two image regions match perfectly. In particular, in the case of the affine model the normalized cross-correlation becomes

$$\mathrm{NCC}(A, \boldsymbol{d}) = \frac{\sum_{W(\boldsymbol{x})} \left(I_1(\tilde{\boldsymbol{x}}) - \bar{I}_1\right)\left(I_2(A\tilde{\boldsymbol{x}} + \boldsymbol{d}) - \bar{I}_2\right)}{\sqrt{\sum_{W(\boldsymbol{x})} (I_1(\tilde{\boldsymbol{x}}) - \bar{I}_1)^2 \sum_{W(\boldsymbol{x})} (I_2(A\tilde{\boldsymbol{x}} + \boldsymbol{d}) - \bar{I}_2)^2}}. \tag{4.27}$$

So, we look for $(\hat{A}, \hat{\boldsymbol{d}}) = \arg\min_{A,\boldsymbol{d}} \mathrm{NCC}(A, \boldsymbol{d})$. In Chapter 11, we will combine NCC with robust statistics techniques to derive a practical algorithm that can match features between two images with a large baseline.

4.3.3 *Point feature selection*

In previous sections we have seen how to compute the translational or affine deformation of a photometric feature, and we have distinguished the case where the computation is performed at a fixed set of locations (optical flow) from the case where point features are tracked over time (feature tracking). One issue we have not addressed in this second case is how to initially select the points to be tracked. However, we have hinted on various occasions at the possibility of selecting as "feature points" the locations that allow us to solve the correspondence problem easily. In this section we make this more precise by giving a numerical algorithm to select such features.

As the reader may have noticed, the description of any of those feature points relies on knowing the gradient of the image. Hence, before we can give any numerical algorithm for feature selection, the reader needs to know how to compute the image gradient $\nabla I = [I_x, I_y]^T$ in an accurate and robust way. The description of how to compute the gradient of a discretized image is in Appendix 4.A.

The solution to the tracking or correspondence problem for the case of pure translation relied on inverting the matrix G made of the spatial gradients of the image (4.20). For G to be invertible, the region must have nontrivial gradients along two independent directions, resembling therefore a "corner" structure, as shown in Figure 4.5. Alternatively, if we regard the corner as the "intersection"

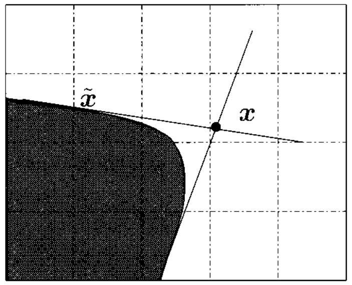

Figure 4.5. A corner feature $\boldsymbol{x}$ is the virtual intersection of local edges (within a window).

of all the edges inside the window, then the existence of at least a corner point

$\boldsymbol{x} = [x, y]^T$ means that over the window $W(\boldsymbol{x})$, the following minimization has a solution:

$$\min_{\boldsymbol{x}} E_c(\boldsymbol{x}) \doteq \sum_{\tilde{\boldsymbol{x}} \in W(\boldsymbol{x})} \left[\nabla I^T(\tilde{\boldsymbol{x}})(\tilde{\boldsymbol{x}} - \boldsymbol{x})\right]^2, \tag{4.28}$$

where $\nabla I(\tilde{\boldsymbol{x}})$ is the gradient calculated at $\tilde{\boldsymbol{x}} = [\tilde{x}, \tilde{y}]^T \in W(\boldsymbol{x})$. It is then easy to check that the existence of a local minimum for this error function is equivalent to the summation of the outer product of the gradients, i.e.

$$G(\boldsymbol{x}) = \sum_{\tilde{\boldsymbol{x}} \in W(\boldsymbol{x})} \nabla I(\tilde{\boldsymbol{x}}) \nabla I^T(\tilde{\boldsymbol{x}}) = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \in \mathbb{R}^{2\times 2}, \tag{4.29}$$

being nonsingular. If σ_2, the smallest singular value of G, is above a specified threshold τ, then G is invertible, (4.20) can be solved, and therefore, we say that the point $\boldsymbol{x}$ is a feature point. If both singular values of G are close to zero, the feature window has almost constant brightness. If only one of the singular values is close to zero, the brightness varies mostly along a single direction. In both cases, the point cannot be localized or matched in another image. This leads to a simple algorithm to extract point (or corner) features; see Algorithm 4.2.

Algorithm 4.2 (Corner detector).

