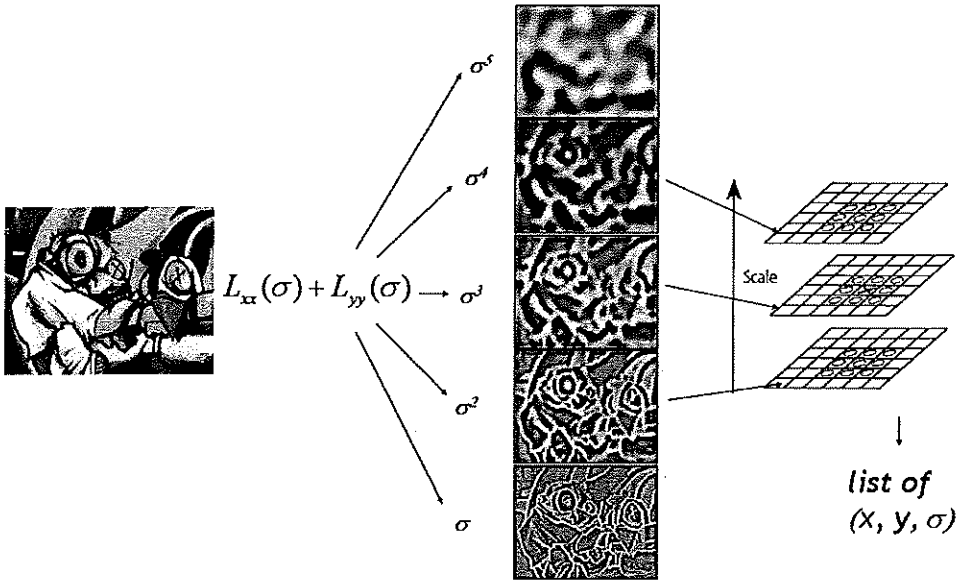


Figure 1.5: Automatic scale selection based on the Laplacian-of-Gaussians. (Image courtesy of K. Mikolajczyk)

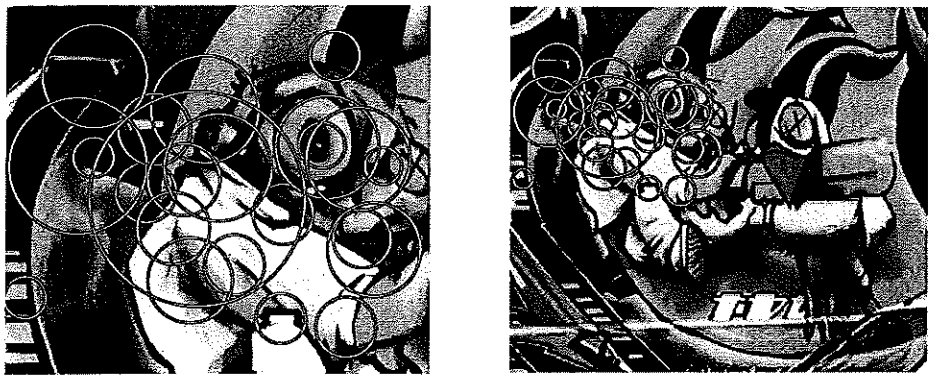


Figure 1.6: Corresponding features found with the Harris-Laplace detector. Only a subset is displayed to avoid clutter. The circles indicate the scale of the features.

The method of automatic scale selection can, in principle, also be generalized to affine transformations, using an affine scale space. However, searching for local maxima when dealing with more parameters may become cumbersome. Therefore, it's typically done in stages: first determine the location (e.g. using the Harris cornerness measure), and then estimate the size and shape of the local feature, followed by iterative refinement of all parameters. We will not discuss these methods in detail here. Instead, we describe another method to obtain affine invariant local features, known as MSER or Maximally Stable Extremal Regions.

1.4.1 Maximally Stable Extremal Regions

A *Maximally Stable Extremal Region* is a connected component of an appropriately thresholded image. The word 'extremal' refers to the property that all pixels inside the MSER have either higher (bright extremal regions) or lower (dark extremal regions) intensity than all the pixels on its outer boundary. The 'maximally stable' in MSER describes the property optimized in the threshold selection process (see below).

The set of extremal regions $\mathcal{E}$, i.e., the set of all connected components obtained by thresholding, has a number of desirable properties. Firstly, a monotonic change of image intensities leaves $\mathcal{E}$ unchanged (since it depends only on the ordering of pixel intensities which is preserved under monotonic transformation). This ensures that common photometric changes modelled locally as linear or affine leave $\mathcal{E}$ unaffected, even if the camera is non-linear (gamma-corrected). Secondly, continuous geometric transformations preserve topology – pixels from a single connected component are transformed to a single connected component. Thus after a geometric change locally approximated by an affine transform, homography or even continuous non-linear warping, a matching extremal region will be in the transformed set $\mathcal{E}'$. Finally, it can be shown that there are no more extremal regions than there are pixels in the image. So a set of regions has been defined that is preserved under a broad class of geometric and photometric changes and yet has the same cardinality as e.g. the set of fixed-sized square windows commonly used in narrow-baseline matching.

The enumeration of the set of extremal regions $\mathcal{E}$ is very efficient, almost linear in the number of image pixels. This is done as follows. First, pixels are sorted by intensity. After sorting, pixels are marked in the image (either in decreasing or increasing order) and the list of growing and merging connected components and their areas is maintained. During the enumeration process, the area of each connected component as a function of intensity is stored. Among the extremal regions, the 'maximally stable' ones are those corresponding to thresholds for which the relative area change as a function of relative change of threshold is at a local minimum. In other words, the MSER are the parts of the image where local binarization is stable over a large range of thresholds. This definition of MSER stability based on relative area change is invariant to affine transformations. Consequently, the process of MSER detection is affine invariant.

Detection of MSER is related to *thresholding*, since every extremal region is a connected component of a thresholded image. However, no global or 'optimal' threshold is sought, all thresholds are tested and the stability of the connected components evaluated. The output of the MSER detector is not a binarized image. For some parts of the image, multiple stable thresholds exist and a system of nested subsets is output in this case.



Figure 1.7: **Regions detected with MSER on the graffiti images (subset).**

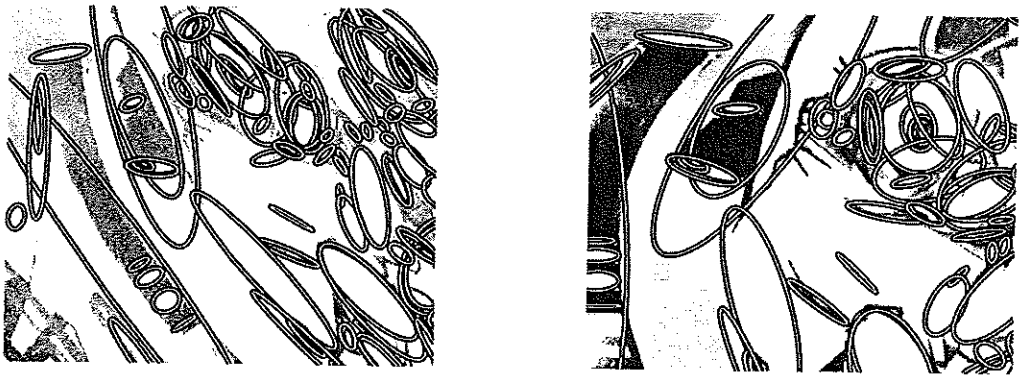


Figure 1.8: **Final MSER regions for the graffiti images (subset).**

Examples of the regions so detected are given in figure 1.7. These arbitrarily shaped regions may not be very practical. Therefore, they are usually replaced by ellipses fitted to them. This results in a set of features as shown in figure 1.8.

