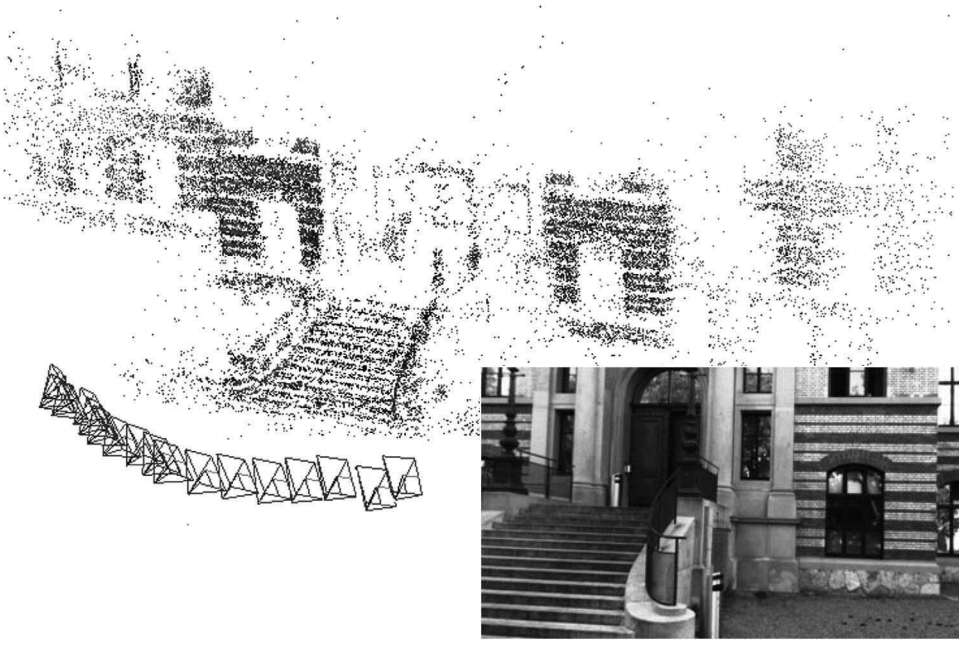


Figure 4.50 An example of structure-from-motion: salient image points (image features, see section 4.5) are extracted and matched across multiple frames. Wrong data associations (outliers) are removed and the relative motion among the views is determined. Finally, the points are reconstructed by triangulation. The reconstruction of the building was obtained using dozens of images. The camera poses are also displayed. Image courtesy of Friedrich Fraundorfer. Structure from motion also allows dense reconstruction of entire cities and monuments using just images.

$$\lambda_1 \tilde{p}_1 = A_1[I|0]\tilde{P}_w = A_1 P_w \text{ (for the first camera position),} \tag{4.67}$$

$$\lambda_2 \tilde{p}_2 = A_2[R|t]\tilde{P}_w \text{ (for the second camera position).} \tag{4.68}$$

In order to simplify our problem, let us make some assumptions. Let us assume that we use the same camera for the first and the second position and that the intrinsic parameters do not change in between; therefore, $A_1 = A_2 = A$. Let us also assume that the camera is calibrated, and that therefore A is known. In this case, it is more convenient to work with *normalized* image coordinates. Let $\tilde{x}_1$ and $\tilde{x}_2$ be the *normalized* coordinates of $\tilde{p}_1$ and $\tilde{p}_2$ respectively, where

$$\tilde{x}_1 = A^{-1}\tilde{p}_1 \text{ and } \tilde{x}_2 = A^{-1}\tilde{p}_2, \tag{4.69}$$

and $\tilde{x}_1 = (x_1, y_1, 1)$, $\tilde{x}_2 = (x_2, y_2, 1)$. Then, we can rewrite (4.67) and (4.68) as:

$$\lambda_1 \tilde{x}_1 = P_w \text{ (for the first camera position),} \tag{4.70}$$

$$\lambda_2 \tilde{x}_2 = [R|t]\tilde{P}_w = RP_w + t \text{ (for the second camera position).} \tag{4.71}$$

As we did before for computing the epipolar lines (page 176), let us map the optical ray corresponding to x_1 into the second image. Thus, by substituting (4.70) into (4.71), we obtain:

$$\lambda_2 \tilde{x}_2 = \lambda_1 R\tilde{x}_1 + t\,. \tag{4.72}$$

Let us now take the *cross* product of both sides with *t*. This in order to cancel *t* on the right-hand side. Then, we obtain:

$$\lambda_2 [t] \times \tilde{x}_2 = \lambda_1 ([t] \times R) \cdot \tilde{x}_1\,, \tag{4.73}$$

where $[t]\times$ is an antisymmetric matrix defined as

$$[t] \times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}. \tag{4.74}$$

Now, taking the *dot* product of both sides (4.73) with x_2 yields:

$$\lambda_2 \tilde{x}_2^T \cdot ([t] \times \tilde{x}_2) = \lambda_1 \tilde{x}_2^T \cdot ([t] \times R) \cdot \tilde{x}_1\,. \tag{4.75}$$

Observe that $\tilde{x}_2^T \cdot ([t] \times \tilde{x}_2) = 0$, and therefore from (4.75) we obtain:

$$\tilde{x}_2^T \cdot ([t] \times R) \cdot \tilde{x}_1 = 0\,, \tag{4.76}$$

which is called *epipolar constraint*. Observe that the epipolar constraint is valid for every pair of conjugate points.

Let us define the *essential matrix* $E = ([t] \times R)$, the epipolar constraint reads as;

$$\tilde{x}_2^T \cdot E \cdot \tilde{x}_1 = 0 \tag{4.77}$$

It can be shown that the essential matrix has two singular values which are equal and another which is zero [29].

Computing the essential matrix. Given this fundamental relationship (4.77), how can we use it to recover the camera motion encoded in the essential matrix E? If we have N corresponding measurements $\{(x_1^i, x_2^i)\}$, we can form N homogeneous equations in the nine elements of $E = [e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}, e_{31}, e_{32}, e_{33}]^T$, of the type

$$\tilde{x}_1^i \tilde{x}_2^i e_{11} + \tilde{y}_1^i \tilde{x}_2^i e_{12} + \tilde{x}_2^i e_{13} + \tilde{x}_1^i \tilde{y}_2^i e_{21} + \tilde{y}_1^i \tilde{y}_2^i e_{22} + \tilde{y}_2^i e_{23} + \tilde{x}_1^i e_{31} + \tilde{y}_1^i e_{32} + e_{33} = 0 . \quad (4.78)$$

This can be rewritten in a more compact way as:

$$D \cdot E = 0 . \quad (4.79)$$

Given $N \geq 8$ such equations, we can compute an estimate (up to a scale) for the entries in E using the *Singular Values Decomposition* (*SVD*). The solution of (4.79) will therefore be the eigenvector of D corresponding to the smallest eigenvalue. Because at least eight point correspondences are needed, this algorithm is known as the *eight-point algorithm* [194]. This algorithm is one of the milestones of computer vision. The main advantages of the eight-point algorithm are that it is very easy to implement and that it works also for an uncalibrated camera, that is, when the camera intrinsic parameters are unknown. The drawback is that it does not work for degenerate point configurations such as planar scenes, that is, when all the scene points are coplanar.

In the case of a calibrated camera, at least five point correspondences are required [178]. An efficient algorithm for computing the essential matrix from at least five point correspondences was proposed by Nister [246]. The *five-point algorithm* works only for calibrated cameras but is more complicated to implement. However, in contrast to the eight-point algorithm, it also works for planar scenes.

Decomposing *E* into *R* and *t*. Let us now assume that the essential matrix E has been determined from known point correspondences. How do we determine R and t? Because a complete derivation of the proof is beyond the scope of this book, we will give directly the final expression. The interested reader can find the proof of these equations in [29].

Before decomposing E, we need to enforce the constraint that two of its singular values are equal and the third one is zero. In fact, in presence of image noise this constraint will never be verified in practice. To do this, we compute the closest[9] essential matrix $\hat{E}$ which

9. Closest in terms of the Frobenius norm.

satifies this constraint. One popular technique is to use SVD and force the two larger singular values to be equal and the smallest one to be zero. Therefore:

$$[U, S, V] = SVD(E), \tag{4.80}$$

where $S = diag([S_{11}\ S_{ss}\ S_{33}])$ with $S_{11} \geq S_{22} \geq S_{33}$. Then, the closest essential matrix eo E in the Frobenius norm is given by

$$\hat{E} = U \cdot diag\left(\left[\frac{S_{11}+S_{22}}{2},\ \frac{S_{11}+S_{22}}{2},\ 0\right]\right) \cdot V^T \tag{4.81}$$

Then, we replace E with $\hat{E}$. At this point, we can decompose E into R and t.

The decomposition of E returns four solutions for (R,t), two for R and two for t. Let us define

$$B = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and } [U, S, V] = SVD(E), \tag{4.82}$$

where U, S, and V are such that $U \cdot S \cdot V^T = E$. It can be shown (see [29]) that the two solutions for R are:

$$R_1 = det(U \cdot V^T) \cdot U \cdot B \cdot V^T, \tag{4.83}$$

$$R_2 = det(U \cdot V^T) \cdot U \cdot B^T \cdot V^T. \tag{4.84}$$

Now, let us define

$$L = U \cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot U^T \quad \text{and } M = -U \cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot U^T. \tag{4.85}$$

The two solutions for t are:

$$t_1 = \frac{\left[L_{32}\ L_{13}\ L_{21}\right]^T}{\left\|\left[L_{32}\ L_{13}\ L_{21}\right]\right\|}, \tag{4.86}$$

$$t_2 = \frac{\left[M_{32}\ M_{13}\ M_{21}\right]}{\left\|\left[M_{32}\ M_{13}\ M_{21}\right]^T\right\|}. \tag{4.87}$$

These four solutions can be disambiguated using the so-called *cheirality constraint*, which requires that reconstructed point correspondences lie in front of the cameras. In fact, if you analyse the four solutions of the SfM problem, you will always find that three solutions are such that the reconstructed point correspondences appear behind at least one of the two cameras, while only one solution guarantees that they lie in front of both cameras. Thus, testing with a single point correspondence to determine if it is reconstructed in front of both cameras is sufficient to identify the right solution out of the four possible choices. Also, observe that the solution for t is known up to a scale. In fact, with a single camera it is not possible to recover the absolute scale. For the same reason, the recovered structure will also be known up to a scale.

The last step in two-view structure-from-motion is the reconstruction of the scene. Once R and t have been found, the 3D structure can be computed via triangulation of the feature points as done for the stereo camera (page 173).

Free software for multi-view structure from motion. To conclude this section, we would like to point the reader to some interesting, free software to performe structure-from motion from unordered image collections. The most popular is Microsoft Photosynth (http://photosynth.net)—inspired by the research work on Photo Tourism [355]—which is based on the very popular open-source software Bundler (available at http://photo-tour.cs.washington.edu/bundler) and described in [297] and [298].

Very useful and fully open-source tools for on-line processing are: the Parallel Tracking and Mapping (PTAM) tool [358], the Vodoo camera tracker [356], and the ARToolkit [357].

Useful Matlab toolboxes for structure from motion are:

- FIT3D: http://www.fit3d.info
- Structure from Motion toolbox by V. Rabaud: http://code.google.com/p/vincents-structure-from-motion-matlab-toolbox
- Matlab Functions for Multiple View Geometry by A. Zissermann: http://www.robots.ox.ac.uk/~vgg/hzbook/code
- Structure and Motion Toolkit by P. Torr: http://cms.brookes.ac.uk/staff/PhilipTorr/Code/code_page_4.htm
- Matlab Code for Non-Rigid Structure from Motion using Factorisation by L. Torresani http://movement.stanford.edu/learning-nr-shape

Finally, see also the companies 2d3 (http://www.2d3.com) and Vicon (http://www.vicon.com).

4.2.6.2 Visual odometry

Directly linked to structure from motion is *visual odometry*. Visual odometry consists in estimating the motion of a robot or that of a vehicle by using visual input alone. The term "visual odometry" was coined in 2004 by Nister with his homonym landmark paper [245], where he showed successful results on different vehicles (on-road and off-road) using either a single camera or a stereo camera. The basic principle behind visual odometry is a simple iteration of two-view structure from motion that we have seen in the previous section.

Most of the work done about visual odometry has been produced using stereo cameras and can be traced back to 1980 with Moravec's work [236]. Similar work has also been reported elsewhere also (see [160, 174, 181, 244]). Furthermore, stereo visual odometry has also been successfully used on Mars by the NASA rovers since early 2004 [203]. Nevertheless, visual odometry methods for outdoor applications have also been produced, which use a single camera alone (see [107, 244, 278, 279, 307]).

The advantage of using a stereo camera compared to a single camera is that the measurements are directly provided in the absolute scale. Conversely, when using a single camera the absolute scale must be estimated in other ways (e.g., from knowledge of an element in the scene, or the distance between the camera and the ground plane) or using other sensors such as GPS, IMU, wheel odometry, or lasers.

Visual odometry aims at recovering only the trajectory of the vehicle. Nevertheless, it is not uncommon to see results showing also the 3D map of the environment which is usually a simple triangulation of the feature points from the estimated camera poses. An example visual odometry result using a single omnidirectional camera is shown in figure 4.51. Here the scale was obtained by exploiting the nonholonomic constraints of the vehicle as described in [277]. In this figure, visual odometry is performed over a 3 km trajectory. Notice the visible drift toward the end of the trajectory.

All visual odometry algorithms suffer from motion drift due to the integration of the relative displacements between consecutive poses which unavoidably accumulates errors over time. This drift becomes evident usually after a few hundred meters. But the results may vary depending on the abundance of features in the environment, the resolution of the cameras, the presence of moving objects like people or other passing vehicles, and the illumination conditions. Remember that motion drift is also present in wheel odometry, as will be described in section 5.2.4. However, the reason visual odometry is becoming more and more popular in both robotics and automotive is that drift can be canceled if the vehicle revisits a place that has already been observed previously. The possibility of performing location recognition (or place recognition) is one of the main advantages of vision com-

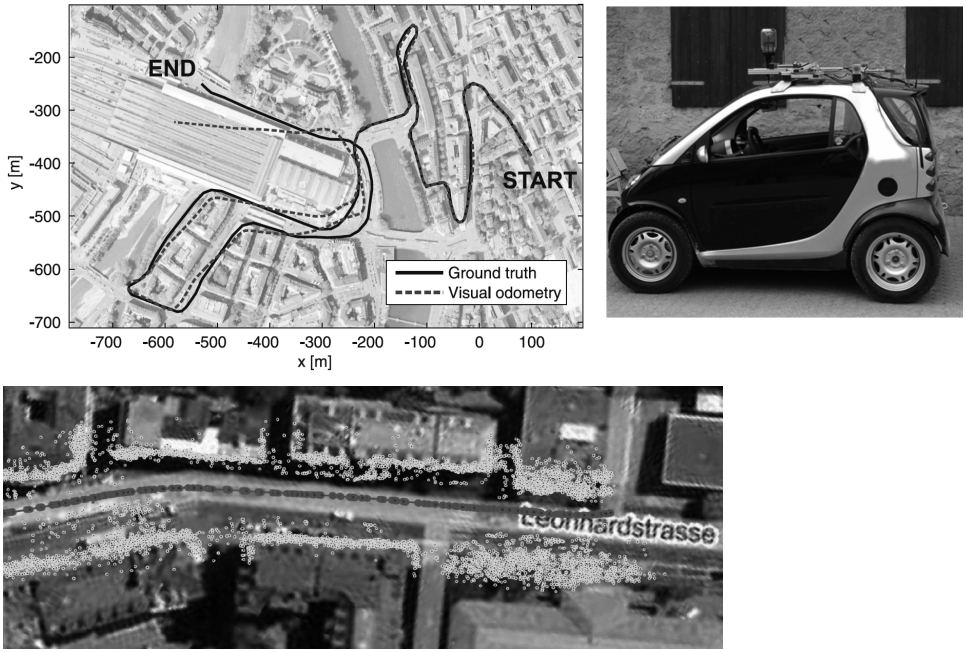


Figure 4.51 (upper left) An example visual odometry result with related map (bottom) obtained using a single omnidirectional camera mounted on the roof of the vehicle (right). The absolute scale was computed automatically by taking advantage of the fact that a wheeled vehicle is constrained to follow a circular coarse, locally, about the instantaneous center of rotation (see Ackerman steering principle in section 3.3.1). This visual odometry result was obtained using the 1-point RANSAC method described in [278]. The advantage of this algorithm compared to the state of the art is that visual odometry runs at 400 frames per second, while standard method work at 20–40 Hz.

pared to other sensor modalities. The most popular computer vision approaches for location recognition will be described in section 4.6. Once a place previously observed is visited a second time by the robot, the accumulated error can be reduced by adding the constraint that the positions of the vehicle at these two places (the previously visited and the revisited one) should actually coincide. This obviously requires an algorithm that modifies ("relaxes") all the previous robot poses until the error between the current and previously visited location is minimized.

The problem of location recognition is also called loop-detection, because a loop is a closed trajectory of a vehicle that returns to a previously-visited point. The problem of minimizing the error at the loop closure is instead called loop-closing. There are several algorithms in the literature to perform loop-closing. Some of them come from the computer vision community and rely on the so called *bundle adjustment*,[10] while others have been

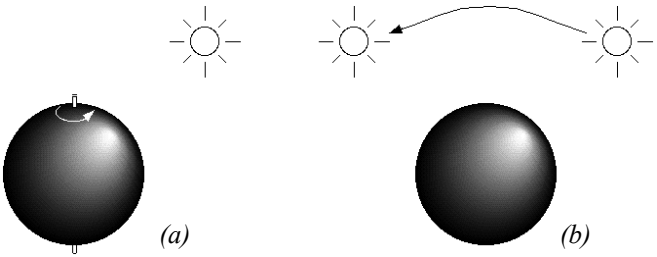


Figure 4.52
Motion of the sphere or the light source here demonstrates that optical flow is not always the same as the motion field.

developed within the robotics community to solve the Simultaneous Localization and Mapping (SLAM) problem (see section 5.8.2). Some of the most popular algorithms can be found in [318] and [352].

4.2.7 Motion and optical flow

A great deal of information can be recovered by recording time-varying images from a fixed (or moving) camera. First, we distinguish between the motion field and optical flow:

- Motion field: this assigns a velocity vector to every point in an image. If a point in the environment moves with velocity v_0, then this induces a velocity v_i in the image plane. It is possible to determine mathematically the relationship between v_i and v_0.
- Optical flow: it can also be true that brightness patterns in the image move as the object that causes them moves (light source). Optical flow is the apparent motion of these brightness patterns.

In our analysis here we assume that the optical flow pattern will correspond to the motion field, although this is not always true in practice. This is illustrated in figure 4.52a, where a sphere exhibits spatial variation of brightness, or shading, in the image of the sphere since its surface is curved. If the surface moves, however, this shading pattern will not move hence the optical flow is zero everywhere even though the motion field is not zero. In figure 4.52b, the opposite occurs. Here we have a fixed sphere with a moving light source. The shading in the image will change as the source moves. In this case the optical

10. Given a set of images observing a certain number of 3D points from different viewpoints, bundle adjustment is the problem of simultaneously refining the 3D coordinates of the scene geometry as well as the relative motion and the camera intrinsic parameters. This is done according to an optimality criterion involving the corresponding image projections of all points.

flow is nonzero but the motion field is zero. If the only information accessible to us is the optical flow and we depend on this, we will obtain incorrect results in both cases.

4.2.7.1 Optical flow

There are a number of techniques for attempting to measure optical flow and thereby obtain the scene's motion field. Most algorithms use local information, attempting to find the motion of a local patch in two consecutive images. In some cases, global information regarding smoothness and consistency can help to disambiguate further such *matching* processes. Below we present details for the optical flow constraint equation method. For more details on this and other methods, refer to [69, 151, 316].

Suppose first that the time interval between successive snapshots is so fast that we can assume that the measured intensity of a portion of the same object is effectively constant. Mathematically, let $I(x, y, t)$ be the image irradiance at time t at the image point (x, y). If $u(x, y)$ and $v(x, y)$ are the x and y components of the optical flow vector at that point, we need to search a new image for a point where the irradiance will be the same at time $t + \delta t$, that is, at point $(x + \delta t, y + \delta t)$, where $\delta x = u\delta t$ and $\delta y = v\delta t$. That is,

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t) \tag{4.88}$$

for a small time interval, δt. This will capture the motion of a constant-intensity *patch* through time. If we further assume that the brightness of the image varies smoothly, then we can expand the left-hand side of equation (4.88) as a Taylor series to obtain

$$I(x, y, t) + \delta x\frac{\partial I}{\partial x} + \delta y\frac{\partial I}{\partial y} + \delta t\frac{\partial I}{\partial t} + e = I(x, y, t), \tag{4.89}$$

where e contains second- and higher-order terms in δx, and so on. In the limit as δt tends to zero we obtain

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0, \tag{4.90}$$

from which we can abbreviate

$$u = \frac{dx}{dt}\ ;\quad v = \frac{dy}{dt} \tag{4.91}$$

and

$$I_x = \frac{\partial I}{\partial x} \; ; \quad I_y = \frac{\partial I}{\partial y} ; \quad I_t = \frac{\partial I}{\partial t} = 0 \, , \tag{4.92}$$

so that we obtain

$$I_x u + I_y v + I_t = 0 \, . \tag{4.93}$$

The derivative I_t represents how quickly the intensity changes with time while the derivatives I_x and I_y represent the spatial rates of intensity change (how quickly intensity changes across the image). Altogether, equation (4.93) is known as the *optical flow constraint equation*, and the three derivatives can be estimated for each pixel given successive images.

We need to calculate both u and v for each pixel, but the optical flow constraint equation only provides one equation per pixel, and so this is insufficient. The ambiguity is intuitively clear when one considers that a number of equal-intensity pixels can be inherently ambiguous—it may be unclear which pixel is the resulting location for an equal-intensity originating pixel in the prior image.

The solution to this ambiguity requires an additional constraint. We assume that in general the motion of adjacent pixels will be similar, and that therefore the overall optical flow of all pixels will be smooth. This constraint is interesting in that we know it will be violated to *some* degree, but we enforce the constraint nonetheless in order to make the optical flow computationally tractable. Specifically, this constraint will be violated precisely when different objects in the scene are moving in different directions with respect to the vision system. Of course, such situations will tend to include edges, and so this may introduce a useful visual cue.