Given an image $I(x, y)$, follow the steps to detect whether a given pixel (x, y) is a corner feature:

- set a threshold $\tau \in \mathbb{R}$ and a window W of fixed size, and compute the image gradient (I_x, I_y) using the filters given in Appendix 4.A;
- at all pixels in the window W around (x, y) compute the matrix
$$G = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}; \tag{4.30}$$
- if the smallest singular value $\sigma_2(G)$ is bigger than the prefixed threshold τ, then mark the pixel as a feature (or corner) point.

Although we have used the word "corner," the reader should observe that the test above guarantees only that the irradiance function I is "changing enough" in two independent directions within the window of interest. Another way in which this can happen is for the window to contain "sufficient texture," causing enough variation along at least two independent directions.

A variation to the above algorithm is the well-known Harris corner detector [Harris and Stephens, 1988]. The main idea is to threshold the quantity

$$C(G) = \det(G) + k \times \mathrm{trace}^2(G), \tag{4.31}$$

where $k \in \mathbb{R}$ is a (usually small) scalar, and different choices of k may result in favoring gradient variation in one or more than one direction, or maybe both. To see this, let the two eigenvalues (which in this case coincide with the singular

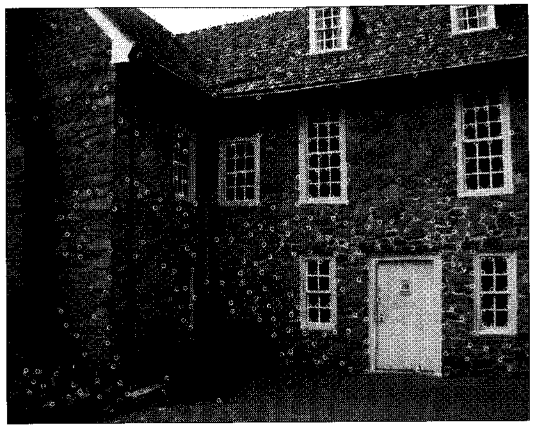

Figure 4.6. An example of the response of the Harris feature detector using 5×5 integration windows and parameter $k = 0.04$. Some apparent corners around the boundary of the image are not detected due to the size of window chosen.

values) of G be σ_1, σ_2. Then

$$C(G) = \sigma_1\sigma_2 + k(\sigma_1 + \sigma_2)^2 = (1 + 2k)\sigma_1\sigma_2 + k(\sigma_1^2 + \sigma_2^2). \tag{4.32}$$

Note that if k is large and either one of the eigenvalues is large, so will be $C(G)$. That is, features with significant gradient variation in at least one direction will likely pass a threshold. If k is small, then both eigenvalues need to be big enough to make $C(G)$ pass the threshold. In this case, only the corner feature is favored. Simple thresholding operations often do not yield satisfactory results, which lead to a detection of too many corners, which are not well localized. Partial improvements can be obtained by searching for the local minima in the regions, where the response of the detector is high. Alternatively, more sophisticated techniques can be used, which utilize contour (or edge) detection techniques and indeed search for the high curvature points of the detected contours [Wuescher and Boyer, 1991]. In Chapter 11 we will explore further details that are crucial in implementing an effective feature detection and selection algorithm.

4.4 Tracking line features

As we will see in future chapters, besides point features, line (or edge) features, which typically correspond to boundaries of homogeneous regions, also provide important geometric information about the 3-D structure of objects in the scene. In this section, we study how to extract and track such features.

4.4.1 *Edge features and edge detection*

As mentioned above, when the matrix G in (4.29) has both singular values close to zero, it corresponds to a textureless "blank wall." When one of the singular values is large and the other one is close to zero, the brightness varies mostly along a single direction. But that does not imply a sudden change of brightness value in the direction of the gradient. For example, an image of a shaded marble sphere does vary in brightness, but the variation is smooth, and therefore the entire surface is better interpreted as one smooth region instead of one with edges everywhere. Thus, by "an edge" in an image, we typically refer to a place where there is a distinctive "peak" in the gradient. Of course, the notion of a "peak" depends on the resolution of the image and the size of the window chosen. What appears as smooth shading on a small patch in a high-resolution image may appear as a sharp discontinuity on a large patch in a subsampled image.

We therefore label a pixel $\boldsymbol{x}$ as an "edge feature" only if the gradient norm $\|\nabla I\|$ reaches a local maximum compared to its neighboring pixels. This simple idea results in the well-known Canny edge-detection algorithm [Canny, 1986].

Algorithm 4.3 (Canny edge detector).