Discussion The MSER features typically anchor on region boundaries, thus the resulting regions are accurately localized. The method works best for structured images which can be segmented well – ideally an image with uniform regions separated by strong intensity changes. On the downside, it was found to be sensitive to image blur, which can be explained by the fact that image blur undermines the stability criterion. The method is also relatively efficient.

1.5 Local feature descriptors

Now that we have different methods available for the extraction of local features, the next question is: "How can we best describe the intensity pattern within such a local feature?". This is a necessary step to be able to find corresponding features. It is usually split in two steps. First, we normalize for the geometric deformations, by mapping the local feature onto a feature with fixed shape and size. Next, we compute the actual descriptor.

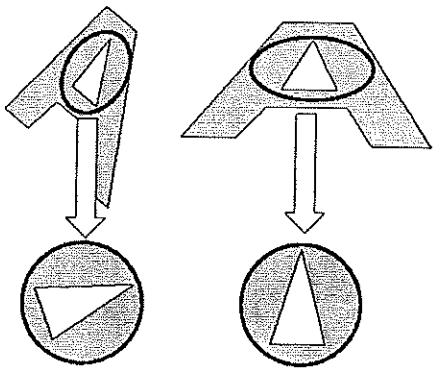


Figure 1.9: **Geometric normalization to compensate for geometric deformations.**

1.5.1 Geometric normalization

The easiest way to compensate for the geometric deformations is by applying an additional transformation that maps the area covered by the local feature onto a reference region with fixed size and shape (see figure 1.9). This can e.g. be a circular region of say 16 pixels radius.

For scale-invariant features, this means we have to locally rescale the image. When the local feature is smaller than the reference region, we need to interpolate between nearby pixels. When the local feature is larger than the reference region, we need to downsample (after smoothing the image appropriately, to avoid aliasing effects).

For affine invariant features, we need to determine the affine transformation mapping the ellipse defining the local feature's shape onto the circle. This can be done by selecting e.g. the center point, a point on the ellipse's major axis and a point on the ellipse's minor axis and mapping these onto $(0,0)$, $(0,1)$ and $(1,0)$ respectively (yielding six equations for the six unknown parameters of the affine transformation).

Compensating for in-plane rotations The procedure described above does not compensate for all the geometric deformations. Indeed, when normalizing two corresponding local features to a circular region of fixed size, the patterns so obtained can still be rotated with respect to each other. This is usually solved by determining the dominant orientation within the image patch, and rotating the image patch such that the dominant orientation becomes aligned with the x-axis. In practice, the dominant orientation is determined by first computing the gradient in each pixel of the local image patch, then building a histogram of gradient orientations, and checking which orientation bin gets most votes.

Photometric normalization Apart from the geometric deformations, also photometric transformations need to be taken into account. All three methods for feature extraction described above are also invariant to linear changes in intensity (i.e. when the change in intensities can be modeled with a constant scalefactor and offset). It is possible to normalize for these photometric transformations, by applying an additional linear transformation, e.g. such that the mean intensity becomes 0.0 and the standard deviation 1.0. However, in practice, one usually sticks to the original intensities, and compensates for the photometric transformations e.g. by using gradients or differences of image intensities (both of which compensate for the offset) or by using ratios of intensities or ratios of intensity gradients (which compensates for the scalefactor).

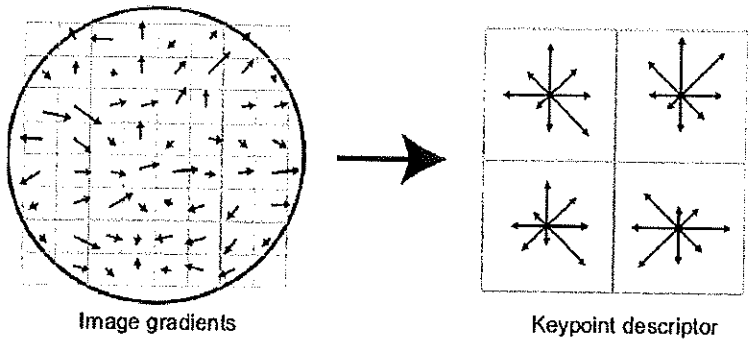


Figure 1.10: **Illustration of the SIFT descriptor (for simplicity, only a grid of 2×2 cells is shown).**

1.5.2 Normalized crosscorrelation

Now that the geometric deformations have been compensated for, the easiest way to obtain a descriptor would be to simply concatenate the different pixel values. Such feature vectors can then be compared using normalized cross-correlation, i.e. by checking if there is a linear relation between the intensity patterns of the two local features being compared.

However, this method has proven to be very sensitive to small misalignments. So instead, more robust descriptors have been introduced.

1.5.3 SIFT

SIFT is the most widely used local feature descriptor. Its name is an acronym for 'Scale Invariant Feature Transform', but that actually refers to the related scale-invariant feature detector, which we won't discuss here. SIFT is a carefully designed descriptor, that tries to capture the essence of the intensity pattern while at the same time to be robust for misalignments, noise, and unmodelled photometric and geometric deformations (e.g. due to non-planarities, or due to affine deformations when using a scale invariant detector).

The basic idea underlying SIFT is the use of local histograms of gradient orientations. The 360 degrees range over all orientations is discretized into 8 intervals of 45 degrees each. For each pixel, the gradient is computed. Then, a histogram can be built by counting how many gradients fall within each interval.

To make the descriptor more informative, SIFT does not compute a single histogram of gradient orientations, but multiple of them. To this end, the image patch is divided in a number of cells, and for each cell a local histogram of gradient orientations is computed. In practice, usually a grid of 4×4 cells is used (see also figure 1.10). The descriptor then becomes a vector of all the values of these histograms. Since each histogram has 8 orientation intervals, this results in a final descriptor of $4 \times 4 \times 8 = 128$ dimensions.

To make this procedure more robust, the histogram construction is modified in a number of ways, as follows:

- The contribution of each gradient is *weighted by the gradient magnitude.* This takes into account that strong edges are more stable and their orientation is defined more accurately, while the orientation of weak edges is mostly determined by noise.
- The contribution of each gradient is not assigned to a single orientation interval but split over two neighboring intervals, using *bilinear weighting.* Suppose that for a gradient with

magnitude g and orientation θ the centers of the two nearest orientation intervals are orientations θ_i and θ_{i+1}. Then the weight for interval i is computed as $g\frac{\theta_{i+1}-\theta}{\theta_{i+1}-\theta_i}$ and the weight for interval $i+1$ is computed as $g\frac{\theta-\theta_i}{\theta_{i+1}-\theta_i}$. This limits the discretization error: suppose the orientation of a strong edge is close to the boundary between two orientation intervals. Without bilinear interpolation, a small rotation would have a very big impact on the descriptor. With bilinear interpolation, on the other hand, the effect of the strong edge will be split almost equally over the two intervals and a small rotation will have only a small impact. The same bilinear weighting scheme is also applied spatially, i.e. a gradient contributes not just to one local histogram, but it's contribution is spread over the four nearest cells (two nearest for each dimension). This again reduces discretization effects for both the horizontal and vertical subdivisions.