Because we know that this smoothness constraint will be somewhat incorrect, we can mathematically define the degree to which we violate this constraint by evaluating the formula

$$e_s = \iint (u^2 + v^2) dx dy \, , \tag{4.94}$$

which is the integral of the square of the magnitude of the gradient of the optical flow. We also determine the error in the optical flow constraint equation (which in practice will not quite be zero).

$$e_c = \iint (I_x u + I_y v + I_t)^2 dx dy \, . \tag{4.95}$$

Both of these equations should be as small as possible, so we want to minimize $e_s + \lambda e_c$, where λ is a parameter that weights the error in the image motion equation relative to the departure from smoothness. A large parameter should be used if the brightness measurements are accurate and small if they are noisy. In practice, the parameter λ is adjusted manually and interactively to achieve the best performance.

The resulting problem then amounts to the calculus of variations, and the Euler equations yield

$$\nabla^2 u = \lambda(I_x u + I_y v + I_t) I_x, \tag{4.96}$$

$$\nabla^2 v = \lambda(I_x u + I_y v + I_t) I_y, \tag{4.97}$$

where

$$\nabla^2 = \frac{\partial^2}{\delta x^2} + \frac{\partial^2}{\delta y^2}, \tag{4.98}$$

which is the Laplacian operator.

Equations (4.96) and (4.97) form a pair of elliptical second-order partial differential equations that can be solved iteratively.

Where occlusions (one object occluding another) occur, discontinuities in the optical flow will occur. This, of course, violates the smoothness constraint. One possibility is to try to find edges that are indicative of such occlusions, excluding the pixels near such edges from the optical flow computation so that smoothness is a more realistic assumption. Another possibility is to make use of these distinctive edges opportunistically. In fact, corners can be especially easy to *pattern-match* across subsequent images and thus can serve as fiducial markers for optical flow computation in their own right.

Optical flow is an important ingredient in vision algorithms that combine cues across multiple algorithms. Obstacle avoidance and navigation control systems for mobile robots (especially flying robots) using optical flow have proved to be broadly effective as long as texture is present [23, 54].

4.2.8 Color tracking

An important aspect of vision sensing is that the vision chip can provide sensing modalities and cues that no other mobile robot sensor provides. One such novel sensing modality is detecting and tracking color in the environment.

Color is an environmental characteristic and represents both a natural cue and an artificial cue that can provide new information to a mobile robot. For example, the annual robot

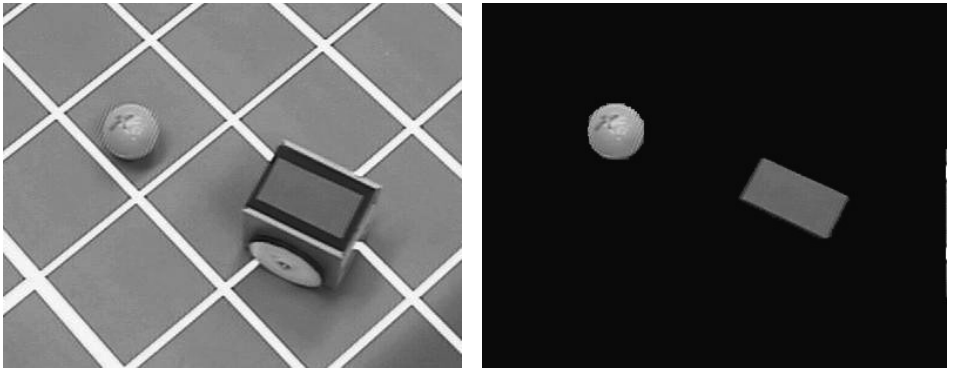


Figure 4.53
Color markers on the top of EPFL's STeam Engine soccer robots enable a color-tracking sensor to locate the robots and the ball in the soccer field.

soccer events (RoboCup) make extensive use of color both for environmental marking and for robot localization (see figure 4.53).

Color sensing has two important advantages. First, detection of color is a straightforward function of a single image, therefore no correspondence problem needs be solved in such algorithms. Second, because color sensing provides a new, independent environmental cue, if it is combined (i.e., *sensor fusion*) with existing cues, such as data from stereo vision or laser rangefinding, we can expect significant information gains.

Efficient color-tracking sensors are also available commercially—such as the CMUcam from Carnegie Mellon University—but they can also be implemented straightforwardly using a standard camera. The simplest way of doing this is using *constant thresholding*: a given pixel point is selected if and only if its RGB values (r, g, b) fall simultaneously in the chosen R, G, and B ranges, which are defined by six thresholds $[R_{min}, R_{max}]$, $[G_{min}, G_{max}]$, $[B_{min}, B_{max}]$. Therefore

$$R_{min} < r < R_{max} \text{ and } G_{min} < g < G_{max} \text{ and } B_{min} < b < B_{max} . \quad (4.99)$$

If we represent the RGB color space as a three-dimensional Euclidean space, the aforementioned method selects those pixels whose color components belong to the cube specified by the given thresholds. Alternatively, a sphere could be used. In this case, a pixel would be selected only if its RGB components are within a certain distance from a given point in the RGB space.

Alternatively to RGB, the YUV color space can be used. While R, G, and B values encode the intensity of each color, YUV separates the color (or *chrominance*) measure

Figure 4.54
Examples of adaptive floor plane extraction. The trapezoidal polygon identifies the floor sampling region.

from the brightness (or *luminosity*) measure. Y represents the image's luminosity while U and V together capture its chrominance. Thus, a bounding box expressed in YUV space can achieve greater stability with respect to changes in illumination than is possible in RGB space.

A popular application of color segmentation in robotics is floor plane extraction (figure 4.54). In this case, color segmentation techniques more complex than color thresholding are used, like adaptive thresholding, or *k-means clustering*[11] [18]. Floor plane extraction is a vision approach for identifying the traversable portions of the ground. Because it makes use of edges (section 4.3.2) and color in a variety of implementations, such obstacle detection systems can easily detect obstacles in cases that are difficult for traditional ranging devices. As is the case with all vision algorithms, floor plane extraction succeeds only in environments that satisfy several important assumptions:

- Obstacles differ in appearance from the ground.
- The ground is flat, and its angle to the camera is known.

11. In statistics and machine learning, *k-means clustering* is a method of cluster analysis that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean. The k-means clustering algorithm is commonly used in computer vision as a form of image segmentation.

- There are no overhanging obstacles.

The first assumption is a requirement in order to discriminate the ground from obstacles using its appearance. A stronger version of this assumption, sometimes invoked, states that the ground is uniform in appearance and different from all obstacles. The second and third assumptions allow floor-plane-extraction algorithms to estimate the robot's distance to obstacles detected.

4.3 Fundamentals of Image Processing

Image processing is a form of signal processing where the input signal is an image (such as a photo or a video) and the output is either an image or a set of parameters associated with the image. Most image-processing techniques treat the image as a two-dimensional signal $I(x, y)$ where x and y are the *spatial* image coordinates and the amplitude of I at any pair of coordinates (x, y) is called *intensity* or *gray level* of the image at that point.

Image processing is a huge field and typical operations, among many others, are:

- Filtering, image enhancing, edge detection
- Image restoration and reconstruction
- Wavelets and multiresolution processing
- Image compression (e.g., JPEG)
- Euclidean geometry transformations such as enlargement, reduction, and rotation
- Color corrections such as brightness and contrast adjustments, quantization, or color translation to a different color space
- Image registration (the alignment of two or more images)
- Image recognition (for example, extracting a face from the image by using some face recognition algorithm)
- Image segmentation (partitioning the image in characteristic regions according to color, edges, or other features)

Because a review of all these techniques goes beyond the scope of this book, here we focus only on the most important image processing operations that are relevant for robotics. In particular, we describe image filtering operations such as smoothing and edge detection. We will then describe some image similarity measures for finding point correspondences between images, which are helpful in structure from stereo and structure from motion. For an in-depth study of image processing in general, we refer the reader to [26].

4.3.1 Image filtering

Image filtering is one of the principal tools in image processing. The word *filter* comes from frequency domain processing, where "filtering" refers to the process of accepting or rejecting certain frequency components. For example, a filter that passes low frequencies is called a *lowpass* filter. The effect produced by a lowpass filter is to blur (smooth) an image, which has the main effect of reducing image noise. Conversely, a filter that passes high frequencies is called *highpass* filter and is typically used for edge detection. Image filters can be implemented both in the frequency domain and in the spatial domain. In the latter case, the filter is called *mask* or *kernel*. In this section, we will review the fundamentals of spatial filtering.

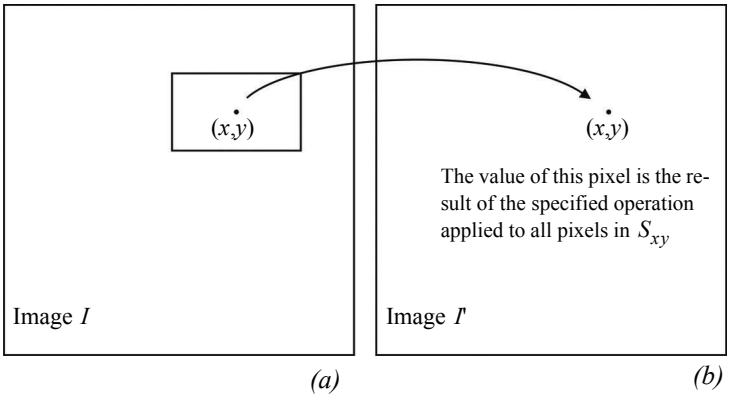
In figure 4.55, the basic principle of spatial filtering is explained. A spatial filter consists of (1) a neighborhood of the pixel under examination, (typically a small rectangle), and (2) a predefined operation T that is performed on the image pixels encompassed by the neighborhood. Let S_{xy} denote the set of coordinates of a neighborhood centered on an arbitrary point (x,y) in an image I. Spatial filtering generates a corresponding pixel at the same coordinates in an output image I' where the value of that pixel is determined by a specified operation on the pixels in S_{xy}. For example, suppose that the specified operation is to compute the average value of the pixels in a rectangular window of size $m \times n$ centered on (x,y). The locations of pixels in this region constitute the set S_{xy}. Figure 4.55a–b illustrates the process. We can express this operation in equation form as

$$I'(x,y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} I(r,c) \tag{4.100}$$

where r and c are the row and column coordinates of the pixels in the set S_{xy}. The new image I' is created by varying the coordinates (x, y) so that the center of the window moves from pixel to pixel in image I. For instance the image in figure 4.55d was created in this manner using a window of size 21×21 applied on the image in figure 4.55c.

The filter used to illustrate the example above is called *averaging filter*. More generally, the operation performed on the image pixels can be linear or nonlinear. In these cases, the filter is called either a *linear* or *nonlinear* filter. Here, we concentrate on linear filters. In general, linear spatial filtering of an image with a filter w of size $m \times n$ is given by the expression

$$I'(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot I(x+s, y+t), \tag{4.101}$$



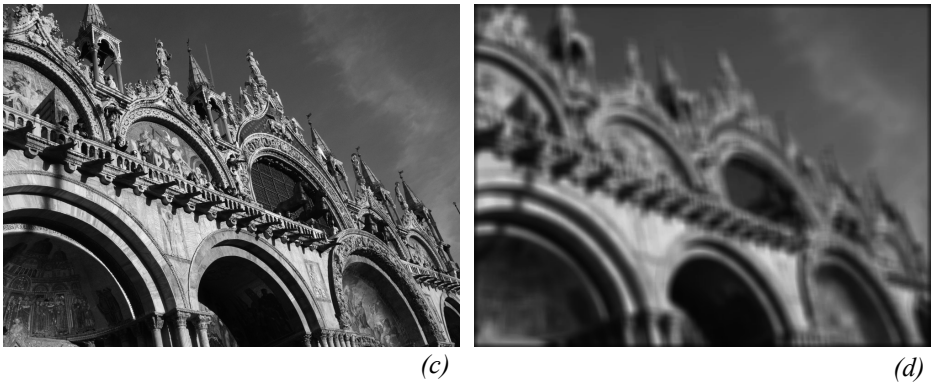


Figure 4.55 Illustration of the concept of spatial filtering. (c) Input image. (d) Output image after application of average filter.

where $m = 2a + 1$ and $n = 2b + 1$ are usually assumed odd integers. The filter w is also called *kernel*, *mask*, or *window*. As observed in (4.101), linear filtering is the process of moving a filter mask over the entire image and computing the sum of products at each location. In signal processing, this particular operation is also called *correlation* with the kernel w. It is, however, opportune to specify that an equivalent linear filtering operation is the *convolution*

$$I'(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot I(x-s, y-t) \tag{4.102}$$

where the only difference with the correlation is the presence of the minus sign, meaning that the image must be flipped. Observe that for symmetric filters *convolution* and *correlation* return the same result and the two terms can therefore be used interchangeably. The operation of convolution with the kernel w can be written in a more compact way as

$$I'(x, y) = w(x, y) * I(x, y), \tag{4.103}$$

where $*$ denotes the convolution operator.

Generating linear spatial filters requires that we specify the mn coefficients of the kernel. These coefficients are chosen based on what the filter is supposed to do. In the next section, we will see how to select these coefficients.

4.3.1.1 Smoothing filters

Smoothing filters are used for blurring and for noise reduction. Blurring is used in tasks such as removal of small details or filling of small gaps in lines or curves. Both blurring and noise reduction can be accomplished via linear or nonlinear filters. Here, we review some linear filters.

The output of a smoothing filter is simply the weighted average of the pixels contained in the filter mask. These filters are sometimes called *averaging filters* or *lowpass filters*. As we explained before, every pixel in an image is replaced by the average of the intensity of the pixels in the neighborhood defined by the filter mask. This process results in a new image with reduced sharp transitions. Accordingly, image noise gets reduced. As a side effect, however, edges—which are usually a desirable feature of an image—also get blurred. This side effect can be limited by choosing the filter coefficients appropriately. Finally observe that a *nonlinear* averaging filter can also be easily implemented by taking the *median* of the pixels contained in the mask. Median filters are particularly useful to remove *salt and pepper* noise.[12]

In the previous section, we already saw an example of constant averaging filter (figure 4.55) that simply yields the standard average of the pixels in the mask. Assuming a 3×3 mask, the filter can be written as

$$w = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \tag{4.104}$$

12. Salt and pepper noise represents itself as randomly occurring white and black pixels and is a typical form of noise in images.

where all the coefficients sum to 1. This normalization is important to keep the same value as the original image if the region by which the filter is multiplied is uniform. Also note that, instead of being 1/9, the coefficients of the filter are all 1s. The idea is that the pixels are first summed up and the result is then divided by 9. Indeed, this is computationally more efficient than multiplying each element by 1/9.

Many image-processing algorithms make use of the second derivative of the image intensity. Because of the susceptibility of such high-order derivative algorithms to changes in illumination in the basic signal, it is important to smooth the signal so that changes in intensity are due to real changes in the luminosity of objects in the scene rather than random variations due to imaging noise. A standard approach is the use of a Gaussian averaging filter whose coefficients are given by

$$G_\sigma(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}}. \tag{4.105}$$

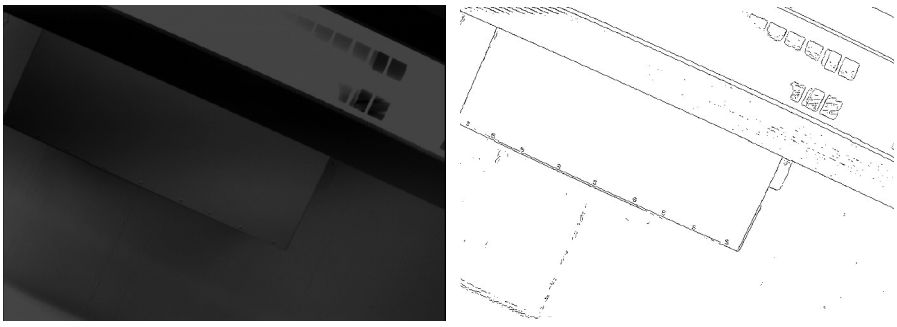
To generate, say, a 3×3 filter mask from this function, we sample it about its center. For example, with $\sigma = 0.85$, we get

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \tag{4.106}$$

where, again, the coefficients were rescaled so that they sum to 1. Also notice that the coefficients are all powers of 2, which makes it extremely efficient to compute. This filter is actually very popular. Such a lowpass filter effectively removes high-frequency noise, and this in turn causes the first derivative and especially the second derivative of intensity to be far more stable. Because of the importance of gradients and derivatives to image processing, such Gaussian smoothing preprocessing is a popular first step of virtually all computer vision algorithms.

4.3.2 Edge detection

Figure 4.56 shows an image of a scene containing a part of a ceiling lamp as well as the edges extracted from this image. Edges define regions in the image plane where a *significant* change in the image brightness takes place. As shown in this example, edge detection significantly reduces the amount of information in an image, and is therefore a useful potential feature during image interpretation. The hypothesis is that edge contours in an image correspond to important scene contours. As figure 4.56b shows, this is not entirely true. There is a difference between the output of an edge detector and an ideal line drawing.



(a) *(b)*

Figure 4.56
(a) Photo of a ceiling lamp. (b) Edges computed from (a).

Typically, there are missing contours, as well as noise contours, that do not correspond to anything of significance in the scene.

The basic challenge of edge detection is visualized in figure 4.57. The top left portion shows the 1D section of an ideal edge. But the signal produced by a camera will look more like figure 4.57 (top right) because of noise. The location of the edge is still at the same x value, but a significant level of high-frequency noise affects the signal quality.

A naive edge detector would simply differentiate, since an edge by definition is located where there are large transitions in intensity. As shown in figure 4.57 (bottom right), differentiation of the noisy camera signal results in subsidiary peaks that can make edge detection very challenging. A far more stable derivative signal can be generated simply by preprocessing the camera signal using the Gaussian smoothing function described above. Below, we present several popular edge detection algorithms, all of which operate on this same basic principle, that the derivative(s) of intensity, following some form of smoothing, comprises the basic signal from which to extract edge features.

Optimal edge detection: the Canny edge detector. The current reference edge detector throughout the vision community was invented by John Canny in 1983 [91]. This edge detector was born out of a formal approach in which Canny treated edge detection as a signal-processing problem in which there are three explicit goals:

- Maximizing the signal-to-noise ratio;
- Achieving the highest precision possible on the location of edges;
- Minimizing the number of edge responses associated with each edge.

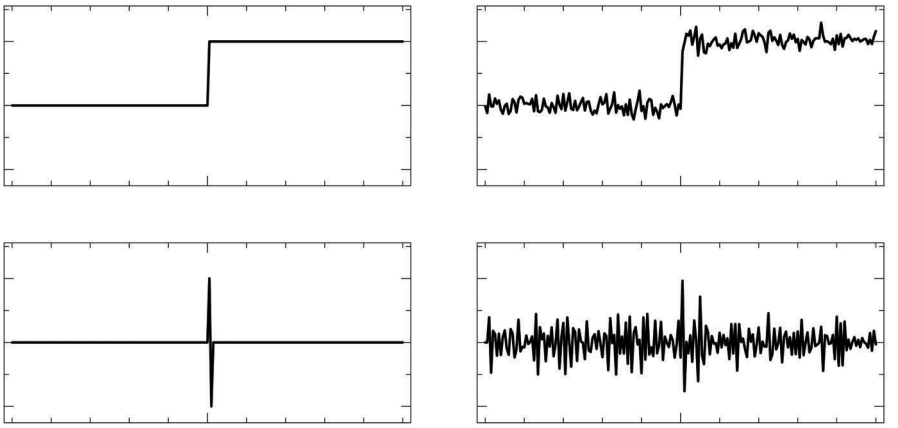


Figure 4.57
Step function example of second derivative shape and the impact of noise.

The Canny edge extractor smooths the image I via Gaussian convolution and then looks for maxima in the (rectified) derivative. In practice, the smoothing and differentiation are combined into one operation because

$$(G*I)' = G'*I \quad .^{13} \tag{4.107}$$

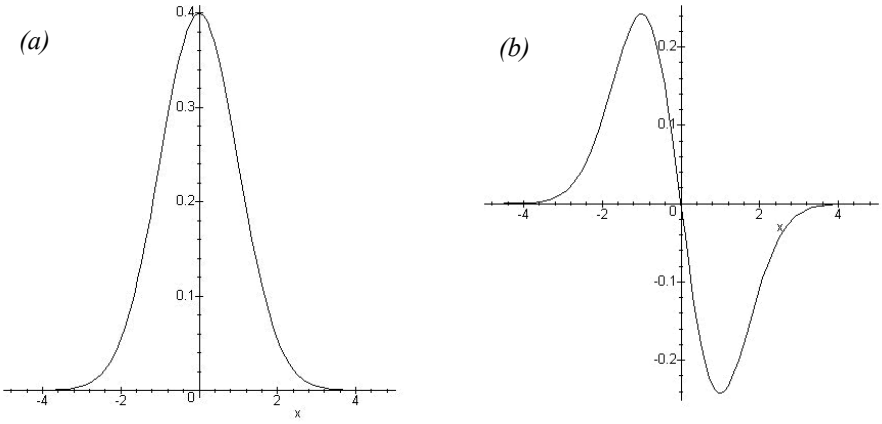
Thus, smoothing the image by convolving with a Gaussian G_σ and then differentiating is equivalent to convolving the image with G'_σ, which is the first derivative of G_σ (figure 4.58b).