Given an image $I(x, y)$, follow the steps to detect whether a given pixel (x, y) is on an edge

- set a threshold $\tau > 0$ and standard deviation $\sigma > 0$ for the Gaussian function g_σ used to derive the filter (see Appendix 4.A for details);
- compute the gradient vector $\nabla I = [I_x, I_y]^T$ (see Appendix 4.A);
- if $\|\nabla I(x, y)\|^2 = \nabla I^T \nabla I$ is a local maximum along the gradient and larger than the prefixed threshold τ, then mark it as an edge pixel.

Figure 4.7 demonstrates edges detected by the Canny edge detector on a gray-level image.

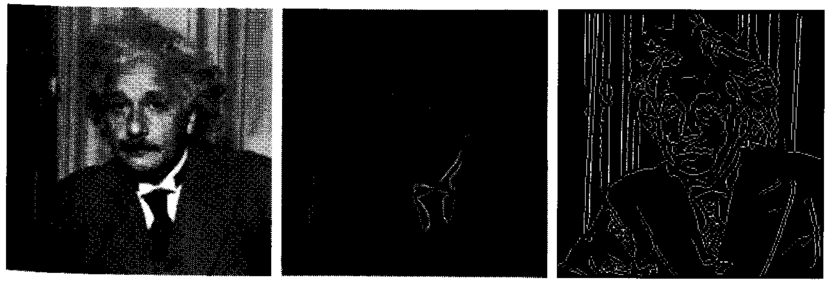

Figure 4.7. Original image, gradient magnitude, and detected edge pixels of an image of Einstein.

The next sections follow the program above, with an eye toward implementation. For the sake of completeness, the description of some of the algorithms is repeated from previous chapters.

11.1 Feature selection

Given a collection of images, for instance a video sequence captured by a handheld camcorder, the first step consists of selecting candidate features in one or more images, in preparation for tracking or matching them across different views. For simplicity we concentrate on the case of point features, and discuss the case of lines only briefly (the reader can refer to Chapter 4 for more details).

The quality of a point with coordinates $\boldsymbol{x} = [x, y]^T$ as a candidate feature can be measured by Harris' criterion,

$$C(\boldsymbol{x}) = \det(G) + k \times \operatorname{trace}^2(G), \tag{11.1}$$

defined in Chapter 4, equation (4.31), computed on a region $W(\boldsymbol{x})$, for instance a rectangular window centered at $\boldsymbol{x}$, of size between 3×3 and 11×11 pixels; we use 7×7 in this example. In the above expression, k is a constant to be chosen by the designer,[2] and G is a 2×2 matrix that depends on $\boldsymbol{x}$, given by

$$G = \begin{bmatrix} \sum_{W(\boldsymbol{x})} I_x^2 & \sum_{W(\boldsymbol{x})} I_x I_y \\ \sum_{W(\boldsymbol{x})} I_x I_y & \sum_{W(\boldsymbol{x})} I_y^2 \end{bmatrix} \in \mathbb{R}^{2\times 2},$$

where I_x, I_y are the gradients obtained by convolving the image I with the derivatives of a pair of Gaussian filters (Section 4.A). A point feature $\boldsymbol{x}$ is selected if $C(\boldsymbol{x})$ exceeds a certain *threshold* τ. Selection based on a single global threshold, however, is not a good idea, because one region of the image may contain objects with strong texture, whereas another region may appear more homogeneous and therefore it may not trigger any selection (see Figure 11.3). Therefore, we recommend partitioning the image into *tiles* (e.g., 10×10 regions of 64×48 pixels each, for a 640×480 image), sorting the features according to their quality $C(\boldsymbol{x})$ in each region, and then selecting as many features as desired, provided that they exceed a minimum threshold (to avoid forcibly selecting features where there are none). For instance, we typically start with selecting about 200 to 500 point features.

In addition, to avoid associating multiple features with the same point, we also impose a *minimum separation* between features. Consider, for instance, a white patch with a black dot in the middle. Every window of size, say, 11×11, centered at a point within 5 pixels of the black dot, will satisfy the requirements above and pass the threshold. However, we want to select only *one* point feature for this dot. Therefore, once the best point feature is selected, according to the criterion $C(\boldsymbol{x})$, we need to "suppress" feature selection in its neighborhood.

[2]A value that is often used is $k = 0.03$, which is empirically verified to yield good results.