- The contribution of each gradient is additionally *weighted by a Gaussian* centered on the image patch and with σ equal to one half the width of the descriptor window. As a result, pixels towards the boundaries of the patch have a relatively small contribution. This makes the descriptor more robust to small misalignments (translations or scale changes) that cause these pixels to fall outside of the patch.

- Finally, the feature vector is *normalized to unit length*, making it invariant to linear photometric transformations. Moreover, to avoid one orientation bin to dominate the descriptor, a threshold of 0.2 is applied to all elements of the normalized descriptor, after which it is again normalized to unit length.

The careful design of the SIFT descriptor makes it a very powerful tool in finding correspondences between images. The robustness built into this descriptor allows to deal with quite some geometric and photometric deformations, and to compensate for inaccuracies in the feature detection stage. In fact, robustness can often replace invariance. For instance, as long as the expected viewpoint changes are not extreme, using a scale-invariant detector combined with a robust descriptor can outperform an affine-invariant detector.

1.6 Local feature matching

Now that we have methods to extract local features and a method to describe them in a robust way, we are ready to find correspondences between two images of the same object or scene. This procedure consists of the following steps:

1. Extract the local feature for the first image.
2. For each local feature extracted in the first image
   - Map the patch onto a reference patch of fixed size and shape to compensate for the geometric deformations.
   - Find the dominant orientation and rotate the patch accordingly.
   - Compute the SIFT descriptor, resulting in a 128 dimensional feature vector.
3. Repeat steps 1 and 2 for the second image.
4. For each local feature in the first image
   - Find the most similar feature in the second image.
   - If the distance to this feature is below a threshold value, accept it as a match.
5. Verify the geometric consistency of the correspondences.

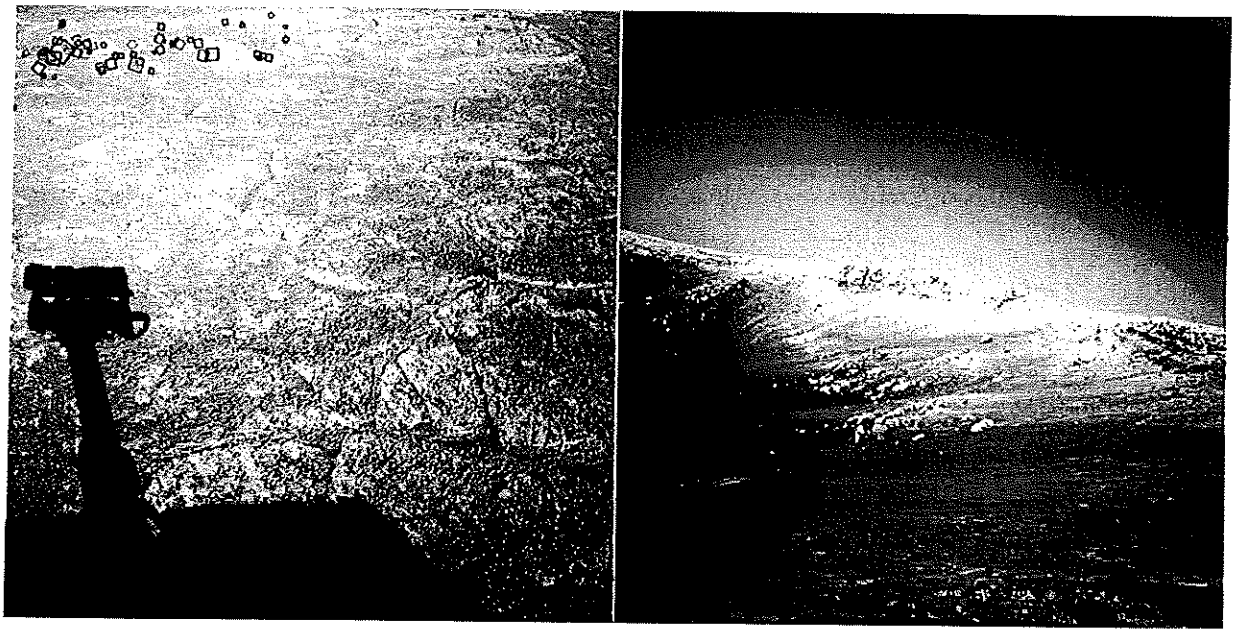


Figure 1.11: **Correspondences found with a scale-invariant local feature detector combined with SIFT (images courtesy of N. Snavely).**

To find the most similar feature in the second image (step 4), SIFT descriptors are compared using Euclidean distance. Efficient methods for nearest neighbor search can be used to speed up this step. Instead of accepting the closest feature if the distance is below a threshold, an alternative that is often used consists of comparing the distance to the closest feature with the distance to the second closest feature. If the closest feature is signficantly closer than the second closest, this provides evidence that it is actually a correct match. If both features are almost equally close, this indicates that there is some uncertainty about the match. So instead of thresholding the distance to the closest feature, one often applies a threshold to the ratio of the distance to the closest feature and the distance to the second closest feature.

Step 5 is crucial to reject incorrect matches. Since local features describe only a relatively small part of the image, wrong correspondences are difficult to avoid. Moreover, scenes often contain repetitive structures, such that local appearance alone contains insufficient information to decide on what is a correct match. Therefore, as a final verification step, the geometric consistency of the correspondences needs to be checked. This can be done e.g. by exploiting epipolar geometry (i.e. constraints on 3D points projected in two different images) or by imposing the points are lying on a planar surface (e.g. a building facade) and checking whether they are consistent under a projective transformation. How this can be done, in a manner that is robust to the outliers, will bring us too far and falls beyond the scope of this text.

Figure 1.11 shows an example of two images and the correspondences found between them, using a scale-invariant feature detector combined with SIFT.

Local features are not only useful for object recognition, but have been applied to various computer vision applications, including image mosaicing, robot navigation, image retrieval, etc. For some example applications, the interested reader can check the following websites:

http://www.cs.ubc.ca/ mbrown/autostitch/autostitch.html, or

http://www.robots.ox.ac.uk/ vgg/research/vgoogle/ .

Chapter 2

Object recognition

In this chapter we discuss approaches to *object recognition*. This is, together with scene and action recognition, one of the main aspects of *image understanding* or *image interpretation*. Given an image (or video) we want to find out what it represents, i.e. looking behind the pixels at the actual image content, as humans do when looking at a picture. Since scenes are usually composed of objects, recognizing objects plays a crucial role in this endeavor.