We wish to detect edges in any direction. Since G' is directional, this requires application of two perpendicular filters (figure 4.59). We define the two filters as $f_V(x,y) = G'_\sigma(x)G_\sigma(y)$ and $f_H(x,y) = G'_\sigma(y)G_\sigma(x)$. The result is a basic algorithm for detecting edges at arbitrary orientations:

The algorithm for detecting edge pixels at an arbitrary orientation is as follows:

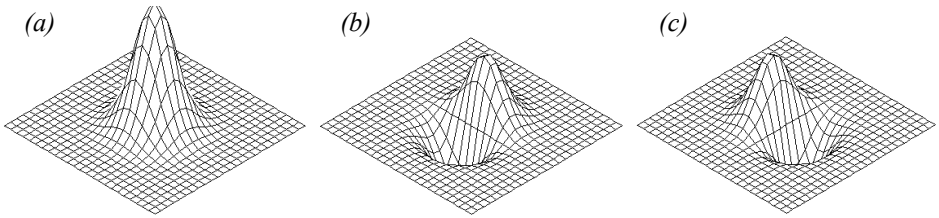
1. Convolve the image $I(x,y)$ with $f_V(x,y)$ and $f_H(x,y)$ to obtain the gradient components $R_V(x,y)$ and $R_H(x,y)$, respectively.
2. Define the square of the gradient magnitude $R(x,y) = R_V^2(x,y) + R_H^2(x,y)$.
3. Mark those peaks in $R(x,y)$ that are above some predefined threshold T.

13. This is a known property of convolution.



$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \qquad G_\sigma'(x) = \frac{-x}{\sigma^3\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

Figure 4.58
(a) A Gaussian function. (b) The first derivative of a Gaussian function.



$$G_\sigma(x,y) = G_\sigma(x)G_\sigma(y) \qquad f_V(x,y) = G'_\sigma(x)G_\sigma(y) \qquad f_H(x,y) = G'_\sigma(y)G_\sigma(x)$$

Figure 4.59
(a) Two-dimensional Gaussian function. (b) Vertical filter. (c) Horizontal filter.

Once edge pixels are extracted, the next step is to construct complete edges. A popular next step in this process is *nonmaxima suppression*. Using edge direction information, the process involves revisiting the gradient value and determining whether or not it is at a local maximum. If not, then the value is set to zero. This causes only the maxima to be preserved, and thus reduces the thickness of all edges to a single pixel (figure 4.60).

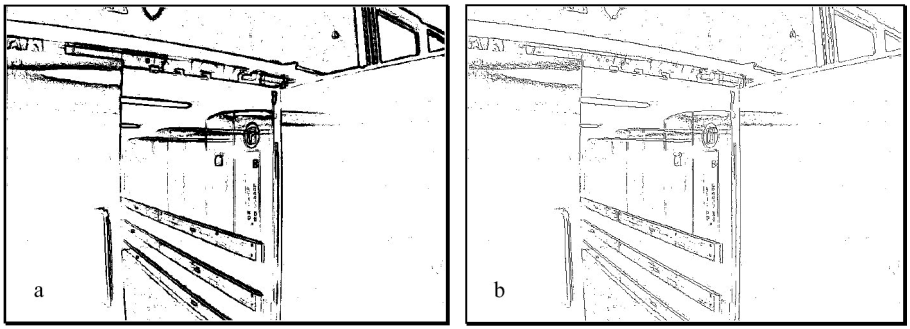


Figure 4.60
(a) Example of an edge image; (b) Nonmaxima suppression of (a).

Finally, we are ready to go from edge pixels to complete edges. First, find adjacent (or connected) sets of edges and group them into ordered lists. Second, use thresholding to eliminate the weakest edges.

Gradient edge detectors. On a mobile robot, computation time must be minimized to retain the real-time behavior of the robot. Therefore simpler, discrete kernel operators are commonly used to approximate the behavior of the Canny edge detector. One such early operator was developed by Roberts in 1965 [43]. He used two 2×2 masks to calculate the gradient across the edge in two diagonal directions. Let r_1 be the value calculated from the first mask and r_2 that from the second mask. Roberts obtained the gradient magnitude $|G|$ with the equation

$$|G| \cong \sqrt{r_1^2 + r_2^2} \;\; ; \;\; r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \;\; ; \;\; r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{4.108}$$

Prewitt (1970) [43] used two 3×3 masks oriented in the row and column directions. Let p_1 be the value calculated from the first mask and p_2 the value calculated from the second mask. Prewitt obtained the gradient magnitude $|G|$ and the gradient direction θ taken in a clockwise angle with respect to the column axis shown in the following equation.

$$|G| \cong \sqrt{p_1^2 + p_2^2} \;\;;$$

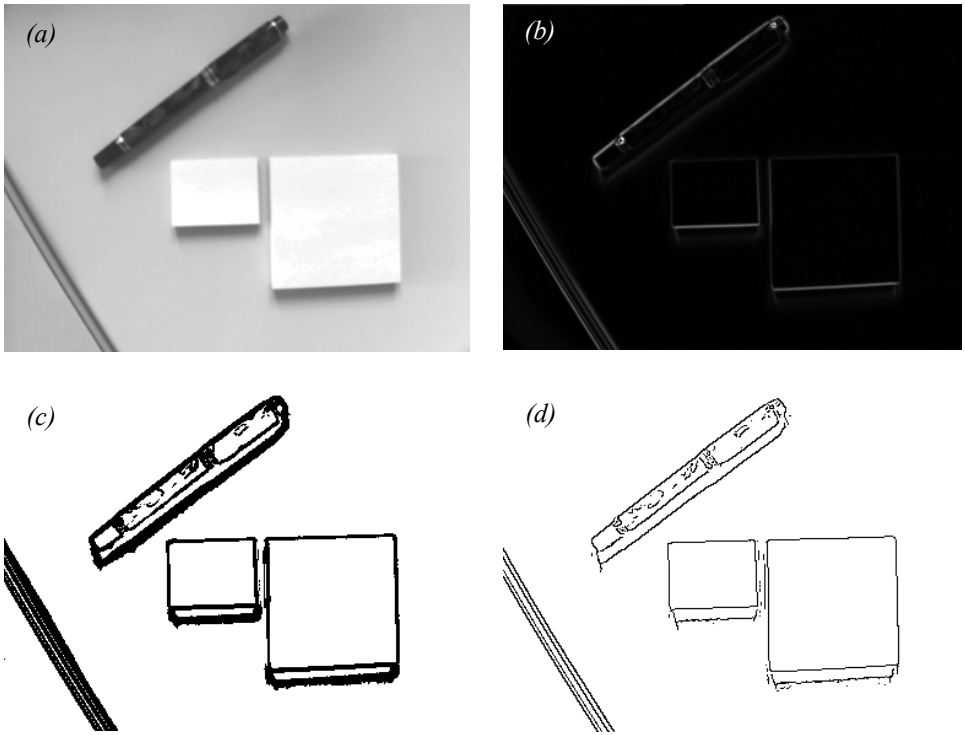


Figure 4.61
Example of visual feature extraction with the different processing steps: (a) raw image data; (b) filtered image using a Sobel filter; (c) thresholding, selection of edge pixels (d) nonmaxima suppression.

$$\theta \cong \operatorname{atan}\left(\frac{p_1}{p_2}\right) \;; \quad p_1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \;; \quad p_2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.109}$$

In the same year, Sobel [43] used, like Prewitt, two 3×3 masks oriented in the row and column direction. Let s_1 be the value calculated from the first mask and s_2 the value calculated from the second mask. Sobel obtained the same results as Prewitt for the gradient magnitude $|G|$ and the gradient direction θ taken in a clockwise angle with respect to the column axis. Figure 4.61 shows application of the Sobel filter to a visual scene.

$$|G| \cong \sqrt{s_1^2 + s_2^2} \ ;$$

$$\theta \cong \operatorname{atan}\left(\frac{s_1}{s_2}\right) \ ; \quad s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \ ; \quad s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} . \tag{4.110}$$

Dynamic thresholding. Many image-processing algorithms have generally been tested in laboratory conditions or by using static image databases. Mobile robots, however, operate in dynamic real-world settings where there is no guarantee regarding optimal or even stable illumination. A vision system for mobile robots has to adapt to the changing illumination. Therefore a constant threshold level for edge detection is not suitable. The same scene with different illumination results in edge images with considerable differences. To adapt the edge detector dynamically to the ambient light, a more adaptive threshold is required, and one approach involves calculating that threshold based on a statistical analysis of the image about to be processed.

To do this, a histogram of the gradient magnitudes of the processed image is calculated (figure 4.62). With this simple histogram it is easy to consider only the n pixels with the highest gradient magnitude for further calculation steps. The pixels are counted backward, starting at the highest magnitude. The gradient magnitude of the point where n is reached will be used as the temporary threshold value.

The motivation for this technique is that the n pixels with the highest gradient are expected to be the most relevant ones for the processed image. Furthermore, for each image, the same number of relevant edge pixels is considered, independent of illumination. It is important to pay attention to the fact that the number of pixels in the edge image delivered by the edge detector is not n. Because most detectors use nonmaxima suppression, the number of edge pixels will be further reduced.

Straight edge extraction: Hough transform. In mobile robotics, the straight edge is often extracted as a specific feature. Straight vertical edges, for example, can be used as clues to the location of doorways and hallway intersections. The Hough transform is a simple tool for extracting edges of a particular shape [21, 28]. Here we explain its application to the problem of extracting straight edges.

Suppose a pixel (x_p, y_p) in the image I is part of an edge. Any straight-line edge including point (x_p, y_p) must satisfy the equation: $y_p = m_1 x_p + b_1$. This equation can only be satisfied with a constrained set of possible values for m_1 and b_1. In other words, this equation is satisfied only by lines through I that pass through (x_p, y_p).

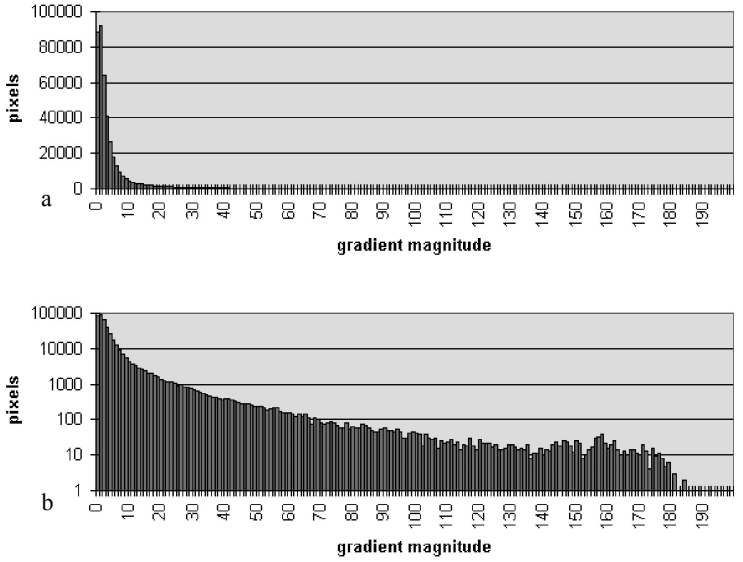


Figure 4.62
(a) Number of pixels with a specific gradient magnitude in the image of figure 4.61b. (b) Same as (a), but with logarithmic scale

Now consider a second pixel, (x_q, y_q) in I. Any line passing through this second pixel must satisfy the equation: $y_q = m_2 x_q + b_2$. What if $m_1 = m_2$ and $b_1 = b_2$? Then the line defined by both equations is one and the same: it is the line that passes through both (x_p, y_p) and (x_q, y_q).

More generally, for all pixels that are part of a single straight line through I, they must all lie on a line defined by the *same* values for m and b. The general definition of this line is, of course, $y = mx + b$. The Hough transform uses this basic property, creating a mechanism so that each edge pixel can "vote" for various values of the (m, b) parameters. The lines with the most votes at the end are straight edge features:

- Create a 2D array A with axes that tessellate the values of m and b.
- Initialize the array to zero: $A[m, b] = 0$ for all values of m, b.
- For each edge pixel (x_p, y_p) in I, loop over all values of m and b:
  if $y_p = mx_p + b$ then $A[m, b]$+=1.
- Search the cells in A to identify those with the largest value. Each such cell's indices (m, b) correspond to an extracted straight-line edge in I.

4.3.3 Computing image similarity

In this section, we review the three most popular image similarity measures used for solving the correspondence problem in structure from stereo (section 4.2.5) and structure from motion (section 4.2.6). The methods we are about to describe are all *area-based* (page 174). Suppose that we want to compare a $m \times n$ patch in image I_1 centered on (u,v) with another patch of the same size centered on (u', v') in image I_2. We assume that these are odd integers, therefore $m = 2a + 1$ and $n = 2b + 1$. The similarity is then computed between the gray intensity levels of the two patches. Some of the most popular criteria are:

Sum of Absolute Differences (SAD)

$$SAD = \sum_{k=-a}^{a} \sum_{l=-b}^{b} \left| I_1(u+k, v+l) - I_2(u'+k, v'+l) \right| . \tag{4.111}$$

Sum of Squared Differences (SSD)

$$SSD = \sum_{k=-a}^{a} \sum_{l=-b}^{b} [I_1(u+k, v+l) - I_2(u'+k, v'+l)]^2 . \tag{4.112}$$

Normalized Cross Correlation (NCC)

$$NCC = \frac{\sum_{k=-a}^{a} \sum_{l=-b}^{b} [I_1(u+k, v+l) - \mu_1] \cdot [I_2(u'+k, v'+l) - \mu_2]}{\sqrt{\sum_{k=-a}^{a} \sum_{l=-b}^{b} [I_1(u+k, v+l) - \mu_1]^2 \sum_{k=-a}^{a} \sum_{l=-b}^{b} [I_2(u'+k, v'+l) - \mu_2]^2}} , \tag{4.113}$$

where

$$\mu_1 = \frac{1}{mn} \sum_{k=-a}^{a} \sum_{l=-b}^{b} I_1(u+k, v+l) \tag{4.114}$$

$$\mu_2 = \frac{1}{mn} \sum_{k=-a}^{a} \sum_{l=-b}^{b} I_2(u'+k, v'+l) \tag{4.115}$$

are the mean values of the two image patches.

The SAD is the simplest among these similarity measures. It is calculated by subtracting pixels between the reference image I_1 and the target image I_2 followed by the aggregation of absolute differences within the patch. The SSD has a higher computational complexity compared to the SAD, since it involves numerous multiplication operations (i.e., squared). Notice that if the left and right images match perfectly, the resultant of SAD and SSD will be zero.

The NCC is even more complex than both SAD and SSD algorithms, since it involves numerous multiplication, division, and square root operations; however, it but provides more distinctiveness than SSD and SAD and also invariance to affine intensity changes (see also figure 4.69). Finally, note that the value of NCC ranges between −1 and 1 where 1 corresponds to maximum similarity between the two image patches.

4.4 Feature Extraction

An autonomous mobile robot must be able to determine its relationship to the environment by making measurements with its sensors and then using those measured signals. A wide variety of sensing technologies are available, as shown in section 4.1. But every sensor we have presented is imperfect: measurements always have error and, therefore, uncertainty associated with them. Therefore, sensor inputs must be used in a way that enables the robot to interact with its environment successfully in spite of measurement uncertainty.

There are two strategies for using uncertain sensor input to guide the robot's behavior. One strategy is to use each sensor measurement as a raw and individual value. Such raw sensor values could, for example, be tied directly to robot behavior, whereby the robot's actions are a function of its sensor inputs. Alternatively, the raw sensor values could be used to update an intermediate model, with the robot's actions being triggered as a function of this model rather than of the individual sensor measurements.

The second strategy is to extract information from one or more sensor readings first, generating a higher-level *percept* that can then be used to inform the robot's model and perhaps the robot's actions directly. We call this process *feature extraction*, and it is this next, optional step in the perceptual interpretation pipeline (figure 4.63) that we will now discuss.

In practical terms, mobile robots do not necessarily use feature extraction and scene interpretation for every activity. Instead, robots will interpret sensors to varying degrees depending on each specific functionality. For example, in order to guarantee emergency stops in the face of immediate obstacles, the robot may make direct use of raw forward-facing range readings to stop its drive motors. For local obstacle avoidance, raw ranging sensor strikes may be combined in an occupancy grid model, enabling smooth avoidance of obstacles meters away. For map-building and precise navigation, the range sensor values and even vision-sensor measurements may pass through the complete perceptual pipeline,

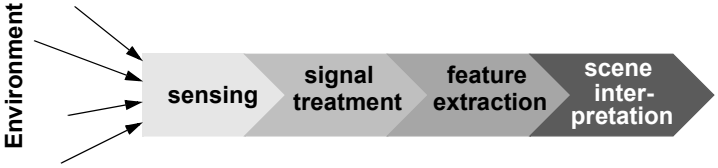


Figure 4.63
The perceptual pipeline: from sensor readings to knowledge models.

being subjected to feature extraction followed by scene interpretation to minimize the impact of individual sensor uncertainty on the robustness of the robot's mapmaking and navigation skills. The pattern that thus emerges is that, as one moves into more sophisticated, long-term perceptual tasks, the feature-extraction and scene-interpretation aspects of the perceptual pipeline become essential.

Feature definition. Features are recognizable structures of elements in the environment. They can usually be extracted from measurements and mathematically described. Good features are always perceivable and easily detectable from the environment. We distinguish between *low-level features* (*geometric primitives*) such as lines, points, corners, blobs, circles, or polygons and *high-level features* (*objects*) such as doors, tables, or trash cans. At one extreme, raw sensor data provide a large volume of data, but with low distinctiveness of each individual quantum of data. Making use of raw data has the potential advantage that every bit of information is fully used, and thus there is a high conservation of information. Low-level features are abstractions of raw data, and as such they provide a lower volume of data while increasing the distinctiveness of each feature. The hope, when one incorporates low-level features, is that the features are filtering out poor or useless data, but of course it is also likely that some valid information will be lost as a result of the feature-extraction process. High-level features provide maximum abstraction from the raw data, thereby reducing the volume of data as much as possible while providing highly distinctive resulting features. Once again, the abstraction process has the risk of filtering away important information, potentially lowering data utilization.

Although features must have some spatial locality, their geometric extent can range widely. For example, a corner feature inhabits a specific coordinate location in the geometric world. In contrast, a visual "fingerprint" identifying a specific room in an office building applies to the entire room, but it has a location that is spatially limited to the one particular room.

In mobile robotics, features play an especially important role in the creation of environmental models. They enable more compact and robust descriptions of the environment,

helping a mobile robot during both map-building and localization. When designing a mobile robot, a critical decision revolves around choosing the appropriate features for the robot to use. A number of factors are essential to this decision.

Target environment. For geometric features to be useful, the target geometries must be readily detected in the actual environment. For example, line features are extremely useful in office building environments due to the abundance of straight wall segments, while the same features are virtually useless when navigating on Mars. Conversely, point features (such as corners and blobs) are more likely to be found in any textured environment. As an example, consider that the two NASA Mars explorations rovers, Spirit and Opportunity, used corner features (section 4.5) for visual odometry [203].

Available sensors. Obviously, the specific sensors and sensor uncertainty of the robot impacts the appropriateness of various features. Armed with a laser rangefinder, a robot is well qualified to use geometrically detailed features such as corner features owing to the high-quality angular and depth resolution of the laser scanner. In contrast, a sonar-equipped robot may not have the appropriate tools for corner feature extraction.

Computational power. Visual feature extraction can effect a significant computational cost, particularly in robots where the vision sensor processing is performed by one of the robot's main processors.

Environment representation. Feature extraction is an important step toward scene interpretation, and by this token the features extracted must provide information that is consonant with the representation used for the environmental model. For example, nongeometric visual features are of little value in purely geometric environmental models but can be of great value in topological models of the environment. Figure 4.64 shows the application of two different representations to the task of modeling an office building hallway. Each approach has advantages and disadvantages, but extraction of line and corner features has much more relevance to the representation on the left. Refer to chapter 5, section 5.5 for a close look at map representations and their relative trade-offs.

In sections 4.5−4.7, we present specific feature extraction techniques based on the two most popular sensing modalities of mobile robotics: vision and range sensing.

Visual interpretation is, as we have mentioned before, an extremely challenging problem to fully solve. Significant research effort has been dedicated over the past several decades to inventing algorithms for understanding a scene based on 2D images, and the research efforts have slowly produced fruitful results.

In section 4.2 we saw vision ranging and color-tracking sensors that are commercially available for mobile robots. These specific vision applications have witnessed commercial

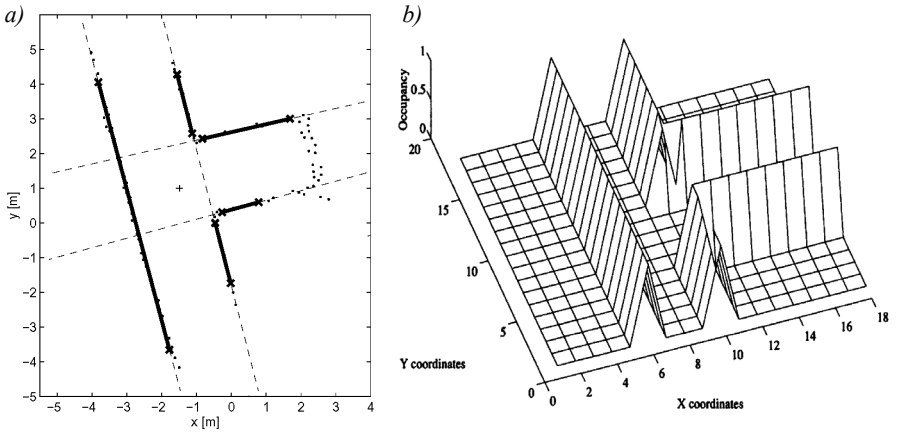


Figure 4.64
Environment representation and modeling: (a) feature-based (continuous metric); (b) occupancy grid (discrete metric). Courtesy of Sjur Vestli.

solutions primarily because the challenges are in both cases relatively well focused and the resulting, problem-specific algorithms are straightforward. But images contain much more than implicit depth information and color blobs. We would like to solve the more general problem of extracting a large number of feature types from images.

The next section presents some point feature extraction techniques that are relevant to mobile robotics along these lines. Two key requirements must be met for a visual feature extraction technique to have mobile robotic relevance. First, the method must operate in real time. Mobile robots move through their environment, and so the processing simply cannot be an offline operation. Second, the method must be robust to the real-world conditions outside of a laboratory. This means that carefully controlled illumination assumptions and carefully painted objects are unacceptable requirements.