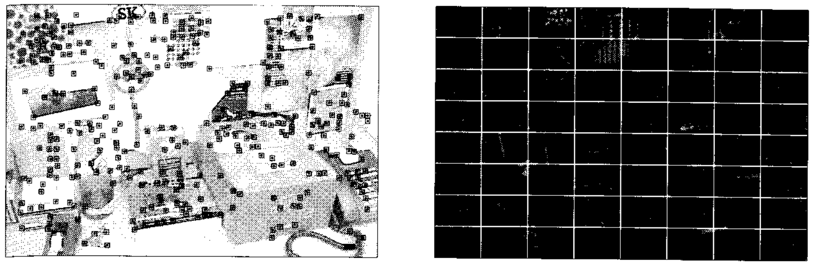

Figure 11.3. Examples of detected features (left) and quality criterion $C(\boldsymbol{x})$ (right). As it can be seen, certain regions of the image "attract" more features than others. To promote uniform selection, we divide the image into tiles and select a number of point features in each tile, as long as they exceed a minimum threshold.

The choice of quality criterion, window size, threshold, tile size, and minimum separation are all part of the design process. There is no right or wrong choice at this stage, and one should experiment with various choices to obtain the best results on the data at hand. The overall feature selection process is summarized in Algorithm 11.1.

Algorithm 11.1 (Point feature detection).

1. Compute image gradient $\nabla I = [I_x, I_y]^T$ as in Section 4.A.
2. Choose a size of the window $W(\boldsymbol{x})$ (e.g., 7×7). Compute the quality of each pixel location $\boldsymbol{x}$, using the quality measure $C(\boldsymbol{x})$ defined in equation (11.1).
3. Choose a threshold τ; sort all locations $\boldsymbol{x}$ that exceed the threshold $C(\boldsymbol{x}) > \tau$, in decreasing order of $C(\boldsymbol{x})$.
4. Choose a tile size (e.g., 64×48). Partition the image into tiles. Within each tile, choose a minimum separation space (e.g., 10 pixels) and the maximum number of features to be selected within each tile (e.g., 5). Select the highest-scoring feature and store its location. Go through the list of features in decreasing order of quality; if the feature does not fall within the minimum separation space of any previously selected features, then select it. Otherwise, discard it.
5. Stop when the number of selected features has exceeded the maximum, or when all the features exceeding the threshold have been discarded.

Further issues

In order to further improve feature localization one can interpolate the function $C(\boldsymbol{x})$ *between pixels*, for instance using quadratic polynomial functions, and choose the point $\boldsymbol{x}$ that maximizes it. In general, the location of the maximum will not be exactly on the pixel grid, thus yielding *subpixel accuracy* in feature localization, at the expense of additional computation. In particular, the approximation of C via a quadratic polynomial can be written as $C(\tilde{\boldsymbol{x}}) =$

$a\tilde{x}^2 + b\tilde{y}^2 + c\tilde{x}\tilde{y} + d\tilde{x} + e\tilde{y} + f$ for all $\tilde{x} \in W(x)$. As long as the window chosen is of size greater than 3×3, we can find the minimum of $C(\tilde{x})$ in $W(x)$ with respect to a, b, c, d, e and f using linear least-squares as described in Appendix A.

A similar procedure can be followed to detect line segments. The Hough transform is often used to map line segments to points in the transformed space. More details can be found in standard image processing textbooks such as [Gonzalez and Woods, 1992]. Once line segments are detected, they can be clustered into longer line segments that can then be used for matching. Since matching line segments is computationally more involved, we do not emphasize it here and refer the reader to [Schmidt and Zisserman, 2000] instead.

11.2 Feature correspondence

Once candidate point features are selected, the goal is to track or match them across different images. We first address the case of small baseline (e.g., when the sequence is taken from a moving video camera), and then the case of moderate baseline (e.g., when the sequence is a collection of snapshots taken from disparate vantage points).

11.2.1 Feature tracking

We first describe the simplest feature-tracking algorithm for small interframe motion based on a purely translational model. We then describe a more elaborate but effective tracker that also compensates for contrast and brightness changes.

Translational motion model: the basic tracker

The displacement $\boldsymbol{d} \in \mathbb{R}^2$ of a feature point of coordinates $\boldsymbol{x} \in \mathbb{R}^2$ between consecutive frames can be computed by minimizing the sum of squared differences (SSD) between the two images $I_i(\boldsymbol{x})$ and $I_{i+1}(\boldsymbol{x} + \boldsymbol{d})$ in a small window $W(\boldsymbol{x})$ around the feature point $\boldsymbol{x}$. For the case of two views, $i = 1$ and $i + 1 = 2$, we can look for the displacement $\boldsymbol{d}$ that solves the following minimization problem (for multiple views taken by a moving camera, we will consider correspondence of two views at a time)

$$\min_{\boldsymbol{d}} E(\boldsymbol{d}) \doteq \sum_{\tilde{\boldsymbol{x}} \in W(\boldsymbol{x})} \left[I_2(\tilde{\boldsymbol{x}} + \boldsymbol{d}) - I_1(\tilde{\boldsymbol{x}}) \right]^2. \tag{11.2}$$