2.1 Introduction

2.1.1 Challenges

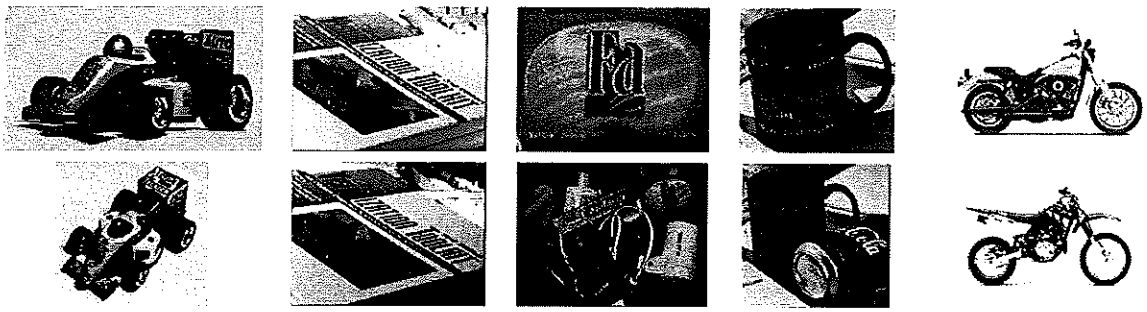
Recognizing objects in a scene is quite a challenging task, and is, even after almost 50 years of research, still not completely solved. In our daily life, we recognize objects, places and faces almost instantaneously and without much effort. As a result, the complexity of this process is often underestimated. Figure 2.1 shows some of the challenges we're confronted with when building a system for object recognition:

- *changes in object pose:* Changes in the relative position and orientation between the camera and the object result in local deformations as well as scale changes. Moreover, apart from these geometric transformations, we also have to cope with different parts of the objects becoming visible/unvisible.

- *changes in illumination:* Photometric changes are caused by changes in the lighting conditions (e.g. shadows, inside or outside, sunny or cloudy, or simply interreflections of light on nearby surfaces) as well as by changes in the geometry (e.g. under diffuse illumination, the part of a uniform sphere facing the camera will appear brighter than the parts oriented at larger angles to the camera). The human visual system is very good in compensating for these effects. However, for a computer system this is quite complex, making color or intensity information not very reliable.

- *image clutter:* Other objects may be present in the scene, distracting the attention of the system. This is especially harmful when using global image descriptors. Local features are, by their very nature, more robust to image clutter, as at least some of them tend to lie completely on the object, unaffected by the background.

- *occlusions:* Other objects in the scene can also occlude part of the object, i.e. hide it from the camera. This results in partial visibility. Again, local features are key to deal with this situation.



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

Figure 2.1: **Some of the sources of variability in appearance of objects: a) changes in viewpoint/object pose, b) changes in illumination, c) image clutter, d) occlusions, e) within-class variability**

- *noise, discretization errors, compression artefacts, motion blur, out-of-focus, ...* The lower the image quality, the harder it gets to recognize the objects it contains. This is often a limiting factor in applications such as surveillance.
- *within-class variability* When recognizing a class of objects, another important source of variability is the variation that can be observed between different instances of the same object class. This is an especially complicating factor, since it cannot be modeled mathematically, but needs to be learned from training data.
- *non-rigid deformations:* Another complicating factor is the fact that some classes are not rigid, but can deform - think e.g. of animals or scissors, further complicating the recognition task.

Dealing with within-class variation is a particularly challenging issue. As a result, research into object recognition is usually divided into two main subtasks: recognition of *specific objects* (e.g. 'my car') and recognition of object classes (e.g. 'a car'). The latter is also referred to as *category-level object recognition.* Different techniques have been developed for these two cases.

Also, one often makes a distinction between the recognition of 2D objects (e.g. planar contours, but also CD covers, film posters, or building facades) and the recognition of 3D objects. Recognizing planar shapes is easier than recognizing 3D shapes since for 2D objects no information is lost during the projection from the 3D world onto the image plane, and the geometric deformation due to viewpoint changes can be modelled accurately (see previous chapter on image transformations for planar surfaces). For 3D objects, on the other hand, much of the problems arise from the fact that information is lost under projection, which is a non-invertible operation. Any two-dimensional projection could have arisen from an infinity of three-dimensional scenes. Under very simple viewing conditions the effect of the loss of information can be kept to a minimum. Such conditions are typical for machine vision systems in industry. For instance, if the object is completely flat and no surface characteristics have to be judged, all shape information can be obtained from a binary silhouette image if looked from a perpendicular direction. Backlighting combined with thresholding yields a simple and robust basis for recognition if the environment can sufficiently be constrained. Here we will focus on more advanced methods that are applicable under more general conditions.

2.1.2 A taxonomy

Usually the goal is not only to recognize objects in the image, but also to determine their pose, i.e. their (3D) position and orientation. As we will see the complexity of the recognition problem depends on whether the 'what' and 'where' problems can be separated. Recognition of an isolated object against a homogeneous background is far easier than recognition in cluttered environments. If one would be able to segment the foreground object from the background, the subsequent recognition task would be significantly simplified. Unfortunately, image segmentation by itself is a very challenging task as well, and usually can only be performed accurately under very limiting assumptions (e.g. if the object has a uniform color that clearly distinguishes it from the background) or if information about the object class can be exploited (e.g. knowing which class the foreground object belongs to). Hence, image segmentation and object recognition go hand in hand and should be solved simultaneously.

As such, we can distinguish different levels of object recognition tasks:

- **Object categorization:** Here the task only looks at the 'what' part of the question: *"Does this image contain the object ?"*.
- **Object classification:** This is similar to object categorization, except that it is known beforehand that the object belongs to one of a fixed set of objects / object classes: e.g. *"Is this an image of a dog or a cat ?"*.
- **Object detection:** Here, we do not only want to know if the image contains the object, but also where in the image the object is found: *"Where are the cars in this image ?"*. Usually, the objects found are delineated with a *bounding box*. This is a rectangular window containing the object and fitting tightly around it. This problem is sometimes also referred to as *object localization*.
- **Object segmentation:** This goes one step further than mere object detection, indicating at the pixel level where the object can be found: *"Which pixels belong to the object?"*.
- **Pose estimation:** While solving any of the tasks above, one often needs to determine the pose of the object as well, since the appearance of an object depends a lot on its pose (think e.g. of side views of cars vs. frontal views). This is referred to as pose estimation.

A method that is often used for object detection consists of trying out all possible bounding boxes, and for each and everyone of them check whether they contain the object or not. This effectively turns the object detection task into multiple object classification tasks. One usually refers to this method as a *sliding window* approach. Typically, when the object is present in the image, many overlapping detections will give rise to positive classifications, so as a postprocessing step a non-maximum suppression step is needed (i.e. find local maxima of the classification score and keep only these as final detections).

Finally, it should be noted that the image content outside of the bounding box may also contain useful information that can help in the object recognition task. This is referred to as *context*. For instance, when recognizing cows, the grass surrounding it can be a useful cue. Likewise, the shadow beneath a car does not actually belong to the object, but still provides a discriminative feature that can be exploited for recognition. Going a step further, higher level information can be included as well, e.g. the presence of other objects, the 3D geometric layout (e.g. the fact that objects are not floating around, but should be on the ground plane), etc.

2.2 A brief history of object recognition

Object recognition has been studied since the late sixties and is one of the main research goals of computer vision. Generally speaking, two different schools can be distinguished: model-based approaches and appearance-based approaches. At first, appearance-based approaches were limited to template matching, achieving good results in controlled environments. Model-based approaches, on the other hand, seemed most promising for generic objects. Yet in the end, it is the appearance-based approach that has led to most breakthroughs and progress. Both will be discussed from a historical point of view in the next two sections.

2.2.1 Model-based approaches

Approach

The classical, model-based approaches to object recognition start from an explicit (3D) model of an object's shape. 2D features or primitives such as lines, holes, circular segments, etc. are extracted from the image and matched to the 3D model of the object, taking into account the projection from 3D to 2D. Alternatively, 3D volumetric primitives (e.g. spheres, parallelepipeda) can be extracted directly from the image to be matched with the various models. These primitives can then be used as the nodes in a graph, where the edges indicate adjacency relations possibly with additional labels indicating concepts like "to the right of", "below", etc. Recognition in those cases is a question of (sub)graph matching. Both bottom-up and top-down implementations have been used, mostly following a hypothesize-and-test paradigm.