Throughout the following descriptions, keep in mind that vision interpretation is primarily about the challenge of *reducing information*. A sonar unit produces perhaps fifty bits of information per second. By contrast, a CCD camera can output 240 *million* bits per second! The sonar produces a tiny amount of information from which we hope to draw broader conclusions. But the CCD chip produces too much information, and this overabundance of information mixes together relevant and irrelevant information haphazardly. For example, we may intend to measure the color of a landmark. The CCD camera does not simply report its color, but also measures the general illumination of the environment, the direction of illumination, the defocusing caused by optics, the side effects imposed by nearby objects with different colors, and so on. Therefore, the problem of visual feature extraction is

largely one of removing the majority of irrelevant information in an image so that the remaining information unambiguously describes specific features in the environment.

4.5 Image Feature Extraction: Interest Point Detectors

In this section, we define the concept of the local feature and review some of the most consolidated feature extractors. As the computer vision literature in this field is very large, we will only describe in detail the two most popular feature detectors, namely Harris and SIFT, and will briefly introduce the others by explaining the main advantages and disadvantages and domain of application. For the interested reader, a comprehensive survey on local feature detectors can be found in [320].

4.5.1 Introduction

A local feature is an image pattern that differs from its immediate neighborhood in terms of intensity, color, and texture. Local features can be small image patches (such as regions of uniform color), edges, or points (such as corners originated from line intersections). In the modern terminology, local features are also called interest points, interest regions, or keypoints.

Depending on their semantic content, local features can be divided into three different categories. In the first category are features that have a semantic interpretation such as, for instance, edges corresponding to lanes of the road or blobs corresponding to blood cells in medical images. This is the case in most automotive applications, airborne images, and medical image processing. Furthermore, these were also the first applications for which local feature detectors have been proposed. In the second category are features that do not have a semantic interpretation. Here, what the features actually represent is not relevant. What matters is that their location can be determined accurately and robustly over time. Typical applications are feature tracking, camera calibration, 3D reconstruction, image mosaicing, and panorama stitching. Finally, in the third category are features that still do not have a semantic interpretation if taken individually, but that can be used to recognize a scene or an object if taken all together. For instance, a scene could be recognized counting the number of feature matches between the observed scene and the query image. In this case, the location of the feature is not important; only the number of matches is relevant. Application domains include texture analysis, scene classification, video mining, and image retrieval (see, for instance, Google Images, Microsoft Bing Images, Youtube, or Tineye.com). This principle is the basis of the *visual-word*-based place recognition that will be described in section 4.6.

4.5.2 Properties of the ideal feature detector

In this section we summarize the properties that an ideal feature detector should have. Let us start with a concrete example from digital image photography. Most of today's digital consumer cameras come with software for automatic stitching of panoramas from multiple photos. An example is shown in figure 4.65. The user simply takes several shots of the scene with little overlap between adjacent pictures and the software automatically aligns and fuses them all together into a cylindrical panorama (figure 4.65a). The key challenge is to identify corresponding regions between overlapping images. As the reader may perceive, one way to solve this problem is to extract feature points from adjacent pictures, find corresponding pairs according to some similarity measure (figure 4.65b), and compute the transformation (e.g., homography) to align them (figure 4.65c). The first problem is how to detect the same points independently in both images. In figure 4.65d, for instance, the features from the left image are not redetected in the right image. Because of this, we need a "repeatable" feature detector. The second problem is: for each point in the first image we need to correctly recognize the corresponding one in the second image. Thus, the detected features should be very distinctive (i.e., highly distinguishable).

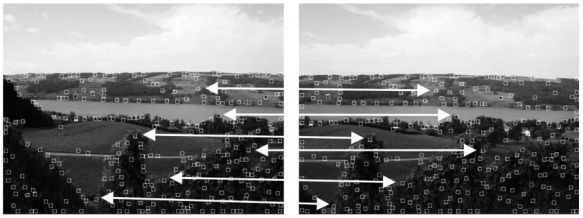
"Repeatability" is probably the most important property of a good feature detector. Given two images of the same scene taken under different viewing and illumination conditions, it is desirable that a high percentage of the features of the first image can be redetected in the second image. This requires the feature be invariant to view point changes, such as camera rotation or zoom (i.e., scale), and illumination changes.

The second important property is "distinctiveness," that is, the information carried by the patch surrounding the feature point should be as distinctive as possible so that the features can be distinguished and matched. For instance, the corners of a chessboard are not distinctive because they cannot be distinguished from each other. As we will see later, distinctiveness is also the main difference between Harris and SIFT features. Harris privileges corners (such as edge intersections), while SIFT privileges image patches with highly informative content (i.e., not corners).

Other important properties of a good feature detector are:

- Localization accuracy: the detected features should be accurately localized, both in image position and scale. Accuracy is especially important in camera calibration, 3D reconstruction from images ("structure from motion"), and panorama stitching.
- Quantity of features: the ideal number of detected features depends on the application. For most of the tasks like object or scene recognition, image retrieval, and 3D reconstruction it is important to have a sufficiently large number of features, this in order to increase the recognition rate or the accuracy of the reconstruction. However, if the feature had a semantic interpretation, then a small number of features would be enough to

(a)

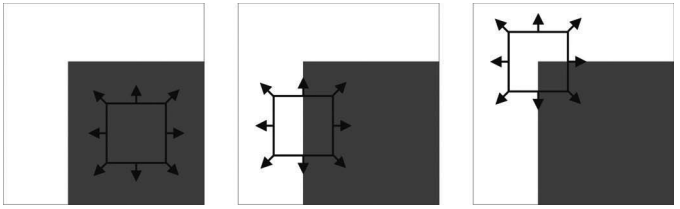


(b)

(c)

(d) No chance to match!

Figure 4.65 (a) Panorama built from multiple overlapping images using Autostitch software. (b) First step: select salient features in both images and match corresponding ones. (c) Second step: compute the transformation between the two corresponding sets and align the images. (d) Two example images where features were not redetected and therefore there is no chance to match them.



(a) Flat region *(b)* Edge *(c)* Corner

Figure 4.66 (a) "Flat" region: no change in all directions. (b) "Edge": no change along the edge direction. (c) "Corner": significant change in all directions.

recognize a scene (as an example, some semantic "high level" features could be individual objects or objects parts such as table, chair, table leg, door, and so on).

- Invariance: good features should be invariant to changes of camera viewpoint, environment illumination, and scale (like zoom or camera translation). Invariance can be achieved within a certain range when these changes can be modeled as mathematical transformations (see section 4.5.4). A successful result in this direction has been successfully demonstrated by some of the recent detectors like SIFT (section 4.5.5.1).
- Computational efficiency: it is also desirable that features can be detected and matched very efficiently. In the framework of the project undertaken by Google Images of organizing all the images on the web, computation efficiency is a critical component as its database—nowadays composed of billions of images—grows more and more every year. This is even important in robotics, where most of the applications need to work in real-time. However, the time of detection and matching of a feature is strictly related to the degree of invariance desired: the higher the level of invariance, the more image transformations to check, and, thus, the longer the computation time.
- Robustness: the detected features should be robust to image noise, discretization effects, compression artifacts, blur, deviations from the mathematical model used to obtain invariance, and so on.

4.5.3 Corner detectors

A corner in an image can be defined as the intersection of two or more edges. Corners are features with high repeatability.

The basic concept of corner detection. One of the earliest corner detectors was invented by Moravec [234, 235]. He defined a corner as a point where there is a large intensity variation in every direction. An intuitive explanation of his corner detection algorithm is given in figure 4.66. Intuitively, one could recognize a corner by looking through a small window

centered on the pixel. If the pixel lies in a "flat" region (i.e., a region of uniform intensity), then the adjacent windows will look similar. If the pixel is along an edge, then adjacent windows in the direction perpendicular to the edge will look different, but adjacent windows in a direction parallel to the edge will result only in a small change. Finally, if the pixel lies on a corner, then none of the adjacent windows will look similar. Moravec used the Sum of Squared Differences (SSD, section 4.3.3) as a measure of the similarity between two patches. A low SSD indicates more similarity. If this number is locally maximal, then a corner is present.

4.5.3.1 The Harris corner detector

Harris and Stephens [146] improved Moravec's corner detector by considering the partial derivatives of the SSD instead of using shifted windows.

Let I be a grayscale image. Consider taking an image patch centered on (u, v) and shifting it by (x, y). The Sum of Squared Differences SSD between these two patches is given by:

$$SSD(x, y) = \sum_u \sum_v ((I(u, v)) - I(u + x, v + y))^2 . \tag{4.116}$$

$I(u + x, v + y)$ can be approximated by a first-order Taylor expansion. Let I_x and I_y be the partial derivatives of I, such that

$$I(u + x, v + y) \approx I(u, v) + I_x(u, v)x + I_y(u, v)y . \tag{4.117}$$

This produces the approximation

$$SSD(x, y) \approx \sum_u \sum_v (I_x(u, v)x + I_y(u, v)y)^2 , \tag{4.118}$$

which can be written in matrix form:

$$SSD(x, y) \approx \begin{bmatrix} x & y \end{bmatrix} M \begin{bmatrix} x \\ y \end{bmatrix} , \tag{4.119}$$

where M is the second moment matrix

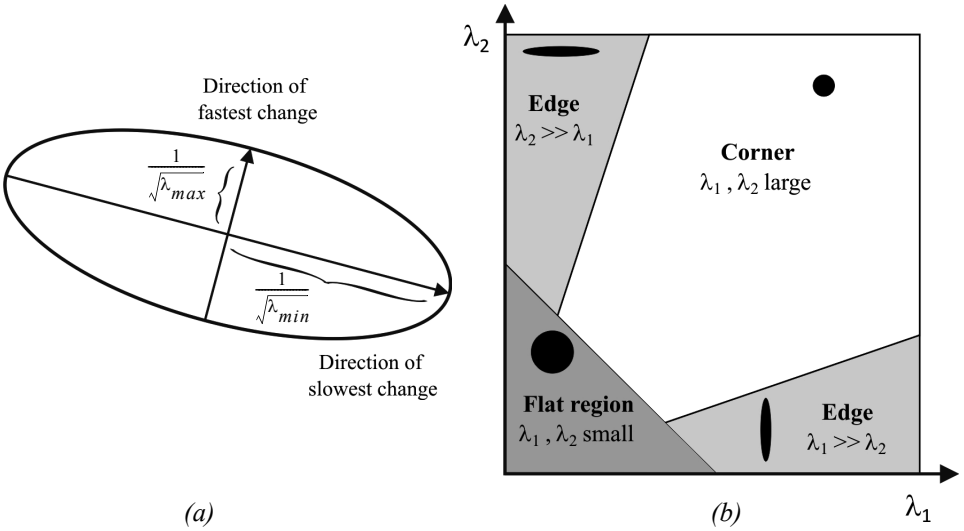


Figure 4.67 (a) This ellipse is built from the second moment matrix and visualizes the directions of fastest and lowest intensity change. (b) The classification of corner and edges according to Harris and Stephens.

$$M = \sum_u \sum_v \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \sum\sum I_x^2 & \sum\sum I_x I_y \\ \sum\sum I_x I_y & \sum\sum I_y^2 \end{bmatrix}. \tag{4.120}$$

And since M is symmetric, we can rewrite M as

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R, \tag{4.121}$$

where λ_1 and λ_2 are the eigenvalues of M.

As mentioned before, a corner is characterized by a large variation of SSD in all directions of the vector (x, y). The Harris detector analyses the eigenvalues of M to decide if we are in presence of a corner or not. Let us first give an intuitive explanation before showing the mathematical expression.

Using equation (4.119) we can visualize M as an ellipse (figure 4.67a) of equation:

$$\begin{bmatrix} x & y \end{bmatrix} M \begin{bmatrix} x \\ y \end{bmatrix} = const. \tag{4.122}$$

The axis lengths of this ellipse are determined by the eigenvalues of M and the orientation is determined by R.

Based on the magnitudes of the eigenvalues, the following inferences can be made based on this argument:

- If both λ_1 and λ_2 are small, SSD is almost constant in all directions (i.e., we are in presence of a flat region).
- If either $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$, we are in presence of an edge: SSD has a large variation only in one direction, which is the one perpendicular to the edge.
- If both λ_1 and λ_2 are large, SSD has large variations in all directions and then we are in presence of a corner.

The three situations mentioned above are pictorially summarized in figure 4.67b.

Because the calculation of the eigenvalues is computationally expensive, Harris and Stephens suggested the use of the following "cornerness function" instead:

$$C = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(M) - \kappa \cdot trace^2(M), \tag{4.123}$$

where κ is a tunable sensitivity parameter. This way, instead of computing the eigenvalues of M, we just need to evaluate the determinant and trace of M. The value of κ has to be determined empirically. In the literature, values are often reported in the range 0.04–0.15.

The last step of the Harris corner detector consists in extracting the local maxima of the cornerness function, using *nonmaxima suppression*[14]. Finally only the local maxima which are above a given threshold are retained. The processing steps are illustrated in figure 4.68.

Figure 4.68c shows the corners detected for the two example images. Notice that the images are related by a rotation and a slight change of illumination. As can be seen, many of the features detected in the left image have also been redetected in the right image. This means that the repeatability of the Harris detector under rotations and small changes of illumination is high. In the next section, we will point out the properties of the Harris detector as well as its drawbacks.

14. *Nonmaxima suppression* involves revisiting every pixel of the cornerness function and determining whether or not it is at a local maximum. If not, then the value is set to zero. This causes only the maxima to be preserved.

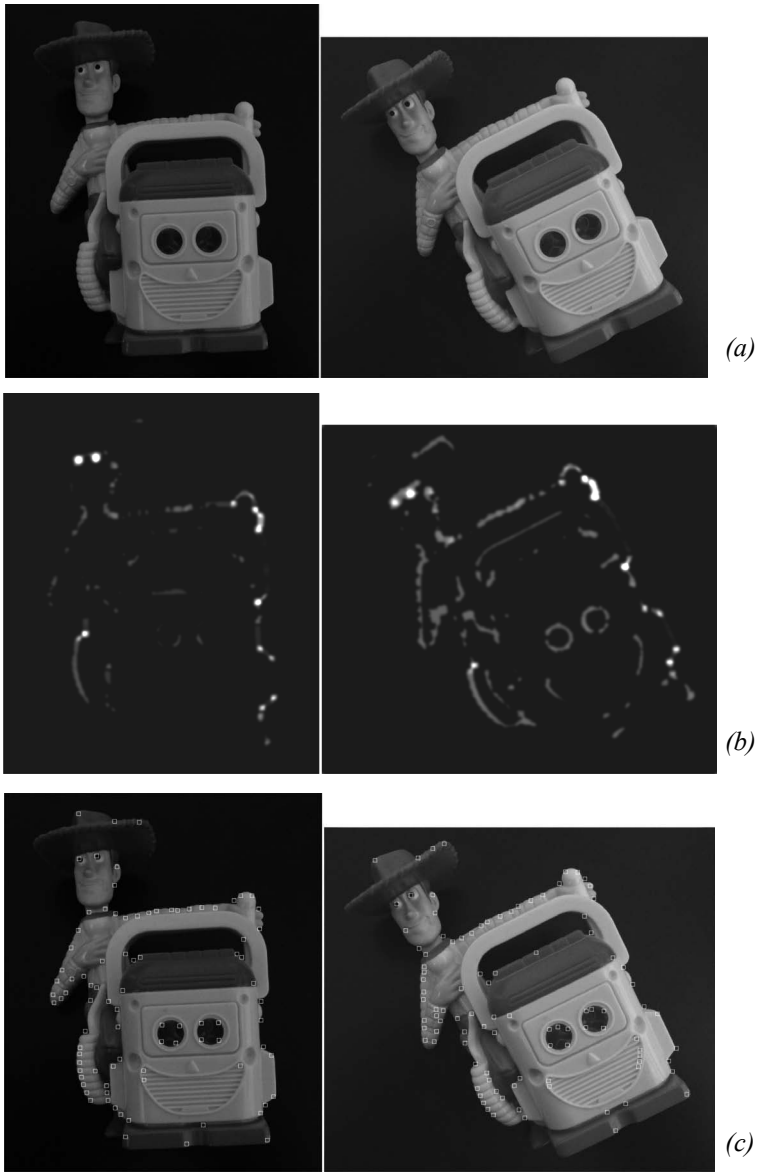


Figure 4.68 Extraction of the Harris corners. (a) Original image. (b) Cornerness function. (c) Harris corners are identified as local maxima of the cornerness function (only local maxima larger than a given threshold are retained). Two images of the same object are shown, which differ by illumination and orientation.

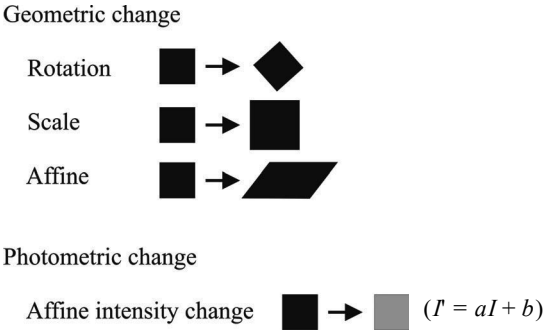


Figure 4.69 Models of image changes.

4.5.4 Invariance to photometric and geometric changes

As observed in section 4.5.2, in general we want features to be detected despite geometric and photometric changes in the image: if we have two transformed versions of the same image, features should be detected in corresponding locations. Image transformations can affect the geometric or the photometric properties of an image. Consolidated models of image transformations are the following (see also figure 4.69):

- Geometric changes:
  - 2D Rotation
  - Scale (uniform rescaling)
  - Affine
- Photometric changes
  - Affine intensity

Observe that we did not mention changes of camera viewpoint (i.e., perspective distortions). In this case the transformation valid only for planar objects would be a homography. However, when the viewpoint changes are small and the object is locally planar, the affine transformation is a good approximation of the homography. Observe also that 2D rotation occurs only if the camera rotates purely about its optical axis. Uniform rescaling, instead, appears when the camera zooms (in or out) or translates along the direction of its optical axis, but the latter is valid only for locally planar objects.

As an example, let us now examine the invariance of the Harris detector to the above mentioned transformations. We can observe that the Harris detector is invariant to 2D image rotations. This can be explained by observing that the eigenvalues of the second moment matrix do not change under pure rotation. Indeed, the ellipse rotates, but its shape

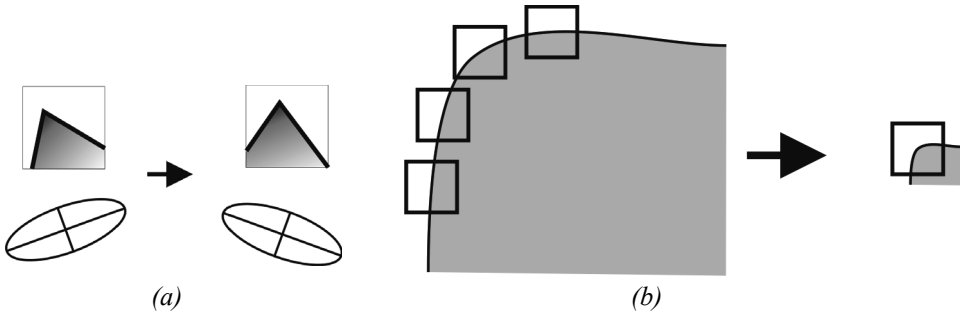


Figure 4.70 (a) Harris detector is invariant to image rotations: the ellipse rotates but its shape (i.e. eigenvalues) remains the same. (b) Conversely, it is not invariant to image scale: at a large scale (left) all points along the corner would be classified as edges, while at a smaller scale case (right) the point would be classified as corner.

(i.e., the eigenvalues) remains the same (see figure 4.70a). As a matter of fact, observe that to make the Harris detector isotropic (i.e., uniform for all rotations), the second moment matrix should be computed in a circular region rather in a squared window. This is usually done by averaging equation (4.120) with a circular symmetric Gaussian function.

The Harris detector is also invariant to affine intensity changes. In this case, the eigenvalues, and so the cornerness function, are rescaled by a constant factor, but the position of the local maxima of the cornerness function remains the same. Conversely, the Harris detector is not invariant to geometric affine transformations or scale changes. Intuitively, an affine transformation distorts the neighborhood of the feature along the x and y directions and, accordingly, a corner can get reduced or increased its curvature. Regarding scale changes, this is immediately clarified as observed in figure 4.70b. In this figure, the corner would be classified as an edge at a high scale and as a corner at a smaller scale.

The performance of the Harris detector against scale changes is shown in figure 4.71 according to a comparative study made in [216]. In this figure, the repeatability rate is plotted versus the scale factor. The repeatability rate between two images is computed as the ratio between the number of found correspondences and the number of all possible correspondences. As observed, after rescaling the image by a factor 2, only 20% of the possible correspondences are redetected.

Although it is not invariant to scale changes, the Harris detector, as in its original implementation by Harris and Stephens, is still widely used and can be found in the well-known Intel open-source computer vision library (OpenCV [343]). Furthermore, in a comparative study of different interest point detectors, Schmid et al. [285] showed that the Harris corners are among the most repeatable and most informative features. As we will see in the next sections, some modifications to the original implementation have made the Harris detector also invariant to scale and affine changes. Additionally, the location accuracy of

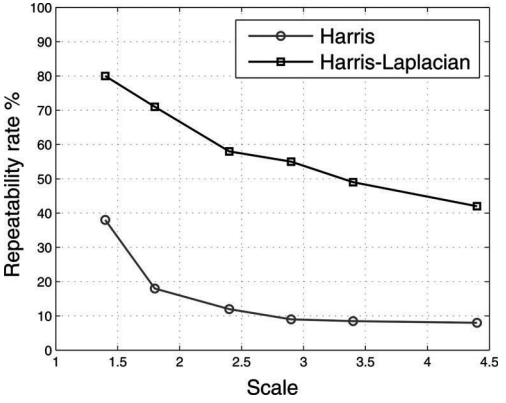


Figure 4.71 Repeatability rate of Harris detector. Comparison with Harris-Laplacian. This plot is the result of a comparative study presented in [216].