As we have seen in Chapter 4, the closed-form solution to this problem is given by

$$\boldsymbol{d} = -G^{-1}\boldsymbol{b}, \tag{11.3}$$

where

$$G \doteq \begin{bmatrix} \sum_{W(\boldsymbol{x})} I_x^2 & \sum_{W(\boldsymbol{x})} I_x I_y \\ \sum_{W(\boldsymbol{x})} I_x I_y & \sum_{W(\boldsymbol{x})} I_y^2 \end{bmatrix}, \quad \boldsymbol{b} \doteq \begin{bmatrix} \sum_{W(\boldsymbol{x})} I_x I_t \\ \sum_{W(\boldsymbol{x})} I_y I_t \end{bmatrix},$$

and $I_t \doteq I_2 - I_1$ is an approximation of the temporal derivative,[3] computed as a first-order difference between two views. Notice that G is the same matrix we have used to compute the quality index of a feature in the previous section, and therefore it is guaranteed to be invertible, although for tracking we may want to select a different window size and a different threshold. In order to obtain satisfactory results, we need to refine this primitive scheme in a number of ways.

First, when the displacement of features between views exceeds 2 to 3 pixels, first-order differences of pixel values cannot be used to compute temporal derivatives, as we have suggested doing for I_t just now. The proposed tracking scheme needs to be implemented in a *multiscale* fashion. This can be done by constructing a "pyramid of images," by smoothing and downsampling the original image, yielding, say, I^1, I^2, I^3, and I^4, of size 640×480, 320×240, 160×120, 80×60, respectively.[4] Then, the basic scheme just described is first applied at the coarsest level to the pair of images (I_1^4, I_2^4), resulting in an estimate of the displacement $\boldsymbol{d}^4 = -G^{-1}\boldsymbol{b}$. This displacement is scaled up (by a factor of two) and the window $W(\boldsymbol{x})$ is moved to $W(\boldsymbol{x} + 2\boldsymbol{d}^4)$ at the next level (I^3) via a warping of the image[5] $\tilde{I}_2^3(\boldsymbol{x}) \doteq I_2^3(\boldsymbol{x} + 2\boldsymbol{d}^4)$. We then apply the same scheme to the pair $(I_1^3, \tilde{I}_2^3)$ in order to estimate the displacement $\boldsymbol{d}^3$. The algorithm is then repeated until the finest level, where it is applied to the pair $I_1^1(\boldsymbol{x})$ and $\tilde{I}_2^1(\boldsymbol{x}) \doteq I_2^1(\boldsymbol{x} + 2\boldsymbol{d}^2)$. Once the displacement $\boldsymbol{d}^1$ is computed, the total estimated displacement is given by $\boldsymbol{d} \doteq \boldsymbol{d}^1 + 2\boldsymbol{d}^2 + 4\boldsymbol{d}^3 + 8\boldsymbol{d}^4$. In most sequences captured with a video camera, two to four levels of the pyramid are typically sufficient.

Second, in the same fashion in which we have performed the iteration across different scales (i.e. by warping the image using the estimated displacement and reiterating the algorithm), we can perform the iteration repeatedly at the finest scale: In this iteration, $\boldsymbol{d}^{i+1}$ is computed between $I_1(\boldsymbol{x})$ and the warped interpolated[6] image $\tilde{I}_2^i(\boldsymbol{x}) = I_2(\boldsymbol{x} + \boldsymbol{d}^1 + \cdots + \boldsymbol{d}^i)$. Typically, 5 to 6 iterations of this kind[7] are sufficient to yield a localization error of a tenth of a pixel with a window

[3]A better approximation of the temporal derivative can be computed using the derivative filter, which involves three images, following the guidelines of Appendix 4.A.

[4]A more detailed description of multiscale image representation can be found in [Simoncelli and Freeman, 1995] and references therein.

[5]Notice that interpolation of the brightness values of the original image is necessary: Even if we assume that point features were detected at the pixel level (and therefore $\boldsymbol{x}$ belongs to the pixel grid), there is no reason why $\boldsymbol{x} + \boldsymbol{d}$ should belong to the pixel grid. In general, $\boldsymbol{d}$ is not an integer, and therefore, at the next level, all the intensities in the computation of the gradients in G and $\boldsymbol{b}$ must be interpolated outside the pixel grid. For our purposes here, it suffices to use standard linear or bilinear interpolation schemes.

[6]As usual, the image must be interpolated outside the pixel grid in order to allow computation of G and $\boldsymbol{b}$.

[7]This type of iteration is similar in spirit to Newton-Raphson methods.