The *top-down* or *model-driven* version works as follows. Assuming that both the model and the scene are composed of features such as vertices, holes, straight lines, circular arcs, etc. and a specified object is searched for, then one knows in advance what types of features to look for. This means that problem-specific information is used to reduce the feature search in the image. After making an initial feature match hypothesis, the arrangement of other features in the model is used to verify the assumption: it is checked whether the other features in the model can also be found at the predicted places and in the appropriate combinations (according to some syntactic rules and with the possibility that they are occluded). Notice that an assumption is not only about type but also about pose. If additional features are in place, i.e. if they are found at the predicted positions this reinforces the assumption and a refined hypothesis about type and pose can be made. The algorithm cycles through this *hypothesize-and-verify* loop. Looking selectively for a few, specific intial features is an interesting strategy if one only wants to find a piece of a certain type without being interested in the other objects present.

Bottom-up or *data-driven search* is a second strategy. First, all features in the scene are searched. No problem-specific knowledge is used at that stage. That also means that features will be searched which are not relevant to the application. Then a particularly salient one is selected and all possible models to which it could belong are listed. By considering several features more and more false candidate models can be eliminated. Again, the recognition-localization is realized through the hypothesize-and-verify paradigm. Bottom-up approaches can be interesting if it is not specified which type of object to look for or when all objects in the scene should be recognized. This approach tries to minimize the time needed to analyze the total scene, rather than to minimize the time needed to find an object of a predefined type. Also hybrid forms have been proposed, which combine top-down and bottom-up techniques.

Discussion

The main drawback of the model-based approaches is that they firmly rely on the feature extraction, which is often a vulnerable, error-prone step. Also, this approach is only feasible for special object classes, composed of primitives that allow easy and robust detection. In the case of invariant based descriptions, only limited classes of shapes can be dealt with. Moreover, the object model needs to be available. For complex and natural objects, this may be problematic.

2.2.2 Appearance-based approaches

In contrast to the more classic model-based approaches, *image-based* or *appearance-based approaches* to object recognition have become increasingly popular. Instead of representing an object by some abstract, often three-dimensional model, they simply model an object by a set of images thereof. In this way, they directly exploit what *can* actually be seen in the image instead of what the system developer *hopes* that will be seen, based on the 3D model and some theoretic assumptions on the image formation process.

Template matching

The simplest example of an appearance-based approach is *template matching*. This technique has been (and still is) widely used, especially in visual inspection applications.

Template matching is only used with simple changes in object pose, e.g. pure translations of the shape in the image. Template matching consists of comparing, usually by correlation or subtraction techniques, sections of the image with images of the objects one is looking for. The image(s) of the object(s) are called template(s). Hence, the 'model' of an object is simply its image.

In order to find the pattern corresponding to the object, the template is shifted over the image until both overlap precisely. In its general form, this approach is very time consuming, especially if also the orientation (2D) or scale of the pattern are unknown. Then the process has to be repeated for several orientations or scales. Processing can be accelerated by first only comparing a subset of sparsely sampled points in the pattern mask and the image. Further processing is only envisaged when the match with the subtemplate exceeds some threshold.

The sensitivity of template matching to minor mismatches increases with perimeter/area, which can be regarded as a measure of complexity. As a consequence, complex patterns which will often require more computations with other methods, simplify the determination of their precise position, orientation, and/or scale with template matching.

Correlation with eigenimages

For a long time, researchers assumed the appearance-based approach was only suited for such controlled settings, where the parameter space can be explored exhaustively at run-time. In 1995, Murase and Nayar from Columbia University introduced their SLAM-system and proved the same idea can also be used under more general imaging conditions.

As was the case with template matching, the SLAM system uses images of an object or scene as model. Recognition then becomes a matter of comparing images. During a training phase, a sufficiently wide variety of model images is taken, to make sure that images presented for recognition will be similar to at least one of the model images. Recognition should normally work if images are taken from many viewpoints and under several lighting conditions. There are

six degrees of freedom to account for object pose (three for translation and three for rotation) and three degrees of freedom to deal with changes in the position of a point light source (linear combinations of which yield arbitrary light sources). As described later, image intensities will be normalised to deal with changes in lightsource radiance. Suppose 128×128 images are used. Then each image represents a point in a 16384-dimensional space. When continuously changing the pose and illumination parameters it can be expected that the image and hence this point will generate a 9-dimensional manifold embedded in the 16384-dimensional space. Of course, in practice one can only sample this manifold, i.e. take images for a finite number of pose and illumination parameters. The manifold needs to be created through interpolation.

If different objects generate different images, then the manifolds corresponding to each of the objects will not intersect. Given an image of an object that has to be recognized (and for which it is assumed a manifold has already been created), the question is to which manifold its point is closest in the 16384-dimensional space. Working in such high-dimensional space is computationally demanding. PCA is used to lower the dimensionality of the problem. PCA captures the variations in the image data by projecting the image onto a lower-dimensional subspace of eigenvectors extracted from the covariance matrix C of the image intensities:

$$C_{ij} = \sum_k (I_i(k) - \mu_i) \ (I_j(k) - \mu_j)$$

where I is a vector that contains all image intensities in lexicographic order and where the sum is taken over the different images k. Typically, 10 to 20 eigenvectors suffice because images show strong correlations between the pixels. Note that these eigenvectors can be interpreted as images with which the given image is correlated, hence the name 'eigenimages' that is sometimes used.

The 'SLAM' (Software Library for Appearance Matching) algorithm then consists of the following steps

1. *Image normalization:* A bounding box is constructed around the object. The data within this box is then represented using a standard image resolution (e.g. 128×128). The average intensity and its variance are normalized to zero and unity, resp.

2. *Building the database:* Each normalized image is added to the image database, with a label identifying the object.

3. *Creating the subspace:* A PCA is applied to the database images and a subspace is created based on the 10 eigenvectors with the largest eigenvalues.

4. *Object manifold generation:* For each object the sample images are combined into a manifold, through spline fitting. In the original SLAM algorithm objects only rotated about a vertical axis under fixed illumination, hence the manifolds are curves (i.e. one-dimensional manifolds).

5. *Object manifold sampling:* The fitted manifold is then sampled to provide sufficient coverage for all envisaged viewing conditions. If, for instance, images where only taken every 5^o of rotation, the manifold can be sampled every 1^o.

6. *Object recognition:* Given an image of an object to be recognized, the object first is delineated with a bounding box, sampled at the fixed resolution, normalized and then projected onto the subspace of the eigenvectors. The nearest manifold sample point yields the object identity.