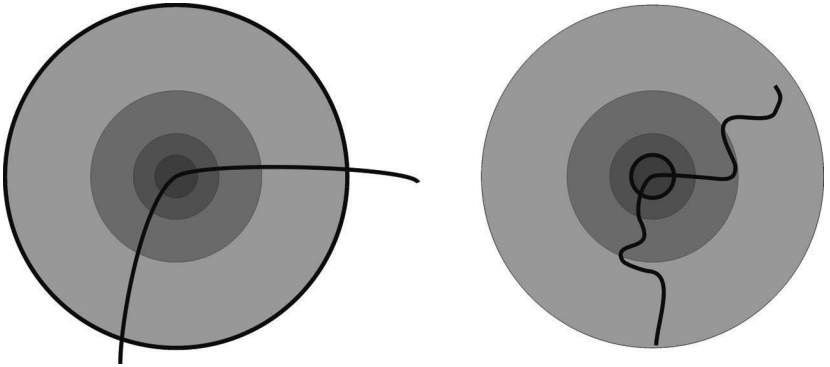


Figure 4.72 To achieve scale-invariant detection, the image is analyzed at different scales. This means, the Harris detector is applied several times on the image, each time with a different window size.

the Harris corners can be improved up to subpixel precision. This can be achieved by approximating the cornerness function in the neighborhood of a local maximum through a quadratic function.

4.5.4.1 Scale-invariant detection

In this section, we will describe the modifications that have been devised to make the Harris detector invariant to scale changes. If we look at figure 4.72, we will notice that one way to detect the corner at higher scale is to use a multiscale Harris detector. This means that

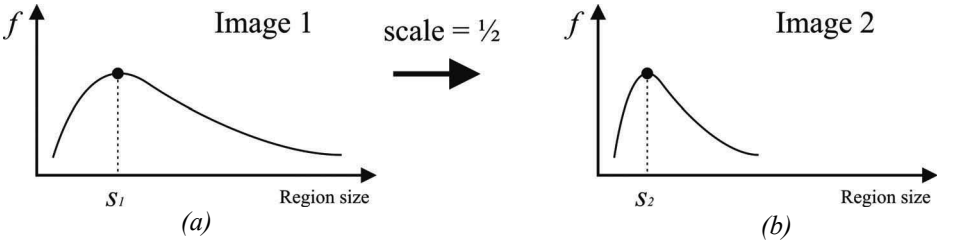


Figure 4.73 Average intensity as a function of the region size. (a) Original image. (b) Resized image.

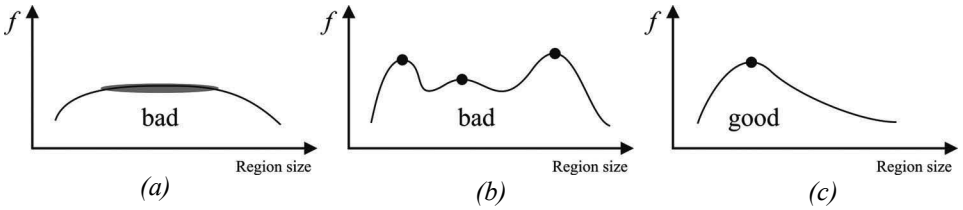


Figure 4.74 (a)-(b) are bad scale invariant functions. (c) is a good.

the same detector is applied several times on the image, each time with a different window (circle) size. Note, an efficient implementation of multiscale detection uses the so called scale-space pyramid: instead of varying the window size of the feature detector, the idea is to generate upsampled or downsampled versions of the original image (i.e., pyramid). Using multiscale Harris detector, we can be sure that at some point the corner of image 4.72a will get detected. Once the point has been detected in both images a question arises as: how do we select the corresponding scale? In other words, how do we choose corresponding circles independently in each image?

In computer vision, the correct-scale selection is usually done by following the approach proposed in 1998 by Lindeberg [193]: the appropriate scale of a local feature can be chosen as the one for which a given function reaches a maximum or minimum over scales. Let us give an intuitive explanation. For every circle in the two images in figure 4.72, let us plot the average intensity of the pixels within the circle as a function of the circle size. For these images, we will get the two plots shown in figure 4.73. As expected, these two functions look the same up to a rescaling in the x-axis. The solution to our problem is therefore to take as corresponding scales the circle sizes for which these functions reach their maximum. Depending of the chosen function, we might take the minimum instead of the maximum.

The problem is how to design a good function. The first observation we can make is that a "good" function for scale detection should have one stable sharp peak like that in figure 4.74. Despite the one used in our example, the average intensity is not good because it can

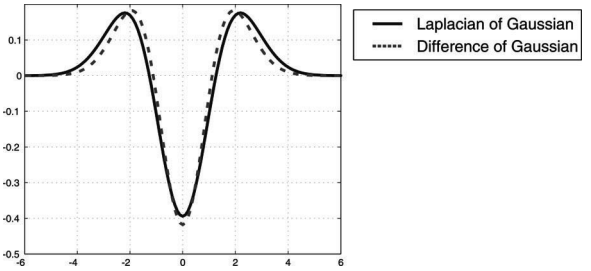


Figure 4.75 Comparison between Laplacian of Gaussian and Difference of Gaussian

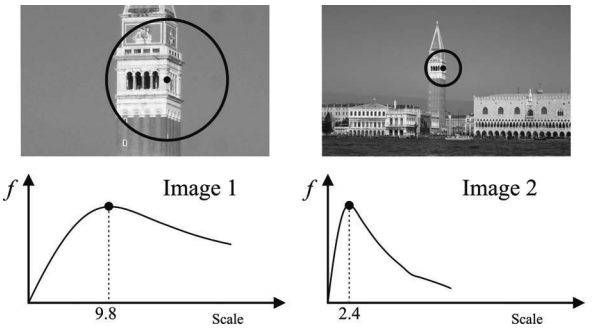


Figure 4.76 Response of the LOG operator over two corresponding points from two images taken at different scales.

return multiple peaks or even no peaks at all. As a matter of fact, it turns out that for usual images a good function is one that corresponds to contrast, that is, sharp local intensity changes. The Laplacian of Gaussian (LoG) (figure 4.75) is a good operator for identifying sharp intensity changes and is currently the one used for scale selection by the Harris corner detector. In a comparative study presented in [216, 101], the LoG operator has been shown to give the best results with respect to other functions. The response of the LoG over two image features is shown in figure 4.76.

The multiscale Harris detector is known as Harris-Laplacian and was implemented by Mikolajczyk and Schmid [217]. The comparison between the standard Harris and the Harris-Laplacian over scale is shown in figure 4.71.

4.5.4.2 Affine invariant detection

As mentioned in section 4.5.4, affine transformation is a good approximation of perspective distortion of locally planar patches under small viewpoint changes. In the previous section, we considered the problem of detection under uniform rescaling. The problem now is how

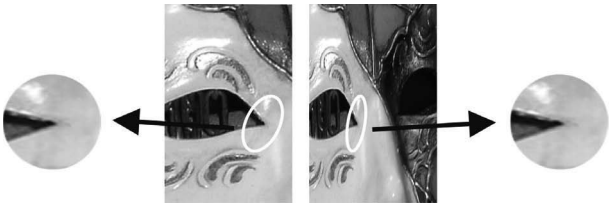


Figure 4.77 Computation of the affine invariant ellipse in two images related by an affine transformation.

to detect the same features under affine transformation, which can be seen as a nonuniform rescaling. The procedure consists in the following steps:

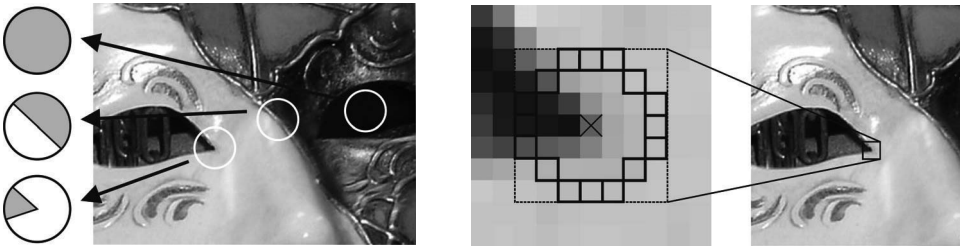
- First, the features are identified using the scale invariant Harris-Laplacian detector.
- Then, the second moment matrix (4.120) is used to identify the two directions of slowest and fastest change of intensity around the feature.
- Out of these two directions, an ellipse is computed to the same size as the scale computed with the LoG operator.
- The region inside the ellipse is normalized to a circular one.
- The initial detected ellipse and the resulting normalized circular shape are shown in figure 4.77.

The affine invariant Harris detector is known as Harris-Affine and was devised by Mikolajczyk and Schmid [218].

4.5.4.3 Other corner detectors

The Shi-Tomasi corner detector. This is also sometimes referred to as the Kanade-Tomasi corner detector [284]. This detector is strongly based on the Harris corner detector. The authors show that for image patches undergoing affine transformations, $min(\lambda_1, \lambda_2)$ is a better measure than the cornerness function C (4.123).

The SUSAN corner detector. SUSAN stands for Smallest Univalue Segment Assimilating Nucleus and, besides being used for corner detection, it is also used for edge detection and noise suppression. The SUSAN corner detector has been introduced by Smith and Brady [296]. Its working principle is different from the Harris detector. As we have seen, Harris is based on local image gradients, which are computationally expensive to compute. Conversely, SUSAN is based on a morphological approach, which is computationally much more efficient than Harris.



(a) SUSANT corners *(b)* FAST corners

Figure 4.78 (a) SUSAN detector compares pixels within a circular region, while FAST (b) compares them only on a circle.

The working principle of SUSAN is very simple (see also figure 4.78a). For each pixel in the image, SUSAN considers a circular window of fixed radius centered on it. Then, all the pixels within this window are divided into two categories, depending on whether they have "similar" or "different" intensity values as the center pixel. Accordingly, on uniform intensity regions of the image, most pixels within the window will have a similar brightness as the center pixel. Near edges, the fraction of pixels with similar intensity will drop to 50%, while near corners it will decrease further to about 25%. Thus, SUSAN corners are identified as image locations where the number of pixels with similar brightness in a local neighborhood attains a local minimum and is below a specified threshold. As a final step, nonmaxima suppression (page 218) is used to identify local minima.

The SUSAN corners show a high repeatability, however they are heavily sensitive to noise. Indeed, many of the features are often located on edges than on real corners.

The FAST corner detector. The FAST (Features from Accelerated Segment Test) detector, was introduced by Rosten and Drummond [267, 268]. This detector builds upon the SUSAN detector. As we have seen, SUSAN computes the fraction of pixels within a circular window, which have similar intensity as the center pixel. Conversely, FAST compares pixels only on a circle of 16 pixels around the candidate corner (see figure 4.78b). This results in a very efficient detector that is up to thirty times faster than Harris: FAST takes only 1–2 milliseconds on a 2GHz Dual Core laptop and is currently the most computationally efficient feature detector available. However, like the SUSAN, it is not robust at high levels of noise.

4.5.4.4 Discussion about corner detectors

The Harris detector, with its scale and affine invariant extensions, is a convenient tool for extracting a large number of corners. Additionally, it has been identified as the most stable corner detector, as reported in several evaluations [28, 219, 285]. Alternatively, the

SUSAN or the FAST detectors can be used. They are much more efficient but also more sensitive to noise.

Shi-Tomasi, SUSAN, and FAST can also be made scale invariant like the Harris-Laplacian by analyzing the image at multiple scales, as seen in section 4.5.4.1. However, the scale estimation of corners is less accurate than blobs (e.g., SIFT, MSER, or SURF) due to the multiscale nature of corners: by definition, a corner is found at the intersection of edges, therefore its appearance changes very little at adjacent scales.

Finally, it is important to remind that the affine transformation model holds only for small viewpoint changes and in case of locally planar regions, that is, assuming the camera is relatively far from the object.

In the next section, we will describe the SIFT detector. Despite being a blob detector, the SIFT features incorporate all the properties of the scale-affine-invariant Harris but they are much more distinctive and robust to image noise, small illumination changes, and large changes of camera viewpoint.

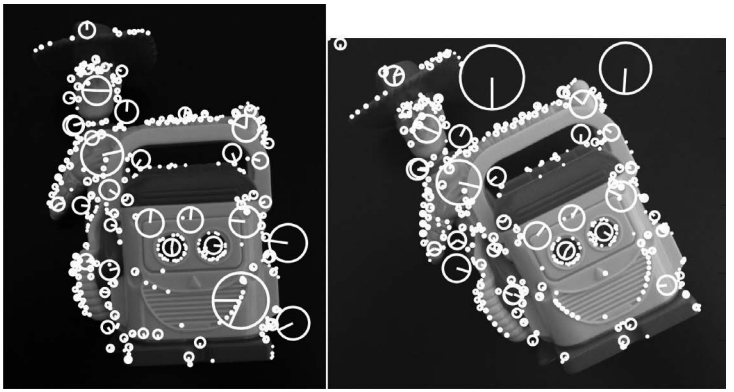
4.5.5 Blob detectors

A blob is an image pattern that differs from its immediate neighborhood in terms of intensity, color, and texture. It is not an edge, nor a corner. The location accuracy of a blob is typically smaller than that of a corner, but its scale and shape are better defined. To be clearer, a corner can be localized by a single point (e.g., the intersection of two edges), while a blob can only be localized by its boundary. On the other hand, a corner is less accurately localized over the scale because, as we pointed out before, a corner is found at the intersection of edges and therefore its appearance changes very little at adjacent scales. Conversely, a blob is more accurately localized over the scale because the boundary of a blob defines immediately its size and so its scale.

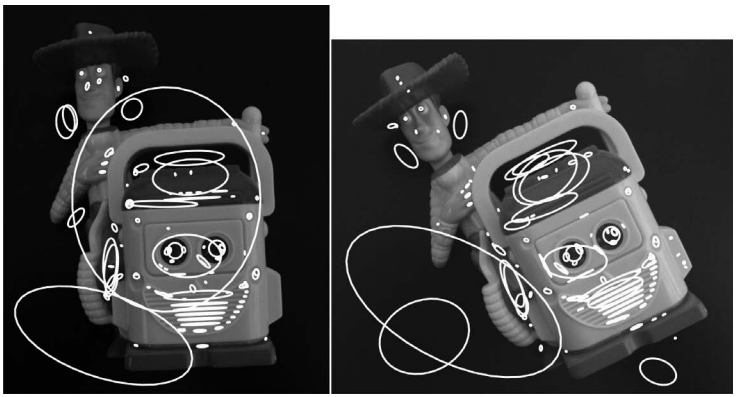
Using the new terminology, blob detectors can also be referred to as interest point operators, or alternatively interest region operators. Some examples of bloblike features are shown in figure 4.79. In this figure you can see two feature types that will be described in this section, namely SIFT and MSER. As observed, MSER privileges regions with uniform intensity, while SIFT does not.

4.5.5.1 SIFT features

SIFT stands for Scale Invariant Feature Transform and is a method to detect and match robust keypoints, which was invented in 1999 by Lowe [196, 197]. The uniqueness of SIFT is that these features are extremely distinctive and can be successfully matched between images with very different illumination, rotation, viewpoint, and scale changes. Its high repeatability and high matching rate in very challenging conditions have made SIFT the best feature detector so far. It has found many applications in object recognition, robotic



(a) SIFT features



(b) MSER features

Figure 4.79 Extraction of SIFT and MSER features from the same sample image used for the Harris detector in figure 4.68. Observe that both SIFT and MSER avoid corners. Furthemore, MSER privileges regions with uniform intensity. Both these feature detectors are robust to large changes of intensity, scale, and viewpoint.

mapping and navigation, image stitching (e.g. panoramas, mosaics), 3D modeling, gesture recognition, video tracking, and face recognition.

The main advantage of the SIFT features in comparison to all previously explained methods is that a "descriptor" is computed from the region around the interest point, which distinctively describes the information carried by the feature. As we will see, this descriptor is a vector that represents the local distribution of the image gradients around the interest point. As proven by its inventor, it is actually this descriptor that makes SIFT robust to rotation and small changes of illumination, scale, and viewpoint.

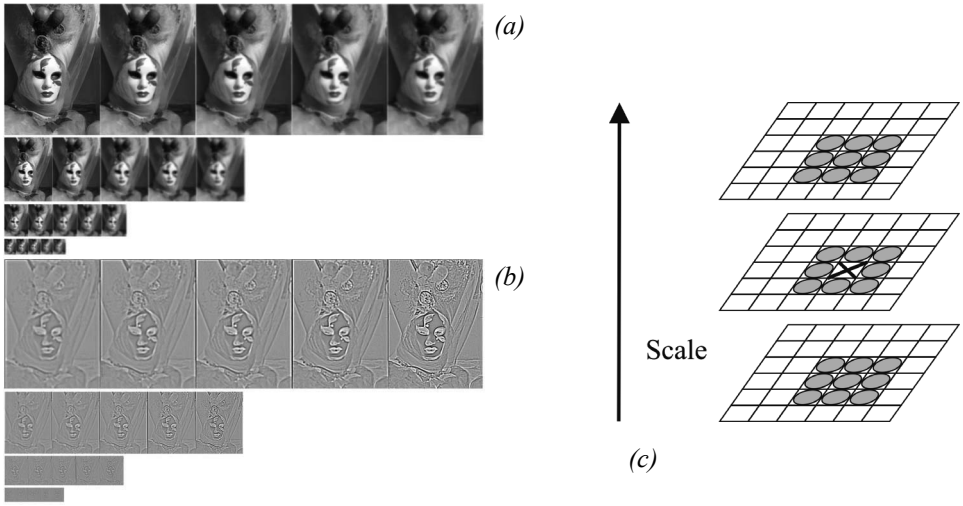


Figure 4.80 (a) Gaussian blurred images at different scales. (b) Difference of Gaussian images. (c) Keypoint selection as local maxima or minima of the DoG images across adjacent scales.

In the following, we will analyze the main steps of the SIFT algorithm, which are:

- Identification of keypoint location and scale
- Orientation assignment
- Generation of keypoint descriptor

Identification of keypoint location and scale. The first step toward the identification of SIFT keypoints is the generation of the so-called Difference of Gaussian (DoG) images. This is done by first blurring the original image with Gaussian filters at different scales (i.e., different sigma) and then by taking the difference of successive Gaussian-blurred images. This process is shown in figure 4.80a: the original image (top left) is blurred with four Gaussian filters with different sigma, and this is repeated after downsampling the image of a factor 2. Finally, DoG images are computed by simply taking the difference between successive blurred images 4.80b.

The second step is the selection of the keypoints. SIFT keypoints are identified as local maxima or minima of the DoG images across scales. In particular, each pixel in the DoG images is compared to its eight neighbors at the same scale, plus the nine neighbors at adjacent scales (figure 4.80c). If the pixel is a local maximum or minimum, it is selected as a candidate keypoint.

The third step consists in refining the location, in both space and scale, of the keypoints by interpolation of nearby data. Finally, keypoints with low contrast or along edges are removed because of their low distinctiveness and due to their instability to image noise.

Note that another way of generating DoG images consists in convolving the image with a DoG operator, which is nothing but the difference between to Gaussian filters (figure 4.75). As shown in figure 4.75, the DoG function is actually a very good approximation of the Laplacian of Gaussian (LoG). However, DoG images are more efficient to compute, and therefore they have been used in SIFT in lieu of LoG. At this point the attentive reader will recognize that the scale extrema selection of the SIFT is very similar to the scale extrema selection of the Harris-Laplacian. Indeed, the main difference with the Harris-Laplacian is the identification of the keypoint location. While in Harris the keypoint is identified in the image plane as local maximum of the cornerness function, in SIFT the keypoint is a local minimum or maximum of the DoG image in both position and scale. To recap, in SIFT the DoG operator is used to identify both position and scale of the keypoints.

Orientation assignment. This step consists in assigning each keypoint a specific orientation in order to make it invariant to image rotation.

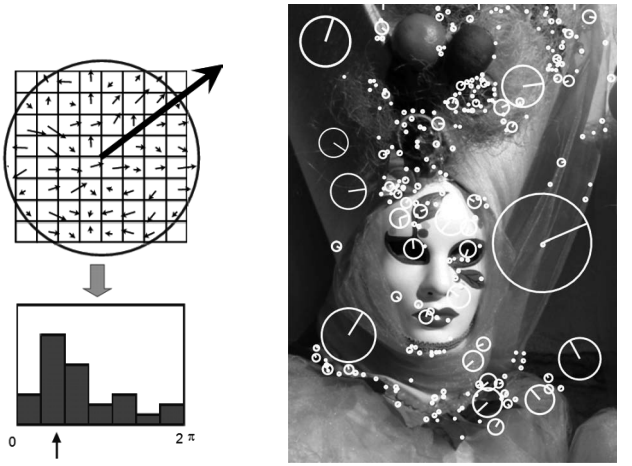
To determine the keypoint orientation, a gradient orientation histogram is computed in the neighborhood of the keypoint. In other words, for every pixel in the neighborhood of the keypoints, the intensity gradient (magnitude and orientation) is computed. Then a histogram of orientations is built such that the contribution of each pixel is weighted by the gradient magnitude.

Peaks in the histogram correspond to dominant orientations (figure 4.81a). Once the histogram is filled, the orientation corresponding to the highest peak is assigned to the keypoint. In the case of multiple peaks that are within 80% of the highest peak, an additional keypoint is created for each additional orientation, having the same location and scale as the original keypoint. All the properties of the keypoint will be measured relative to the keypoint orientation. This provides invariance to rotation.

Final keypoints with selected orientation and scale are shown in figure 4.81b.

Generation of keypoint descriptor. In the previous steps, we have described how to detect SIFT keypoints in both location and scale spaces and how to assign orientations to them. The last step of the SIFT algorithm is to compute descriptor vectors for these keypoints such that the descriptors are highly distinctive and partially invariant to illumination and viewpoint.

The descriptor is based on gradient orientation histograms. In order to achieve orientation invariance, the gradient orientations are rotated relative to the keypoint orientation. The neighboring region of the keypoint is then divided into 4×4 smaller regions, and a gradient histogram with eight orientation bins is computed within each of these regions.



(a) *(b)*

Figure 4.81 (a) Orientation assignment. (b) Some SIFT features with detected orientation and scale.

Finally, the descriptor is built by stacking all the orientation histogram entries. Therefore, the final length of the descriptor vector is $4 \times 4 \times 8 = 128$ elements. To achieve partial illumination invariance, the descriptor vector is finally normalized to have unit norm.

Observe, lower dimension descriptors could be built by using smaller region partitions or less histogram bins. However according to the literature, the 128 element vector is the one for which the best results in terms of robustness to image variations were reported.

In [197], it was also shown that feature matching accuracy is above 50% for viewpoint changes of up to 50 degrees (see figure 4.82). Therefore SIFT descriptors are invariant to minor viewpoint changes. To evaluate the distinctiveness of the SIFT descriptor, many tests were performed by counting the number of correct matches in a database with a varying number of keypoints. These tests revealed that matching accuracy of SIFT descriptor decreases only very little for very large database sizes. This implies that SIFT features are highly distinctive.

Because of its high repeatability and distinctiveness, SIFT has demonstrated in the last ten years to be the best feature detectors in a wide range of applications, although it is outperformed in efficiency by SURF (section 4.5.5.2). Excellent results have been achieved in robot navigation, 3D object recognition, place recognition, SLAM, panorama stitching, image retrieval, and many others.

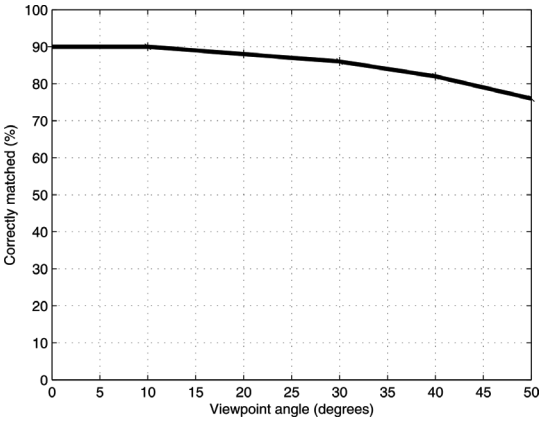


Figure 4.82 Fraction of SIFT keypoints correctly matched as a function of the viewpoint angle.

4.5.5.2 Other blob detectors

The MSER detector. Maximally Stable Extremal Regions (MSER) have been proposed by Matas et al. [210] for matching features that are robust under large viewpoint changes. A *maximally stable extremal region* is a connected component of pixels that have either higher or lower intensity than all the pixels on its outer boundary (figure 4.79b). These *extremal regions* are selected using an appropriate intensity thresholding and have a number of desirable properties. First, they are completely invariant to monotonic changes of intensity. Second, they are invariant to affine image transformations.

The SURF detector. SURF stands for Speeded Up Robust Features and have been proposed by Bay et al. [71]. This scale-invariant feature detector is strongly inspired by SIFT but is several times faster. Basically, it uses *Haar wavelets*[15] to approximate DoG filters and *integral images*[16] for convolution, which make the filtering process much more efficient at the expense of a minor robustness with respect to SIFT.

4.5.5.3 Summary on features detectors

Table 4.2 gives an overview of the most important properties for the feature detectors described in the previous sections. The highest repeatability and localization accuracy is

15. A Haar wavelet is a piecewise constant function.
16. Integral image is an algorithm for quickly and efficiently generating the sum of values in a rectangular subset of a grid.

Table 4.2
Comparison of feature detectors: properties and performance.

	Corner detector	Blob detector	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
Harris-Laplacian	x	x	x	x		+++	+++	++	+
Harris-Affine	x	x	x	x	x	+++	+++	++	++
SUSAN	x		x			++	++	++	+++
FAST	x		x			++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
MSER		x	x	x	x	+++	+	+++	+++
SURF		x	x	x	x	++	++	++	++

obtained by the Harris detector and its scale and affine invariant versions. The SUSAN and FAST detectors avoid computation of image derivatives and are therefore more efficient than Harris but the absence of smoothing makes them more sensitive to noise. The original Harris, Shi-Tomasi, SUSAN, and FAST are not scale-invariant, however some literature exists on how achieving scale invariance using the approach described in section 4.5.4.1.

In contrast to the original Harris, Harris-Laplace attains scale invariance; however, its scale estimation is less accurate than SIFT, MSER, or SURF due to the multiscale nature of corners. Finally, the SURF detector shows high repeatability, scale, and viewpoint invariance. However, it was devised for efficiency, and therefore it does not perform as well as SIFT.

GPU and FPGA implementations. Some of these feature detectors have been implemented to take advantage of the parallelism offered by modern Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGA). GPU implementations of SIFT are described in [150, 292]. An FPGA implementation of the Harris-Affine feature detector is discussed in [89] and of the SIFT detector in [286]. The availability of these algorithms for GPUs and FPGA make computer vision algorithm able to work at high frame rates.

Open source software: Web resources.

- Most of the feature detectors described in this section (Harris, MSER, FAST, SURF, and several others) are available as ready-to-use code in the Intel open-source computer vision library (OpenCV): http://opencv.willowgarage.com/wiki
- SUSAN, original source code: http://users.fmrib.ox.ac.uk/~steve/susan
- FAST, original source code: http://mi.eng.cam.ac.uk/~er258/work/fast.html
- SIFT, the original executable from David Lowe: http://people.cs.ubc.ca/~lowe/keypoints
- A reimplementation of SIFT, MSER and other featured detectors by Andrea Vedaldi. Source code in C and Matlab: http://www.vlfeat.org
- 3D object recognition toolkit, based on SIFT. Developed at the Autonomous Systems Lab at the ETH Zurich: http://robotics.ethz.ch/~ortk
- ERSP Vision Tool. An exiting demo for live object recognition by Evolution Robotics: http://www.evolution.com/product/oem/download/?ch=Vision (it becomes downloadable after user registration)
- SURF, precompiled software (GPU implementation also available): http://www.vision.ee.ethz.ch/~surf

4.6 Place Recognition

4.6.1 Introduction

Location recognition (or *place recognition*) describes the capability of naming discrete places in the world. A requirement is that it is possible to obtain a discrete partitioning of the environment into places and a representation of the place and that the places with the corresponding representations are stored in a database. The location recognition process then works by computing a representation from the current sensor measurements of the robot and searching the database for the most similar representation stored. The retrieved representation then tells us the location of the robot.

Location recognition is the natural form of robot localization in a topological environment map as described by many authors [108, 131, 138, 208, 214, 325]. Visual sensors (i.e., cameras) are perfectly suited to create a rich representation that is both descriptive and discriminative. Most visual representations proposed so far can be divided into *global* representations and *local* representations. Global representations use the whole camera image as a representation of the place, most in a domain-transformed way, for instance, as PCA transformed image [159], Fourier-transformed image [213], image histograms, image fingerprints [182], GIST descriptors [253, 254], and so on. Local representations instead iden-

tify salient regions of the image first and create the representation out of this only. This approach largely depends on the detection of salient regions using interest point or interest region detectors, which we have seen in section 4.5. With the development of many effective interest point detectors, local methods have proven to be practical and are nowadays applied in many systems. We will therefore present this method first as the preferred way to location recognition. However, in the last two sections, we will also review some of the earliest approaches to place recognition using image histograms and fingerprints. In fact, although largely outperformed by the local *visual-word*-based approaches, these methods are still used in some robot applications.

4.6.2 From bag of features to visual words

A representation of an image by a set of interest points only is usually called a bag of features. For each interest, point a descriptor is usually computed in a manner that is invariant to rotation, scale, intensity, and viewpoint change (section 4.5.4). A popular way is to use gradient histograms, e.g. SIFT (section 4.5.5.1) or SURF (section 4.5.5.2). This set of descriptors is the new representation of the image. It is called a *bag of features* because the original spatial relation between the interest points is removed and only the descriptors are remembered. The similarity between two sets of descriptors can be computed by counting the number of common feature descriptors. For this, a matching function needs to be defined, which allows us to determine whether two feature descriptors are the same. This matching function usually depends on the type of feature descriptor. But in general a feature descriptor is a high-dimensional vector, and matching features can be found by computing the distance using the L_2 norm. *Visual words* are a 1-dimensional representation of the high-dimensional feature descriptor. This means that the visual word for a 128-dimensional SIFT descriptor is just a single integer number. The conversion to visual words creates a bag of visual words instead of a bag of features. For this conversion, the high-dimensional descriptor space is divided into nonoverlapping cells. This division is computed by *k-means clustering* [18]. For the clustering, a large number of feature descriptors is necessary. The computed cluster borders form the cell divisions of the feature space. Each of the cells is now assigned a number that will be assigned to any feature descriptor within the cell. This number is referred to as visual word. Similar feature descriptors will be then sorted into the same cell and therefore get the same visual word assigned. This is illustrated in figure 4.83, which is a very efficient method of finding matching-feature descriptors. The visual words created by the partitioning is called *visual vocabulary*.

For *quantization*, a prototype vector for each cell is stored, which is the mean descriptor vector of all training descriptors from the cell. To assign a feature descriptor to its cell it needs to be compared to all prototype vectors. For a large number of cells this can be a very expensive operation. It can be sped up by creating a hierarchical splitting of the feature space called *vocabulary tree* [243].

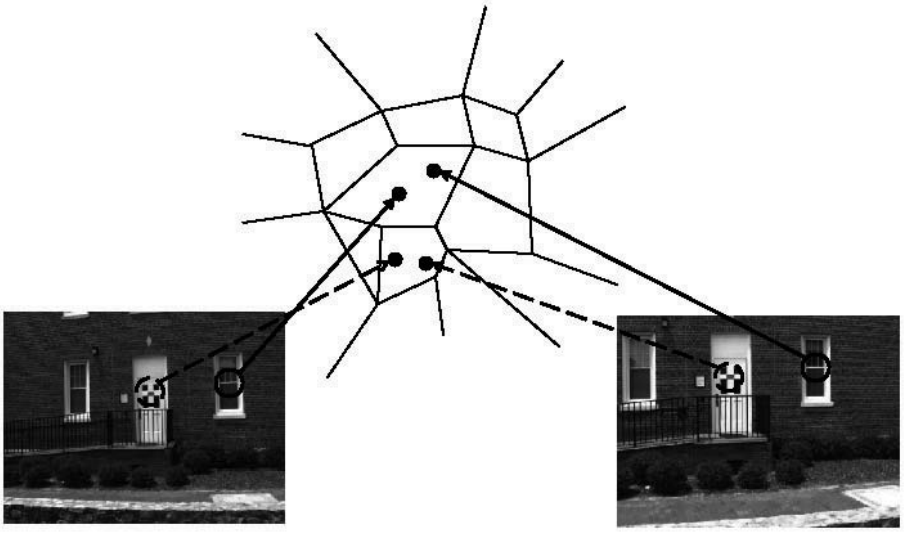


Figure 4.83 Partition of the descriptor feature space. Each cell stands for a visual word. Similar feature descriptors will be sorted into the same cell and therefore get the same visual word assigned.

4.6.3 Efficient location recognition by using an inverted file

Feature quantization into visual words is one key ingredient for efficient location recognition. Another one is the use of an *inverted file* for the database and a voting scheme for similarity computations. The database organized as an inverted file consists of a list of all possible visual words. Each element of this list points to another list that holds all the image identifiers in which this particular visual word appeared. This is illustrated in figure 4.84.

The voting scheme to find the most similar set of visual words in the database to a given query set works as follows. A voting array is initialized, which has as many cells as images in the database. A visual word from the query image is taken, and the list of image identifiers attached to this visual word is processed. For all the image identifiers in the list, a vote is cast by increasing the value at the corresponding position in the voting array. The most similar image to the query image is then the one with the highest vote. This voting scheme can exactly compute the L_2 norm if the descriptor vectors are correctly normalized [243].

This algorithm not only gives the most similar image in the database but also creates a ranking of all images in the database by similarity without any additional computational cost. This can be used to robustify place recognition.

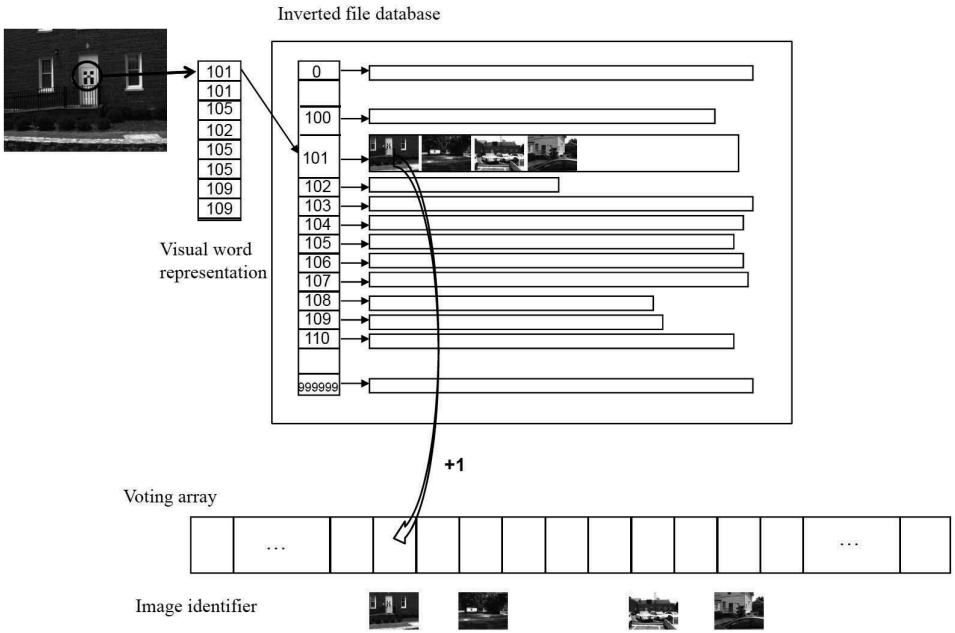


Figure 4.84 Visual word based location recognition using an inverted file system. An image in the database gets a vote if the same visual word is present in the query image.

4.6.4 Geometric verification for robust place recognition

The set of visual words does not contain the spatial relations anymore, thus an image that has the same visual words but in a different spatial arrangement would also have high similarity. The spatial relations however can be enforced again by a final geometric verification. For this, the k most similar images in a query are tested for geometric consistency. The geometric consistency test computes geometric transformations using the x and y image coordinates of matching visual words. Transformations used are affine transformations, homographies, or the essential matrix between images (section 4.2.6, page 184). The computation is performed in a robust way using RANSAC [128] (section 4.7.2.4), and the number of inliers to the transformation is counted. The image that achieves the largest number of inliers with the query image is then reported as the final match. This returns the desired location in place recognition.

4.6.5 Applications

This method for place recognition has already been used successfully in several applications. It was used for topological localization and mapping in [131]. It is also the core algo-

rithm of FABMAP [108], for which it was extended by a probabilistic formulation. Other methods that use this scheme are described in [56, 276].

Available source code on the Web:

- Vocabulary-tree-based image search by F. Fraundorfer et al. [131]. It is a framework for fast image retrieval and place recognition very popular in robotics (useful for loop detection in visual SLAM):
  http://www1.ethz.ch/cvg/people/postgraduates/fraundof/vocsearch
- FABMAB, by M. Cummins et al. [108], is another framework for fast image retrieval and place recognition also very popular in robotics: http://www.robots.ox.ac.uk/~mobile/wikisite/pmwiki/pmwiki.php?n=Software.FABMAP
- Bag of features: another powerful tool for image retrieval and visual recognition using vocabulary trees: http://www.vlfeat.org/~vedaldi/code/bag/bag.html

These algorithms are all very useful for loop detection in the problem of simultaneous localization and mapping, which we will see in section 5.8.

4.6.6 Other image representations for place recognition

In this section, we review two of the early and most successful approaches to place recognition before the advent of the visual words based methods described above. The first method uses image histograms, while the second one uses image fingerprints.

4.6.6.1 Image histograms

A single visual image provides so much information regarding a robot's immediate surroundings that an alternative to searching the image for spatially localized features is to make use of the information captured by the entire image (i.e., all the image pixels) to extract a *whole-image feature* or *global image feature*. Whole-image features are not designed to identify specific spatial structures such as obstacles or the position of specific landmarks. Rather, they serve as compact representations of the entire local region. From the perspective of robot localization, the goal is to extract one or more features from the image that are correlated well with the robot's position. In other words, small changes in robot position should cause only small changes to whole-image features, while large changes in robot position should cause correspondingly large changes to whole-image features.

A logical first step in designing a vision sensor for this purpose is to maximize the field of view of the camera. As the field of view increases, a small-scale structure in the robot's environment occupies a smaller proportion of the image, thereby mitigating the impact of individual scene objects on image characteristics. A catadioptric camera system, nowadays very popular in mobile robotics, offers an extremely wide field of view (section 4.2.4).

A catadioptric image is a 360-degree image warped onto a 2D image surface. Because of this, it offers another critical advantage in terms of sensitivity to small-scale robot motion. If the camera is mounted vertically on the robot so that the image represents the environment surrounding the robot (i.e., its horizon; figure 4.37a), then rotation of the camera and robot simply results in image rotation. In short, the catadioptric camera can be invariant to rotation of the field of view.

Of course, mobile robot rotation will still change the image; that is, pixel positions will change, although the new image will simply be a rotation of the original image. But we intend to extract image features via histogramming. Because histogramming is a function of the set of pixel values and not of the position of each pixel, the process is pixel position-invariant. When combined with the catadioptric camera's field of view invariance, we can create a system that is invariant to robot rotation and insensitive to small-scale robot translation.

A color camera's output image generally contains useful information along multiple *bands*: r, g, and b values as well as hue, saturation, and luminance values. The simplest histogram extraction strategy is to build separate 1D histograms characterizing each band. Given a color camera image, G, the first step is to create mappings from G to each of the n available bands. We use G_i to refer to an array storing the values in band i for all pixels in G. Each band-specific histogram H_i is calculated as before:

- As preprocessing, smooth G_i using a Gaussian smoothing operator.
- Initialize H_i with n levels: $H[j] = 0$ for $j = 1, \ldots, n$.
- For every pixel (x,y) in G_i, increment the histogram: $H_i[G_i[x, y]]+=1$.

Given the image shown in figure 4.37a, the image histogram technique extracts six histograms (for each of r, g, b, hue, saturation, and luminance) as shown in figure 4.85. In order to make use of such histograms as whole-image features, we need ways to compare to histograms to quantify the likelihood that the histograms map to nearby robot positions. The problem of defining useful histogram distance metrics is itself an important subfield within the image retrieval field. For an overview refer to [270]. One of the most successful distance metrics encountered in mobile robot localization is the *Jeffrey divergence*. Given two histograms H and K, with h_i and k_i denoting the histogram entries, the Jeffrey divergence $d(H, K)$ is defined as

$$d(H, K) = \sum_i \left(h_i \log \frac{2h_i}{h_i + k_i} + k_i \log \frac{2k_i}{h_i + k_i} \right). \tag{4.124}$$

Using measures such as the Jeffrey divergence, mobile robots have used whole-image histogram features to identify their position in real time against a database of previously

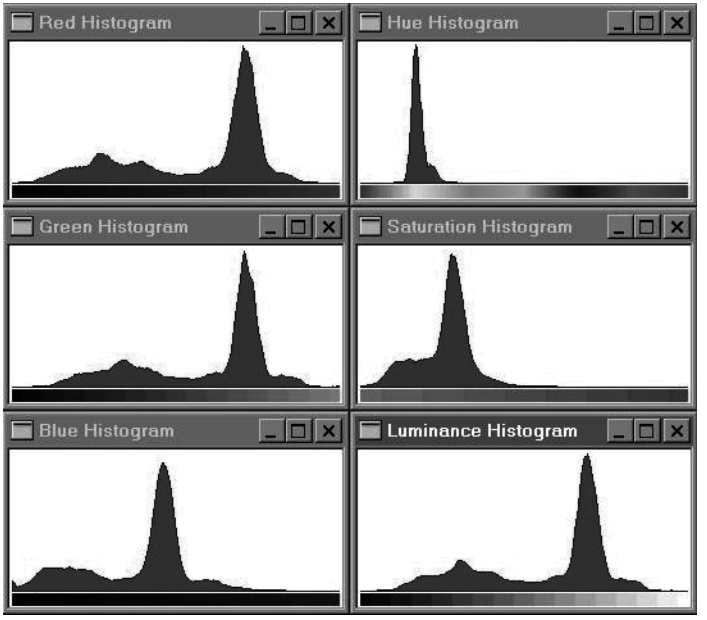


Figure 4.85
Six 1D histograms of the image above. A 5×5 smoothing filter was convolved with each band before histogramming.

recorded images of locations in their environment. Using this whole-image extraction approach, a robot can readily recover the particular hallway or particular room in which it is located [325].

Finally, note that in the last decade another global image descriptor—known as GIST—has been devised. The image is represented by a 320 dimensional vector per color band. The feature vector corresponds to the mean response to steerable filters at different scales and orientations computed over 4×4 sub-windows. Because a complete explanation of the GIST descriptor goes beyond the scope of this book, for an in-depth study we refer the reader to [253, 254] and [238].

4.6.6.2 Image fingerprints

This method is similar to the *visual word* approach with the difference that here the features are not interest points but rather morphological features, lines and colored blobs. Although outperformed by the new *visual word* based place recognition methods, this approach is still quite used in many applications of mobile robotics in both indoor and outdoor.

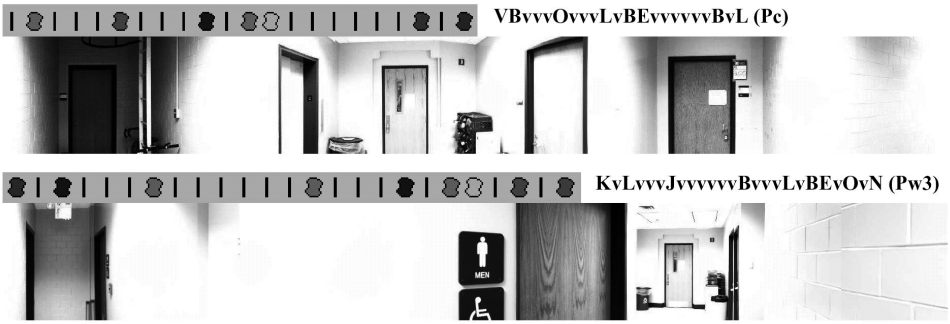


Figure 4.86
Two panoramic images and their associated fingerprint sequences [182].