Figure 2.2: *Objects used for the recognition tests with SLAM.*

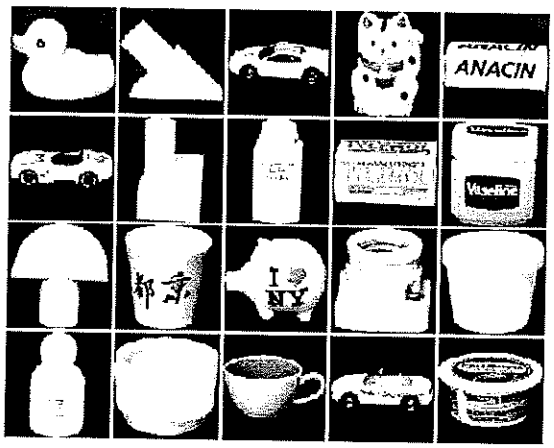


Figure 2.3: *Images used in the recognition tests with SLAM.*

Example Fig. 2.4 shows 100 objects that have been used in tests with the SLAM system. Given the specific restrictions of SLAM, recognition was actually performed on images with isolated objects, on a uniform background. Such images are shown in fig. 2.3. Fig. 2.4 shows SLAM's graphical user interface (GUI). At the bottom left window one sees an example of a manifold representing the object in the top right window.

Discussion There are several advantages to this approach. First of all, it can deal with a wide variety of shapes. Recognition has been demonstrated for complex objects that would be very difficult to handle with model-based techniques. Also, it copes with several types of variations in the same way. Not only different viewpoints are taken into account, the same approach can also deal e.g. with changing illumination. There is no need for an accurate mathematical model of the variations. This makes the method also suitable for category-level object recognition, where within-class variations often play an important role. Finally, training the system only requires taking images. There is no need for a three-dimensional, abstract model of the objects.

There are also important disadvantages, however. Building the image database can be a rather painstaking process. The more degrees of freedom there are, the larger becomes the number of images that need to be taken. The method does not extrapolate well, in the sense that recognition needs a model image that is close to it. Hence, model images should densely sample the space of all possible views. to cover the complete manifold quite completely. Moreover, the object needs to be separated from the background. This is not an easy task to automate as segmentation is not robust unless objects are e.g. lying on a homogeneous background. Using

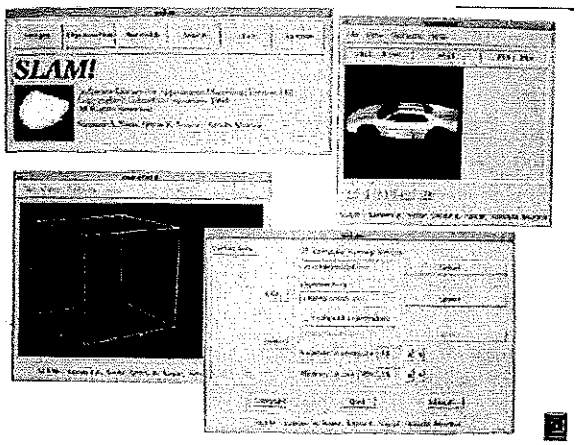


Figure 2.4: *SLAM's graphical user interface.*

a simple bounding box is not necessarily enough if backgrounds change a lot.

Model-based or appearance-based: pros and cons

Comparing the model-based and image-based techniques, the first have the advantage that they can deal with cluttered scenes because they operate on features that are extracted from the image. The image-based techniques, on the other hand, work globally on (a part of) the image containing the object. These methods cannot cope with scene clutter, unless one would manually segment the object from its background and the object is not occluded. On the other hand, by focusing on raw intensities instead of features, these methods are not dependent on the presence of any predefined feature or model and, as a result, can deal with broad classes of objects.

So the question is: Can we combine the best of both worlds ? This has led to the more recent approaches to object recognition, based on *local features*, i.e. relatively small image patches around interest points that can be found in virtually any image (see previous chapter).

These new methods also work with sets of images to represent the object, just like the appearance-based approaches. In these images the object is still assumed to ly on a homogeneous background. But the sequel differs in that the model images are analysed further to extract features, both during modeling and later during recognition.

Using local features, the problems related to image clutter (at test time) and occlusions can be reduced. Indeed, while some of the local features will be affected by these phenomena, there will usually also be a number of features that are not affected, falling entirely on (unoccluded parts of) the object of interest. This way, the object can still be recognized. It is not even necessary to first segment the object from its background during recognition.

Moreover, assuming the object surface is locally planar, the extraction of these local features can be made invariant to viewpoint changes (see chapter 1). This way, the number of model images needed can be kept relatively low (e.g. $10 - 20$ images to describe a 3D object from arbitrary viewpoints, depending on the complexity of the object).

We first describe methods for specific objects (section 2.3), and then continue with methods for object classes (section 2.4).

2.3 Recognition of specific objects

As explained above, we go for appearance-based approaches, i.e. the object to be recognized is modeled by a (set of) image(s). Object recognition could then be as simple as trying to find corresponding features between the model image(s) and a test image. If a sufficiently large number of correspondences are found, this can be considered as a clear indication that the image indeed contains the same object. This is the basic idea behind most local-feature based object recognition systems. However, when the amount of image clutter is large, the number of false matches can be significant. Moreover, the object has not yet been localized in the image. Here, we describe two methods that allow to simultaneously filter out the mismatches (and the false detections they bring along) as well as to localize the object in the image.

2.3.1 Alignment

A first method is known as *alignment*. This only works when a global transformation can be found that (approximately) maps the object in the model image onto the object in the test image. As we have seen earlier, this holds for planar objects, for which the image transformation is a projective transformation, often approximated by an affine transformation. This also holds, approximately, for 3D objects, as long as the two images are taken from similar viewpoints. When there are many images from different viewpoints in the model database, one can usually assume that there is at least one for which this condition is met.

The basic idea is then to estimate this transformation based on a subset of the correspondences found between the test image and model image. For instance, given three correspondences, we can fit an affine transformation. Next, this transformation can be used to map features from the model image onto the test image. This is known as *backprojection*. This can be done for *any* feature, i.e. not only interest points but also edges etc. After backprojection, one can then check whether similar structures are found at corresponding locations.

To find the parameters of the affine transformation, one can rewrite the equations for an affine transformation

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathrm{c} & \mathrm{d} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

as a set of equations in the 6 unknowns a, b, c, d, t_x, t_y:

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} \mathrm{a} \\ \mathrm{b} \\ \mathrm{c} \\ \mathrm{d} \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

Of course, this method only works if the subset of correspondences selected to estimate the global transformation consists of correct matches. Since this cannot be guaranteed, in practice many different subsets need to be tried out.

Figure 2.5 shows an example with different model images of book covers (upper left), the features extracted from these model images (lower left), and the features backprojected onto a test image, showing all three books have been correctly recognized.

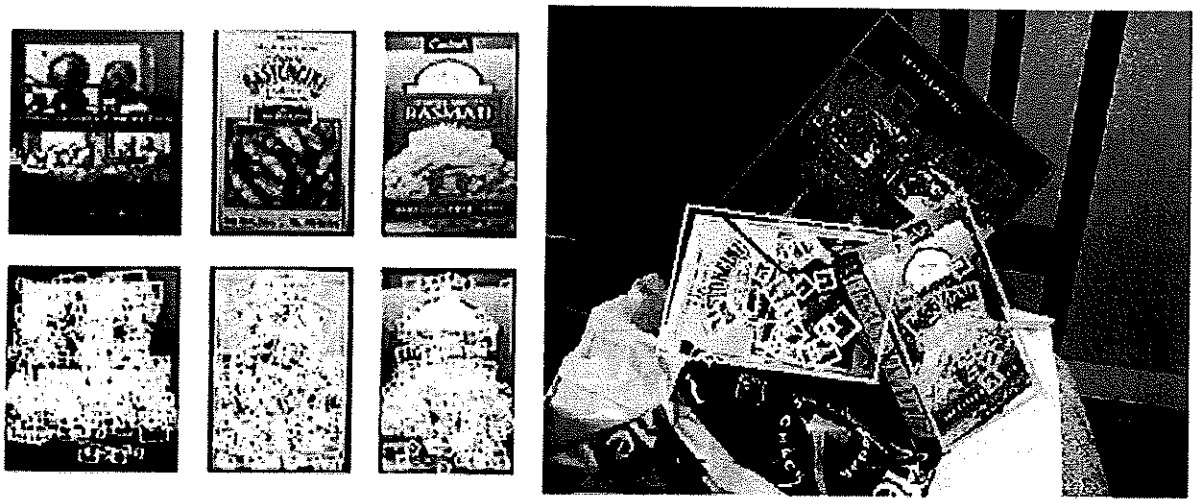


Figure 2.5: **Object recognition by alignment (images courtesy of D. Lowe).**

2.3.2 Voting

Another very robust, but unfortunately also computationally demanding technique is the generalized Hough transform (see also section ??, where the Hough transform was introduced to detect straight lines in an image). This technique comes down to a *voting mechanism*, where consistently positioned features all vote for the correct model-instance transformation parameters, whereas false guesses will only receive limited support.

Voting is a general technique where we let the features vote for all models that are compatible with it. It basically consists of two steps. First, we cycle through all features and cast votes for model parameters that are consistent with it. Next, we look for model parameters that have received a lot of votes. Of course, noise and clutter features will cast votes too, but typically their votes are not consistent with the majority of "good" features, so they can be rejected as outliers. This makes the voting strategy robust to even a large amount of false correspondences.

When applied to object recognition, the model parameters are the parameters of the global transformation mapping the object in the model image onto a particular location and pose in the test image. If several model images are available, an additional parameter is added that selects the model image. To keep the number of parameters (and consequently, the dimensionality of the voting space) low, we use similarity transformations (i.e. rotation, translation and scale). This results in a four-dimensional voting space. This space is discretized using broad bin sizes (e.g. 30 degrees for orientation, a factor of 2 for scale, and 0.25 times the image size for location).

First, we match scale-invariant features extracted from the test image to the scale-invariant features found in the model images. Each correspondence then gives an alignment hypothesis (for scale, translation and rotation, see also figure 2.6) and votes for the two closest bins in each dimension, using bilinear weighting. When all correspondences have voted, we look for all bins with three or more votes. These are alignment hypotheses that are passed on to a geometric verification. To this end, we compute a more accurate affine transformation using all the correspondences that contributed to the bin, using a least squares fitting algorithm. Then, we backproject the model image onto the test image and search for additional features that agree with the alignment.

Figure 2.7 shows some results obtained with this method.

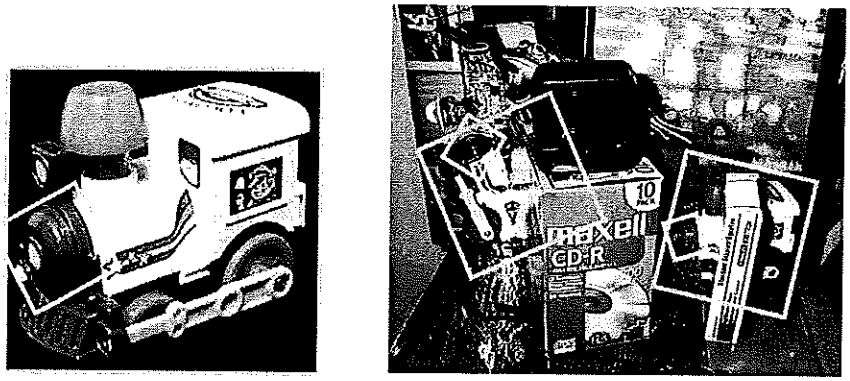


Figure 2.6: Object recognition by voting: each correspondence gives an alignment hypothesis (images courtesy of D. Lowe).

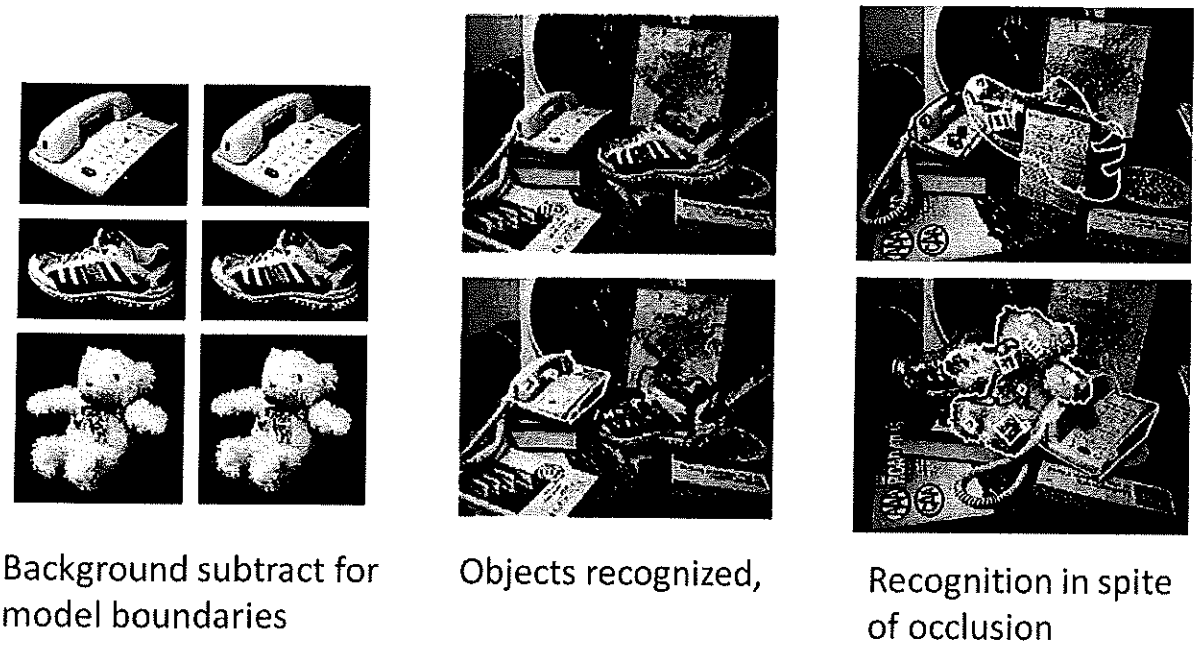


Figure 2.7: Object recognition by voting: example result (images courtesy of D. Lowe).