We describe one particular implementation of this approach called the image *fingerprint*, which was developed first in [182]. Such as the previous method, the system makes use of a 360-degree panoramic image. The first extraction tier searches the panoramic image for spatially localized features: vertical edges and sixteen discrete hues of color. The vertical edge detector is a straightforward gradient approach implementing a horizontal difference operator. Vertical edges are "voted upon" by each edge pixel just as in a vertical edge Hough transform. An adaptive threshold is used to reduce the number of edges. Suppose the Hough table's tallies for each candidate vertical line have a mean μ and a standard deviation σ. The chosen threshold is simply $\mu + \sigma$.

Vertical color bands are identified in largely the same way, identifying statistics over the occurrence of each color, then filtering out all candidate color patches except those with tallies greater than $\mu + \sigma$. Figure 4.86 shows two sample panoramic images and their associated fingerprints. Note that each fingerprint is converted to an ASCII string representation.

Just as with histogram distance metrics in the case of image histogramming, we need a quantifiable measure of the distance between two fingerprint strings. String-matching algorithms are yet another large field of study, with particularly interesting applications today in the areas of genetics [55]. Note that we may have strings that differ not just in a single element value, but even in their overall length. For example, figure 4.87 depicts three actual sequences generated using the preceding algorithm. The top string should match *Place 1*, but note that there are deletions and insertions between the two strings.

The technique used in the fingerprinting approach for string differencing is known as a *minimum energy algorithm*. Taken from the stereo vision community, this optimization algorithm will find the minimum energy required to "transform" one sequence into another sequence. The result is a distance metric that is relatively insensitive to the addition or sub-

Figure 4.87
Three actual string sequences. The top two are strings extracted by the robot at the same position [182].

traction of individual local features while still able to identify robustly the correct matching string in a variety of circumstances.

It should be clear to the reader that the image histogram and image fingerprint place representations are straightforward to implement. For this reason these methods became very popular, although recently the *visual-word*-based approaches for a greater variety of applications have outperformed them.

4.7 Feature Extraction Based on Range Data (Laser, Ultrasonic)

Most of today's features extracted from ranging sensors are geometric primitives such as line segments or circles. The main reason for this is that for most other geometric primitives the parametric description of the features becomes too complex and no closed-form solution exists. In this section, we will focus on line extraction, since line segments are the simplest features to extract. As we will see in chapter 5, lines are used to match laser scans for performing tasks like robot localization or automatic map building.

There are three main problems in line extraction in unknown environments:

- How many lines are there?
- Which points belong to which line?
- Given the points that belong to a line, how to estimate the line model parameters?

For answering these questions, we will present the description of the six most popular line extraction algorithms for 2D range scans. Our selection is based on their performance and popularity in both mobile robotics, especially feature extraction, and computer vision. Only basic versions of the algorithms are given, even though their details may vary in different applications and implementations. The interested reader should refer to the indicated

references for more details. Our implementation follows closely the pseudocode described below in most cases, otherwise it will be stated.

Before describing the six algorithms, we will first explain the line fitting problem, which answers the third question: "Given the points that belong to a line, how to estimate the line model parameters?" In describing line fitting, we will demonstrate how the uncertainty models presented in section 4.1.3 can be applied to the problem of combining multiple sensor measurements. Then, we will answer the first two questions by describing six line-extraction algorithms from noisy range measurements. Finally, we will briefly present other very successful features of indoor mobile robots using range data, the corner and the plane features, and demonstrate how these features can be combined into a single representation.

4.7.1 Line fitting

Geometric feature fitting is usually the process of comparing and matching measured sensor data against a predefined description, or template, of the expected feature. Usually, the system is overdetermined in that the number of sensor measurements exceeds the number of feature parameters to be estimated. Since the sensor measurements all have some error, there is no perfectly consistent solution and, instead, the problem is one of optimization. One can, for example, fit the feature that minimizes the discrepancy with all sensor measurements used (e.g,. *least-squares estimation*).

In this section we present an optimization solution to the problem of extracting a line feature from a set of uncertain sensor measurements. For greater detail than what is presented below, refer to [17, pages 15 and 221].

4.7.1.1 Probabilistic line fitting from uncertain range sensor data

Our goal is to fit a line to a set of sensor measurements as shown in figure 4.88. There is uncertainty associated with each of the noisy range sensor measurements, and so there is no single line that passes through the set. Instead, we wish to select the best possible match, given some optimization criterion.

More formally, suppose n ranging measurement points in polar coordinates $x_i = (\rho_i, \theta_i)$ are produced by the robot's sensors. We know that there is uncertainty associated with each measurement, and so we can model each measurement using two random variables $X_i = (P_i, Q_i)$. In this analysis we assume that the uncertainty with respect to the actual value of P and Q is independent. Based on equation (4.11), we can state this formally:

$$E[P_i \cdot P_j] = E[P_i]E[P_j] \qquad \forall\, i,j = 1, \ldots, n \mid i \neq j . \tag{4.125}$$

$$E[Q_i \cdot Q_j] = E[Q_i]E[Q_j] \qquad \forall\, i,j = 1, \ldots, n \mid i \neq j . \tag{4.126}$$

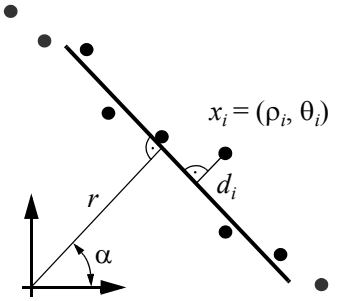


Figure 4.88
Estimating a line in the least-squares sense. The model parameters r (length of the perpendicular) and α (its angle to the abscissa) uniquely describe a line.

$$E[P_i \cdot Q_j] = E[P_i]E[Q_j] \qquad \forall\ i,j = 1, \ldots, n\,. \tag{4.127}$$

Furthermore, we assume that each random variable is subject to a Gaussian probability density curve, with a mean at the true value and with some specified variance:

$$P_i \sim N(\rho_i, \sigma^2_{\rho_i})\,. \tag{4.128}$$

$$Q_i \sim N(\theta_i, \sigma^2_{\theta_i})\,. \tag{4.129}$$

Given some measurement point (ρ, θ), we can calculate the corresponding Euclidean coordinates as $x = \rho\cos\theta$ and $y = \rho\sin\theta$. If there were no error, we would want to find a line for which all measurements lie on that line:

$$\rho\cos\theta\cos\alpha + \rho\sin\theta\sin\alpha - r = \rho\cos(\theta - \alpha) - r = 0\,. \tag{4.130}$$

Of course, there is measurement error, and so this quantity will not be zero. When it is nonzero, this is a measure of the error between the measurement point (ρ, θ) and the line, specifically in terms of the minimum orthogonal distance between the point and the line. It is always important to understand how the error that shall be minimized is being measured. For example, a number of line-extraction techniques do not minimize this orthogonal point-line distance, but instead the distance parallel to the y-axis between the point and the line. A good illustration of the variety of optimization criteria is available in [25], where several

algorithms for fitting circles and ellipses are presented that minimize algebraic and geometric distances.

For each specific (ρ_i, θ_i), we can write the orthogonal distance d_i between (ρ_i, θ_i) and the line as

$$\rho_i \cos(\theta_i - \alpha) - r = d_i . \tag{4.131}$$

If we consider each measurement to be equally uncertain, we can sum the square of all errors together, for all measurement points, to quantify an overall fit between the line and all of the measurements:

$$S = \sum_i d_i^2 = \sum_i (\rho_i \cos(\theta_i - \alpha) - r)^2 . \tag{4.132}$$

Our goal is to minimize S when selecting the line parameters (α, r). We can do so by solving the nonlinear equation system

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0 . \tag{4.133}$$

This formalism is considered an *unweighted least-squares* solution because no distinction is made from among the measurements. In reality, each sensor measurement may have its own, unique uncertainty based on the geometry of the robot and environment when the measurement was recorded. For example, we know with regard to vision stereo ranging that uncertainty and, therefore, variance increase as a square of the distance between the robot and the object. To make use of the variance σ_i^2 that models the uncertainty regarding distance ρ_i of a particular sensor measurement, we compute an individual weight w_i for each measurement using the formula[17]

$$w_i = 1 / \sigma_i^2 . \tag{4.134}$$

Then, equation (4.132) becomes

$$S = \sum w_i d_i^2 = \sum w_i (\rho_i \cos(\theta_i - \alpha) - r)^2 . \tag{4.135}$$

17. The issue of determining an adequate weight when σ_i is given (and perhaps some additional information) is complex in general and beyond the scope of this text. See [11] for a careful treatment.

It can be shown that the solution to equation (4.133) in the *weighted* least-squares sense[18] is

$$\alpha = \frac{1}{2}\text{atan}\left(\frac{\sum w_i\rho_i^2\sin 2\theta_i - \frac{2}{\sum w_i}\sum\sum w_i w_j \rho_i\rho_j\cos\theta_i\sin\theta_j}{\sum w_i\rho_i^2\cos 2\theta_i - \frac{1}{\sum w_i}\sum\sum w_i w_j \rho_i\rho_j\cos(\theta_i+\theta_j)}\right). \tag{4.136}$$

$$r = \frac{\sum w_i\rho_i\cos(\theta_i-\alpha)}{\sum w_i}. \tag{4.137}$$

In practice, equation (4.136) uses the four-quadrant arc tangent (atan2).[19]

Let us demonstrate equations (4.136) and (4.137) with a concrete example. The seventeen measurements (ρ_i, θ_i) in table 4.3 have been taken with a laser range sensor installed on a mobile robot. The measurements are shown in figure 4.89. The measurement uncertainty is usually considered proportional to the measured distance, but, to simplify the calculation, in this case we assume that the uncertainties of all measurements are equal. We also assume that the measurements are uncorrelated, and that the robot was static during the measurement process. Direct application of this solution equations yields the line defined by $\alpha = 37.36$ and $r = 0.4$. This line represents the best fit in a least-squares sense and is shown visually in figure 4.89.

4.7.1.2 Propagation of uncertainty during line fitting

Returning to the subject of section 4.1.3, we would like to understand how the uncertainties of specific range sensor measurements propagate to govern the uncertainty of the extracted line. In other words, how does uncertainty in ρ_i and θ_i propagate in equations (4.136) and (4.137) to affect the uncertainty of α and r?

This requires direct application of equation (4.15) with A and R representing the random output variables of α and r respectively. The goal is to derive the 2×2 output covariance matrix

18. We follow here the notation of [17] and distinguish a weighted least-squares problem if C_X is diagonal (input errors are mutually independent) and a generalized least-squares problem if C_X is non-diagonal.

19. Atan2 computes $\tan(x/y)^{-1}$ but uses the signs of both x and y to determine the quadrant in which the resulting angles lies. For example $\text{atan}2(-2,-2) = -135°$, whereas $\text{atan}2(2, 2) = 45°$, a distinction which would be lost with a single-argument arc tangent function.

Table 4.3 Measured values

pointing angle of sensor θ_i [deg]	range ρ_i [m]
0	0.5197
5	0.4404
10	0.4850
15	0.4222
20	0.4132
25	0.4371
30	0.3912
35	0.3949
40	0.3919
45	0.4276
50	0.4075
55	0.3956
60	0.4053
65	0.4752
70	0.5032
75	0.5273
80	0.4879

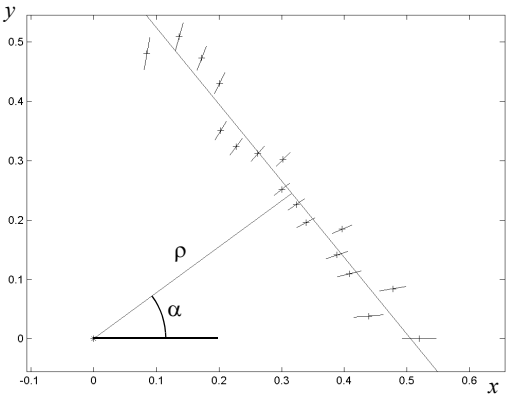


Figure 4.89
Extracted line from laser range measurements (+). The small lines at each measurement point represent the measurement uncertainty σ that is proportional to the measured distance.

$$C_{AR} = \begin{bmatrix} \sigma_A^2 & \sigma_{AR} \\ \sigma_{AR} & \sigma_R^2 \end{bmatrix}, \tag{4.138}$$

given the $2n \times 2n$ input covariance matrix

$$C_X = \begin{bmatrix} C_P & \mathbf{0} \\ \mathbf{0} & C_Q \end{bmatrix} = \begin{bmatrix} diag(\sigma_{\rho_i}^2) & \mathbf{0} \\ \mathbf{0} & diag(\sigma_{\theta_i}^2) \end{bmatrix} \tag{4.139}$$

and the system relationships [equations (4.136) and (4.137)]. Then by calculating the Jacobian,

$$F_{PQ} = \begin{bmatrix} \frac{\partial \alpha}{\partial P_1} & \frac{\partial \alpha}{\partial P_2} & \cdots & \frac{\partial \alpha}{\partial P_n} & \frac{\partial \alpha}{\partial Q_1} & \frac{\partial \alpha}{\partial Q_2} & \cdots & \frac{\partial \alpha}{\partial Q_n} \\ \frac{\partial r}{\partial P_1} & \frac{\partial r}{\partial P_2} & \cdots & \frac{\partial r}{\partial P_n} & \frac{\partial r}{\partial Q_1} & \frac{\partial r}{\partial Q_2} & \cdots & \frac{\partial r}{\partial Q_n} \end{bmatrix}, \tag{4.140}$$

we can instantiate the uncertainty propagation equation (4.15) to yield C_{AR}:

$$C_{AR} = F_{PQ} C_X F_{PQ}^T \tag{4.141}$$

Thus we have calculated the probability C_{AR} of the extracted line (α, r) based on the probabilities of the measurement points. For more details about this method, refer to [8, 59].

4.7.2 Six line-extraction algorithms

The previous section described how to fit a line feature given a set of range measurements. Unfortunately, the feature extraction process is significantly more complex than that. A mobile robot does indeed acquire a set of range measurements, but in general the range measurements are not all part of one line. Rather, only some of the range measurements should play a role in line extraction and, further, there may be more than one line feature represented in the measurement set. This more realistic scenario is shown in figure 4.90.

The process of dividing up a set of measurements into subsets that can be interpreted one by one is termed *segmentation* and is the most important step of line extraction. In the fol-

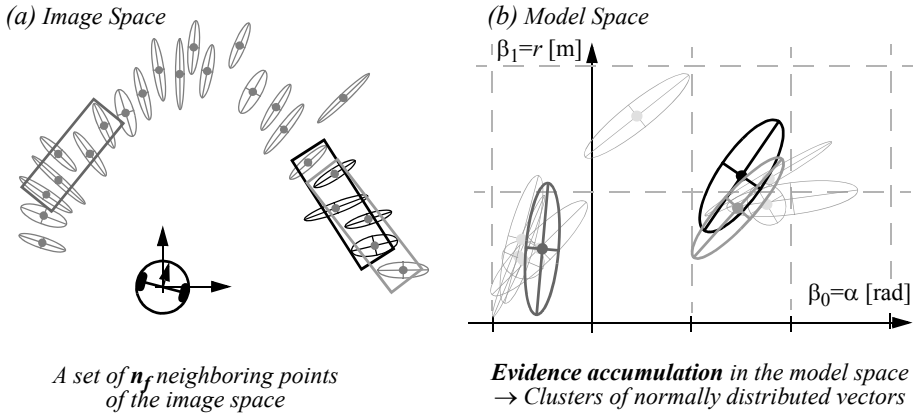


Figure 4.90
Clustering: finding neighboring segments of a common line [59].

Algorithm 1: *Split-and-Merge*

1. Initial: set s_1 consists of N points. Put s_1 in a list L
2. Fit a line to the next set s_i in L
3. Detect point P with maximum distance d_P to the line
4. If d_P is less than a threshold, continue (go to step 2)
5. Otherwise, split s_i at P into s_{i1} and s_{i2}, replace s_i in L by s_{i1} and s_{i2}, continue (go to 2)
6. When all sets (segments) in L have been checked, merge collinear segments.

lowing, we describe six popular line-extraction (segmentation) algorithms. For both an overview and a comparison among these algorithms, we refer the reader to [247].

4.7.2.1 Algorithm 1: Split-and-merge

Split-and-Merge is the most popular line extraction algorithm. This algorithm has originated from computer vision [257] and has been studied and used in many works [96, 121, 287, 78, 336]. The algorithm is outlined in algorithm 1.

Notice that this algorithm can be slightly modified on line 3 to make it more robust to noise. Indeed, sometimes the splitting position can be the result of a point which still belongs to the same line but which, because of noise, appears too far away from this line.

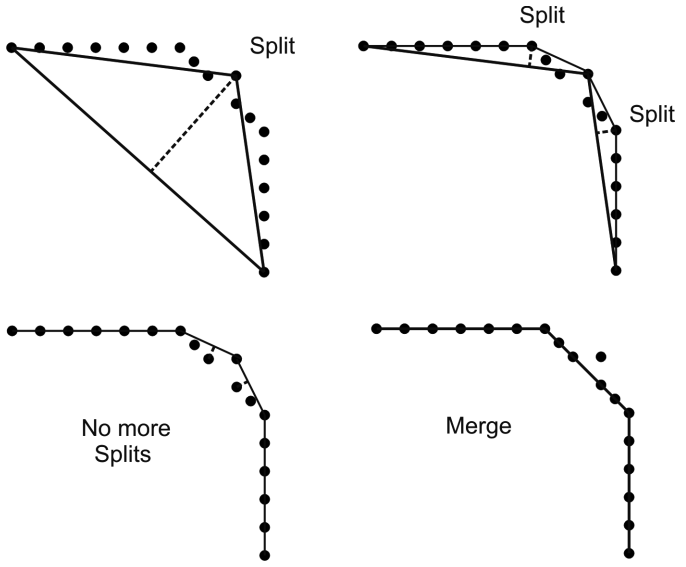


Figure 4.91 Split-and-merge implemented in the Iterative-End-Point-Fit fashion. In this case, the line is not fitted to the points but is constructed by connecting the first and last points.

In this case, we scan for a splitting position where two adjacent points P_1 and P_2 are on the same side of the line and both have distances to the line greater than the threshold. If we find only one such point, then we automatically discard it as a noisy point.

Observe that in line 2 one can use for line fitting the least-squares method described in section 4.7.1. Alternatively, one can construct the line by simply connecting the first and the last points. In this case, the algorithm is named *Iterative-End-Point-Fit* [19, 287, 78, 336] and is a well consolidated approach to implement split-and-merge. This procedure is illustrated in figure 4.91.

Finally, an application of split-and-merge to a 2D laser scan is shown in figure 4.92.

4.7.2.2 Algorithm 2: Line regression

This algorithm was proposed in [59]. It uses a sliding window of size N_f. At every step, a line is fitted to the N_f points within the window. The window is then shifted one point forward (this is why it is called sliding window), and the line-fitting operation is repeated again. The goal is to find adjacent line segments and merge them together. To do this, at every step the *Mahalanobis*[20] distance between the last two windows is computed and is stored in a *fidelity* array. When all the points have been analyzed, the fidelity array is scanned for consecutive similar elements. This is done by using an appropriate clustering

20. The Mahalanobis distance is defined in section "Matching" on page 334.

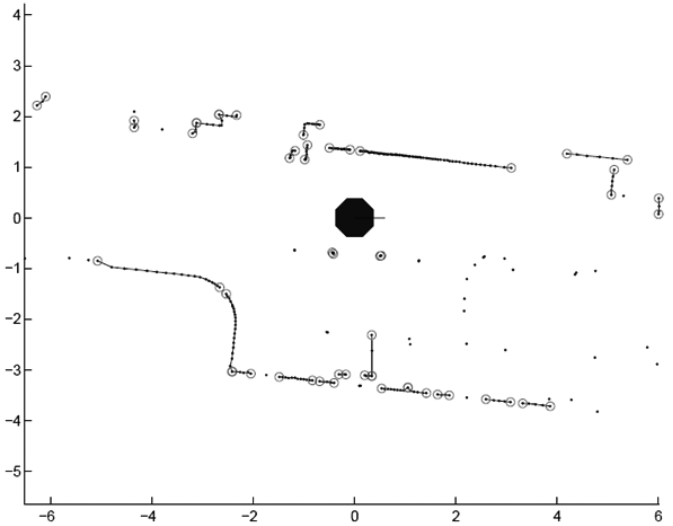


Figure 4.92 Split-and-merge applied to a 2D laser scan (courtesy of B. Jensen).

Algorithm 2: *Line-Regression*

1. Initialize sliding window size N_f
2. Fit a line to every N_f consecutive points
3. Compute a line fidelity array. Each element of the array contains the sum of Mahalanobis distances between every three adjacent windows
4. Construct line segments by scanning the fidelity array for consecutive elements having values less than a threshold
5. Merge overlapped line segments and recompute line parameters for each segment

algorithm. At the end, the clustered consecutive line segments are merged together using again line regression. This algorithm is outlined in algorithm 2, while the main steps are depicted in figure 4.93.

Notice that the sliding window size N_f is very dependent on the environment and has a strong influence on the algorithm performance. In typical applications, $N_f = 7$ is used.

4.7.2.3 Algorithm 3: Incremental

This algorithm is straightforward to implement and has been used in many applications [24, 328, 308]. The algorithm is outlined in algorithm 3. At the beginning, the set consists of

Algorithm 3: *Incremental*

1. Start by the first 2 points, construct a line
2. Add the next point to the current line model
3. Recompute the line parameters by line fitting
4. If it satisfies the line condition, continue (go to step 2)
5. Otherwise, put back the last point, recompute the line parameters, return the line
6. Continue with the next two points, go to step 2

ments may contain distinct lines from the surrounding walls but also points from other static and dynamic objects (like chairs or humans). In this case, an outlier is any entity which does not belong to a line (i.e., chair, human, and so on).

RANSAC is an iterative method and is nondeterministic in that the probability to find a line free of outliers increases as more iterations are used. RANSAC is not restricted to line extraction from laser data but it can be generally applied to any problem where the goal is to identify the inliers which satisfy a predefined mathematical model. Typical applications in robotics are: line extraction from 2D range data (sonar or laser); plane extraction from 3D laser point clouds; and structure-from-motion (section 4.2.6), where the goal is to identify the image correspondences which satisfy a rigid body transformation.