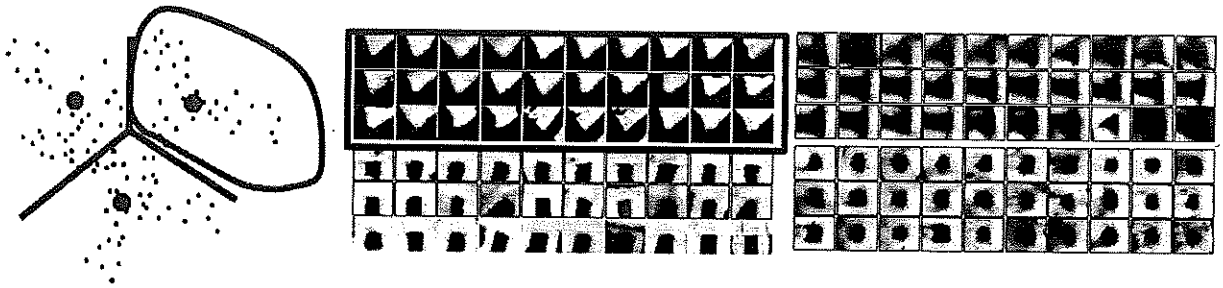


Figure 2.8: **Local features are clustered in feature space (left). All features assigned to the same cluster center then represent the same visual word (right). (image courtesy of J. Sivic).**

2.3.3 Using a visual vocabulary for efficient recognition/retrieval

In many practical applications, images of a particular object need to be searched for in a large database of images. The methods of alignment and voting described so far are not well suited for this setting, since the needed computation time scales linearly with the number of images in the database. With potentially thousand of features per image, and hundreds to millions of images to search, how to efficiently find those that are relevant to a new query image ? Efficient methods for image or object retrieval have been proposed to address this question. Using indexing and inverted file systems, they manage to search a database of millions of images in less than a second.

Indexing and inverted file systems are methods that have originally been developed to retrieve text documents: an index of a book, for instance, makes it possible to quickly find all pages on which a particular word occurs. Likewise, we want to find all images in which a particular feature occurs. To use this idea, we need to map the notion of visual features onto *visual words*, the visual equivalent to words in a text document.

However, while words are discrete, the feature vectors describing our local features are continuous. Two feature vectors are never exactly the same, but only similar at best. This is solved by vector quantizing the feature space: We collect a lot of example feature vectors, and cluster them using k-means. Then, each feature vector can be assigned to its nearest cluster center, and all features that have been assigned to the same cluster center, are considered to be representing the same visual word. This is illustrated in figure 2.8.

The set of all visual words is then referred to as the *visual vocabulary* or *visual codebook*. The size of this visual vocabulary corresponds to the number of clusters specified for the k-means algorithm. For the recognition of specific objects, best results are obtained with large vocabularies, with up to 100.000 different visual words. For category-level recognition, on the other hand, the concept of visual words will be used as well, but then using just a couple of thousands of visual words.

Given the concept of visual words, applying the retrieval techniques developed for text documents is relatively straigtforward. First, in an offline stage, all the database images are loaded in the index, mapping words to image numbers. Then, the system is ready for retrieval. Given a new query image, local features are extracted. Then for each local feature, we assign it to a visual word, and vote for all images in the database that share the same word. These can be retrieved efficiently using the index. Database images that share many words will receive many

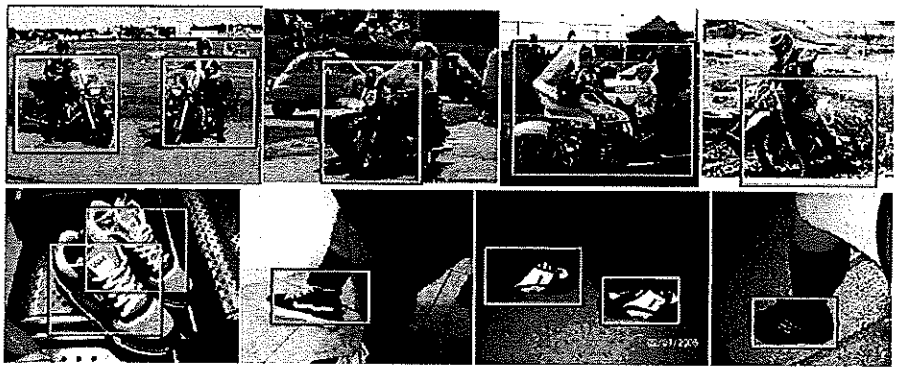


Figure 2.9: **Some detection results with a state-of-the-art method for category-level object recognition.**

votes. By ranking the database images based on the number of votes, the most similar images can be retrieved.

As a special case of this image retrieval scheme, a database can be constructed containing example images of many different objects, along with their object label. Then the object(s) in the query image can be recognized efficiently by retrieving the most similar images and looking at the object they represent.

As with all the methods described in this section, this works best in case of highly textured objects. While there are systems in operation that can recognize almost any CD cover, film poster or book cover (see e.g. www.kooaba.com), recognition of textureless objects such as a coffee cup is still problematic.

2.4 Recognition of object classes

Recognition of object classes is even more challenging than the recognition of specific objects, as the within-class variability can often be larger than some of the inter-class variability. This calls for machine learning techniques, that learn from a set of training examples which variations are relevant and which are not.

Various classification methods have been applied in this context, including maximum-margin classifiers (support vector machines), graphical models, boosting, Markov random fields, etc. However, explaining these would lead us too far. So we limit ourselves to a very basic method instead.

How well these methods perform often depends, to a large extent, on the availability of training data (including both positive as well as negative examples, i.e. examples of images containing the object as well as examples not containing the object). Some examples of detection results obtained with a state-of-the-art method are shown in figure 2.9.

To somehow simplify the problem, methods for category-level object recognition usually work for only a single viewpoint (with e.g. one classifier for side-views of cars, one for front-views of cars, etc.).

Given the large variability in appearance of objects belonging to the same class, matching individual features between a test image and a model image is no longer a reliable strategy.

Roughly speaking, two approaches can be distinguished. On the one hand, there are methods that try to combine information from multiple training images. On the other hand, we have methods that do not focus so much on individual feature correspondences, but rather look at the overall distribution of features. The bag-of-visual-words method that we will describe next

belongs to this second category. The Implicit Shape Model, which is a probabilistic extension of the voting strategy described earlier for specific objects, is an example of the first category.

2.4.1 Bag-of-visual-words

The basic idea of the bag-of-visual-words approach is to summarize the entire image based on its distribution (histogram) of visual word occurrences. This is in analogy with the bag-of-words representation commonly used for text documents. This yields a single vector describing the whole image content. The similarity between different images can then be estimated by comparing their bag-of-visual-words vectors, e.g. using a normalized scalar product

$$sim(d_j, q) = \frac{\sum w_{i,j} \times w_{i,q}}{\sqrt{\sum w_{i,j}^2} \sqrt{\sum w_{i,q}^2}}$$

with $\{w_{i,j}\}$ the i-th element of the bag-of-visual-words representation of the j-th model image (i.e. how many times the i-th visual word occurs in image d_j) and $\{w_{i,q}\}$ the same for the query image q.

A very simple method for category-level object recognition could then be implemented as follows, using a k-nearest neighbor classifier:

1. Rank the training images based on the similarity between their bag-of-visual-words representation and the bag-of-visual-words representation of the test image.
2. Select the top k most similar training images (nearest neighbours).
3. Apply a voting strategy based on the categories of those k training images.

Moreover, since we now have a simple vector representation describing the image, this can be used in combination with most classifiers. Using a bag-of-visual-words description in combination with a support vector machine has lead to very good results for a wide range of object classes.

A downside of the bag-of-visual-words representation is that all spatial information has been ignored. Therefore, extensions have been proposed that again bring in some of the spatial configuration. The most widely used example is the spatial pyramid, where different bag-of-visual-words are computed for different parts of the image.

Also, instead of local invariant features, one sometimes applies the same framework of visual vocabulary construction and bag-of-visual-words representations starting from patches that sample the image at regular intervals (e.g. every n-th pixel). These may not be very repeatable, but have the advantage that the whole image is covered more uniformly, including also information on textureless regions.