Let us see how RANSAC works for the simple case of line extraction from 2D laser scan points. The algorithm starts by randomly selecting a sample of two points from the dataset. Then a line is constructed from these two points and the distance of all other points to this line is computed. The inliers set comprises all the points whose distance to the line is within a predefined threshold d. The algorithm then stores the inliers set and starts again by selecting another minimal set of two points at random. The procedure is iterated until a set with a maximum number of inliers is found, which is chosen as a solution to the problem. The algorithm is outlined in algorithm 4, while figure 4.94 illustrates its working principle.

Because we cannot know in advance if the observed set contains the maximum number of inliers, the ideal would be to check all possible combinations of 2 points in a dataset of N points. The number of combinations is given by $N \cdot (N-1)/2$, which makes it computationally unfeasible if N is too large. For example, in a laser scan of 360 points we would need to check all $360 \cdot 359/2 = 64{,}620$ possibilities!

At this point, a question arises: Do we really need to check all possibilities, or can we stop RANSAC after k iterations? The answer is that indeed we do not need to check all combinations but just a subset of them if we have a rough estimate of the percentage of inliers in our dataset. This can be done by thinking in a probabilistic way.

Algorithm 4: *RANSAC*

1. Initial: let A be a set of N points
2. **repeat**
3. Randomly select a sample of 2 points from A
4. Fit a line through the 2 points
5. Compute the distances of all other points to this line
6. Construct the inlier set (i.e. count the number of points with distance to the line < d)
7. Store these inliers
8. **until** Maximum number of iterations k reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

Let p be the probability of finding a set of points free of outliers. Let w be the probability of selecting an inlier from our dataset of N points. Hence, w expresses the fraction of inliers in the data, that is, w = number of inliers/N. If we assume that the two points needed for estimating a line are selected independently, w^2 is the probability that both points are inliers and $1-w^2$ is the probability that at least one of these two points is an outlier. Now, let k be the number of RANSAC iterations executed so far, then $(1-w^2)^k$ will be the probability that RANSAC never selects two points that are both inliers. This probability must be equal to $1-p$. Accordingly,

$$1-p = (1-w^2)^k, \tag{4.142}$$

and therefore

$$k = \frac{\log(1-p)}{\log(1-w^2)}. \tag{4.143}$$

This expression tells us that knowing the fraction w of inliers, after k RANSAC iterations we will have a probability p of finding a set of points free of outliers. For example, if we want a probability of success equal to 99% and we know that the percentage of inliers in the dataset is 50%, then according to (4.143) we could stop RANSAC after 16 iterations, which is much less than the number of all possible combinations that we had to check in the previous example! Also observe that in practice we do not need a precise knowledge of

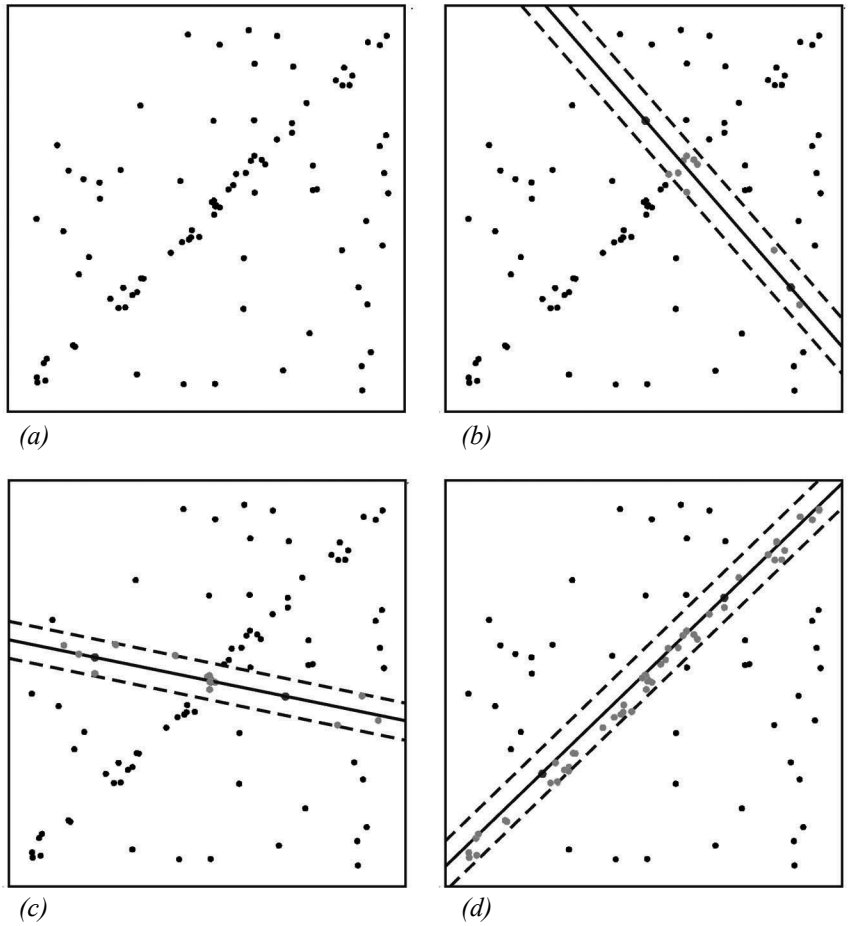


Figure 4.94 Working principle of RANSAC. (a) Dataset of N points. (b) Two points are randomly selected, a line is fitted through them, and the points within a predefined distance to it are identified. (c) The procedure is repeated (iterated) several times. (d) The set with the maximum number of inliers is chosen as a solution to the problem.

the fraction of inliers but just a rough estimate. More advanced implementations of RANSAC estimate the fraction of inliers by changing it adaptively iteration after iteration.

The main advantage of RANSAC is that it is a generic extraction method and can be used with many types of features once we have the feature model. Because of this, it is very popular in computer vision [29]. It is also simple to implement. Another advantage is its ability to cope with large amount of outliers, even more than 50%. Clearly, if we want to extract multiple lines, we need to run RANSAC several time and remove sequentially all

Algorithm 5: *Hough Transform*

1. Initial: let A be a set of N points
2. Initialize the accumulator array by setting all elements to 0
3. Construct values for the array
4. Choose the element with max. votes V_{max}
5. If V_{max} is less than a threshold, terminate
6. Otherwise, determine the inliers
7. Fit a line through the inliers and store the line
8. Remove the inliers from the set, go to step 2

the lines extracted so far. A disadvantage of RANSAC is that when the maximum number of iterations k is reached, the solution obtained may not be the optimal one (i.e., the one with the maximum number of inliers). Furthermore, this solution may not even be the one that fits the data in the best way.

4.7.2.5 Algorithm 5: Hough Transform (HT)

This algorithm was already described for straight edge detection in intensity images (page 205) but it can be applied without any modification to 2D range images. The algorithm is outlined in algorithm 5. Although it has been developed within the computer vision community, it has been brought into robotics for extracting lines from scan images [158] and [261]. In fact, 2D scan images are nothing but binary images.

Typical drawbacks with the Hough transform are that it is usually difficult to choose an appropriate grid size and the fact that this transform does not take noise and uncertainty into account when estimating the line parameters. To overcome the second problem, in line 7 one can use the line fitting method described in section 4.7.1, which takes into account feature uncertainty.

4.7.2.6 Algorithm 6: Expectation maximization (EM)

Expectation Maximization (EM), is a probabilistic method commonly used in missing variable problems. EM has been used as a line extraction tool in computer vision [24] and robotics [261]. There are some drawbacks with the EM algorithm. First, it can fall into local minima. Second, it is difficult to choose a good initial value. The algorithm is outlined in algorithm 6. For a detailed implementation of this algorithm for extracting lines, we refer the reader to [24].

Algorithm 6: *Expectation Maximization*

1. Initial: let A be a set of N points
2. **repeat**
3. Randomly generate parameters for a line
4. Initialize weights for remaining points
5. **repeat**
6. *E-Step*: Compute the weights of the points from the line model
7. *M-Step*: Recompute the line model parameters
8. **until** Maximum number of steps reached or convergence
9. **until** Maximum number of trials reached or found a line
10. If found, store the line, remove the inliers, go to step 2
11 Otherwise, terminate

4.7.2.7 Implementation details

Clustering. In most cases, 2D laser scans present some agglomerations of a few sparse points (figure 4.92). These points can be caused for instance by small objects or moving people. In this case, a simple clustering algorithm is usually used for preprocessing: it divides the raw points into groups of close points and discards groups consisting of too few points. Basically, this algorithm scans for big jumps in radial differences of consecutive points and puts breakpoints in those positions. As a result, the scan is segmented into contiguous clusters of points. Clusters having too few number of points are removed.

Merging. Due to occlusions, a line may be observed and extracted as several segments. When this happens, it is likely good to merge collinear line segments into a single line segment. This merging routine should be applied at the output end of each previously seen algorithm, after segments have been extracted. To decide if two consecutive line segments have to be merged, the Mahalanobis distance[21] between each pair of line segments is typically used. If the two line segments have Mahalanobis distance less than a predefined

21. The Mahalanobis distance depends on the covariance matrix of the parameters of each line segment as explained on page 334.

threshold, then they are merged. Using line fitting, the new line parameters are finally recomputed from the raw scan points that constitute the two segments.

4.7.2.8 A comparison of line extraction algorithms

These six algorithms can be divided into two categories: deterministic and nondeterministic methods:

1. Deterministic: Split-and-Merge, Incremental, Regression, Hough transform.
2. Nondeterministic: RANSAC, EM.

RANSAC and EM are nondeterministic because their results can be different at every run. This is because these two algorithms generate random hypotheses.

A comparison between all six algorithms has been done by Nguyen et al. [247]. They evaluated four quality measures: complexity, speed, correctness (false positives), and precision. The results of that study are shown in table 4.4. The terminology used is explained as follows:

- N: Number of points in the input scan (e.g., 722)
- S: Number of line segments extracted (e.g., 7 in average depending on the algorithm)
- N_f: Sliding window size for Line-Regression (e.g., 9)
- N_{Trials}: Number of trials for RANSAC (e.g., 1000)
- N_C, N_R: Number of columns, rows respectively for the Hough accumulator array (N_C = 401, N_R = 671 for resolutions of 1 cm and 0.9 degrees)
- N_1, N_2: Number of trials and convergence iterations, respectively, for EM (e.g. N_1 = 50, N_2 = 200).

Observe that the values shown in parentheses are typical numbers used in practical implementations.

As shown in the third column (Speed) of table 4.4, Split-and-Merge, Incremental, and Line-Regression perform much faster than the others. The Split-and-Merge algorithm takes the lead. The reason why these three algorithms are much faster is mainly because they are deterministic and, especially, because they take advantage of the sequential ordering of the raw scan points (the points are not captured randomly but according to the rotation direction of the laser beam). If these three algorithms were applied on randomly distributed points (e.g., general binary images), they would not be able to segment all lines, while RANSAC, EM, and Hough would. Indeed, these last three algorithms are popular for their ability to extract lines in binary images which obviously present a large number of outliers.

The Incremental algorithm seems to perform the best in terms of correctness. In fact, it has a very low number of false positives, which is very important for localization, mapping,

Table 4.4 Comparison of algorithms for line extraction from 2D laser data.

	Complexity	Speed [Hz]	False positives	Precision
Split-and-Merge	$N \cdot \log N$	1500	10%	+++
Incremental	$S \cdot N^2$	600	6%	+++
Line-Regression	$N \cdot N_f$	400	10%	+++
RANSAC	$S \cdot N \cdot N_{Trials}$	30	30%	++++
Hough-Transform	$S \cdot N \cdot N_C + S \cdot N_R \cdot N_C$	10	30%	++++
Expectation Maximization	$S \cdot N_1 \cdot N_2 \cdot N$	1	50%	++++

and SLAM (section 5.8). Conversely, RANSAC, HT, and EM seem to produce many more false positives. This is due to the fact that they do not take advantage of the sequential ordering of the scan points and therefore they often try to fit lines falsely across the scan map. Their behavior could be improved by increasing the minimum number of points per line segment, but the drawback of this would then be that short segments might be left out.

Despite their bad correctness, as observed in the fourth column of table 4.4, RANSAC, HT, and EM produce more precise lines than the other algorithms. This is due to their ability to get rid of outliers or largely noisy inliers. For instance, with RANSAC the probability of extracting a stable line increases with the number of iterations, while with HT the outlier (or a largely noise inlier) would vote another grid cell than that representing the line.

In conclusion, Split-and-Merge and Incremental are the best choice in terms of correctness and efficiency and are therefore the best candidates for 2D laser-based robot localization and mapping. However, the right choice depends highly on the type of application and the desired precision.

4.7.3 Range histogram features

A histogram is a simple way to combine characteristic elements of an image. An angle histogram, as presented in figure 4.95, plots the statistics of lines extracted by two adjacent range measurements. First, a 360-degree scan of the room is taken with the range scanner, and the resulting "hits" are recorded in a map. Then the algorithm measures the relative angle between any two adjacent hits (see figure 4.95b). After compensating for noise in the readings (caused by the inaccuracies in position between adjacent hits), the angle histogram

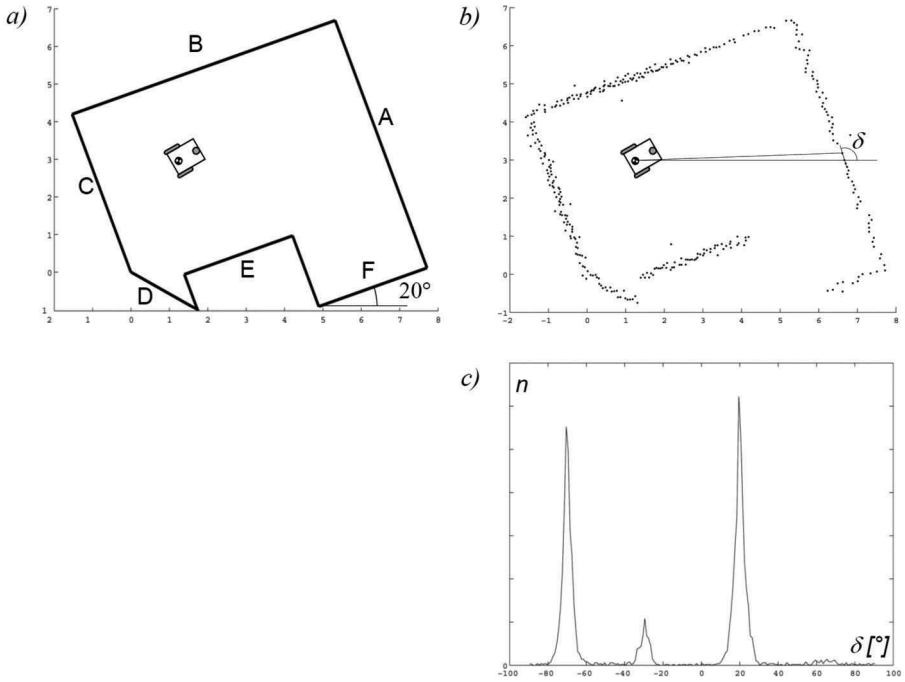


Figure 4.95
Angle histogram [329].

shown in figure 4.95c can be built. The uniform direction of the main walls are clearly visible as peaks in the angle histogram. Detection of peaks yields only two main peaks: one for each pair of parallel walls. This algorithm is very robust with regard to openings in the walls, such as doors and windows, or even cabinets lining the walls.

4.7.4 Extracting other geometric features

Line features are of particular value for mobile robots operating in man-made environments, where, for example, building walls and hallway walls are usually straight. In general, a mobile robot makes use of multiple features simultaneously, comprising a *feature set* that is most appropriate for its operating environment. For indoor mobile robots, the line feature is certainly a member of the optimal feature set.

In addition, other geometric kernels consistently appear throughout the indoor man-made environment. *Corner* features are defined as a point feature with an orientation. *Step discontinuities*, defined as a step change perpendicular to the direction of hallway travel,

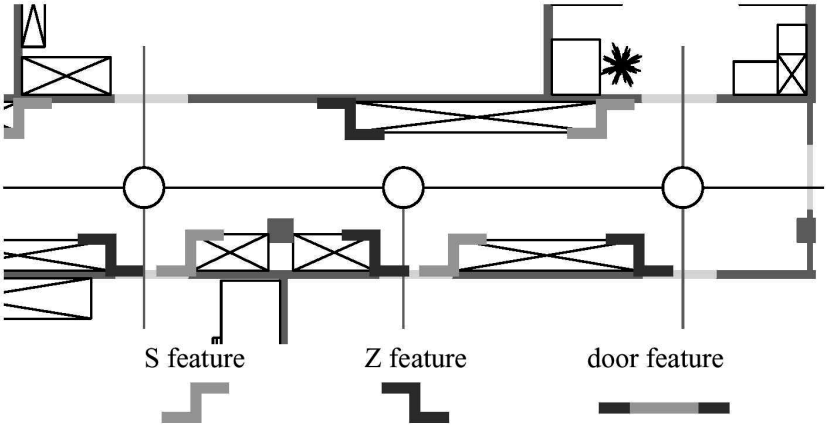


Figure 4.96
Multiple geometric features in a single hallway, including doorways and discontinuities in the width of the hallway.

are characterized by their form (convex or concave) and step size. *Doorways*, defined as openings of the appropriate dimensions in walls, are characterized by their width.

Thus, the standard segmentation problem is not so simple as deciding on a mapping from sensor readings to line segments, but rather it is a process in which features of different types are extracted based on the available sensor measurements. Figure 4.96 shows a model of an indoor hallway environment along with both indentation features (i.e., step discontinuities) and doorways.

Note that different feature types can provide quantitatively different information for mobile robot localization. The line feature, for example, provides two degrees of information, angle and distance. But the step feature provides 2D relative position information as well as angle.

The set of useful geometric features is essentially unbounded, and as sensor performance improves we can only expect greater success at the feature extraction level. For example, an interesting improvement upon the line feature described above relates to the advent of successful vision ranging systems (e.g., stereo cameras and time-of-flight cameras) and 3D laser rangefinder. Because these sensor modalities provide a full 3D set of range measurements, one can extract plane features in addition to line features from the resulting data set. Plane features are valuable in man-made environments due to the flat walls, floors, and ceilings of our indoor environments. Thus they are promising as another highly informative feature for mobile robots to use for mapping and localization. Some

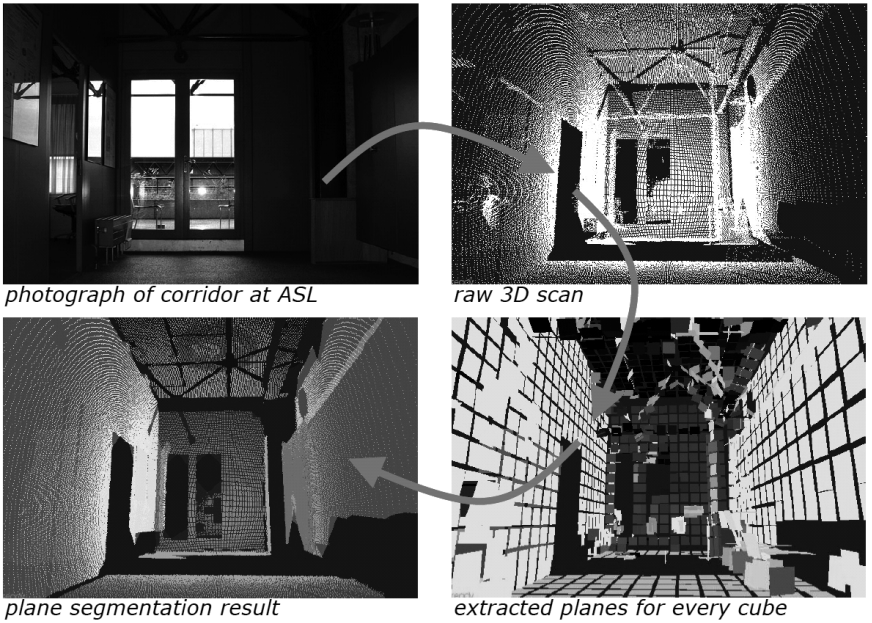


Figure 4.97 Extraction process of plane features: (Upper left) Photograph of the original environment. (Upper right) Raw 3D scan. (Bottom right) Plane feature segmentation and fitting. (Bottom left) final plane segmentation result. Image courtesy of J. Weingarten [331].

experiments using plane features have been done at the ASL (ETH Zurich) [331], the plane feature extraction process is illustrated in figure 4.97.

4.8 Problems

1. Consider an omnidirectional robot with a ring of eight 70 KHz sonar sensors that are fired sequentially. Your robot is capable of accelerating and decelerating at 50 cm/s^2 . It is moving in a world filled with sonar-detectable fixed (nonmoving) obstacles that can only be detected at 5 meters and closer. Given the bandwidth of your sonar sensors, compute your robot's appropriate maximum speed to ensure no collisions.

2. Design an optical triangulation system with the best possible resolution for the following conditions: specify b (as in figure 4.15):
(a) the system must have sensitivity of 1 cm at a range of 2 meters.
(b) The PSD has a sensitivity of 0.1 mm.
(c) f = 10 cm.

3. Identify a specific digital CMOS-based camera on the market. Using product specifications for this camera, collect and compute the following values. Show your derivations:
- Dynamic range
- Resolution (of a single pixel)
- Bandwidth

4. Stereo vision. Solve the system given by equations (4.60) and (4.61) and find the optimal point (x, y, z) that minimizes the distance between the optical rays passing through $\tilde{p}_l$ and $\tilde{p}_r$. For doing this, observe that these two equations define two distinct lines in the 3D space. The problems consists in rewriting these two equations as the difference between 3D points along these two lines. Then, impose that the partial derivatives of this distance with respect to λ_l and λ_r equal zero. From this, you will obtain the two points along the two lines at minimum distance between each other. The optimal point (x, y, z) can then be found as the middle point between those points.

5. Challenge Question.
Implement a basic two-view structure-from-motion algorithm from scratch:
(a) Implement the basic Harris corner detector in Matlab.
(b) Take two images of the same scene from different view points.
(c) Extract and match Harris features using SSD.
(d) Implement the 8-point algorithm to compute the essential matrix.
(e) Compute rotation and translation up to a scale from the essential matrix. Disambiguate the four solutions using the cheirality constraint.